9900

Family Systems Design

and Data Book

MICROPROCESSOR SERIES™

1st Edition

LEARNING CENTER

TEXAS INSTRUMENTS

9900

Family Systems Design

and Data Book

MICROPROCESSOR SERIES™

Your Local Sales Office:
Texas Instruments Ltd.,
Supply Division,
St. James House,
Wellington Road North,
Stockport, SK4 2RT.
'phone: 061-432 0645. Telex: 667478.

1st Edition

Basic Decisions in System Design 1

Product Selection Guide 2

A First Encounter:
Getting Your Hands on a 9900 3

Hardware Design:
Architecture and Interfacing Techniques 4

Software Design:
Programming Methods and Techniques 5

Instruction Set 6

Program Development: Software Commands—
Descriptions and Formats 7

Product Data Book 8

Applications 9

Glossary, Appendix

9900 Family Products

PRODUCT AND AVAILABILITY INFORMATION:

- Contact a TI Sales Office or authorized Distributor.

Additional copies of the 9900 Family System Design Book LCC4400:

- Contact an authorized distributor, or
  send purchase order to:

  Texas Instruments Incorporated
  P.O. Box 225012 M/S 54
  Dallas, Texas 75265

Complete worldwide listing of TI Sales Offices and Distributors
in the appendix on pages A-15 and A-16.

9900 Family Systems Design

and Data Book

FIRST EDITION

By:

William D. Simpson, MSEE, Staff Consultant
Gerald Luecke, MSEE, Manager of Technical Product Development
Don L. Cannon, PhD., Staff Consultant
David H. Clemens, Staff Consultant

Texas Instruments Learning Center

and
The Engineering Staff of Texas Instruments Incorporated

TEXAS INSTRUMENTS
INCORPORATED

P.O. Box 1443 M/S 6404
Houston, Texas 77001

LCC4400
97049-118-NI

Printed in U.S.A.

Acknowledgement:

Many members of the engineering and marketing staff of Texas Instruments Incorporated have contributed previously authored materials for the content of this book. The contributions, which are significant but are too numerous to identify individually, have been edited and combined with original authored material into the present book written by the Texas Instruments Learning Center and its staff consultants.

A final review and edit was done by the 9900 Family marketing and engineering staffs.

Design and artwork by:

Schenck, Plunk & Deason

ISBN 0-89512-026-7
Library of Congress Catalog Number: 78-058005

IMPORTANT

Texas Instruments makes no warranty, either express or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding these materials and makes such materials available solely on an "as-is" basis.

In no event shall Texas Instruments be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of the purchase or use of these materials and the sole and exclusive liability to Texas Instruments, regardless of the form of action, shall not exceed the purchase price of this book.

Copyright © 1978 Texas Instruments Incorporated. All rights reserved.

Unless otherwise noted, this publication, or parts thereof, may not be reproduced in any form by photographic, electrostatic, mechanical, or any other method, for any use, including information storage and retrieval.

For condition of use and permission to use materials contained herein for publication in other than the English language, apply to Texas Instruments Incorporated.

For permissions and other rights under this copyright, please write Texas Instruments Learning Center, P.O. Box 225012 MS-54, Dallas, Texas 75265.

TABLE OF CONTENTS

Chapter 1. Basic Decisions in System Design 1-1

Introduction 1-2
The Impact of Semiconductors 1-3
Applications of Programmable Semiconductors 1-11
Single-Chip Microcomputer Applications 1-12
Multi-Chip Microcomputer Applications 1-13
Building a Microprocessor Based System 1-15
Basic Hardware Components 1-15
Programming for Microcomputers 1-17
Which Microprocessor or Microcomputer to Use 1-20
Evolution of Memory-to-Memory Architecture 1-22
Getting up to Speed on Microprocessors 1-30
Bibliography 1-31
Books 1-31
Articles 1-32
List of Periodicals to be Monitored 1-32

Chapter 2. Product Selection Guide 2-1

The 9900 Family — What Is It? 2-2
Family Overview 2-2
The Hardware Family 2-2
The Software and Development Systems Support 2-2
Typical Applications 2-8
Hardware Selection 2-9
The Component Route: CPU 2-10
CPU Selection 2-12
Flexible I/O 2-14
Family Members Fitted to the Application 2-15
Interrupt Flexibility 2-15
Advantages of 9900 Family CPUs 2-15
True Compatibility 2-15
Lower Costs 2-15
Instruction Set 2-16
Memory-to-Memory Architecture 2-16
The Component Route: Peripherals 2-17
Interface Techniques 2-18
Serial I/O for Data Communications 2-18
Parallel I/O 2-18
Clock and Support Logic 2-21
Cost Effectiveness of NMOS LSI 2-22
CRU Interface 2-23
The Component Route: Memory 2-23
The Component Route: Miscellaneous Components 2-28
The Modular Route: Microcomputer Modules 2-29
The Minicomputer Route 2-34

TABLE OF CONTENTS

9900 Family Software and Development Systems . . . 2-36
Importance of Software . . . 2-36
Software Development Systems . . . 2-37
Support Software and Firmware . . . 2-46

Chapter 3. A First Encounter: Getting Your Hands on a 9900 . . . 3-1

Purpose . . . 3-2
Where to Begin . . . 3-2
What You Have . . . 3-3
Getting It Together . . . 3-5
Unpacking and Checking the Microcomputer (TM 990/100M-1) . . . 3-13
Connecting the Microterminal TM 990/301 . . . 3-15
Operating the Microcomputer . . . 3-15
Telling the Microcomputer What to Do . . . 3-18
How Was It Done? . . . 3-24
Back to Basics . . . 3-25
Registers . . . 3-26
Workspace . . . 3-28
SBZ and SBO Instructions . . . 3-30
I/O Selection . . . 3-32
TB Instruction . . . 3-34
Idea to Flowchart . . . 3-35
Flow Charts . . . 3-36
WAIT Subroutine . . . 3-38
Subroutine Jump . . . 3-39
A Loop Within the WAIT Subroutine . . . 3-41
Loading a Register for the Time Delay . . . 3-42
Where Does the Program Start? . . . 3-43
Writing the Program . . . 3-45
WAIT Subroutine Call . . . 3-46
Return from WAIT Subroutine . . . 3-47
Writing the Machine Code . . . 3-52
Immediate Instructions . . . 3-52
Instructions SBO, SBZ . . . 3-54
Instruction BL . . . 3-55
Miscellaneous Instructions . . . 3-55
Jump Instructions . . . 3-56
Summary . . . 3-59
Assembly Language Program: Table 3-2 . . . 3-60

Chapter 4. Hardware Design: Architecture and Interfacing Techniques . . . 4-1

Introduction . . . 4-2
Architecture . . . 4-5
Basic Microprocessor Chip . . . 4-5
Microprocessor Registers . . . 4-5
Memory-to-Memory Architecture . . . 4-9
Context Switching . . . 4-11

Memory 4-12
Memory Organization 4-13
Memory Control Signals 4-15
Static Memory 4-23
Dynamic Memory 4-25
Buffered Memory 4-28
Memory Parity 4-28
Memory Layout 4-30
Instruction Execution 4-32
Timing 4-32
Cyclic Operation 4-35
Input/Output 4-42
Direct Memory Access 4-42
Memory Mapped I/O 4-43
Communications Register Unit (CRU) 4-45
CRU Interface 4-46
CRU Interface Logic 4-46
Expanding CRU I/O 4-47
CRU Machine Cycles 4-47
CRU Data Transfer 4-51
CRU Paper Tape Reader Interface 4-54
Burroughs SELF-SCAN Display Interface 4-57
Interrupts 4-59
Reset 4-60
Load 4-61
Basic Machine Cycle 4-63
Maskable Interrupts 4-64
Interrupt Service 4-64
Interrupt Signals 4-66
Interrupt Masking 4-67
Interrupt Processing Example 4-70
Electrical Requirements 4-71
Understanding the Electrical Specifications 4-71
Detailed Electrical Interface Specifications (TMS 9900) 4-75
TMS 9900 Clock Generation 4-75
TMS 9900 Signal Interfacing 4-78
TMS 9940 Microcomputer 4-82
Pin Assignments and Functional Control 4-83
Interrupts 4-83
Decrementer 4-86
CRU Implementation 4-86
Multiprocessor System Interface (MPSI) 4-87
Summary 4-88

TABLE OF CONTENTS

Complete Listing of Machine Cycles . . . 4-89
- Machine Cycles . . . 4-89
- 9900 Machine Cycle Sequences . . . 4-90
- Terms and Definitions . . . 4-90
- Data Derivation Sequences . . . 4-91
  - Workspace Register . . . 4-91
  - Workspace Register Indirect . . . 4-91
  - Workspace Register Indirect Auto-Increment (Byte Operand) . . . 4-91
  - Workspace Register Indirect Auto-Increment (Word Operand) . . . 4-91
  - Symbolic . . . 4-92
  - Indexed . . . 4-92

Instruction Execution Sequences . . . 4-92
- A, AB, C, CB, S, SB, SOC, SOCB, SZC, SZCB, MOV, MOVB, COC, CZC, XOR . . . 4-92
- MPY (multiply) . . . 4-93
- DIV (divide) . . . 4-94
- XOP . . . 4-94
- CLR, SETO, INV, NEG, INC, INCT, DEC, DECT, SWPB . . . 4-95
- ABS . . . 4-95
- X . . . 4-96
- B . . . 4-96
- BL . . . 4-96
- BLWP . . . 4-97
- LDCR . . . 4-97
- STCR . . . 4-98
- SBZ, SBO . . . 4-99
- TB . . . 4-99
- JEQ, JGT, JH, JHE, JL, JLE, JLT, JMP, JNC, JNE, JNO, JOC, JOP . . . 4-99
- SRA, SLA, SRL, SRC . . . 4-100
- AI, ANDI, ORI . . . 4-100
- CI . . . 4-101
- LI . . . 4-101
- LWPI . . . 4-101
- LIMI . . . 4-101
- STWP, STST . . . 4-102
- CKON, CKOF, LREX, RSET . . . 4-102
- IDLE . . . 4-102
- RTWP . . . 4-103

Machine-Cycle Sequences in Response to External Stimuli . . . 4-103
- RESET . . . 4-103
- LOAD . . . 4-104
- Interrupts . . . 4-105

Timing . . . 4-105

Chapter 5. Software Design: Programming Methods and Techniques 5-1

9900 Architecture 5-2
Instruction Register and Cycle 5-4
Program Counter (PC) 5-6
Status Register (ST) 5-7
Workspace Pointer (WP) 5-8
Program Environment or Context 5-9
Memory Organization 5-10
ROM/RAM Partitioning 5-11
Reserved Memory 5-11
Workspace Utilization 5-12
The Workspace Concept and Uses 5-12
Dedicated Areas of Workspaces 5-15
Workspace Location 5-17
Subroutine Techniques 5-19
Types of Subroutines 5-19
Parameter Passing 5-25
Multiple Level Shared Workspace Subroutines 5-28
Shared Workspace Mapping 5-28
Re-entrant Programming 5-30
Programming Tasks 5-33
Initialization 5-33
Masking and Testing 5-35
Arithmetic Operations 5-38
Input/Output 5-43
Memory Mapped Input/Output 5-43
CRU Input/Output 5-44
Input/Output Methods 5-45

Chapter 6. Instruction Set 6-1

Software Features of the 9900 6-2
Processor Registers and System Memory 6-2
Program Counter 6-3
Workspace 6-3
Status Register 6-4
Addressing Modes 6-6
Workspace Register Addressing 6-6
Workspace Register Indirect Addressing 6-6
Workspace Register Indirect Addressing with Autoincrement 6-6
Symbolic or Direct Addressing 6-7
Indexed Addressing 6-8
Special Addressing Modes 6-8
Assembly Language Programming Information 6-10
Assembly Language Formats 6-10
Terms and Symbols 6-11
Survey of the 9900 Instruction Set 6-13

TABLE OF CONTENTS

Instruction Descriptions 6-16
Data Transfer Instructions 6-18
LI Load Immediate 6-18
LIMI Load Interrupt Mask Immediate 6-18
LWPI Load Workspace Pointer Immediate 6-19
MOV Move Word 6-19
MOVB Move Byte 6-20
SWPB Swap Bytes 6-21
STST Store Status 6-21
STWP Store Workspace Pointer 6-22
Arithmetic Instructions 6-23
A Add Words 6-23
AB Add Bytes 6-24
AI Add Immediate 6-25
S Subtract Words 6-25
SB Subtract Bytes 6-26
INC Increment 6-27
INCT Increment by Two 6-27
DEC Decrement 6-28
DECT Decrement by Two 6-28
NEG Negate 6-29
ABS Absolute Value 6-29
MPY Multiply 6-30
DIV Divide 6-31
Comparison Instructions 6-32
C Compare Words 6-32
CB Compare Bytes 6-33
CI Compare Immediate 6-34
COC Compare Ones Corresponding 6-34
CZC Compare Zeroes Corresponding 6-35
Logic Instructions 6-36
ANDI AND Immediate 6-36
ORI OR Immediate 6-37
XOR Exclusive OR 6-38
INV Invert 6-38
CLR Clear 6-39
SETO Set to One 6-39
SOC Set Ones Corresponding 6-40
SOCB Set Ones Corresponding, Byte 6-40
SZC Set to Zeroes Corresponding 6-41
SZCB Set to Zeroes Corresponding, Bytes 6-42
Shift Instructions 6-43
SRA Shift Right Arithmetic 6-43
SLA Shift Left Arithmetic 6-44
SRL Shift Right Logical 6-45
SRC Shift Right Circular 6-45

Unconditional Branch Instructions 6-46
B Branch 6-46
BL Branch and Link. 6-47
BLWP Branch and Load Workspace Pointer 6-48
XOP Extended Operation 6-49
RTWP Return with Workspace Pointer 6-50
JMP Unconditional Jump. 6-50
X Execute. 6-51
Conditional Jump Instructions. 6-52
JH, JL, JHE, JLE, JGT, JLT, JEQ, JNE,
JOC, JNC, JNO, JOP 6-52
CRU Instructions 6-53
SBO Set Bit to Logic One 6-53
SBZ Set Bit to Logic Zero 6-54
TB Test Bit 6-54
LDCR Load CRU U. 6-55
STCR Store CRU 6-57
Control Instructions 6-58
LREX, CKOF, CKON, RSET, IDLE, ... 6-58
Special Features of the 9940. 6-59
LIIM Load Immediate Interrupt Mask 6-59
XOP Extended Operation 6-60
DCA Decimal Correct Addition 6-61
DCS Decimal Correct Subtraction 6-62

Chapter 7. Program Development: Software Commands — Descriptions and Formats 7-1

Introduction 7-2
Assembly Language Programming: Formats and Directives 7-3
Assembly Language Application 7-4
Assembly Language Formats 7-4
Terms and Symbols 7-6
Assembler Directives. 7-8
Program Linkable Directives 7-15
Assembler Output 7-15
9900 Reference Data 7-17
TM990/402 Line-by-Line Assembler User's Guide 7-25
General 7-26
Installation 7-26
Operation 7-27
Setup 7-27
Inputs to Assembler 7-27
Exiting to the Monitor 7-32
Pseudo-Instructions. 7-32
TIBUG Monitor 7-33
TM990/302 Software Development Board 7-37
TXDS Commands for FS990 Software Development System 7-43
AMPL Reference Data 7-50
POWER BASIC MP 307 7-66
Cross Support 7-76

TABLE OF CONTENTS

Chapter 8. Product Data Book 8-1

Introduction 8-2
Family Description 8-2
Common Key Features 8-3
Key Features of Specific Devices 8-4
Organization of CPU Data Manuals and Instruction Set 8-5
TMS 9900 8-6
Architecture 8-7
TMS 9900 Electrical Specifications 8-24
TMS 9900-40 Electrical Specifications 8-28
SBP 9900A 8-30
Architecture 8-31
Interfacing 8-47
Power Source 8-50
Electrical Specifications 8-52
TMS 9980A/TMS 9981 8-56
Architecture 8-58
Electrical Specifications 8-82
TMS 9940 8-85
Architecture 8-87
Instruction Set 8-103
EPROM Programming 8-106
Test Function 8-109
Electrical Specifications 8-109
Design Support 8-111
TMS 9985 — A Summary 8-113
9900 Instruction Set 8-125
Peripheral and Interface Circuits 8-137
TMS 9901 Programmable Systems Interface 8-138
TMS 9902 Asynchronous Communications Controller 8-160
TMS 9903 Synchronous Communications Controller 8-194
TIM 9904 Four-Phase Clock Generator Driver 8-249
TIM 9905 Data Selectors/Multiplexers 8-263
TIM 9906 8-Bit Addressable Latches 8-266
TIM 9907 8-Line-to-3-Line Priority Encoder 8-269
TIM 9908 8-Line-to-3-Line Priority Encoder 8-274
TMS 9909 Floppy Disk Controller 8-277
TMS 9911 Direct Memory Access Controller 8-282
TMS 9914 General Purpose Interface Bus Adapter 8-288
TMS 9915 Memory Timing (and Refresh) Controller Chip Set 8-296
TMS 9927 Video Timer/Controller 8-297
TMS 9932 Combination ROM/RAM Memory 8-302
TMS 3064 65,536-Bit CCD Memory 8-307
TMS 6011 Asynchronous Data Interface (UART) 8-307

SBP 9960 I/O Expander 8-308
SBP 9961 Interrupt Controller/Timer 8-316
SBP 9964 Timing Controller for the SBP 9900A 8-329
SBP 9965 Peripheral Interface Adapter 8-330
TM 990 Series Microcomputer Modules 8-331
TM 990/100M Microcomputer 8-338
TM 990/101 Microcomputer 8-342
TM 990/180 Microcomputer 8-346
TM 990/201 Memory Expansion Board 8-351
TM 990/206 Expansion Memory Board 8-356
TM 990/310 I/O Expansion Module 8-360
TM 990/301 Microterminal 8-364
TM 990/401 TIBUG 8-368
TM 990/402 Line-by-Line Assembler 8-369
TM 990/500 Accessories 8-370
TM 990/189 University Microcomputer Board 8-373
Memory 8-375
Dynamic Random-Access Memory (RAM) 8-376
Static Random-Access Memory (RAM) 8-380
Read-Only Memory (ROM) 8-384
Programmable ROM 8-387
Eraseable Programmable ROM 8-389
Mechanical Data 8-391
Software 8-397
TMSW101MT Transportable Cross-Support 8-398
TM 990/302 Software Development Board 8-400
TM 990/40DS Design Aid for TMS 9940 Microcomputer 8-406
TM 990/450, 451, 452, TMSW201F POWER BASIC Family 8-409

Chapter 9. Applications 9-1

A Simulated Industrial Control Application (With TM990/100M Microcomputer and 5MT I/O Modules) 9-3
A Low Cost Data Terminal 9-75
TMS 9900 Floppy Disk Controller 9-89

Glossary G-1

Appendix A-1

CHAPTER 1

Basic Decisions In System Design

1

INTRODUCTION

1 Texas Instruments has developed and is manufacturing a family of microprocessor products and systems based on the architecture of its 990 minicomputer. The purpose of this book is to present enough factual information about the 9900 and the family of devices and systems surrounding it to serve not only as a guide for deciding to use the 9900 in an application, but also as the primary reference for design and programming activities. The book is much more than a data book or a collection of application notes. It contains basic concepts, presents methods and techniques, and most important of all, shows how the architecture of the 9900, substantially superior to other microprocessor architectures, can be translated into cost effective applications.

The time investment you make in learning how to use the 9900 will inevitably produce substantial benefits because your designs will be advanced well beyond other microprocessor systems; they will be expandable, flexible, easily upgraded and will not be obsolescent. The capital investment in programming systems will bring powerful computing equipment and software tools to your design team that will have them out-distancing the competition in a very short time.

In reading this book, you will see the 9900 product as more than a single microprocessor. You will find a family of processors, peripherals, boards, minicomputers and systems all based on a single architectural concept called *memory-to-memory architecture.* It is this basic principle which, when fully understood at the fundamental level, will help you understand why and how the 9900 can be used to implement outstanding products. In addition, you will see why Texas Instruments has made the commitment to the continued support of the 9900 family in both hardware and software. New microprocessors and peripheral devices will retain and complement the basic architectural features—the 16-bit word length, the instruction set, the I/O techniques. etc. Texas Instruments software support goes beyond the standard assembler, editor, linker and PROM programmer software. New design tools such as POWER BASIC and PASCAL are now available. These powerful software products bring structured programming disciplines into focus and help you to attain an advanced programming capability.

All in all, the book is a collection of useful factual material which should be of substantial benefit to anyone considering designing with microprocessors. For those who have very limited exposure to designing with semiconductor products, the next few sections will be helpful because the theme of "more functions at lower cost" is demonstrated. These ideas lead to the basic philosophy that designing with *standard hardware* — semiconductor LSI products which are *programmable* — is the most economical procedure, and should be carefully considered for *every* new electronic product.

THE IMPACT OF SEMICONDUCTORS

In the short thirty years since the invention of the transistor (the first semiconductor device to exhibit amplification), there have been more inventions and more scientific and engineering accomplishments than in all time previous. The field of digital electronics (especially computers) has been the greatest contributor of new products for these accomplishments and, therefore, has become one of the most rapidly growing industries. Manufacturers of semiconductor components (transistors, integrated circuits, microprocessors and memories) have been providing the building blocks, and the equipment manufacturers have been taking advantage of the opportunity by developing the most sophisticated systems that are economically feasible.

In his keynote address to the 1977 National Computer Conference, Mark Shepherd, Chairman of the Board of Texas Instruments, made the following points:

> "Until 1971, the semiconductor industry was in the circuits business. Semiconductor circuits, complex though they were, constituted only a fraction of an entire system. The one-chip calculator developed in 1971 was the first significant complete system. Since then many calculators and watches have been developed where the entire system function is accomplished by one or a few semiconductor chips. These were custom chips because the technology did now allow any reserve computing power for other applications.
>
> "The semiconductor industry has now entered an era where the entire system function of an end product can be accomplished by a few semiconductor chips, or a single chip, with enough versatility to permit adaptation to many different applications through the programming.
>
> "This change carries enormous implications for the system designer. 1) The lead time for system implementation is shortened because no special chip development is required. 2) The development cost will be low because it will be limited to software (which may be executed in hardware). 3) The required degree of electronic sophistication on the part of the user is much less. To achieve these advantages the system designer must be prepared to use standard products produced in large volume rather than custom devices.
>
> "The functional equivalent of a medium-scale computer *(Figure 1-1)* cost $30,000 in the early 1960s. Its cost equivalent has now dropped to $4,000 and is projected to be at less than $100 by 1985, penetrating the personal cost threshold. As this is accomplished, greater challenges will be encountered in the cost of sales, service, and maintenance, requiring that we learn to incorporate self-diagnostic and self-repair functions into our systems."

▶1

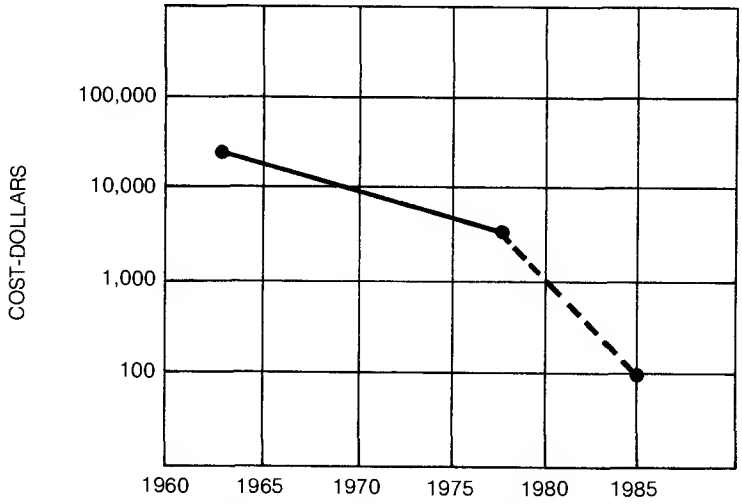


Figure 1-1. *Cost of Medium Scale Computer*
(M. Shepherd, 1977 NCC)

The cost of the hardware components for a typical digital system has been decreasing with time because new and more powerful semiconductor devices have been developed. Equally important is the fact that the development cost for the typical digital system hardware has also been decreasing. *Figure 1-2* illustrates how impressive this cost reduction has been. Contrast the figures of 7-8 *million* dollars in the early fifties with 8-9 *thousand* dollars in the late seventies; digital system development cost has been reduced by a factor of *one thousand* in a period of 25 years! An extension of this trend indicates that typical system hardware development cost will be approximately $1,000 by 1985.

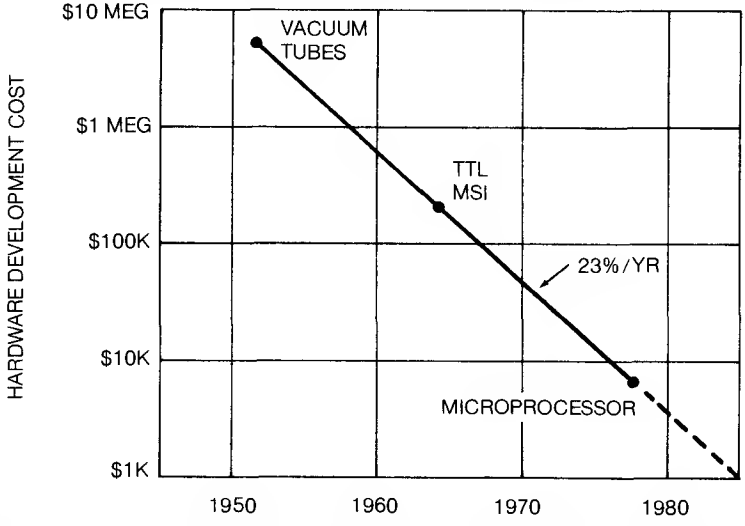


Figure 1-2. *Typical Digital System Hardware Development Cost at OEM Manufacturer*

How has this been accomplished? *Figure 1-3* shows what has been happening. As the number of components per chip of silicon increases, the development cost for each chip also increases. For a semiconductor manufacturer, volume production is required to offset the development cost. Semiconductor devices are therefore being *batched fabricated* — a few hundred, a few thousand per chip — and this means lower cost per *active element group* or AEG. (An AEG is defined as a logic gate, flip-flop, or a memory cell.)

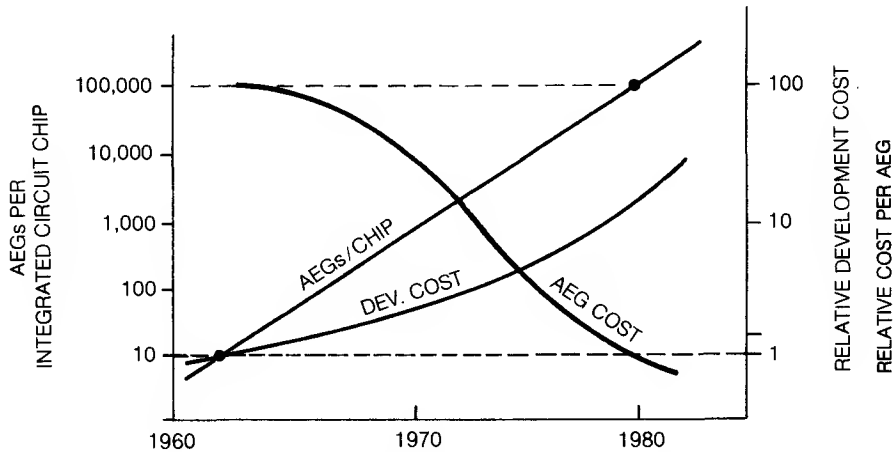


***Figure 1-3.** Evolution of Semiconductor Technology*

Figure 1-4 shows the chronology of semiconductor device development. An AEG in the early 1950's consisted of one or two transistors, several resistors, a capacitor or two, and some area of a printed circuit board to hold the parts together as an assembly. Early integrated circuits contained about 10 AEG's. Then medium scale integration achieved up to 100 AEG's per chip and large scale integration reached 1,000 AEG's per chip.

At this point (the late 1960's), the semiconductor technologists had apparently reached an impasse. If they continued to increase the number of AEG's per chip the high degree of specialization would preclude volume production, and the benefits of LSI would be lost. In fact, the only area in which LSI appeared to be feasible was in memories — primarily read/write memories now called RAM's. Read only memories (ROM's) and programmable read only memories (PROM's) were not required until later (as you will see). But the semiconductor technologists continued their thrust toward greater numbers of AEG's per chip, focusing primarily on memory products.

There was one other product which appeared to be feasible (in 1970) — a single-chip calculator. Here was an opportunity to stretch the imagination to greater degrees of achievement. At the producibility level of about 1,000 AEG's per chip, all of the functions of a microcomputer could be built on one chip — and the application certainly had the required volume potential. So *custom* LSI found a niche in the form of the hand-held calculator.

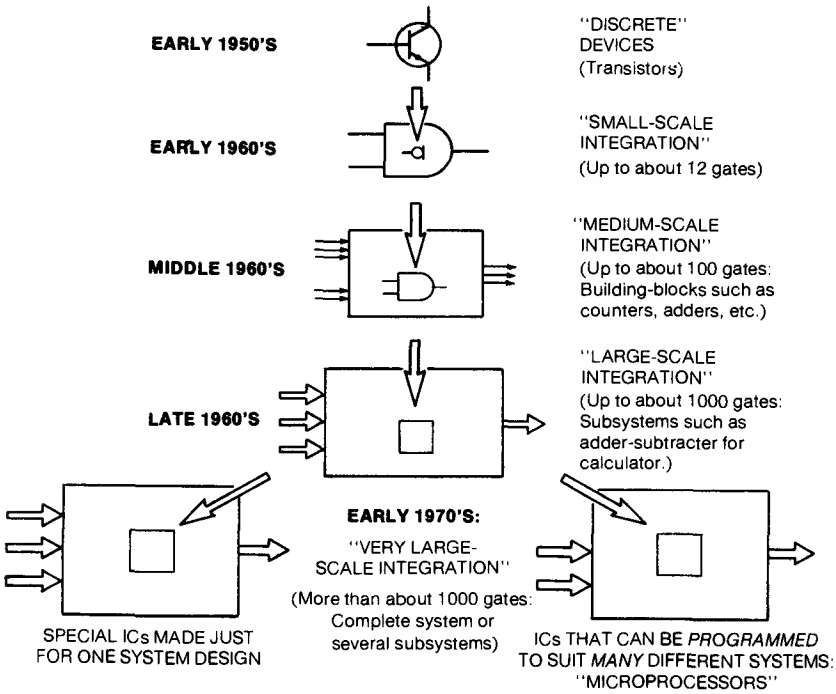


Figure 1-4. *Stages in Evolution of Digital Semiconductor Circuitry.*
(G. McWhorter, Understanding Digital Electronics, Texas Instruments Inc., Dallas, Texas, 1978)

From the very beginning the designers of the single chip microcomputer envisioned new and varied applications of this device, so it was made with a ROM for instructions and RAM for data. It was *programmable,* at least it was "mask programmable." And as we witness the growth of this segment of the semiconductor market, we see a host of dedicated applications for single chip microcomputers such as controllers for microwave ovens, sewing machines, and other appliances.

By designing a "standard" chip that could be *programmed* to do a variety of jobs, semiconductor technologists repeated the step taken by the inventors of the first programmable machine — the first computer — in the late 1940's. The first digital computer was a *stored program* digital calculating machine. Programming provided versatility and variety of applications. Similarly, programmable single chip, LSI semiconductor devices — microcomputers — gave LSI variety of applicability.

The next logical step in the evolution of LSI was the design of the general purpose microprocessor, a computer CPU on a chip. By interfacing the microprocessor to a memory — a set of chips arranged to provide as much storage as needed — one can build larger, more powerful microcomputers which can replace special purpose hardwired logic. In fact, general purpose hardware that is *programmable* provides *multichip* applicability of LSI technology.

With this breakthrough in the concept of LSI application, the semiconductor technologists have been motivated to continue to increase the number of AEG's per device. *Figure 1-5* projects the growth of AEG's per chip to over 10^6 by 1985 — a level sufficient for a single chip 32-bit microcomputer. The 16-bit microprocessor and 4K RAM require about 50,000 AEG's. RAM's of 16K and 64K bits requiring up to 100,000 AEG's are not unrealistic extensions of the trends; they are real products rapidly moving into the marketplace. New advances are being made in semiconductor process technology to achieve the packing densities needed for the future. As *Figure 1-5* indicates, optical techniques for defining regions and interconnections reach a resolution limit at about 10^5 AEG's. E-beam and X-ray technology will be required to further increase component density.

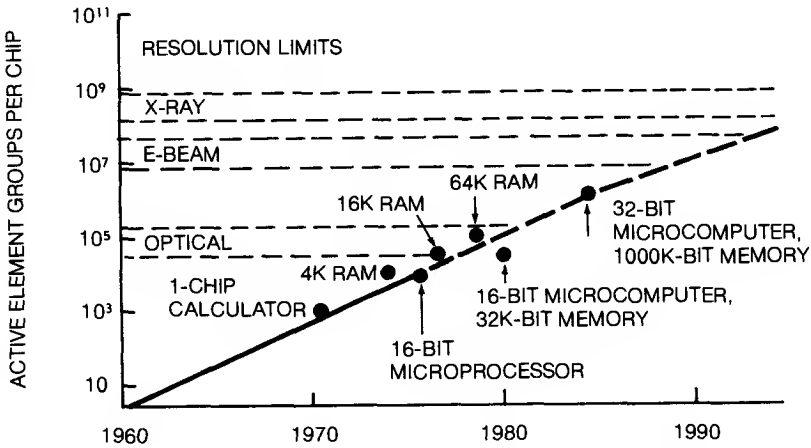


***Figure 1-5.** Semiconductor Chip Complexity*

(M. Shepherd, 1977 NCC)

The impact of programmable semiconductor devices is shown in *Figure 1-6.* Prior to 1972, semiconductor devices were designed as *circuits.* Now they are being designed as *systems* or at least subsystems. As the number of AEG's/chip continues its rise, driving down the cost of CPU and memory devices, unlimited opportunity is being created for an unbelievable variety of new products.

Figure 1-7 shows that a dramatic change is anticipated in the rate of AEG cost reduction with time due to the impact of microprocessors. Functions (AEG's) costing $1.00 in 1966 were obtained for around *5 cents* in 1976. In fact, the cost per AEG is projected to be less than *a tenth of a cent* by 1985.

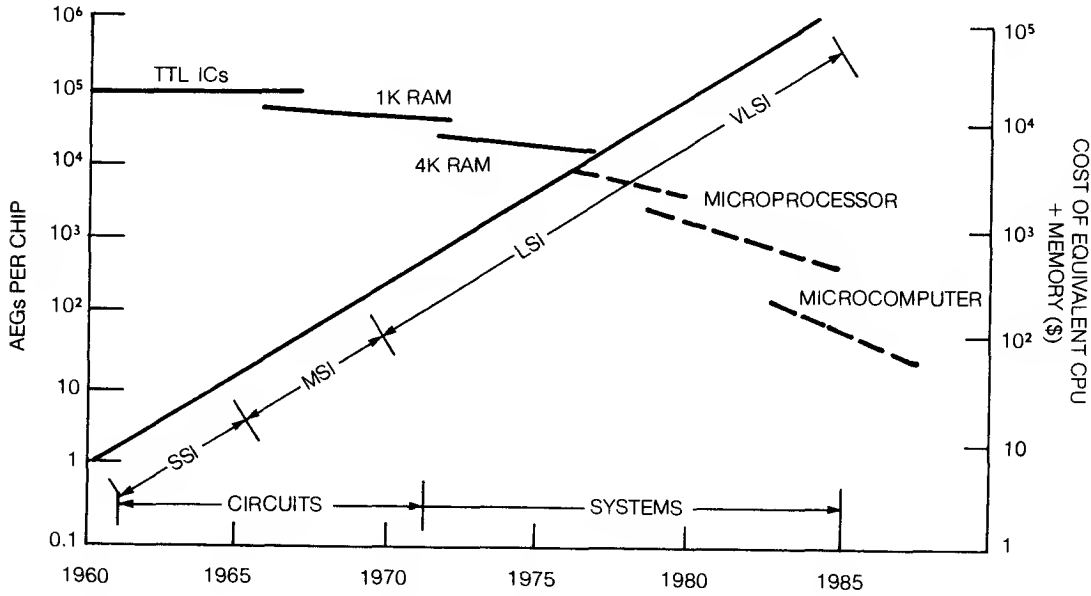


Figure 1-6. Distributed Semiconductor Power

(M. Shepherd, 1977 NCC)

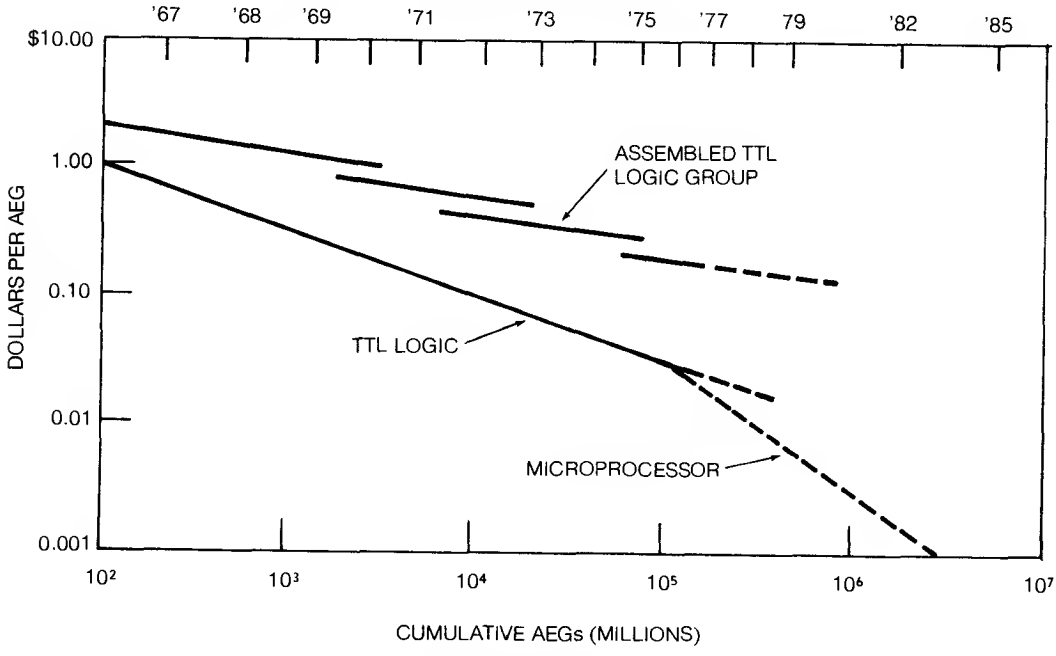


Figure 1-7. Cost Per AEG for TTL and Microprocessor

(M. Shepherd, 1977 NCC)

Memory costs (on a per bit basis) are diminishing, too. Following the projected trends for the cost of AEG's, RAM cost is forecast to be less than .05 cents per bit by 1982 (*Figure 1-8*). The need for various memory types has now been established. Programs for microcomputers are stored in non-volatile memories such as ROM's, PROM's and EPROM's. ROM's are mask programmable by the manufacturer and are best suited for high volume applications. PROM's are programmable after the devices are completely packaged. Either the manufacturer, the distributor or the user may store the desired program (or data) in a PROM. PROM's are suited for medium volume to low volume applications. EPROM's are erasable and so find use during prototyping and development cycles. They are also used in applications where software must be periodically changed, upgraded, or modified in any way. Other memory technologies such as CCD's (charge coupled device) and bubbles will be used for mass storage requirements where speed is not critical.

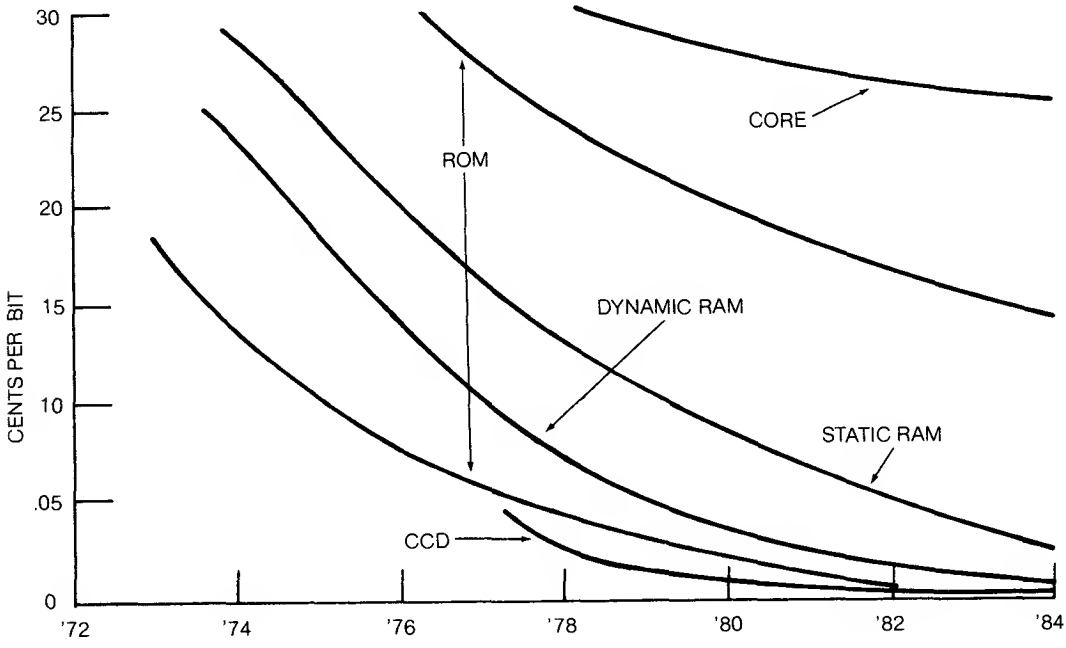


***Figure 1-8.** Memory Cost Comparison*

(M. Shepherd, 1977 NCC)

The effect of modern semiconductor technology has been to alter the roles of the component manufacturer and the OEM (original equipment manufacturer). Component manufacturers are continuing to produce batch-fabricated semiconductor products. But the economic benefit — the low cost per AEG — of batch fabricated semiconductor devices with high functional density cannot be realized except through applications which are program controlled. The component manufacturer must therefore provide programming support via PDS's (program development systems) and software products to enable the OEM to develop applications programs. Thus increased development cost of high functional density devices is found not only in improved process technology and in the design of LSI masks, but also in the attendant software support products. And volume production is required to offset these costs.

The role of the OEM is undergoing a corresponding shift. Component costs and the assembly cost of hardware have been sharply reduced. *Table 1-1* demonstrates the evolutionary steps in hardware costs. The cost improvement ratio of each step as compared with the previous step is dramatic: overall, it is 600:1.

Table 1-1. *System Cost Reduction*

EVOLUTIONARY STEP	COMPONENTS TO ASSEMBLE	TOTAL COST	
		COMPONENTS + ASSEMBLY COST	COST IMPROVEMENT RATIO
DISCRETES	20000-30000	6000-9000	--
IC'S (GATES & FLIP FLOPS)	350-500	600-900	10:1
IC'S + MSI	125-150	250-450	2,5:1
MICROPROCESSORS	7-10	120-190	2:1
MICROCOMPUTERS	1	6-12	12:1

While hardware costs are decreasing, the software costs, as a percentage of the overall design effort, are increasing. *Figure 1-9* illustrates the relationship of hardware to software costs in product development and the change in emphasis. In the 1950's computers were only used in large-scale business and scientific applications. OEM's had no opportunity to use computing power in their systems. When minicomputers were introduced in the 1960's, OEM's found applications in process control and small business EDP functions, and therefore had to provide some special programs for their use. With the advent of microprocessors in the 1970's, the software component of the development cost increased further, and this trend can easily be forecast into the 1980's — less than 25% of the development cost of most products will be for hardware.

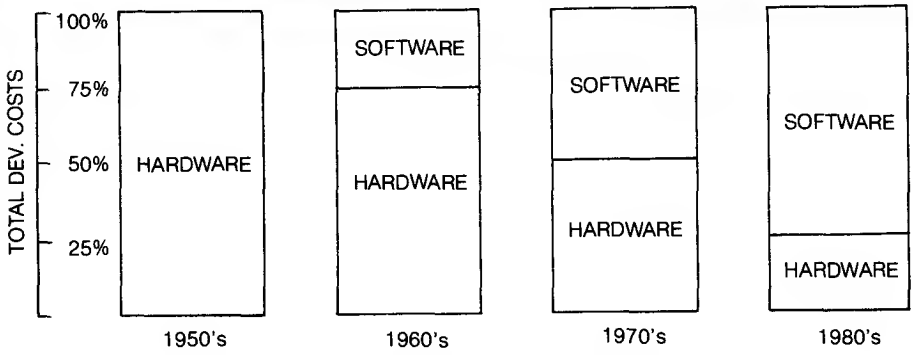


Figure 1-9. Increasing % of Software Development Cost

Development costs are changing — becoming more software oriented — and this has a strong impact on overall product cost. In any product design, the development cost is amortized over some production quantity, and this affects the price of the product. But developing *software* to achieve any design goal is less expensive than developing *hardware* to do the same thing. Therefore, the *total* development cost for "equivalent systems" is decreasing (perhaps by as much as 15-20% per year).

The development of *programmable semiconductors* has been a major accomplishment equivalent in importance to the inventions of the transistor, the integrated circuit, and the stored program computer.

The trends appear to be well established. The number of AEG's per chip will be increasing by at least *75% per year* for at least another two decades. As a result, AEG cost will decline by about *50% per year* and RAM cost per bit will decline by about *20% per year.* The computing power of LSI devices will increase while the price will continue to decrease. The impact will be felt in all walks of life.

APPLICATIONS OF PROGRAMMABLE SEMICONDUCTORS

The application of programmable semiconductors can be considered as an extension of the application of computers. All applications of LSI semiconductor devices are as computers because microprocessors, microcomputers and programmable LSI peripheral chips are *programmed* to perform the special functions required for each application. All the elements of a computer — ALU, control, memory and I/O — are present.

As the price of computing power decreases, the number of applications increases. The number of computers of any given type being applied is inversely proportional to the cost (*Figure 1-10*). As of 1976 there were relatively few systems in the $100-$10000 range. But microcomputers are changing this. Applications are being found in new designs of digital electronic systems, in products previously using electro-mechanical devices, and in new products which previously were not economically feasible.

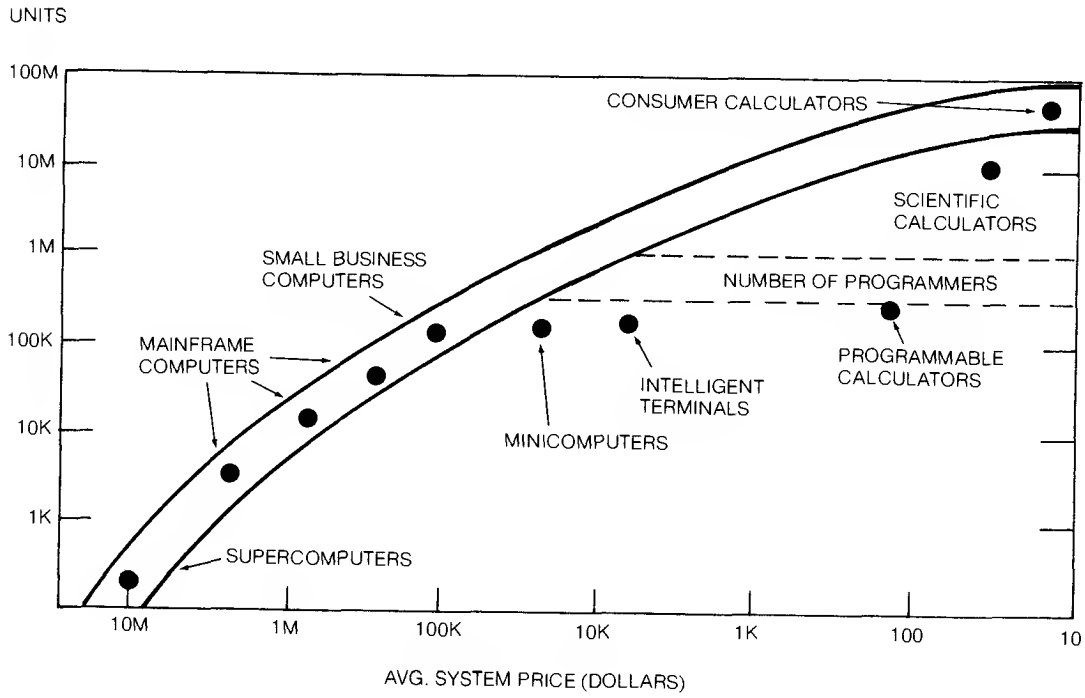


Figure 1-10. *U.S. Installed Computer Base—1976*

(M. Shepherd, 1977 NCC)

While some people may feel that the number of computers cannot exceed the number of "programmers" (approximately one million according to *Figure 1-10*), it is evident that all designers of products which use microcomputers will acquire the necessary programming skills to achieve the desired end product results.

Single Chip Microcomputer Applications

Single chip microcomputers are being used in the small, dedicated, high volume applications such as calculators, microwave ovens, and general appliance controllers. As the computing power of single chip devices increases, the range of applications will obviously expand. Early devices contained about 1K bytes of memory. New devices with 2K bytes of ROM for instructions and small amounts (256 bytes) of RAM for data have been built and designed into more complex applications. One example is a terminal controller using the TMS 9940 microcomputer with one support chip; this is described in Chapter 9.

Multi-Chip Microcomputer Applications

The application areas which involve the greatest number of designers and programmers by far are those using a multi-chip approach — a microprocessor, memory sized to the application, and peripheral interface devices. Limitations are much less in multi-chip systems than for single chip microcomputers. Designs can be accomplished using the general purpose microcomputer boards which have been designed to be applied to a variety of end products. Or the designer can start with individual LSI devices and build a special microcomputer for each application.

The list of applications for microprocessors is long and continues to grow. But a few of the representative areas are these:

Instrumentation
Test Equipment
Industrial Process Control
Point-of-sale Terminals
Cash Registers
Typewriter/Word Processing Equipment
CRT Terminals
Vending Machines
TV Games
Automobile Engine Ignition Controllers
General Automotive Products
CB Equipment
Communications Controllers
Educational Toys
Personal Computers
Special Dedicated EDP Functions

In each application standard programmable semiconductor LSI devices are used to sense input information, process the information according to special procedures (algorithms), and send information to external devices for display, printing, physical control devices, etc. Obviously the need for interface circuits is great. They cover specific functions such as A/D converters, D/A converters, transducers, and special display drivers, etc., as well as standard digital circuits for buffering, multiplexing, latching, etc. *Figure 1-11* shows conceptually how the elements of the microcomputer are arranged for any application.

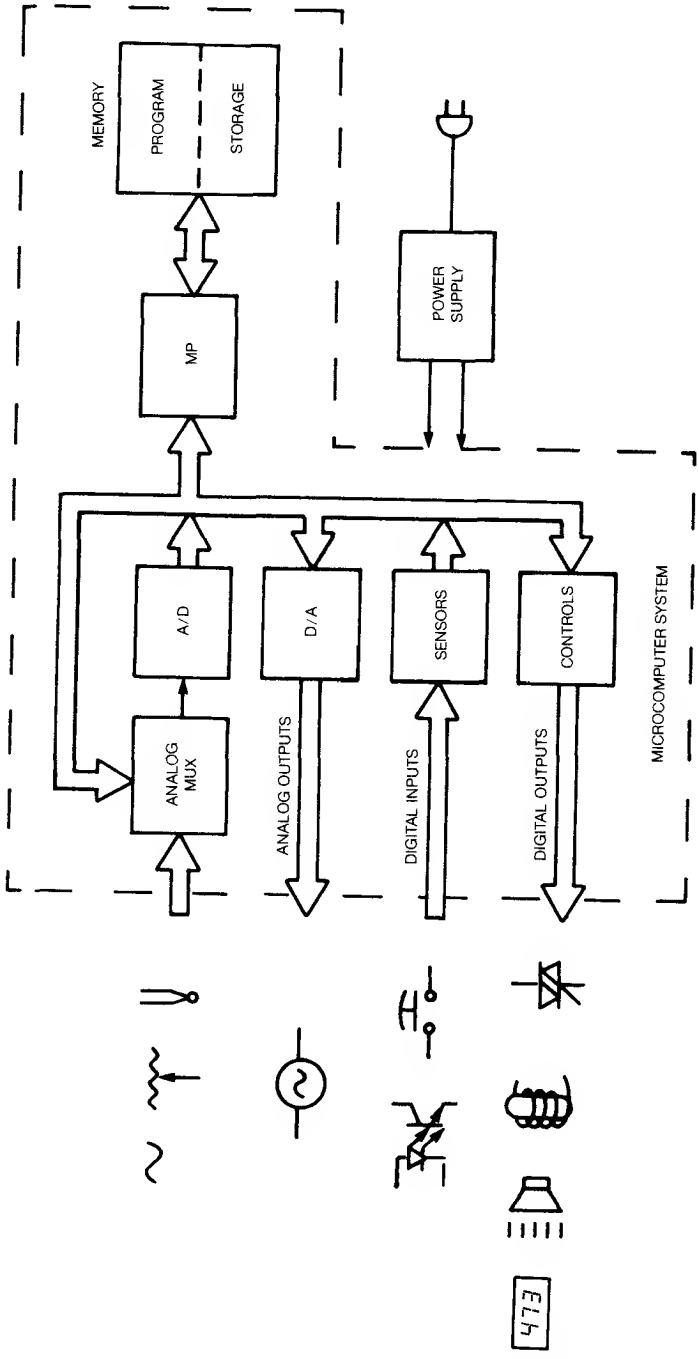


Figure 1-11. Microprocessor Applications in Process Control Systems

BUILDING A MICROPROCESSOR BASED SYSTEM

Given an application idea, how does one proceed toward designing a product in which a microprocessor is the central control device? The design steps may be diagrammed in great detail, but the most important steps are these. 1◂

1. System Specifications — The system requirements include electrical specifications for each input and output, timing details, and overall performance logic.

2. Division into small subsystems — By defining small, easily managed tasks, the hardware and software requirements can be measured, and design can be scheduled.

3. Decisions for hardware and software — This is the appropriate design point at which the tradeoff between hardware and software solutions for each task is evaluated. For economy, the best solution may appear to be software, but there may be a penalty in performance.

4. Hardware and software design — Here the two design activities may be carried out in parallel. The microcomputer parts are assembled on one or more breadboards and tested for signal flow. Software is developed using a software development system (a computer with appropriate peripherals and programs). Software testing may be done to provide algorithm functionality.

5. System integration — Ultimately, the hardware must be tested under program control. At this point the programs must be loaded into the system memory (usually PROM or EPROM) for testing. Often special logic analyzers and other computer based diagnostic tools are needed to debug the complete system (see the description of the AMPL system in Chapters 2 and 7).

It is clear from the foregoing list of steps that a thorough understanding of the hardware components and a knowledge of programming is required to design with microprocessors and microcomputers. But this is not difficult to acquire. By learning the names of standard building blocks and software packages, you will have taken a major step toward understanding what you read about them.

Basic Hardware Components

Since the hardware for digital systems is being standardized, the basic elements and their functions can easily be studied. Comparisons of similar devices from various manufacturers must be made and design tradeoffs evaluated. Here are the basic building blocks, what they do, and how they are used.

Microprocessor or CPU

▸1 This fundamental chip contains the Arithmetic and Logic Unit (ALU) which basically performs addition and comparisons between two numbers. Temporary storage registers are available to hold numbers (called operands) and addresses (memory location numbers) which identify or point to instructions and data. Sometimes the ALU is used to calculate addresses by arithmetic operations on certain register contents. The microprocessor must also contain timing and control circuitry to direct all activities in an orderly step-by-step procedure. The actual control functions are determined by decoding and executing instructions. Instruction execution is a special type of operation on information which comes from memory. The memory stores numerical values which may be interpreted by the processor in one of two ways. Either the number is an instruction, which will direct the sequence of operations over the next few clock cycles, or it is data to be operated upon either arithmetically or logically.

Memory

The main memory of a microcomputer holds the program and data for the system. Because semiconductor RAM devices are volatile (that is, all data is lost when power is removed), it is desirable to use ROM devices (Real Only Memory or non-volatile memory) for the program and RAM for data. ROM devices are programmed (information stored in the cells) by means of a metalization pattern or mask at the time of chip fabrication. Programmable read only memories (PROM's) may be programmed by the manufacturer or the user because information is stored by burning small metalic fuse links via the application of electric current. Programming is performed on the device after it has been packaged. EPROM's are non-volatile read-only memories which may be *erased,* usually via the application of ultraviolet light. These devices are especially useful in prototyping and system development during which program changes are frequent.

Memory devices are designed for cascading so that any size memory may be obtained by adding more devices. Capacities of 4K bits per chip are common; devices with 64K bits per chip are not far away.

Input/Output

For the input and output function — interfacing the microprocessor-memory combination to the "outside world" — usually consists of a variety of devices including programmable LSI devices. Examples of interface requirements are as follows:

1. For communication of digital information over a pair of wires, conversion from 8-bit bytes (parallel) to single bits sent in sequence (serial) is required. The I/O device must receive a number of bits, hold them in a register and then shift them serially to a transmission line. The reverse procedure, serial to parallel conversion, must be performed for receiving information from the transmission line. Since the clock rates,

start and stop characters, and "handshaking" requirements can be complex in communications networks, the protocol is designed into the TMS 9902 and TMS 9903 programmable communications controllers (see Chapter 8 for details).

2. Man-machine interfacing may consist of arrays of switches and indicators or may be performed via a terminal such as a teletype (TTY) or a video display terminal (VDT). Arrays of switches are connected to microcomputers via multiplexers. The address bus may be used to select one of the switches for sampling at any given moment. Addressable latches are useful in supplying on-off data to arrays of indicators. The address bus is again used to select one specific display device (a single lamp) to be turned on (or off) in a given computer cycle. Terminal interfacing can be accomplished via a serial data interface such as the TMS 9902 (see Chapter 9 — example using the TM 990/100M board).

3. The broad category of analog (continuously variable) inputs and outputs requires converters (A/D and D/A) to obtain digital information on the computer side of the interface. Input signals from transducers or output signals to actuators (positioners) require this type of conversion.

Connecting the I/O devices to the CPU and addressing them may present problems in some microcomputer systems. The 9900 solves the problem by providing two types of general purpose I/O. Memory mapped I/O allows a set of memory addresses to identify the I/O devices (as though they were words of memory), while the communications register unit (CRU) provides a separate I/O port specially designed to interface single bit devices, communications devices, standard computer peripherals, etc. Unique to the 9900 architecture, the CRU interface is a powerful and versatile I/O technique; it is easily utilized via the special LSI peripheral supports circuits (such as the TMS 9901, 2, and 3, and the TIM 9905 and 6).

The rules for interconnecting the various elements of the microcomputer include loading specifications and signal level limitations. In observing these rules the designer will occasionally use a few standard devices to reduce loading or perform level shifting. Generally, the addition of such devices is an insignificant part of the overall design. (Details for hardware interfacing are given in Chapter 4.)

Programming for Microcomputers

The writing of programs — often called software development — is the companion activity to hardware breadboarding and testing in computer systems. But software is substantially more flexible than hardware because it consists primarily of *ideas*, documented in strings of characters on a page, or in 1's and 0's in a memory. In fact, until a program is actually loaded into a memory, it is truly a set of ideas on paper, hence the contrasting name, *software*.

BUILDING A MICROPROCESSOR BASED SYSTEM

In developing the individual hardware components of a microcomputer, designers usually subdivide the activities into small, easily managed tasks. These tasks are performed sequentially by one designer or simultaneously by several members of a design team. The same is true of software design. Small, easily defined and understood sub-programs are given as individual assignments to the programmers on the design team.

The disciplines for programming are set up so that each sub-program stands alone, yet couples to the other sub-programs in a harmonious manner. But the overall plan begins at the top (a program to handle all sub-programs) and expands to several lower levels (a "Christmas tree" of programs). This is known as "top-down programming", and it is a form of structured programming.

The term *structured programming* means that discipline in programming in which each program module implements an algorithm with a single entry point, a single exit point and a definitive result for each possible input. Each module must contain its own buffer area so that it cannot alter procedures or data of other modules. (In some cases common buffers are allowed, but complex rules for their use are needed.)

How is programming done? What equipment is needed? And what support can you get from a microcomputer manufacturer? First, there is a preparation phase in which the designer and/or programmer must become familiar with the instruction set and the architectural elements of the microcomputer selected for the design. The second phase involves writing selected short program segments to gain insight into the memory requirements and the execution speed of various sub-programs. Then the main body of the program may be developed.

Writing programs means writing code; writing program steps which must be executed in sequence. Usually these steps are written in a mnemonic language which uses one to four letters as operation codes, and strings of other characters to designate the operand (the number to be operated upon). These program steps must be translated and "assembled" into a set of 1's and 0's — the machine language executable by the microcomputer — by a special program development computer.

The programmer writes the program on paper. Then he enters the program steps via a keyboard into the program development system (PDS), and directs the PDS to "assemble" the instruction into machine code. The output from the PDS is a set of numbers which represent the program steps, and a listing of the input and output codes.

Obviously the PDS uses some special programs (software) for performing the tasks outlined. The programmer writes *source code* statements, submits them to the PDS via a program called the *editor*, then uses the *assembler* program to produce *object code* — the machine code used by the microcomputer. Errors in the program statements are printed along with the object code listing. Errors are corrected by editing the source code (via the editor) and resubmitting it to the assembler.

After a number of program modules are complete, a set of two or more may be "linked" together as a single program. This is done by submitting object code programs to the *linker*. The output from the linker is a single program which may be loaded into the microcomputer. 1◀

The list of support software is just beginning.

The following outline of software products describes the program development cycle further.

- Program development software
  - Editor — for entering and changing source code
  - Assembler — for conversion from symbols and mnemonics into machine code
  - Linker — for connecting several programs into one
  - PROM programmer — for loading numbers (programs) into PROMs
- Program testing software
  - Debug routines — for testing programs
  - AMPL system software — for testing programs and the interaction with the hardware
- Software available for use with user programs
  - Monitor — for checking status of all program modules
  - Executive — for overall control
  - Operating system — for operating peripheral devices
  - Library (utility) programs — for performing special mathematical conversions and calculations
  - High level language software for program development
    - PASCAL — for structured programming
    - POWER-BASIC — for ease of programming in BASIC language
    - FORTRAN — for general computer problem solving

This partial list of software is intended as a categorical outline which should indicate the level of support one finds in the areas of software development. To comprehend the value of any or all of these software products, you must work with them and develop a few programs for microcomputers.

The obvious difficulty with software evaluation is that few designers can afford the capital investment for a large PDS to properly evaluate each of the alternative paths for software development. But Texas Instruments has developed a variety of program development systems. Some of these are very economical and readily available. They were designed specifically for product and programming evaluation and for initial design.

You will find descriptions and approximate prices for each PDS in Chapter 2.

BUILDING A MICROPROCESSOR BASED SYSTEM

Which Microprocessor or Microcomputer to Use

▸1 You may be convinced that designing with programmable semiconductors is the best design philosophy, and you may be attempting to evaluate the various products on the market. But a significant decision point has been reached: "which microprocessor or microcomputer is best for my application?" The selection of the proper device is based on many factors, some of which are not related to architecture or instruction execution speed.

Selection of a microcomputer or microprocessor usually means selection of one primary vendor (and sometimes one or more second sources) who manufactures the product and the compatible peripheral devices. It also means the purchase of a program development system designed especially for the specific microprocessor. Selection of one device means a commitment to using that device for future designs. Changing to another microprocessor is costly both in hardware and in the development of programming skills.

Selection criteria for a microprocessor may be summarized as follows:

1. The microprocessor must be versatile so that it can be used in many applications.
2. The vendor must provide a comprehensive set of support and peripheral circuits.
3. One PDS should serve the programming activity for a significant period of time.
4. The cost of the devices and the PDS must be economically attractive.
5. The performance of the microprocessor must be sufficient to meet the design goals.

Texas Instruments 9900 family of components and software systems clearly meets these selection criteria. Careful evaluation of the price/performance tradeoffs between the various microprocessor products on the market will reveal superior adaptability of the 9900 family to any product design. The selection criteria applied to the 9900 family may be summarized thus.

1. Versatility has been achieved by providing a *family* of processors using one basic 16-bit architecture. Both 16- and 8-bit versions are available as well as a single chip microcomputer. Instead of trying to apply a single microcomputer to a broad scope of applications, the designer may select from the 9900 family the most appropriate microprocessor for each application. Programming and software support is the same for all.

2. Numerous support devices are available from Texas Instruments, and new products are introduced regularly. Chapter 8 contains detailed data sheets for many of them.

3. A single PDS from Texas Instruments can provide programming support for all of the processors in the 9900 family. The powerful support software streamlines program development. And programs written for one microcomputer may be used on another. Software compatibility of the processor family is one of the primary benefits of designing with the 9900.

4. Product pricing of the microprocessors, peripheral devices and PDS's is economically attractive. Designing multiple applications from the same components and using the same development tools means even greater economic benefit.

5. The 16-bit architecture — especially the bus width and the register size — enables the 9900 family of processors to achieve outstanding performance. Performance is measured by throughput and computing power, not by clock speed alone.

A complete evaluation of the 9900 in each of the above categories is not possible in so few words. But one specific feature of the product family should be included as a part of the evaluation. Memory-to-memory architecture, a unique computer concept developed by Texas Instruments for the 9900 minicomputer, is an outstanding feature because it enables the 9900 to achieve the most cost effective product development. The story of the evolution of this architecture will help you understand its importance.

EVOLUTION OF MEMORY-TO-MEMORY ARCHITECTURE

▸1 All things change with time, and computers are no exception. An evolutionary process has been at work in computer design since the beginning. Early machines were designed around a single accumulator which served as the focal point of most of the instructions. Steps such as load the accumulator (LDA), add to the accumulator (ADD), and store the accumulator (STA) were common in programs written for such machines. (The instruction mnemonics used here are simply illustrative and are not intended to be identified with any specific computer or microprocessor.) But there was a fundamental limitation—the bottleneck effect of forcing all transactions to be performed via a single accumulator *(Figure 1-12).*

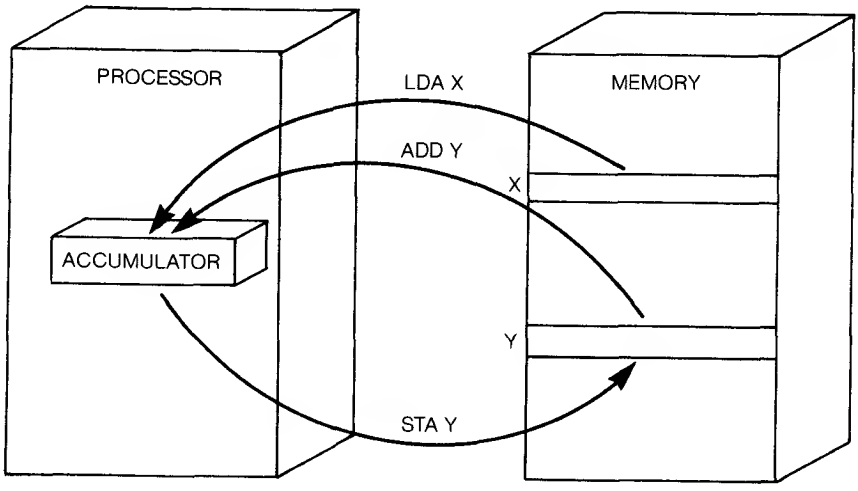


Figure 1-12. Single Accumulator Architecture

EVOLUTION OF MEMORY-TO-MEMORY ARCHITECTURE

As circuit elements became less expensive, especially through the introduction of integrated circuits, multiple accumulator architectures emerged *(Figure 1-13)*. A and B accumulators were the focal points of expanded instruction sets which allowed loading either accumulator (LDA, LDB), adding to either accumulator (ADA, ADB), and storing either accumulator (STA, STB). With this design came the increased use of an accumulator for holding the address of an operand, adding flexibility and power to the instruction set and to the architecture.

It should be clear at this point that the instructions, the dictionary of words used by a computer to implement the ideas of the programmer, are as much a part of the architectural fabric as the registers, the control unit or the bus structure. In fact, by implementing instructions as strings of microinstructions stored in an on-chip control ROM, microprocessor designers have created the opportunity for increasing instruction set power through microprogramming.

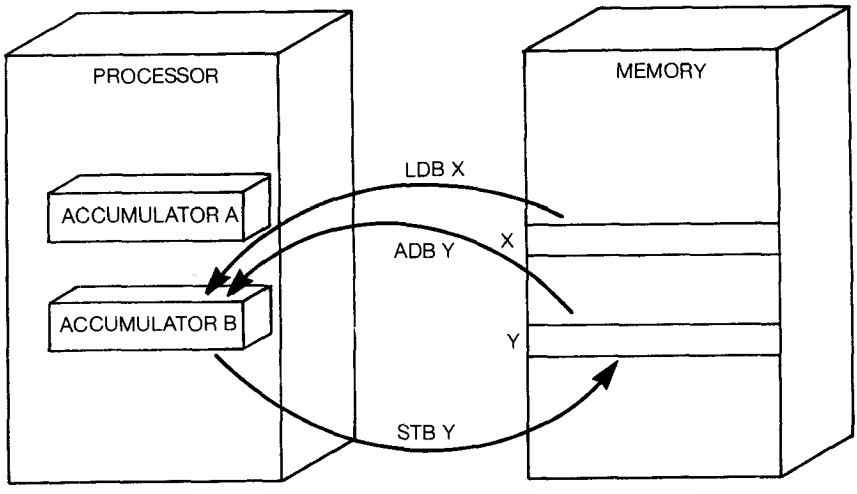


Figure 1-13. Multiple Accumulator Architecture

The next major step in the architectural evolution was the design of machines based on a set of general registers which could be used as accumulators for numerical operations or for storage of operand addresses *(Figure 1-14)*. The expanded capabilities allowed increased flexibility not only in arithmetic functions but also, and more importantly, in the generation of operand addresses via indirect addressing, and indexed addressing.

▸1

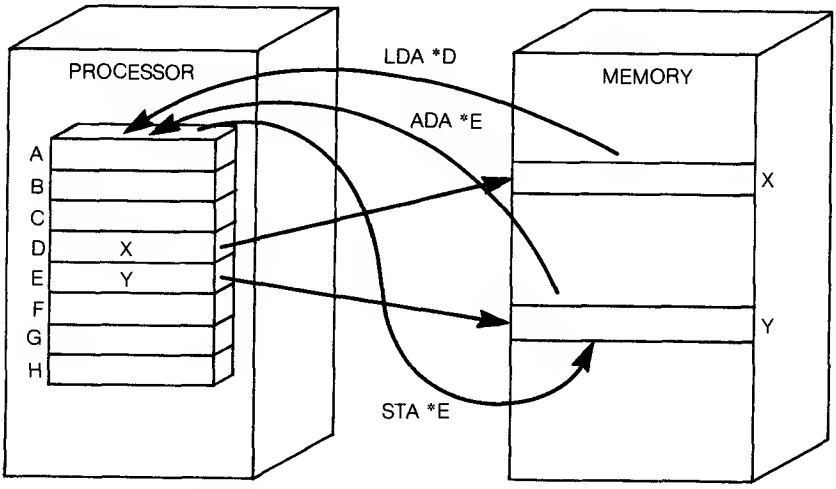
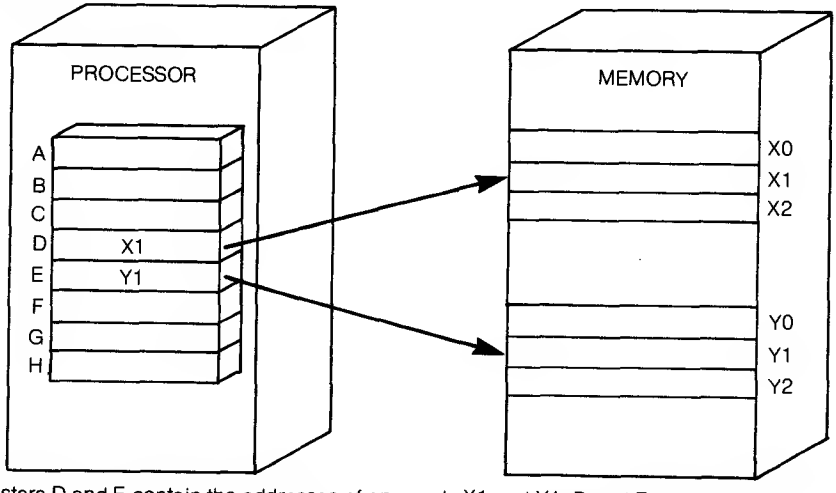


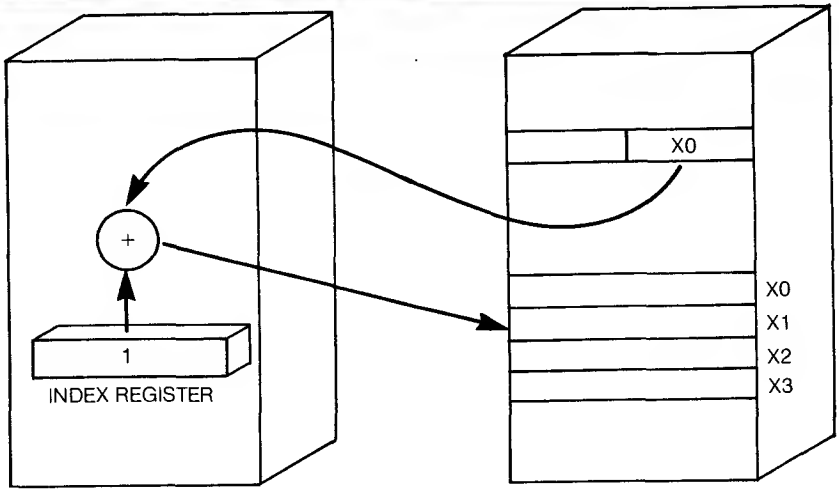
Figure 1-14. General Register Architecture

Perhaps it is well to digress for a moment and explain these terms. Indirect addressing allows a register to serve as a pointer to identify specific elements in a table or an array of data *(Figure 1-15)*. Instructions for an arithmetic operation may be used over and over, with the pointer (or pointers) being adjusted to access different values for each pass. In indexed addressing, the instruction contains a base value while an index register holds the displacement value *(Figure 1-16)*. The base value locates the table, and the index register contains the number of the element in the table (one, two, three, etc.). The base value must be added to the contents of the index register to obtain the actual memory address.



Registers D and E contain the addresses of operands X1 and Y1. D and E may be incremented to address sequential elements in tables.

Figure 1-15. Indirect Addressing



X0 is the address of the first element in the table.

X1 is obtained by adding X0 to the 1 in the index register.

The index register may be incremented to address sequential entries in the table.

Figure 1-16. Indexed Addressing

The general register architectures were made economically feasible by the expanded capabilities of integrated circuits through the technological advancements of Medium scale integration (MSI) and large scale integration (LSI) *(Figure 1-17)*. As more and more circuits were implemented on a chip, it became feasible to expand from two accumulators, to a general register file, to the general register file on a single LSI microprocessor chip.

In discussing LSI, one must not fail to recognize that the single most important impact of LSI is in the development of memories. More bits per unit area of silicon means higher capacity and lower cost, generally without sacrificing speed. The advent of microprocessors was the natural evolutionary step in the utilization of memory for a greater variety of logic and control applications.

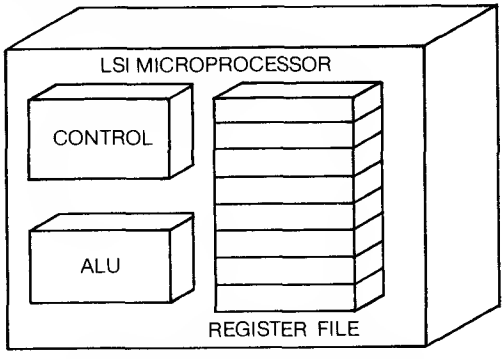


Figure 1-17. LSI Microprocessor

EVOLUTION OF MEMORY-TO-MEMORY ARCHITECTURE

In looking toward the future of memories and microprocessors, the technologists see the implementation of an ever increasing number of memory cells and microprocessor
▶1 calculation and control functions on an ever-shrinking area of silicon *(Figure 1-18).* Registers and memory cells are virtually identical in their implementation at this point, so the words *register* and *memory* no longer connote high speed and low speed storage. In fact, memory speed is rapidly approaching register speed *(Figure 1-19).*

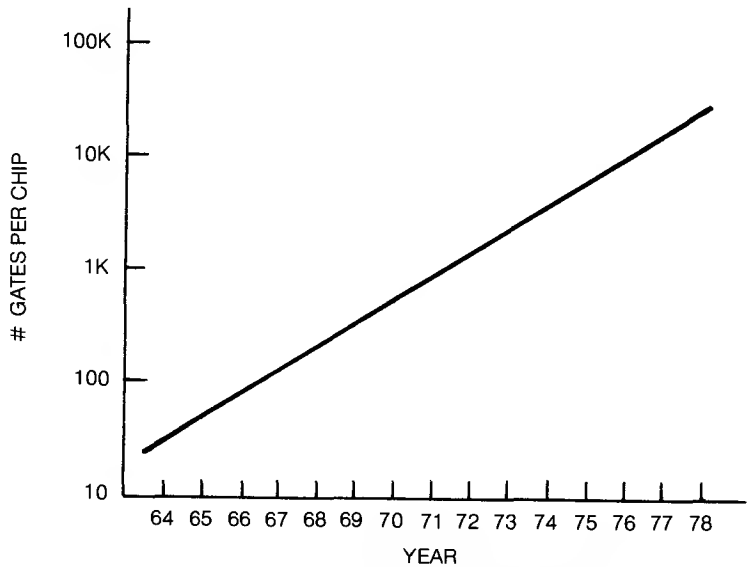


Figure 1-18. *Trend in Gates per Chip*

In view of this convergence of memory speed and register speed, the architects of the 990 minicomputer (from which the 9900 is derived) envisioned an architecture in which the instructions are written with respect to *memory words* rather than *registers.* The architectural concept, called memory-to-memory architecture, was the basis for a new computer design in which all memory reference instructions operate on one or two words of memory and store the result before going on to the next instruction.

There were actually two major reasons for developing such an architecture. First, since all instructions would reference words of memory and complete their cycles by placing results in memory, there would be no requirement for register-save operations in a multitask or interrupt processing environment. Second, while this approach might at first be slightly slower in some cases, the architects envisioned that the technological evolution would continue to narrow the gap between register speed and memory speed, and in the long run this minor disadvantage would vanish.

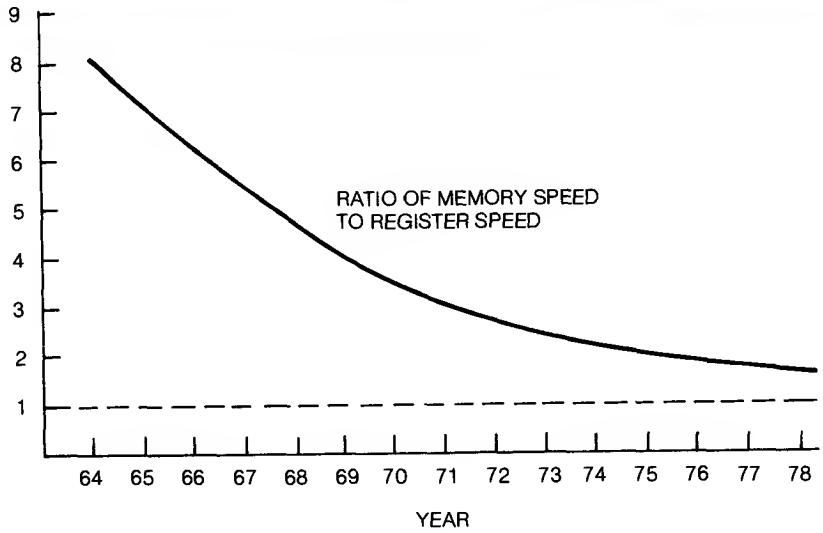


Figure 1-19. Ratio of Memory Speed to Register Speed

Another important advantage of this new architecture, often overlooked, appears to be an even stronger and more important justification for the development of this radical departure from conventional computer architectures. When one instruction can identify two memory words or operands, perform an operation, and store the result in memory, it will replace common sequences such as LDA, ADA, STA found in the instruction sequences of all accumulator-based machines. Furthermore, a single instruction can access additional memory words for use in addressing operands and can even increment pointers and employ index registers all as a part of its execution sequence. If a single instruction can do all this, then the writing of instruction sequences, programming, must be substantially easier. Fewer lines of code are required. (In data manipulation and address computation sequences, the reduction is typically 3:1.) Support software, such as monitors, executives, and operating systems can be smaller, easier to use and understand, and will consume less memory.

For these reasons, benchmarks written to compare the 9900 architecture with conventional register file based microprocessors show the advantage of the 9900's memory-to-memory architecture in three important categories: the number of bytes of memory required, execution speed, and the number of instructions written to accomplish a given task *(Figure 1-20)*. The 9900 comes out ahead in all three categories.

	Program memory requirements (bytes)				Assembler statements				Execution time (microseconds)			
	9900	A	B	C	9900	A	B	C	9900	A	B	C
Input/output handler	24	38	28	17	9	14	17	7	71	154	79	49
Character search	22	24	20	18	8	10	9	8	661	1636	760	808
Computer go to	12	12	17	14	5	5	11	8	98	352	145	145
Vector addition: $A_N \leftarrow B_N = C_N(16)$	20	30	29	46	5	14	20	22	537	2098	1098	1866
Vector addition: $A_N \leftarrow B_N = C_N(8)$	20	32	23	40	5	15	14	22	537	2108	738	936
Shift right 5 bits	10	6	19	20	3	3	12	9	22	56	137	81
Move block	14	18	16	34	4	9	9	16	537	1750	1262	2246
Totals	122	160	152	189	39	70	92	92	2464	8154	4219	6131

Figure 1-20. Benchmark Comparison of 9900 vs. Other Microprocessors

One final note about architecture. Memory-to-memory architecture and instructions in the 9900 do not sacrifice the concept of "registers" as they are conceived in the architectures with general register organizations. The general "register file" is conceptually retained as a block of sixteen words of memory *(Figure 1-21).* Over two thirds of the instructions in the 9900 refer in one way or another to this "register file" in much the same way as prior architectures referenced the general register file in the CPU. Thus, base addresses, subroutine linkage and interrupt save operations can all be accomplished via the "register-file-in-memory" concept.

By using memory for the register file, the advanced memory-to-memory architecture allows new programming flexibility. There is a way to identify *multiple register files* in 9900 based systems *(Figure 1-22).* Each basic process can have its own set of "registers." *There is no limit (except memory size) to the number of "registers" available* for use in programming the functions of a particular application.

The memory-to-memory architecture of the 990 and 9900 products is clearly revolutionary and innovative. Programming effort for the 9900 is typically less than half that for any other microprocessor currently available because the instructions operate on words of memory and store results automatically. This means not only that programs consume less memory, but execution speed (for the 16-bit processors) is faster than that of other processors.

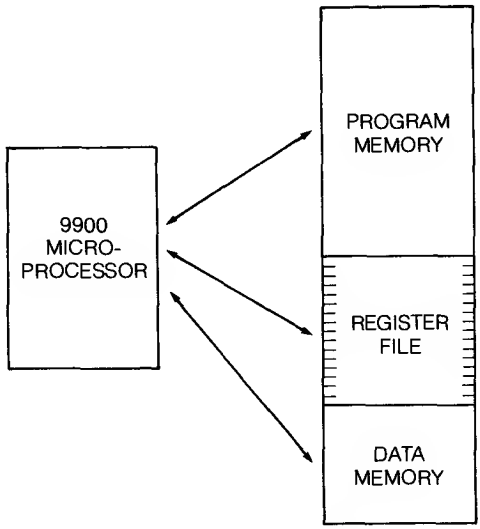


Figure 1-21. Register File in Memory

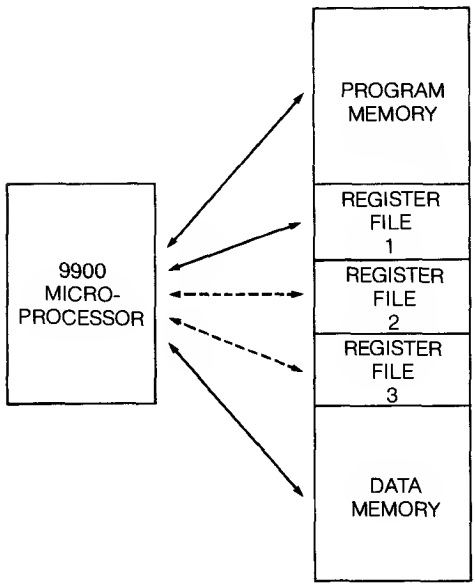


Figure 1-22. Multiple Register Files in Memory

GETTING UP TO SPEED ON MICROPROCESSORS

▶1 By now you may be convinced that this book contains a great amount of information about microprocessors and microcomputers, but you may feel that you are not as well prepared to understand it as you would like to be. This section has the answer. Here are the steps you should take to learn about microprocessors and microcomputers. The knowledge gained will help you in all new designs and will be especially helpful in designing with the 9900 family of processors and peripherals.

Few people have had the opportunity to learn about microcomputers in college. In fact, schools and colleges exist primarily to teach you *how to learn*, and not to teach you everything you need to know to do a particular job. Your effectiveness in performing any job is directly related to your willingness to acquire new specialized knowledge in your particular field. This book will serve as one source of specialized knowledge in the field of microcomputers, but it is focused on the 9900 family. And you may require additional education in this field before achieving a full understanding of the material presented.

It may be that knowledge of MOS and I^2L technologies is needed for a clearer understanding of interfacing techniques. Basic computer fundamentals, such as storage of data and programs and the sequential operations may be an area you would like to study. It could be that you feel a need to improve your understanding of programming and the concepts of building programs via the modular approach. The list of specialized areas within the field of microcomputer technology can be quite long.

Technology advances so rapidly today that it seems virtually impossible to keep up, much less catch up. But you can do both, and without spending an inordinate amount of time. To acquire specialized knowledge in any field, you should devote 30 minutes a day to reading books or periodicals which contain the information you need. Advising you on the implementation of such a program is not the intent of this section. You know where you are and where you are going. What you need is a clear path or plan of action to achieve the goal: the acquisition of specialized knowledge about microprocessors and microcomputers.

The first step is to find authoritative texts on the various subjects in the field. This chapter contains a bibliography of texts and periodicals from which to begin your search for new information. Get your hands on these books and articles. Review them for general content and readability, then decide which ones are best suited to your needs. Set up a plan to read one or more of these books in a definite period of time, devoting a *scheduled*, uninterrupted period of 30 minutes a day to this program. Take notes while you are reading and (if the book belongs to you) underline the information which is especially important to you.

As you are getting up to speed, you will become aware of certain periodicals that contain articles most directly suited to your background and experience. Subscribe to one or more of these or be sure to obtain each issue as it is published so that you are not only reading about fundamentals, but current topics, the latest improvements in devices and systems.

Set up the goal, the plan of action; and then, above all, form the habit of reading for 30 minutes a day. Few people can set up such a plan, and fewer still can continue to execute it for long periods of time. But if you persist, you can learn, not just one, but *all* facets of design with microprocessors and microcomputers, and in time you will achieve the success you desire.

BIBLIOGRAPHY

Books

Altman, L., *Microprocessors*, Electronics Book Series, McGraw-Hill, 1975

Bartree, T. C., I. L. Lebow, and I. S. Reed, *Theory and Design of Digital Machines,* McGraw-Hill, 1969

Bibbero, R., *Microprocessors in Instruments and Control,* John Wiley, 1977

Blakeslee, Thomas R., *Digital Design with Standard MSI and LSI,* John Wiley, 1973

Gear, C. William, *Computer Organization and Programming*, McGraw-Hill, 1969

Greenfield, Joseph D., *Practical Digital Design Using IC's,* John Wiley, 1977

Hansen, P.B., *The Architecture of Concurrent Programs,* Prentice-Hall, Inc., 1977

Hansen, P.B., *Operating System Principles,* Prentice-Hall, Inc., 1973

Knuth, D. E. *The Art of Computer Programming, VOL I, Fundamental Algorithms*, 2nd Edition, Addison-Wesley, 1973.

Knuth, D.E. *The Art of Computer Programming, VOL II, Semi-Numerical Algorithms,* Addison-Wesley, 1969

Knuth, D.E. *The Art of Computer Programming, VOL III, Sorting and Searching,* Addison-Wesley, 1973.

Luecke, G., J. Mize, W. Carr, *Semiconductor Memory Design and Application,* McGraw-Hill, 1973

Malrino, A., *Digital Computer Electronics,* McGraw-Hill, 1977

McWhorter, G., *Understanding Digital Electronics,* Texas Instruments Learning Center, 1976

Morris, R. L., J. D. Miller, *Designing with TTL Intergrated Circuits*, McGraw-Hill, 1971

Norris, B., *Power Transmission and TTL Integrated-Circuit Applications,* McGraw-Hill, 1977

▶1 Silver, G., *Computer Algorithms and Flowcharting,* McGraw-Hill, 1975

Sloan, M.E., *Computer Hardware and Organization,* Science Research Associates, Inc., 1976

Solomon, L., *Getting Involved with Your Own Computer; A Guide for Beginners,* Ridley Enslow Publishing, 1977

Soucek, B., *Microprocessors and Microcomputers,* John Wiley, 1976

Torrero, E., *Microprocessors, New Directions for Designers,* Electronic Design, Hayden, 1975

Wester, John G. and William D. Simpson, *Software Design for Microprocessors,* Texas Instruments Learning Center, 1976

Williams, Gerald E., *Digital Technology,* Science Research Associates, Inc., 1977

Zaks, R., *Microprocessors: From Chips to Systems,* Sybex, 1977

Staff of the Texas Instruments Learning Center, *Understanding Solid-State Electronics, 3rd Edition,* Texas Instruments Learning Center, 1978

ARTICLES

Barna, Arpad, and Dan I. Porat, *"Integrated Circuits in Digital Electronics",* John Wiley, 1973

Reid-Green, K.S., *"A Short History of Computing"*, Byte, Vol. 3, No. 7, July 1978

Special Issue on Microelectronics, *"Scientific American",* Vol. 237, No. 3, September 1977

Electronic Business, *"New Rules in an Old Game",* Vol. 18, No. 6, June 1978

LIST OF PERIODICALS TO BE MONITORED

ACM Computing Surveys, The Survey and Tutorial Journal of the Association for Computing Machinery, ACM, Inc., Mt. Royal and Guilford Avenues, Baltimore, MD 21202

EDN, Cahners Publishing Co., 270 St. Paul St., Denver, Colorado 80206

Electronics, McGraw-Hill Inc., 1221 Avenue of the Americas, New York, N.Y. 10020

Electronics Design, Hayden Publishing Co., 50 Essex St., Rochelle Park, N.J. 07662

IEEE Spectrum, The Institute of Electrical and Electronics Engineers, Inc., 345 East 47 Street, New York, N.Y. 10017

Interface Age, McPheters, Wolfe & Jones, 16704 Marquardt Avenue, Cerritos, CA 90701

CHAPTER 2

Product Selection Guide

THE 9900 FAMILY – WHAT IS IT?

The 9900 Family is a compatible set of microprocessors, microcomputers, microcomputer modules, and minicomputers. It is supported with peripheral devices, development systems, and software. It provides a designer with a system solution having built-in protection against technological obsolescence. The family features true software compatibility, I/O bus compatibility and price/performance ratios which encompass a wide range of applications. The family is designed with a unique flexible architecture to allow technological changes to be easily incorporated while minimizing the impact these changes have on an overall system design.

FAMILY OVERVIEW

The Hardware Family

Figure 2-1 is a diagram of the 9900 Family members. The spectrum of microprocessors and microcomputer products available in a variety of formats is shown in *Figures 2-2* and *2-3.* In the first part of *Figure 2-1,* the microprocessors or microcomputers are combined with microcomputer support components *(Figure 2-3)* to form systems. These systems also include I/O interface, read-only and random access memory, and additional support components such as timing circuits and expanded memory decode .

The family also includes microcomputer board modules containing the 9900 microprocessors and peripheral components *(Figure 2-4).* As shown in the second part of *Figure 2-1,* these modules can be used for product evaluation, combined for system development or applied directly as end equipment components.

When applications require minicomputers, completely assembled units can be purchased and installed. The software will be fully compatible with any associated microprocessor and microcomputer system. *Figure 2-5* gives a brief overview of the minicomputers.

These three levels of compatible hardware — the TMS9900 family parts, the TM990 microcomputer modules, and the 990 minicomputers — provide the flexibility to obtain an optimum match with the user's system application.

The Software and Development Systems Support

New products cannot be made without design, development, test and debug. Development support for all of the levels is shown in *Figure 2-1,* including:

A. Product documentation
B. Software (or firmware)
C. Software development systems
D. Prototyping systems.

Software and development and prototyping systems are outlined in *Figure 2-6.*

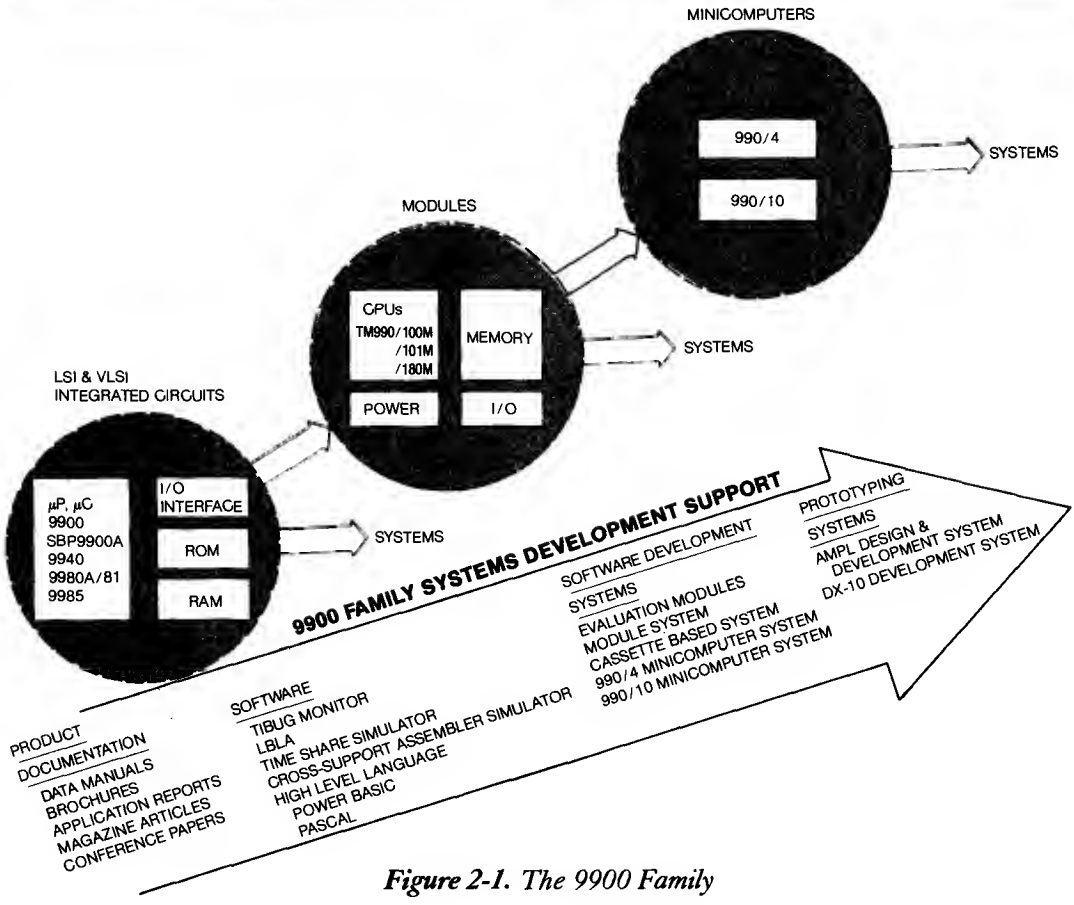


Figure 2-1. The 9900 Family

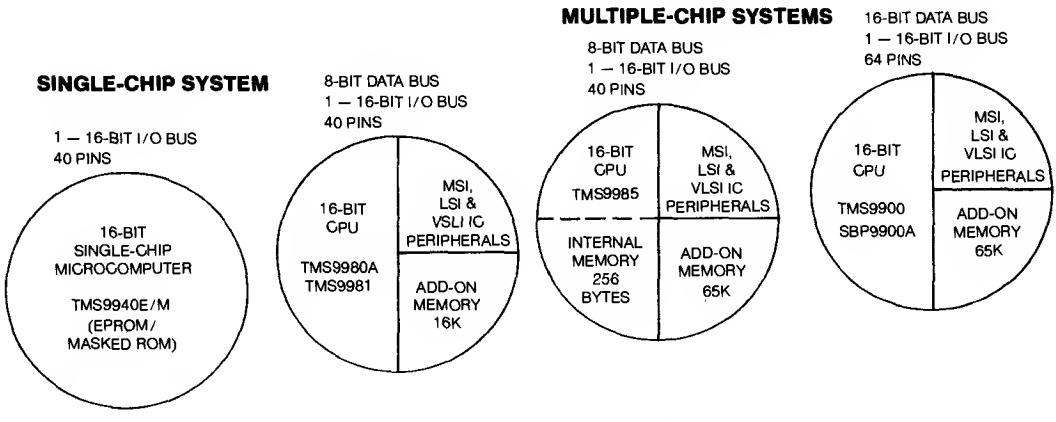


Figure 2-2. 9900 Family CPUs

▶2

CPU's	
TMS9900	NMOS 16-Bit Microprocessor, 64 Pins
TMS9900-40	Higher Frequency Version 9900
SBP9900A	I²L Extended Temperature Range 9900
TMS9980A/ 9981	40-Pin, NMOS 16-Bit Microprocessor with 8-Bit Data Bus. 9981 has XTAL Oscillator
TMS9985	40-Pin, NMOS 16-Bit Microprocessor with Single 5V Supply and 256-Bits of RAM
TMS9940E	40-Pin, NMOS Single Chip Microcomputer, EPROM Version
TMS9940M	40-Pin, NMOS Single Chip Microcomputer, Mask Version

PERIPHERAL DEVICES

TMS9901	Programmable Systems Interface	TMS9914	GPIB Adapter
TMS9901-40	Higher Frequency Version of 9901	TMS9915	Dynamic RAM Controller Chip Set
TMS9902	Asynchronous Communications Controller	TMS9916	92K Magnetic Bubble Memory Controller
TMS9902-40	Higher Frequency Version of 9902	TMS9922	250K Magnetic Bubble Controller
TMS9903	Synchronous Communications Controller	TMS9923	250K Magnetic Bubble Controller
TMS9904	4-Phase Clock Driver	TMS9927	Video Timer/Controller
TMS9905	8 to 1 Multiplexer	TMS9932	Combination ROM/RAM Memory
TMS9906	8-Bit Latch	SBP9960	I/O Expander
TMS9907	8 to 3 Priority Encoder	SBP9961	Interrupt-Controller/Timer
TMS9908	8 to 3 Priority Encoder w/Tri-State Outputs	SBP9964	SBP9900A Timing Generator
TMS9909	Floppy Disk Controller	SBP9965	Peripheral Interface Adapter
TMS9911	Direct Memory Access Controller		

ADD-ON MEMORY

ROMS	EPROMS		DYNAMIC RAMS
TMS4700—1024 X 8	TMS2508	—1024 X 8	TMS4027—4096 X 1
*TMS4710—1024 X 8	TMS2708	—1024 X 8	TMS4050—4096 X 1
TMS4732—4096 X 8	TMS27L08	—1024 X 8	TMS4051—4096 X 1
SBP8316—2048 X 8	TMS2516	—2048 X 8	TMS4060—4096 X 1
SBP9818—2048 X 8	TMS2716	—2048 X 8	TMS4116—16,384 X 1
	TMS2532	—4096 X 8	TMS4164— 65,536 X 1

*Character Generator—ASCII

**PROMS	STATIC RAMS			
SN74S287— 256 X 4	TMS4008	—1024 X 8	TMS4043-2	— 256 X 4
SN74S471— 256 X 8	TMS4016	—2048 X 8	TMS4044	—4096 X 1
SN74S472— 512 X 8	TMS4033	—1024 X 1	TMS40L44	—4096 X 1
SN74S474— 512 X 8	TMS4034	—1024 X 1	TMS4045	—1024 X 4
SN74S476—1024 X 4	TMS4035	—1024 X 1	TMS40L45	—1024 X 4
SN74S478—1024 X 8△	TMS4036-2	— 64 X 8	TMS4046	—4096 X 1
	TMS4039-2	— 256 X 4	TMS40L46	—4096 X 1
△Equivalent to SN74S2708	TMS4042-2	— 256 X 4	TMS4047	—1024 X 4
			TMS40L47	—1024 X 4

**Also available in 54 series

Figure 2-3. Microcomputer Support Components

MICROCOMPUTER MODULES	
TM990/100M	Microcomputer, 1-4K EPROM
TM990/101M	Microcomputer, 1-4K ROM, 1K-2K RAM
TM990/101M-10	Microcomputer, 1-4K ROM, 1K-2K RAM, Evaluation POWER BASIC®
TM990/180	Microcomputer, (8-Bit Data Bus), 1-2K ROM, 256-1K RAM
TM990/189	Microcomputer, University Microcomputer Module
TM990/201	Memory Expansion Module, 4K-16K ROM, 2K-8K RAM
TM990/206	Memory Expansion Module, 4K-8K RAM
TM990/301	Microterminal
TM990/302	Software Development Module
TM990/310	I/O Expansion Module
TM990/401*	TIBUG® Monitor in EPROM
TM990/402*	Line-by-Line Assembler in EPROM
TM990/450*	Evaluation POWER BASIC® —8K Bytes in EPROM
TM990/451*	Development POWER BASIC—12K Bytes in EPROM
TM990/452*	Development POWER BASIC Software Enhancement—4K Bytes in EPROM
TM990/501-521	Chassis, Cable and Power Supply Accessories

*FIRMWARE

Figure 2-4. TM990 Board Modules and Software Support

Software is provided in EPROM (firmware) to operate with the assembled microcomputer modules. It is provided on either "floppy" diskette or on disk pack for use with the minicomputers, and is distributed on magnetic tape for use on in-house computing equipment.

In addition to the development systems available directly from Texas Instruments, a Fortran-IV cross-support package with assembler and simulator is provided by TI for those desiring to use in-house computing equipment. GE, National-CSS and Tymeshare provide similar capabilities on a timeshared basis.

POWER BASIC and PASCAL software systems have just been introduced and will continue to be expanded in the future.

Hardware and software for development and production use is available in appropriate system sizes to support individual designers as well as large design teams.

▶2

CS990/4	• A 990/4 Minicomputer with 4K words of RAM • Expanded memory controller with 4K words of RAM • 733 ASR ROM Loader • 733 ASR Data Terminal • Necessary chassis, power supply, and packaging

FS990/4	• Model 990/4 Minicomputer with 48K bytes of parity memory in a 13-slot chassis with programmer panel and floppy disk loader/self-test ROM • Model 911 Video Display Terminal (1920 character) with dual port controller • Dual FD800 floppy disk drives • Attractive, office-style single-bay desk enclosure • Licensed TX990/TXDS Terminal Executive Development System Software with one-year software subscription service

FS990/10	• Model 990/10 Minicomputer with 64K bytes of error-correcting memory and mapping in a 13-slot chassis with programmer panel and floppy disk loader/self-test ROM • Model 911 Video Display Terminal (1920 character) with dual port controller • Dual FD800 floppy disk drives • Attractive, office-style single-bay desk enclosure • Licensed TX990/TXDS Terminal Executive Development System Software with one-year software subscription service

DS990/10	• Model 990/10 Minicomputer with mapping, 128K bytes of error-correcting memory in a 13-slot chassis with programmer panel and disk loader ROM • Model 911 Video Display Terminal (1920 character) with dual-port controller • Licensed copy of DX10 Operating System on compatible disk media, with one-year software subscription service • DS10 disk drive featuring 9.4M bytes of formatted mass storage, partitioned into one 4.7M-byte fixed disc and a 5440-type removable 4.7M-byte top-loading disk cartridge Options: One additional DS10 disk drive with 9.4M bytes of formatted mass storage, in deskmount, rackmount, or quietized pedestal version

Figure 2-5. 990 Minicomputers

PRODUCT DOCUMENTATION
9900 Family Systems Design and Data Book
9900 Software Design Handbook
TM990 System Design Handbook
990 Computer Family Systems Handbook
Product Data Manuals
Product User's Guides
Product Brochures
Application Notes
Application Sheets

2◀

SOFTWARE AND FIRMWARE	
TM990/401	TIBUG Monitor in EPROM
TM990/402	Line-by-Line Assembler in EPROM
TMSW101MT	ANSI-Fortran Cross-Support Assembler, Simulator and ROM Utility
TM990/450	Evaluation POWER BASIC —8K Bytes in EPROM
TM990/451	Development POWER BASIC — 12K Bytes in EPROM
TM990/452	Development POWER BASIC Software Enhancement Package — 4K Bytes in EPROM
TMSW201F/D	Configurable POWER BASIC in FS990 Diskette
TMSW301F/D	TIPMX — TI PASCAL Executive Components Library

SOFTWARE DEVELOPMENT		SUPPORT SOFTWARE
TM990/302	Software Development Module	Edit, Assembler, Load, Debug, PROM Programming
TM990/40DS	Software Development system for TMS9940 Microcomputer	Assembler, Debug Monitor, Trial-in-System Emulator, PROM Programmer
CS990/4	Single User Software Development System (Cassette Based), uses PX990 software.	Text Editor, Assembler, Linking Loader, Debug Monitor, PROM Programmer
FS990/4	Software Development system (Floppy Disk)	Source Editor, Assembler, Link Editor, PROM Programmer
FS990/10	Software Development System (Floppy Disk)	Same as 990/4, expandable to DS System
DS990/10	Disk Based 990/10 with Macro Assembler	Source Editor, Link Editor, Debug, Librarian, and High-Level Language such as FORTRAN, BASIC, PASCAL, and COBOL

MICROPROCESSOR PROTOTYPING LAB FOR DESIGN AND DEVELOPMENT	
AMPL	FS990 with video display and dual floppy diskettes includes TX990/TXDS system software — Text Editor, Assembler, and Link Utility — and has an in-circuit Emulator Module and a Logic-State Trace Module for proposed system emulation and analysis.

TIMESHARE SYSTEMS	
GE, NCSS, Tymeshare	Assembler, Simulator, ROM Utilities

Figure 2-6. The 9900 Family Software and Development Systems

Typical Applications

The range of applications for microprocessors and microcomputers expands each day; *Figure 2-7* provides a broad scope of the applications extending from those that can be satisfied with single-chip microcomputers to those requiring high performance multichip systems. The market tends to be characterized by lower performance, high volume single-chip systems, and higher performance, low volume multichip systems.

▶2

As shown in *Figure 2-7,* the spectrum of applications is satisfied throughout by 9900 Family members. The single-chip 16-bit microcomputer, the TMS 9940, is used where there is large volume, because it has the lowest cost, yet achieves outstanding performance. At the other end are the system with the 16-bit TMS 9900 and SBP 9900A CPUs, the specially designed family peripherals, and add-on memory. For maximum system performance, the bit slice SN74S481 units are available. For in-between performance limits there are the 16-bit CPUs using 8-bit data buses. The TMS 9980A/81 has lower cost, and the TMS 9985 will accommodate larger memory for extended applications. Both processors use the more economical 40-pin package. Applications in the low and medium performance range include simple instruments, computer peripherals, cash registers and controls for manufacturing.

At the higher performance end, a myriad of products that are emulating many computer-like functions — data terminals, point-of-sale terminals, data acquisition systems, process control systems, military systems — are all gaining performance at lower cost by using microprocessor multichip systems.

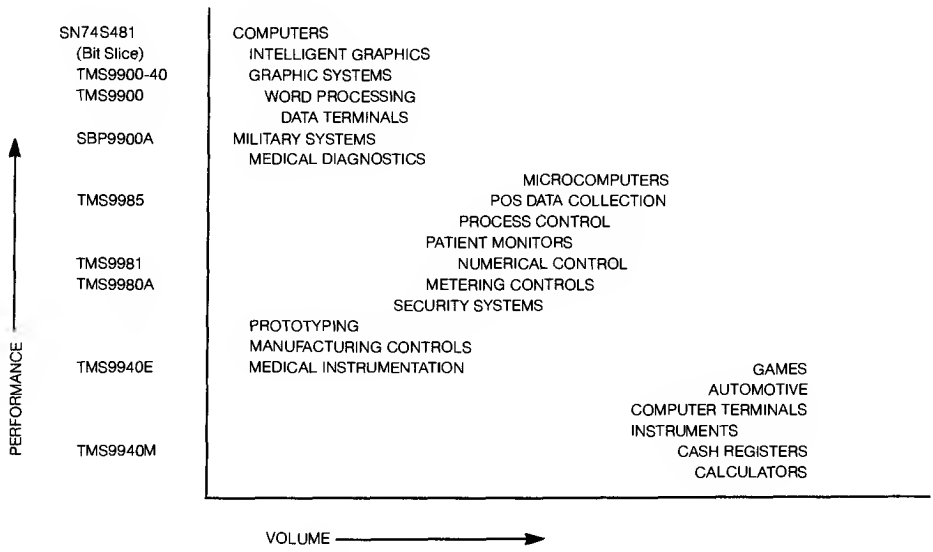


Figure 2-7. Application Spectrum

Figure 2-8 details further the applications for single-chip and multiple-chip systems.

SINGLE-CHIP MICROCOMPUTER	MULTIPLE-CHIP SYSTEMS	
Gas Pump Control	Video Controllers	CPU's-Microcomputer
Alarm Systems	Telephone Switching	Computer Peripherals
Paging Systems	Word Processing Equipment	Intelligent Terminals
Sorters	Manufacturing Material Handlers	Tape Drive Controls
Vending Machines	Electronic Musical Instruments	Graphic Terminals
Microwave Ovens	Small Business/Financial Systems	Communications Network
Appliance Control	Factory Automation	Communications Processing
Power Tools	Instrumentation	Data Concentration
Utility Meter Monitoring	Data Acquisition	Input Terminals
Environmental Controls	Machine Controls	General Purpose Terminals
Automotive	Medical Equipment	
Games	Security Systems	
Cryptography	Machine Tool Controls	
Process Controls		
Navigation Equipment		
Metering Controls		

Figure 2-8. Applications

HARDWARE SELECTION

To reduce the range of detail which must be considered in a given system design, it is often possible to make a definite choice between the three hardware design levels; designing with individual components, designing with prefabricated modules, and designing with minicomputer subsystems. The criteria upon which this choice is based include the number of units to be manufactured per design, complexity of design, performance requirements, special feature requirements, microprocessor system design expertise available, and the importance placed on product introduction — the time to the market place. General tendencies of these decisions are known although the particular choice may be skewed by other considerations. Here are a number of examples.

In terms of production volume, users tend to incorporate minicomputers in their designs up to a volume of 50 to 100 identical systems per year. They tend to use prefabricated modules if the volume is below 500 to 2000 systems per year, and for higher volume, they tend to develop from the component level right from the start. Simple systems may not be able to stand the cost of a minicomputer at any volume, while even at much higher volumes, performance requirements may force the utilization of a disk-based minicomputer. When system specifications require special features, this often forces the use of modules even at low volumes. However, the need for maintenance capability may force the use of minicomputers or prefabricated modules for system construction at extremely large volumes. A firm with expert microprocessor design teams would tend to maximize its value-added by designing from the component level, while a firm without hardware designers would look for completely prefabricated systems.

Finally, product introduction priorities often call for a compromise approach because of an urgent need to get a product to market ahead of competition. It is often ideal to enter the market with prefabricated systems and switch to in-house fabrication as the system is accepted and sales volume builds.

THE COMPONENT ROUTE: CPU

▶2 In the beginning your product selection decisions are tied entirely to the central processor. A very real danger at this point is choosing a processor which is not optimum for the design. Either the cost will be greater than desired, or the processor will not quite meet the required performance. In the TMS9900 Family, each processor is uniquely tuned to its applications environment while maintaining a common architecture, input/output system and instruction set. This commonality allows a simple move up or down the performance scale with a minimum of redesign *(See Figure 2-9).*

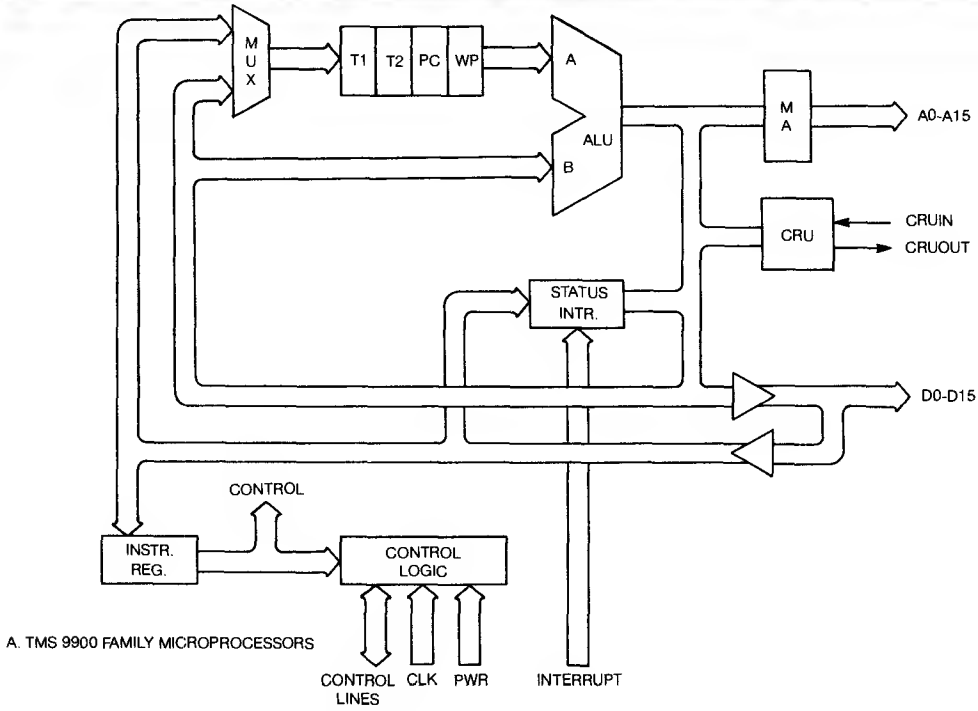
The single packaged CPUs divide into microprocessors and a microcomputer. The TMS 9940 microcomputer is available either with EPROM or with mask programmable ROM.

Microprocessors		*Microcomputer*
TMS9900	TMS9980A/81	TMS9940 E/M
TMS9900-40	TMS9985	
SBP9900A		

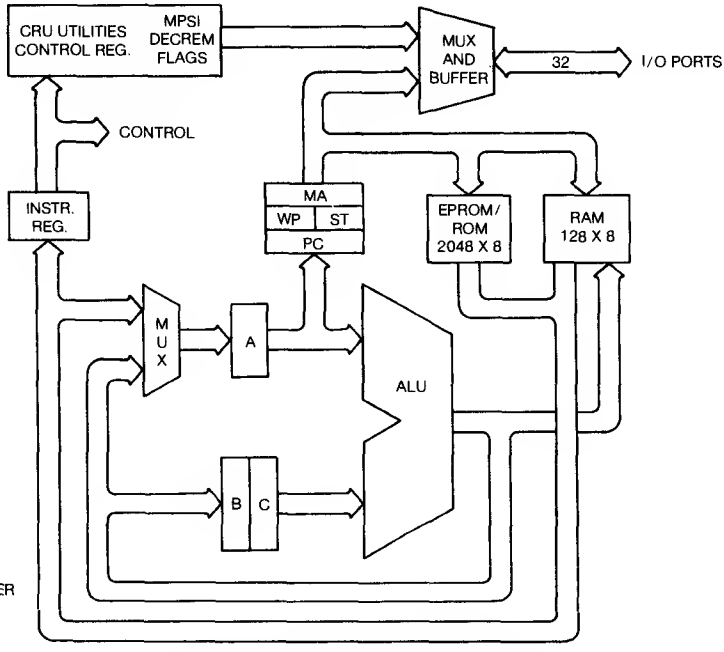
The basic architecture of each is shown in *Figure 2-10.*

CRITERIA	SYSTEM CHOICE	FAMILY PRODUCTS USED
HIGHEST PERFORMANCE	MULTIPACKAGE	1. TMS9900, SBP9900A 2. Microcomputer peripherals for I/O 3. TIM9904 for clock 4. ROM, EPROM 5. RAM
TRADEOFF FOR BEST COST AT PERFORMANCE REQUIRED	A. MINIMUM PACKAGES	1. TMS9980A/81 (with clock) 2. Microcomputer peripherals for I/O 3. Combined ROM & RAM
	B. MINIMUM PACKAGES	1. TMS9985 (with clock and RAM) 2. Microcomputer peripherals for I/O 3. ROM
LOWEST COST	SINGLE PACKAGE	1. TMS9940 Microcomputer with on board I/O, Clock, ROM & RAM, Timer

***Figure 2-9.** Cost/Performance Trade-off*



A. TMS 9900 FAMILY MICROPROCESSORS



B. TMS9940 MICROCOMPUTER

Figure 2-10. Basic Architecture of 9900 Family

CPU Selection

Selecting a CPU for an application requires a study of the CPU characteristics to see which one fits best. *Figure 2-11* provides key characteristics of the 9900 Family CPUs as well as a bit-slice version (SN54/74S481) for the ultimate in performance.

▶2

FEATURE \ DEVICE	SN54S481 SN74S481 Note 1	SBP9900A	TMS9900/ TMS9900-40	TMS9980A/ TMS9981	TMS9985	TMS9940E/M
Number of bytes addressable	65K	65K	65K	16K	65K 256 on chip	2K EPROM/128 RAM 2K/128 RAM/128 RAM
Number of Interrupts	16	16	16	5	5	4
Number of Pins	48/chip	64	64	40	40	40
Power Supply Requirements	+5	Resistor Programmable Note 2	+12, ±5	+12, ±5	+5	+5
Technology	Schottky TTL	I²L	NMOS	NMOS	NMOS	NMOS
Environmental (Temperature, °C)	−55 to 125	−55 to 125	0 to 70	0 to 70	0 to 70	0 to 70
Clock Rate	10MHz	3MHz	3.3MHz/4MHz	10MHz	5MHz	5MHz
Relative Thruput	6	0.9	1.0/1.3	0.6	0.65−0.8 Note 3	1.2
Number of Address Bus Lines	15	15	15	14	16	Note 4
Number of Data Bus Lines	16	16	16	8	8	Note 4
Clock	SN54S124	SN54LS124	TIM9904	On Chip	On Chip	On Chip

Note 1 : Based on four slices microcoded to duplicate TMS9900.
Note 2: Voltage for the SBP9900A is 1.5 to 30 volts with a series resistor.
Note 3: Relative thruput is 0.65 with off-chip RAM and 0.8 with on-chip RAM.
Note 4: No external memory or data bus. 32 general purpose I/O pins 10 of which provide 256 bit CRU I/O expansion if desired.

Figure 2-11. Key Characteristics of 9900 Family CPUs

Figure 2-12 provides, in a "quick look" format, four specifications of the family members that are usually important to all applications — the directly addressable memory, the data bus length, the operating temperature, and the package size.

Figure 2-13 plots the relative thruput of the 9900 Family microprocessors and microcomputers. The thruput, estimated by calculating execution times for a given benchmark program, is plotted relative to the performance of the TMS9900. 30% more thruput is obtained using the TMS9900-40. The thruput of the SBP9900A is 90% of the TMS9900. Both of these processors operate with a full 16-bit data bus and are in 64-pin packages. As mentioned previously, ultimate performance is obtained by using a bit-slice microprocessor. A relative thruput of six is shown for four SN54/74S481 bit-slice packages microcoded to duplicate a TMS9900.

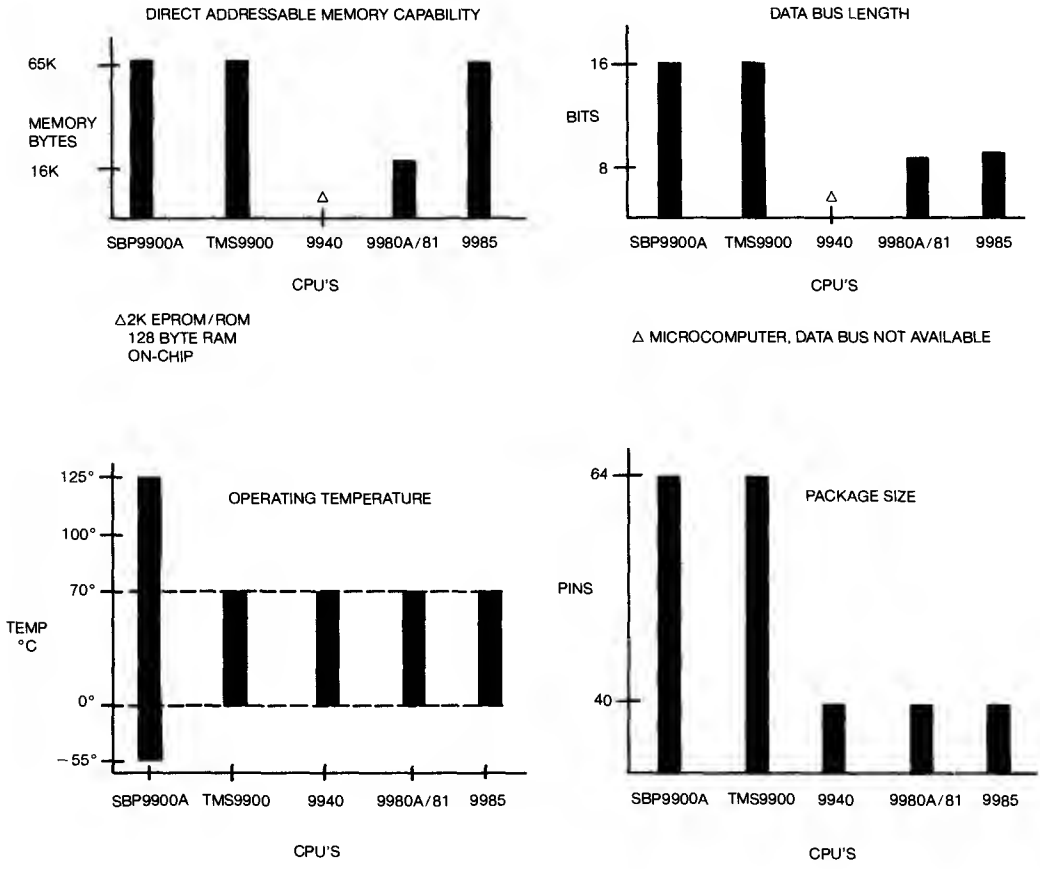


Figure 2-12: "Quick Look" at 9900 Family CPU's

▶2

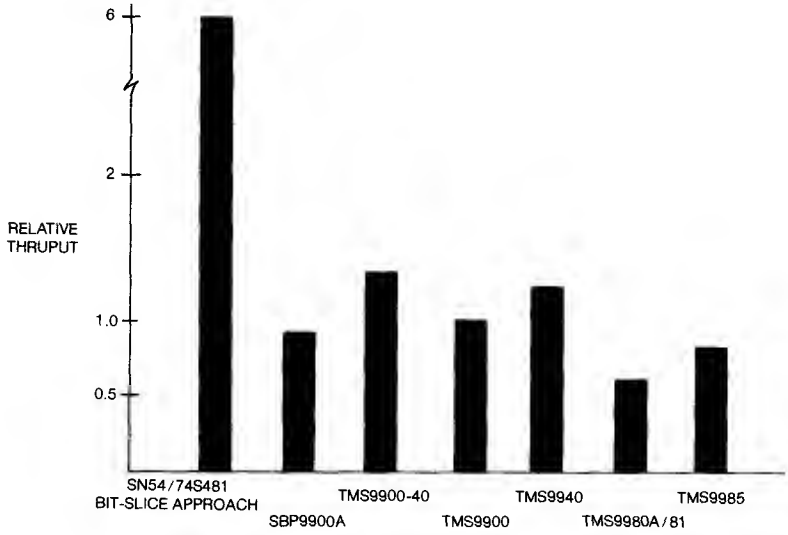


Figure 2-13. Thruput of 9900 Family CPU'S

Cost reduction can be realized via 40-pin packaging. This is accomplished by changing the external operating configuration to an 8-bit format even though the internal processor is a 16-bit processor. This causes a reduction in thruput — the thruput of the 9980A/81 and 9985 is reduced to 60% to 80% of the TMS9900 — because a byte organized memory is required and the number of memory accesses will obviously be increased. The advantage, of course, is that family software can be used even though the 8-bit configuration is used. Note that the 9940 microcomputer thruput is 20% *greater* than the TMS9900. Excellent performance is obtained from this single-chip microcomputer.

The 9980A/81 is designed for the lowest system cost for full family performance while the 9985 spans the link between microprocessor and microcomputer by offering RAM memory on board.

Flexible I/O

The TMS9900 provides non-multiplexed parallel I/O and memory control for maximum performance when needed, with full 16-bit address and data bus. It also has a separate serial bus to allow use of minimum cost, maximum funtionality peripherals for relatively slow I/O processes which will tolerate the reduced speed. This is the Communications Register Unit, CRU.

Family Members Fitted to the Application

The TMS9980A/81 and the TMS9985 multiplex the data bus for reduced cost and package size at some sacrifice in performance. The TMS9940 is the least expensive approach for those applications which will tolerate the limitations of a single-chip. It provides full computer capabilities, albeit of a limited range, on a single integrated circuit. By not taking the address and data bus off-chip, buffer time delays are eliminated resulting in higher performance within a limited memory range (2K EPROM/ROM, 128 bytes RAM). For those applications requiring better temperature or reliability performance than that available from NMOS processors, the SBP9900A provides the same sophisticated processor functions as the TMS9900 over military and industrial temperature and specification ranges.

Interrupt Flexibility

The 9900 Family provides fully prioritized, vectored interrupts as well as software vectored interrupts for maximum flexibility.

Advantages of 9900 Family CPUs

True Compatibility

The greatest advantage of using the 9900 Family as mentioned earlier, is the fact that it is a truly compatible family. Many so-called families of CPUs are not truly compatible in instruction set, in I/O interfacing, or in architecture. The 9900 Family attains compatibility in all three areas. It is difficult and expensive to move from the use of one microcomputer family to the use of another. Of equal importance, in non-compatible families, it is often just as expensive to move from one member to another. When faced with such a move, serious consideration should be given the 9900 family, because doing so can eliminate most of the trauma of future moves, and quite possibly ease the present one.

Lower Costs

As seen in Chapter 1, while system costs are dropping at a 15% to 20% yearly rate, software costs are actually rising. Thus, a family that provides the same and more capabilities with less programming saves software costs. The sophisticated memory-to-memory instruction set of the 9900 Family eases assembly language programming, at the same time reducing the memory storage requirement and increasing execution speed.

Instruction Set

Instruction sets are inherently difficult to compare. *Figure 2-14* is a relatively simple way. It gives three numbers for each of three representative microprocessor families. The second number is the number of instructions used by manufacturers when advertising their product. In many cases it has little to do with the power of the instruction set. The first number is the number of distinct functions included in the instruction set. It represents to a certain extent the uniqueness of the instruction set. In the 8080 and 6800 families, instructions which take care of redundant actions solved automatically by the 9900 Family are not included. The third number represents combinations. Many advertised instructions are obtained by giving a separate name to particular combinations of the basic functions and addressing modes. Many of these are possible. The last set of numbers shows the result of taking the third number to its extreme and listing all possible combinations for each of the families. 62,235 are possible for the 9900 Family. The number of possible combinations is derived from the fact that certain instructions leave several bits unspecified to allow for a variety of addressing modes. In the 9900, 12 instructions (Add, Subtract, etc.) leave 12 bits unspecified, so there are 4096 (2^{12}) variations of each, times 12, or 49,152 combinations. Eight-bit instruction sets simply do not allow this degree of flexibility.

	PROCESSOR		
INSTRUCTIONS	8080	6800	9900
DISTINCT	27	26	36
ADVERTISED	78	72	69
COMBINATIONS	237	169	62,235

Figure 2-14. Instruction Set Comparison

Memory-to-Memory Architecture

Memory-to-memory architecture means high speed context switching in interrupt processing and in subroutine processing. All processors must save the contents of the CPU registers as a prerequisite to processing an interrupt service routine. The register contents to be saved include the PC (program counter), ST (Status register), and one or more general registers. For the 9900, the registers to be saved are *only* the PC, ST and WP (workspace pointer).

The Component Route: Peripherals

Microcomputer component peripherals perform functions that assist the CPU in a microprocessor or microcomputer system. Data communications through serial data links in a synchronous or asynchronous mode; parallel input and output interfaces for general purpose I/O, instrument communications, direct memory access or mass storage control; and display control and memory expansion and control are some of the present peripheral functions provided as shown below.

FAMILY UNITS—INTERFACING TECHNIQUES	
Serial I/O for Data Communications	
Asynchronous Communications Controller	TMS9902
4 MHz Version	TMS9902-40
Synchronous Communications Controller	TMS9903
Parallel I/O	
General Purpose	
Programmable Systems Interface	TMS9901
4 MHz Version	TMS 9901-40
I/O Expander	SBP9960
Interrupt—Controller/Timer	SBP9961
Instrument Communications	
General Purpose Interface Bus Adapter	TMS9914
Direct Memory Access	
Direct Memory Access Controller	TMS9911
Mass Storage	
Floppy Disk Controller	TMS9909
CRT Display (Memory Mapped I/O)	
Video Timer/Controller	TMS9927
Memory	
Combination ROM/RAM Memory	TMS9932
Memory Control	
Dynamic RAM Controller Chip Set	
Refresh Timing Controller	TIM9915A
Memory Timing Controller	TIM9915B
Multiplexer/Latch	TIM9915C
FAMILY UNITS—SUPPORT LOGIC	
Four-Phase Clock Driver	TIM9904
8 to 1 Multiplexer	TIM9905
8-Bit Latch	TIM9906
8 to 3 Priority Encoder	TIM9907
8 to 3 Priority Encoder W/Three State Outputs	TIM9908

Significant progress has been made in implementing these important functions in high-functional-density designs for the 9900 Family. This integration will continue in the future. It provides cost-effective package substitutions for multiple standard TTL units. The result is reduced assembly costs and materials, increased reliability, and shorter time from design to production.

▸2 As the key features of the microcomputer component peripherals are reviewed, note these points: (1) Many of the peripherals units are as complex or even more complex than the CPUs they support; (2) Many of the peripheral units are designed to be *programmable* providing outstanding flexibility to vary their use in system applications. Such design trends reinforce the systems concept of the future—that standard hardware will be used but varied in use by software; (3) Family units will drive two TTL loads, allowing direct interface to low-power Schottky, standard TTL, and even standard Schottky circuits, eliminating the need for many special purpose peripherals which do little else than provide this interface.

Interface Techniques

A computer must be controlled by a person or another machine to be useful. It must be programmed to accept inputs, process data, and give results as outputs. It will do only what it is programmed to do (barring malfunction). Output results must be acted upon otherwise the computer manipulations are worthless. Peripheral components form the required systematic interface between the computer and the outside world and range in functional capability from the general purpose to highly specialized units.

The interface of a microcomputer or microprocessor system to external inputs and outputs is by serial or parallel data lines. Two parallel and two serial techniques are used. The parallel techniques include direct memory access and CPU controlled I/O. The serial techniques include asynchronous and synchronous serial I/O. A final technique called interrupt is used to alert the processor of a change in external conditions.

Serial I/O for Data Communications

Serial I/O for data communications is handled through the TMS9902 and TMS9903. The TMS9902 and TMS9902-40 are for asynchronous serial data that is established around the RS232C protocol and the TMS9903 is for synchronous data, designed for any high-speed communications protocol. CPU control of these devices, as show in *Figure 2-15* via the Communications Register Unit, allows their construction in small, plug-compatible packages.

Parallel I/O

GENERAL PURPOSE

General purpose parallel I/O and interrupt control along with an on-chip timer are provided by the TMS9901 and TMS9901-40, as shown in *Figure 2-15.* The same functions are served in I^2L for extended temperature range operation by the SBP9960 and SBP9961.

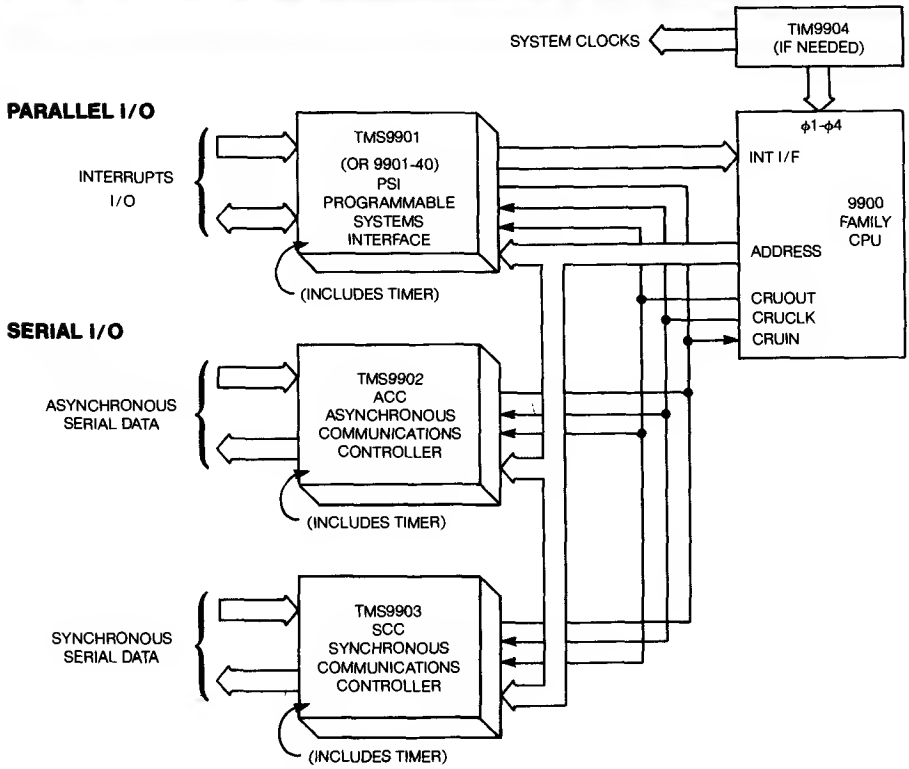


Figure 2-15. Microcomputer Component Peripherals for I/O Interface

A significant advantage of the 9900/9901 parallel I/O interface through the CRU is the ability to transfer fields of from 1 to 16 bits of data as inputs or outputs under the command of one instruction and to modify this structure from instruction to instruction. Additionally, use of the CRU allows implementation of multiple functions in the TMS9901.

MEMORY-MAPPED

Since the CRU is essentially a time-division multiplexed serial port, speed critical applications may require a faster parallel technique. Memory-mapping, the treatment of a parallel I/O port as if it were a memory location, provides this technique. With the memory-to-memory architecture of the 9900 Family, direct manipulation of such an I/O port is practical. The dual-TTL drive of the 9900 Family allows economical construction of memory-mapped I/O ports using standard TTL or LS (Low-Power Schottky) components.

GPIB—GENERAL PURPOSE INTERFACE BUS

In 1975 the IEEE defined a very precise electrical and mechanical protocol designated the IEEE 488 Interface Bus, or commonly known as a General Purpose Interface Bus (GPIB). This protocol allows direct connection of instruments and processors supplied by various manufacturers. The TMS9914 General Purpose Interface Bus Adapter either directly, or under software control, adapts all the capabilities of the GPIB to a microprocessor bus including talker, listener, controller and control passer. This is a general purpose component and will work quite well with any microprocessor, although it is complemented by the speed and power of the 9900 Family.

DMA—DIRECT MEMORY ACCESS

Many I/O devices can be made more effective if transfer rates can be increased beyond the 8 microseconds required for a typical memory-mapped transfer. The GPIB mentioned above, for instance, allows data transfers at rates up to a million bytes per second. The TMS9911 Direct Memory Access Controller allows low cost implementation of two such super high speed ports. The TMS9911 itself is controlled by the CPU via the CRU bus, until one of the DMA channels takes control long enough to process a DMA transfer (either single or block) between I/O port and memory.

FLOPPY DISK

For those applications requiring more storage space than is convenient or economical in a microcomputer, a mass storage device is needed. Floppy disk units provide the benefits of fast access, reliable mass storage using a portable, easily stored media. Interfacing these units to microprocessors is greatly simplified by the TMS9909 Floppy Disk Controller. This device will control up to four floppy disk units using standard or minifloppies, single, double, or triple density, hard or soft sectors. It is also capable of full IBM compatibility (including double-sided, double density at the *same* time). This is a general purpose component and will work quite well with any microprocessor. It is a memory-mapped device and will also interface easily to a DMA controller such as the TMS9911. The TMS9909 can be programmed for:

1. Data encoding formats
2. Number and type of diskette drives
3. Stepper motor control rates
4. Number of sectors and tracks

It can perform the following functions:

1. Step to any track on the diskette
2. Format tracks (set initial conditions on diskette)
3. Read and write diskette data
4. Send status to the host system

CRT CONTROL

The TMS9927 video timer/controller is a memory-mapped device which contains all of the logic necessary to generate all the timing signals for display of video data on CRT monitors, standard or not, and interlaced or not.

This video timer/controller has nine 8-bit registers used for programming; seven for horizontal and vertical formatting and two for the cursor character and address. All the functions for generating the timing signals for video data display are programmable:

1. Characters per row
2. Data rows per frame
3. Raster scans per data row
4. Raster scans per frame

All timing functions are implemented on the chip except the dot generation and dot counting which operate at video frequency. A character generator and shift register are used to shift out video data. The control registers can be loaded by the microprocessor or from PROM. This is a general purpose part for use with any microprocessor.

MEMORY

Contained in the microcomputer component peripherals is a unit for memory expansion, the TMS9932, a combination ROM/RAM memory unit with 1920 bytes of ROM and 128 bytes of RAM. It contains the same key features that characterize the 9900 Family support memory.

MEMORY CONTROL

The TIM9915 chip set consists of 3 packages, a 16-pin Refresh Timing Controller (TIM9915A), a 16-pin Memory Timing Control (TIM9915B), and a 28-pin Multiplexer/Latch with tri-state outputs (TIM9915C). *This chip set becomes the complete packaged set for 4K to 64K of dynamic RAM memory, and provides all the timing and control signals necessary to interface dynamic RAM memory and make it appear as static RAM.*

Clock and Support Logic

Four-Phase Clock Generator/Driver

Microprocessor and microcomputer systems require clock generators and drivers for the timing control of the system. The TMS9904 is such a unit. An oscillator which can be crystal or inductance controlled provides the basic timing source. Four high-level clock phases provide the 9900 microprocessor timing. Four additional TTL-level clocks can be used to time memory or other control functions in a 9900 system.

Support Logic

Common TTL MSI peripherals included in the 9900 Family of microcomputer components are:

TIM9905	8 to 1 Multiplexer
TIM9906	8-Bit Latch
TIM9907	8 to 3 Priority Encoder
TIM9908	8 to 3 Priority Encoder w/Tri-State Outputs

▶2

The reason, of course, is that they are standard units for accomplishing the following tasks:

1. Parallel-to-Serial Conversion
2. Multiplexing from N-lines to one line
3. Providing multiple data selectors
4. Providing bus interface from multiple sources
5. Encoding 10 line decimal to 4 line BCD
6. Encoding 8 lines to 3 lines

All units are fabricated using standard low-power Schottky TTL technology in 16-pin packages. They have tri-state output drivers to interface directly with a system bus and are fully compatible with all TTL interfaces.

Cost Effectiveness of NMOS LSI Peripherals

Figure 2-16 clearly demonstrates the cost effectiveness of the specially designed CRU microcomputer component peripherals. The replacement of large numbers of less complex packages provides a significant reduction in cost due to simplified design, layout, assembly and testing, besides the reduced material costs.

In addition, there are major contributions to improving the reliability of the system just by reducing the number of packages and the associated solder connections and assembly connections external to the IC.

FUNCTION	UNIT USED	SSI AND MSI PACKAGES REPLACED
INTERRUPTS AND I/O	TMS9901	23
ASYNCHRONOUS SERIAL COMMUNICATIONS	TMS9902	45
SYNCHRONOUS SERIAL COMMUNICATIONS	TMS9903	100

***Figure 2-16.** System Package Reduction Using Microcomputer Component Peripherals*

CRU Interface

In the features for the 9900 Family, the Communications Register Unit interface provides:

1. The most cost effective I/O for low and medium speed peripherals via the instruction driven serial data link.
2. Completely separate address space.
3. A choice of transferring fields of 1 to 16 bits per instruction.

The CRU serial data link is an effective mechanism for operation-per-instruction I/O. The CRU interface is simpler and therefore less expensive than memory-mapped I/O. In applications where there are many I/O transfers of one or two bits, the CRU serial data link provides execution times that are better than for memory-mapped I/O, which always transfers 8 or 16 bits at a time.

One way of demonstrating the cost effectiveness of the CRU is shown in *Figure 2-17.* Package pins per function are less using the CRU interface and the 9900 Family units. Thus, costs are saved over memory-mapping in implementing the example I/O functions shown.

FUNCTION	CRU PINS	MEMORY MAPPED PINS
8-Bit Output	16 (TIM9906)	24
UART	18 (TMS9902)	24-40
USRT	20 (TMS9903)	24-40

Figure 2-17. CRU vs Memory Mapped I/O — Package Pins Required Per Function

The Component Route: Memory

Semiconductor memory is the most natural storage media to add to a 9900 system. It has fast access times, an interface that is completely compatible with the microprocessor or microcomputer, and high-density storage per package. Texas Instruments offers a broad spectrum of storage media products in support of the 9900 Family as shown in *Figure 2-18, 2-19* and *2-20.* These products encompass dynamic and static random access memory, mask programmable read-only memory, fused-link programmable read-only memory, and eraseable programmable read-only memory.

▶ 2

WORDS	BITS PER WORD			
	1	2	4	8
16			RAMs SN54/74S189 SN54/74S289	
32				ROM SN54/7488A PROMS SN54/741[illegible] SN5[illegible] [illegible]288
64	RAMs SN10140 SN10142 SN1014[illegible]			RAM TMS 4036 PROM SN54/74[illegible]
128				
256	RAMs SN10144 SN54/74S[illegible] SN5[illegible]		RAMs TMS 4039, TMS 4042, T[illegible] SN74S207 SN74S[illegible] ROMs SN54[illegible] PROMs SN54/74S287 SN54/74S387	ROMs SN54/74S271 SN54/74S371 PROM [illegible]470 [illegible]74S471
512			ROMs SN54/74S270 SN54/74S[illegible]	PROMs SN54/74S472 SN54/74S473 SN54/74[illegible] S[illegible]
1024	RAMs TMS 1103 TMS 4033 TMS 4034 TMS [illegible] TMS 40[illegible] S[illegible]74S309 [illegible]LS214 SN74LS215 SN74S,LS314 SN74LS315		RAMs TMS 4045 TMS 40L45 TMS 4[illegible] T[illegible] PROMs SN[illegible] [illegible]77	RAM TMS 4008 EPROMs TMS 2508 TMS 2[illegible] T[illegible] ROMs TMS[illegible] TMS [illegible]0 PROMs SN74S478 SN74S270B
2048			PROM SN74S454	RAM TMS 4016 EPROMs TMS 25[illegible] T[illegible] ROMs TMS [illegible] [illegible] TSP 9818 PROM SN74S452
4096	RAMs TMS 4027 TMS 4044 TMS [illegible]051 TMS 4060 TMS 40[illegible]46 T[illegible]		ROM TMS 4800	ROM TMS 4732 EPROM T[illegible]
8192				
16384	RAM TMS 4116			
32768				
65536	RAM TMS 4164 CCD TMS 3[illegible]			

Figure 2-18. Matrix of Memory Products

2◀

DYNAMIC RAMS

TYPE	ORGAN.	TECH	PINS	ACCESS TIME-ns	*PKG. P_D-mW	POWER	TEMP
TMS4027	4096 X 1	NMOS	16	150-200	460	+5V, −5V, +12V	0°C-70°C
TMS4050	4096 X 1	NMOS	18	200-300	420	+5V, −5V, +12V	0°C-70°C
TMS4051	4096 X 1	NMOS	18	250-300	460	+5V, −5V, +12V	0°C-70°C
TMS4060	4096 X 1	NMOS	22	150-300	400	+5V, −5V, +12V	0°C-70°C
TMS4116	16,384 X 1	NMOS	16	100-200	462	+5V, −5V, +12V	0°C-70°C
TMS4164	65,536 X 1	NMOS	16	100-150	200	+5V	0°C-70°C

*MAXIMUM

STATIC RAMS

TYPE	ORGAN.	TECH.	PINS	ACCESS TIME-ns	*PKG. P_D-mW	POWER	TEMP.
TMS4008	1024 X 8	NMOS	24	150-450	450	+5V	0°C-70°C
TMS4016	2048 X 8	NMOS	24	150-450	495	+5V	0°C-70°C
TMS4033	1024 X 1	NMOS	16	450	368	+5V	0°C-70°C
TMS4034, 35	1024 X 1	NMOS	16	650-1000	368	+5V	0°C-70°C
TMS4036-2	64 X 8	NMOS	20	450-1000	450	+5V	0°C-70°C
TMS4039-2	256 X 4	NMOS	22	450-1000	368	+5V	0°C-70°C
TMS4042-2	256 X 4	NMOS	18	450-1000	368	+5V	0°C-70°C
TMS4043-2	256 X 4	NMOS	16	450-1000	368	+5V	0°C-70°C
TMS4044	4096 X 1	NMOS	18	150-450	440	+5V	0°C-70°C
TMS40L44	4096 X 1	NMOS	18	200-450	275	+5V	0°C-70°C
TMS4045	1024 X 4	NMOS	18	200-450	495	+5V	0°C-70°C
TMS40L45	1024 X 4	NMOS	18	200-450	300	+5V	0°C-70°C
TMS4046	4096 X 1	NMOS	20	150-450	440	+5V	0°C-70°C
TMS40L46	4096 X 1	NMOS	20	200-450	275	+5V	0°C-70°C
TMS4047	1024 X 4	NMOS	20	150-450	495	+5V	0°C-70°C
TMS40L47	1024 X 4	NMOS	20	200-450	300	+5V	0°C-70°C

*MAXIMUM

Figure 2-19. Dynamic and Static RAM in Support of 9900 Family

READ ONLY MEMORIES

ROM

TYPE	ORGAN.	TECH.	PINS	ACCESS TIME-ns	PKG. P_D-mW	POWER	TEMP.
TMS4700	1024 X 8	NMOS	24	450	*580	+5V, −5V, +12V	0°C-70°C
△TMS4710	1024 X 8	NMOS	24	450	*580	+5V, −5V, +12V	0°C-70°C
TMS4732	4096 X 8	NMOS	24	450	*788	+5V	0°C-70°C
SBP8316	2048 X 8	I²L	24	650	500	+5V (1.5V-30V)	0°C-70°C −55°C to +125°C
SBP9818	2048 X 8	I²L	24	200	500	500 MA	0°C-70°C −55°C to +125°C

△Character Generator
*MAXIMUM

PROMS

TYPE	ORGAN.	TECH.	PINS	ACCESS TIME-ns	PKG. P_D-mW	POWER	TEMP.
SN54/74S287	256 X 4	TTL(s)	16	42	*708	+5V	SN54: 0°C-70°C
SN54/74S471	256 X 8	TTL(s)	20	50	*814	+5V	SN74: −55°C to +125°C
SN54/74S472	512 X 8	TTL(s)	20	55	*814	+5V	"
SN54/74S474	512 X 8	TTL(s)	24	55	*814	+5V	"
SN54/74S476	1024 X 4	TTL(s)	18	35	*735	+5V	"
SN54/74S478	1024 X 8	TTL(s)	24	45	600	+5V	"
SN54/74S2708	1024 X 8	TTL(s)	24	45	600	+5V	"

*MAXIMUM

EPROMS

TYPE	ORGAN.	TECH.	PIN	ACCESS TIME-ns	*PKG. P_D-mW	POWER	TEMP.
TMS2508	1024 X 8	NMOS	24	350	500	+5V	0°C-70°C
TMS2708	1024 X 8	NMOS	24	350-450	800	+5V, −5V, +12V	0°C-70°C
TMS27L08	1024 X 8	NMOS	24	450	475	+5V, −5V, +12V	0°C-70°C
TMS2516	2048 X 8	NMOS	24	450	525	+5V	0°C-70°C
TMS2716	2048 X 8	NMOS	24	450	595	+5V, −5V, +12V	0°C-70°C
TMS2532	4096 X 8	NMOS	24	450	840	+5V	0°C-70°C

*MAXIMUM

Figure 2-20. Read Only Memory in Support of 9900 Family

Standard Memory and Compatibility

A very important characteristic of the 9900 Family is that it uses standard semiconductor memory not memory that depends on a custom multiplexed or latched bus. The full range of MOS, TTL, I^2L and ECL memories are shown in *Figure 2-18.* Many of these units that support the 9900 Family are pin-compatible for ease in conversion from development systems to production systems.

As an example, during development, package flexibility is provided. Initially, a static RAM is used; then EPROM's can be substituted as the system design stabilizes, and when the system is proven and in volume production, mask ROM can be substituted.

SRAM
↓
EPROM
↓
ROM

Here is an example of the socket compatibility:

MEMORY SIZE	SRAM	EPROM	PROM	ROM
1K × 8	4008	2508	SN74S2708/478	
1K × 8		2708	SN74S2708/478	4700
2K × 8	4016	2516		SBP8316, SBP9818
4K × 8		2532		4732

(All of these devices can fit a single socket.)

Even if the memory units are not completely compatible, due to power supplies or control pins, simple jumpers can be used to maintain socket compatibility.

Read-Only Memory: Costs and Flexibility

Figure 2-21 shows the characteristics of read-only memories and their cost per bit vs. design flexibility. Mask programmable read-only memory is lowest cost per bit but also has no flexibility. It is used for high volume production after a design is proved to be correct and no changes are expected. PROMs have excellent performance and have more flexibility because programming is done after they are manufactured. However, once programmed they cannot be changed. PROMs cost somewhat more than ROMs because they use more real estate. EPROM has much more flexibility because design changes are done quickly and because it is reuseable, but EPROM costs more to manufacture than ROM or PROM because it is eraseable. EAROM is also indicated in *Figure 2-21.* This is really "read mostly" memory, because it can be erased in a relatively short period of time (microseconds), but once programmed again, it acts like fixed storage. EAROMs as a practical product are still a bit in the future. The flexibility of EPROMs is well worth the added cost. This is especially true when used as a prototyping tool.

A. READ-ONLY MEMORY CHARACTERISTICS

	ROM	PROM	EPROM	EAROM
COST (RANK)	1	2	3	4
PROGRAM TIME	WEEKS	MINUTES	MINUTES	MICROSECONDS
SETUP CHARGE	YES	NO	NO	NO
REUSABILITY	NO	NO	YES	YES
SPEED	FAST	VERY FAST	MEDIUM	SLOW

B. READ-ONLY MEMORY COST/BIT VS FLEXIBILITY

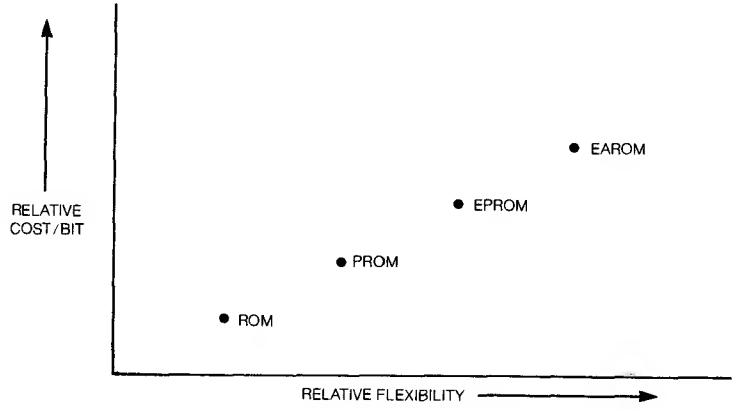


Figure 2-21. Read-Only Memory Overview

The Component Route: Miscellaneous Components

Included in the full support of the 9900 Family is the large array of SSI, MSI and linear integrated circuits. Information on all components manufactured by Texas Instruments may be found in the following data books:

Power Data Book	LCC4041
TTL Data Book (Second Edition)	LCC4112
TTL Data Book (2nd Edition Supplement)	LCC4162
Transistor and Diode Data Book	LCC4131
Semiconductor Memories Data Book	LCC4200
Optoelectronics Data Book (Fourth Edition)	LCC4230
Optoelectronics Data Book (Fifth Edition)	LCC4410
Linear Control Circuits Data Book	LCC4241
Bipolar Microcomputer Components Data Book	LCC4270
Interface Circuits Data Book	LCC4330
Electro Optical Components	LCC4340
Voltage Regulator Handbook	LCC4350
MOS Memory — 1978	LCC4380
9900 Family Systems Design Book	LCC4400

Correspondence and inquiries about these books should be directed to:

Texas Instruments Incorporated
P.O. Box 225012, M/S 54
Dallas, Texas 75265
(214) 238-3894

Most of the above are concerned with SSI and MSI integrated circuits. The following is a list of additional guides to discrete components. 2◄

Optoelectronics Master Selection Guide	CL-346	1978
Discrete Semiconductor Master Selection Guide	CL-347	1978

Correspondence and inquiries about these units should be directed to:

Texas Instruments Incorporated
P.O. Box 225012, M/S 308
Dallas, Texas 75265
(214) 238-2011

The Modular Route: Microcomputer Modules

TM990 microcomputer modules are preassembled, pre-tested, ready-to-use combinations of 9900 Family components which are available to meet the needs of the microprocessor and microcomputer systems designers.

An overview of the TM990 microcomputer module product line, divided into the product series, is shown in *Figure 2-22.* A summary of key parameters are given in *Tables 2-1, 2-2* and *2-3.* The series ranges from microcomputers to expansion boards for memory and I/O, to software support in read-only memory (EPROM), to the accessories required to interconnect the modules. An I/O microterminal, TM990/301, is a low-cost terminal for system development included in the 300 series. A module of I/O and memory for software development, the TM990/302, is included in the 300 series and will be discussed further in the software support section.

The additional software products, TIBUG Monitor (TM990/401), Line-by-Line Assembler (TM990/402), and the POWER BASIC units (TM990/450, 451, and 452) will also be discussed in the software support section.

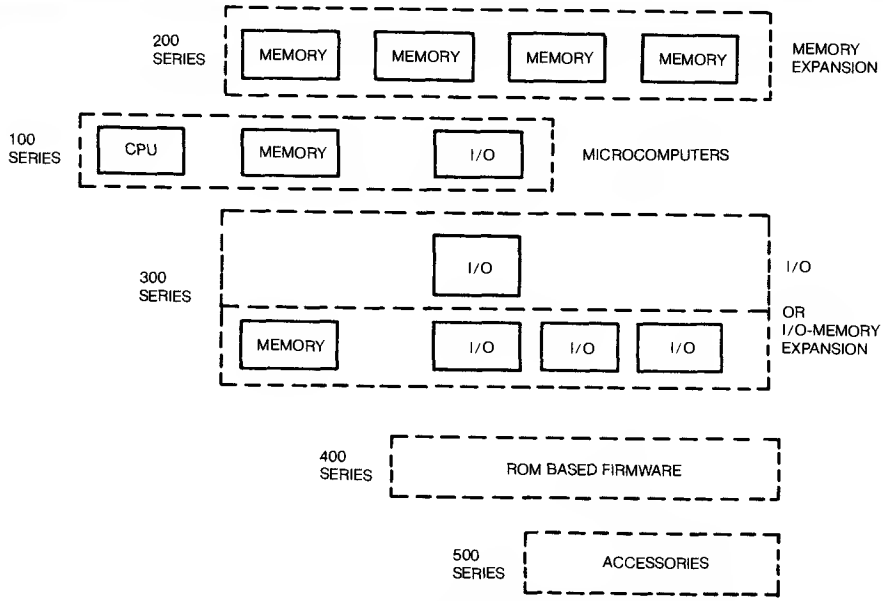


Figure 2-22. TM990 Microcomputer Module Series

The Application

Microcomputer modules are for the system designer who wants to:

1. Apply and evaluate a 9900 Family microcomputer without taking the time for all the engineering, planning, assembly and testing needed to design and assemble the equivalent microcomputer system.
2. Free himself from design details to concentrate on speeding an end product to market.
3. Expand memory of an existing 9900 Family system.
4. Assemble a low-cost software development system to edit, assemble, load and debug programs for PROMs.
5. Expand a university course with low-cost hands-on hardware.
6. Evaluate POWER BASIC programs and apply them to microcomputer systems.

A Special Product

A special product in the microcomputer module series is the TM990/189 University Board. It is designed primarily as a learning tool for the engineer, student or hobbyist. It aids in the instruction of microcomputer fundamentals, machine and assembly language programming and microcomputer interfacing. A tutorial text and a list of assembly procedures are included. More information is found in Chapter 8.

TM990/100 SERIES—MICROCOMPUTER MODULES

Product	CPU	EPROM (Bytes)	RAM (Bytes)	Serial I/O ports	Parallel I/O Lines	Prioritized vectored interrupts	Timers
TM990/100M-1	TMS9900	2K (2708) (Contains TIBUG Monitor) Expandable to 8K (2716)	512 Expandable to 1K	1 RS232C 1 TTY	16	16	2
TM990/100M-2	TMS9900	2K (2708) Blank Expandable to 8K (2716)	512 Expandable to 1K	1 RS232C or 1 Differential line driver	16	16	2
TM990/100M-3	TMS9900	8K (2716) Blank	1K	1 RS232C or 1 Differential line driver	16	16	2
TM990/101M-1	TMS9900	2K (2708) (Contains TIBUG Monitor) Expandable to 8K (2716)	2K Expandable to 4K	Port A RS232C or TTY Port B RS232C or Modem	16	16	3
TM990/101M-2	TMS9900	2K (2708) Blank Expandable to 8K (2716)	2K Expandable to 4K	Port A RS232C or Multidrop Port B RS232C or Modem	16	16	3
TM990/101M-3	TMS9900	8K (2716) Blank	4K	Port A RS232C or TTY Port B RS232C or Modem	16	16	3
TM990/180M-1	TMS9980	2K (2708) (Contains TIBUG Monitor) Expandable to 4K	512 Expandable to 1K	1 RS232C or 1 TTY	16	16	2
TM990/180M-3	TMS9980	4K (2708) Blank	1K	1 RS232C or 1 Differential line driver	16	16	2
TM990/189	TMS9980	4K Expandable to 6K	1K	1 RS232C or TTY	16	Special features: audio cassette and acoustical indicator	

TM 990/200 Series—MEMORY EXPANSION

PRODUCT	MEMORY TYPE	MEMORY SIZE (BYTES)
TM990/201-41	EPROM/Static RAM	BK EPROM, 4K RAM
TM990/201-42	EPROM/Static RAM	16K EPRDM, BK RAM
TM990/201-43	EPROM/Static RAM	32K EPROM, 16K RAM
TM990/206-41	Static RAM	8K RAM
TM990/206-42	Static RAM	16K RAM
TM990/203A*	Dynamic RAM	16/32/64K RAM

*Available second quarter 1979

Table 2-1. Key Parameters of TM990/100 and 200 Series

Table 2-2. Key Parameters of TM 990/ 300 Series Modules

TM 990/300 Series—I/O, I/O-MEMORY EXPANSION

I/O

PRODUCT	DESCRIPTION	FEATURES
TM990/301	Microterminal	Displays Data and Address

I/O, MEMORY (For Software Development)

PRODUCT	DESCRIPTION	FEATURES
TM990/302	Software development module used in conjunction with TM990/100M or TM990/101M for software development system	Dual audio cassette interface, 2K X 16 RAM, 4K X 16 EPROM, and EPROM programming

I/O EXPANSION

PRODUCT	PROGRAMMABLE I/O, INTERRUPT LINES	DEDICATED INTERRUPTS	TIMERS
TM990/310	48 lines programmable as inputs, outputs or up to 27 unlatched interrupts	six (3+, 3−) edge detect latches	3

Table 2-3. Key Parameters of TM990/ 400 and 500 Series

TM 990/400 SERIES—ROM BASED FIRMWARE

Product	Description	Medium	Utilized in
TM990/401-1	TIBUG Monitor	2708 (2)	TM990/100M-X TM990/101M-X
TM990/401-2	TIBUG Monitor	2708 (2)	TM990/180M-X
TM990/402-1	Line by Line Assembler	2708 (2)	TM990/100M-X TM990/101M-X
TM990/402-2	Line by Line Assembler	2708 (1)	TM990/180M-X
TM990/450	8K Byte Evaluation BASIC	2716 (4)	TM990/100M-X TM990/101M-X
TM990/451	12K Byte Development BASIC	2716 (6)	TM990/100M-X (Four 2716's on TM990/100, two 2716's on Memory Expansion Board (TM990/201-XX) or Software Development Board (TM990/302) TM990/101M-X (Four 2716's on TM990/101, two 2716's on Memory Expansion Board (TM990/201-XX) or Software Development Board (TM990/302)
TM990/452	4K Byte Enhancements to Development BASIC	2716 (2)	TM990/302

TM 990/500 SERIES—ACCESSORIES

Card cage

Product	No. of slots	Slot spacing	Outside dimensions
TM990/510	4	1"	5"H, 12.5"W, 8"D
TM990/520	8	.75"	8.25"H, 12.5"W, 8"D

Power supply

Product	Input Requirements		Output			
	Frequency	Voltage	+5V	+12V	−12V	+45V (EPROM programming voltage)
TM990/518	57-63 Hz	115/230 ± 10%	6.0A	0.9A	0.9A	0.1A
TM990/519	57-63 Hz	102/132V	2A	250 mA	180 mA	

Universal prototyping boards

Product	Description	Capacity
TM990/512	Unpopulated board for use with wirewrap or solder sockets.	16 pairs of 50 pin columns that accept .3 or .4 centers
TM990/513	Wire-wrap board populated with gold plated pins	16 pairs of 50 pin columns that accept .3 or .4 centers

Analog I/O expansion

To aid in providing the interface between analog and digital signals several companies are supplying products that complement the 9900 microcomputer components family. Key parameters of a number of these products are shown in *Table 2-4.*

A/D and D/A Converters

Product	Resolution	A/D Input Channels	Input Voltage Range	Input Current Range	Throughput Rates	Programmable Gain	D/A Output Channels	Voltage Output Range	Current Loop Outputs	+5V Requirements	Codes
RT1-1240-S	12 Bits	16SE, 8 Diff Expandable to 32SE, 16 Diff	+5V, +10V ±5V, ±10V	0-50 mA	40K Chan/sec	1,2,4,8	0	–		1.4 A	Binary, Offset Binary, Two's complement
1240-R	"	"	"	"	"	1-1000	0	–		"	"
1241-S	"	"	"	"	"	1,2,4,8	2	+5V, +10V ±2.5V, ±5V ±10V	4-20mA	"	"
1241-R	"	"	"	"	"	1-1000	2	"	4-20 mA	"	"
RT 1242	"	0	–	–	10 μ sec Setting	–	4	"	–	"	"
RT 1243	"	0	–	–	"	–	8	"	–	"	"

Analog Devices—Route 1 Industrial Park, P.O. Box 280, Norwood, Massachusetts 02062, (617) 329-4700

Product	Resolution	A/D Input Channels	Input Voltage Range	Input Current Range	Throughput Rates	Programmable Gain	D/A Output Channels	Voltage Output Range	Current Loop Outputs	+5V Requirements	Codes
ANDS 1001	12	16SE/8 Diff Expandable to 64SE/32 Diff	+5V, +10V ±5V, ±10V	0-50 mA	30K Chan/sec	–	0	–	–	700 mA	Binary, Offset Binary, Two's complement
ANOS 1002	15 Bit +Sign Bit	1-4	±20mv, ±40mv ±80mv thermo-couples	–	10 samples/ sec	–	0	–	–	–	"
ANDS 2001	12	0	–	–	10 μ sec Setting	–	1-4	+5V, +10V ±5V, ±10V	4-20 mA	–	"
ANOS 3001	12	16SE/8 Diff	+5V, +10V ±5V, ±10V	0-50 mA	30K Chan/sec	–	2	"	4-20 mA	–	"

Analogic—Audubon Road, Wakefield, Massachusetts 01880, (617) 246-0300

Table 2-4. Key Parameters for Analog conversion units for I/O Expansion

The Minicomputer Route

For large system applications in which the computer system is a small portion of total system costs, use of prepackaged OEM minicomputers as system components provides a number of advantages. A full complement of system and applications software is readily available for immediate use on the machine, including assemblers, linkers, editors, operating systems, high level languages, a variety of utility packages, many applications packages, and much, much more.

Texas Instruments Digital Systems Group manufactures two minicomputers which are compatible with the TMS9900. The first is the 990/4 minicomputer which uses the TMS9900 as its central processing unit. It utilizes the CRU for control of peripheral devices making this system directly compatible with the 9900 Family. The second minicomputer implements the CPU in TTL, maintaining upward compatibility with the 9900 Family. This unit, the 990/10 uses a DMA peripheral device interface called TILINE™ for control of high speed peripherals such as magnetic tape units and moving head disk drives, and provides extended addressing capability.

A complete discussion of the use of these systems as OEM system components is beyond the scope of this book, but further information may be obtained by writing:

Texas Instruments Incorporated
Digital Systems Group
P.O. Box 1444
Houston, Texas 77001
Attention: Market Communications M/S 784

or contact your local TI sales office or distributor system center listed in the appendix.

The 990 Computer Family Systems Handbook, the 1978 Catalog of the 990 Computer Family, and the 990 Computer Family Price List provide detailed information on the use of 990 computers as OEM system components.

A Selection Process

Criteria for selecting a microprocessor, microcomputer, microcomputer component peripheral, or a minicomputer for a system application are listed in *Figure 2-23*. System performance, cost, reliability, and delivery may also depend on the vendor that designs and supports the products used.

MPU ARCHITECTURE
- Word Size
- Number of Instructions
- Address Bus Length
- Data Bus Length
- I/O Bus Length
- Clock Rate
- Benchmark Performance (Selected Functions)
- Arithmetic Capability
  - Multiply
  - Divide

I/O CAPABILITY AND PERIPHERAL CIRCUITS
- Parallel I/O
  - How Many Bits
  - Data Rate
  - Programmability
  - Drive Required
- Serial I/O
  - Asynchronous
  - Synchronous
  - Baud Rate
  - EIA
  - Current Loop
- Timers and Event Counters
  - Interval
  - Max Count
- Interrupts
  - Number
  - Masking
- DMA
  - Channels
  - Chaining Required
- Other Interfaces
  - Floppy Disk
  - Analog
  - Keyboard
  - CRT
  - Tape

MPU (other specifications)
- Package
- Temperature Range
- Supply Voltages
- Power Consumption
- Special Reliability
- Unit Costs (Selected Volumes)

System Environmental
- Supply Voltages
- Temperature Range
- Power Consumption
- Special Reliability
- Special Size

Support
- Technical Documentation
- Hardware Development Support
  - Emulators
  - Testers
  - Evaluation Modules

Software
- Assemblers
- Text Editors
- Simulators
- Utilities
- Application Libraries
- High-level language

Software Development
- Systems
  - Cross-Support
  - Dedicated

Figure 2-23. Selection Criteria for Microprocessor, Microcomputer Systems

Vendor Selection

One way of evaluating a vendor is to make a list of items similar to the selection criteria for system components. Some of the same items from this list, especially in the support area, can be included. Additional items for consideration are shown in *Figure 2-24.*

▶2

DOCUMENTATION	CREDIBILITY
Product	Reputation
Support Systems	Investment
Applications	Financial Status
MANUFACTURING CAPABILITY	CUSTOMER SUPPORT
Facilities	Application Engineers
Product Levels	Distribution
Backlog	Hot Lines

Figure 2-24. *Vendor Criteria*

Setting weights for each item and summing these for individual vendors allows a direct comparison. The total number accumulated for each vendor establishes a vendor rating.

9900 FAMILY SOFTWARE AND DEVELOPMENT SYSTEMS

Importance of Software

As described in Chapter 1 *(Figure 1-9),* the term software is used to describe the programs and documented ideas which allow small amounts of general purpose hardware (microprocessors, memory, peripherals) to replace large amounts of special purpose hardware. The costs for software are becoming a much larger percentage of the total system development cost. These costs are primarily incurred prior to production of a system. For large volume systems the share of these one-time costs attributed to each unit is small since the total software costs are divided by a very large number. Correspondingly, when the volume of units produced is low, the software cost per unit will be quite high. This factor, coupled with a lack of familiarity, has led many users to underestimate software development costs.

Since software now commands 80% of the design effort of complete systems, and since many software tasks are common to the industry, the level of software support from a vendor can have tremendous impact on total system design cost. Perhaps more importantly, availability of a wide variety of system and application software packages can drastically shorten design time and speed the product to market.

Software Development Systems

Development of software requires equipment — program or software development systems. As a system designer makes a decision to use a microprocessor or microcomputer, all design avenues seem to focus on software development. Questions naturally arise, "How can I do software 'breadboarding' and program testing?", "How can I arrive at a final program and be assured that it is correct?" and "Can it be done economically"?

Figure 2-25 illustrates cost versus capability for each of the program development systems that support the 9900 Family. Lower cost systems tend to have lower capability. The choice of a program development system depends on many factors. Some examples are: (1) *Capital Status* — capital availability determines whether a firm can consider the sophisticated emulator systems which boost designer productivity. (2) *Equipment on Hand* — availability of a terminal, line printer, or EPROM programmer or other useable equipment would likely reduce the required level of investment. (3) *Equipment Longevity* — How long the equipment will be used may allow division of the cost of the equipment over several projects. (4) *System Complexity* — Highly complex applications often require the best development tools possible; therefore, the most sophisticated system is required or the job can't be done. (5) *Production quantities* — High volume applications can more easily bear the cost of top-of-the-line development equipment; the corresponding increase in productivity made possible by this equipment, increases design efficiency. (6) *In-House Computer Capability* — Availability of in-house computer support makes development via timeshare cross-support an efficient alternative.

A brief description of the program development systems follows:

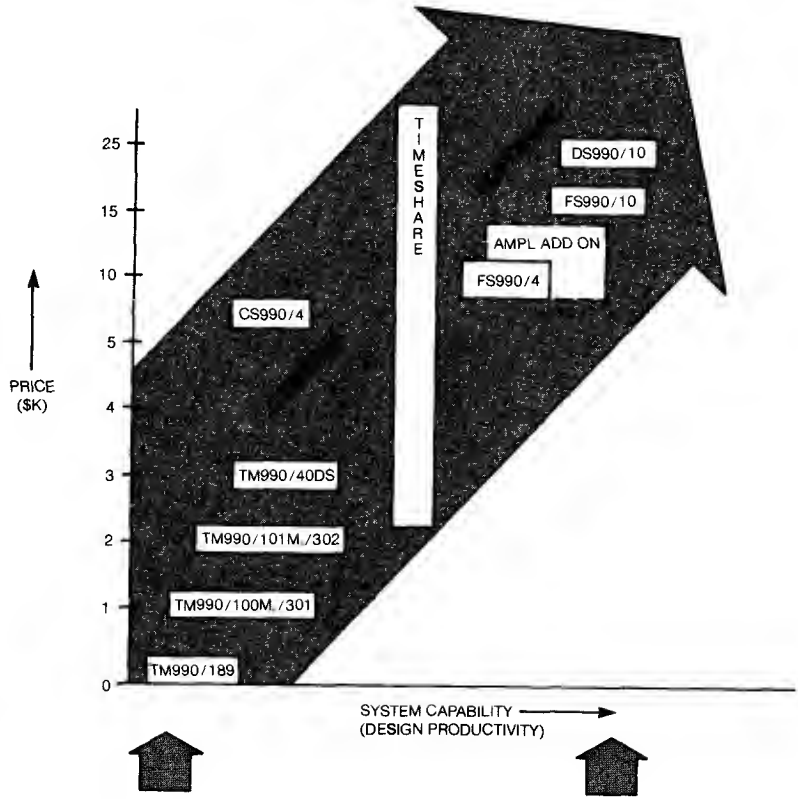
Program Development Systems

1. TM990/189— University Board. (Price: less than $250) This board provides an inexpensive means of evaluating the 9900 Family and learning about microprocessors in general. It comes with a debug monitor and assembler. Key features include full alphanumeric keyboard; display via 10 seven-segment digits; 16-bit parallel, RS232, TTY, and audio cassette interface; and a tutorial text and hardware reference manual. (See Chapter 8.)

2. TM990/100M— Microcomputer with line-by-line assembler
   TM990/301— Microterminal for programming (combined price: less than $500)

(Figure 2-26). These components are described in detail in Chapters 3 and 8. Basic program benchmarks may be written and tested with the 9900 microcomputer. A terminal such as a 743 KSR may be connected to the board and additional development software used. (This technique is described in Chapter 9.)

▶2



1. Initial Evaluation
2. One Time User
3. Small Firm
4. Minimum Capital Investment

1. Detailed System Analysis
2. Long Term Use
3. Large Experienced Firm
4. Large Complex System

Figure 2-25. Cost vs. System Capability for 9900 Family Program Development Systems

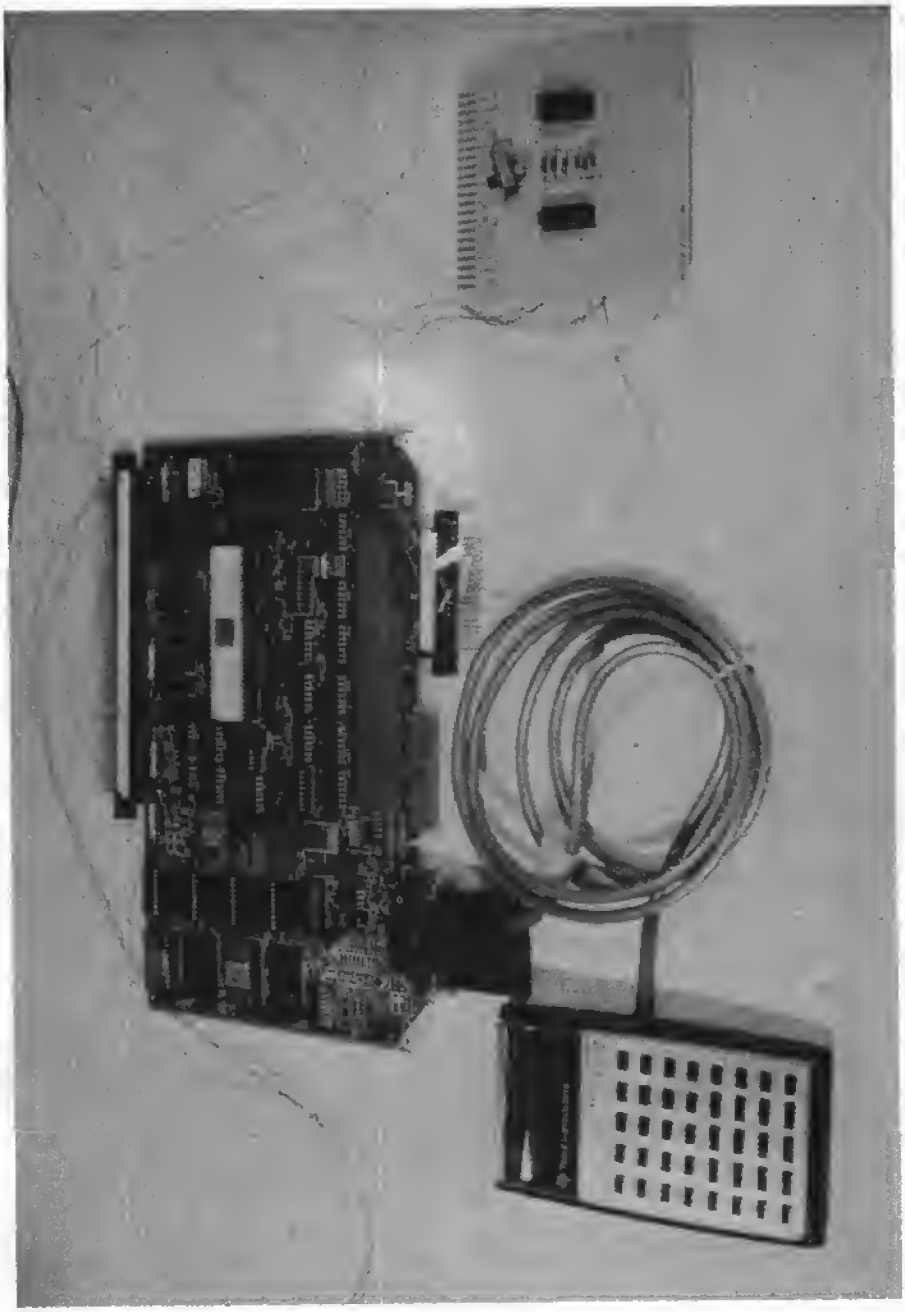


Figure 2-26. TM990/100M Microcomputer with TM990/301 Microterminal

3. TM990/302— Microcomputer board (price: less than $600)

Software is on the board in the form of EPROM devices for editor, assembler, linker, debugger, and EPROM programmer functions. A terminal and one or two cassette recorders are needed to complete a very powerful, yet very low cost program development system. The /302 is a companion to (or extension of) the TM990/100M or /101M board. *(Figure 2-27).*

▶2

4. TM990/40DS— TMS9940 development system (price: less than $2800) containing an EPROM programmer for the TMS9940E, Debug Monitor, Assembler and Trial In-System Emulation; the /40DS provides development capability and emulation of most of the TMS9940's operations *(Figure 2-28).*

5. CS990/4— 990/4 minicomputer with a 733 ASR dual cassette terminal (price: less than $6000) *(Figure 2-29).*

Program development software is available on cassettes to perform every task outlined previously.

6. FS990/4— 990/4 minicomputer, terminal, and dual floppy disk storage unit (price: less than $12,000) *(Figure 2-30).*

Complete program development system with peripheral add-on capacity.

7. FS990/AMPL— Same as FS990 but with AMPL hardware and software added (price: less than $20,000) *(Figure 2-31).*

The primary advantage of the AMPL system is the complete hardware debugging capability via the AMPL software and 9900 emulator and trace functions.

8. FS990/10— 990/10 minicomputer *(Figure 2-32),* terminal, and dual floppy disk storage unit (base system starts at $15,000).

Complete program development system which can be upgraded to include moving-head disk mass storage. AMPL is available as an option.

9. DS990/10— 990/10 minicomputer *(Figure 2-33),* terminal, moving head-disk mass storage with complete multi-user system software (base system starts at $25,000). Supports Macro-Assembler, FORTRAN, BASIC, PASCAL and COBOL.

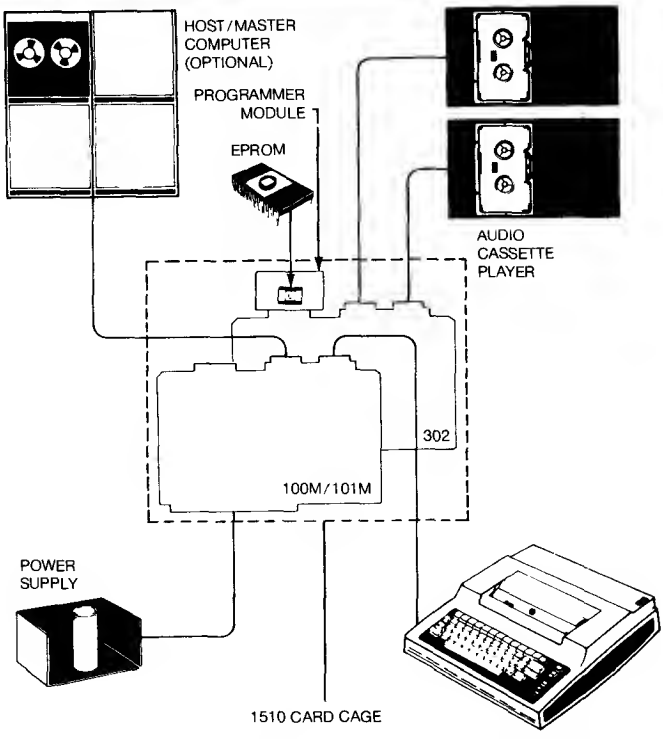


Figure 2-27. TM990/302 Program Development System

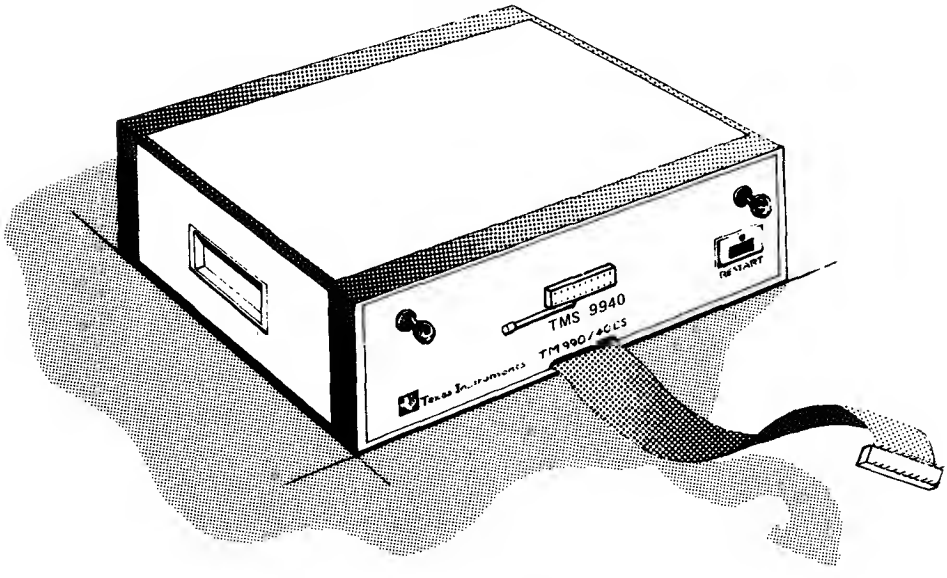


Figure 2-28. TM990/40DS cables and card chassis

▶2

Figure 2-29. CS 990/4 Software System

*Figure 2-30. FS990 Software Development System
(with optional printer)*

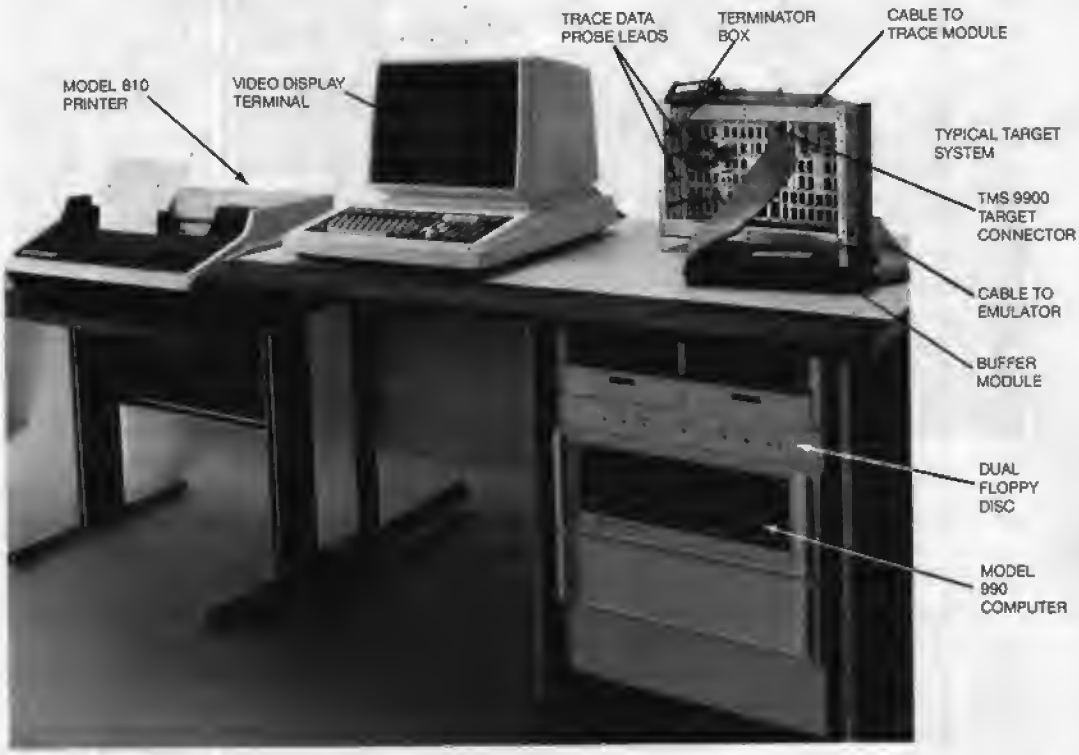


Figure 2-31. Typical AMPL Microprocessor Prototyping Laboratory

▸2

Figure 2-32. FS990/10 Minicomputer System

Figure 2-33. DS990/10 Minicomputer System

Which Program Development System to Use

The choice of a program development system requires evaluation of an application's specific requirements. The lowest cost system (TM 990/100M board and /301 microterminal) will allow a very basic level of programming, and is suitable for writing short routines to test algorithms or evaluate execution speed. Since labels are not allowed and there is no editing program to help add or delete program steps, programming is relatively difficult.

By adding the TM990/302 Software Development Module (with the TM990/100M or 101M) programming becomes much easier. An editor program helps you modify the program steps, the assembler allows labels, and the other elements—debug, EPROM programmer, relocating loader, and I/O handlers—add substantial programming flexibility. A programmer might well evaluate this system as being an order of magnitude better than the /100M board alone. It is best suited for one designer working on a single prototype.

But there are limitations to the /302. The system depends on cassette recorders for storage of development software and user programs. And cassettes are slow. The number of times per day that a programmer can make a change in his program, process it through the system, and test the results is generally in the range of three to five.

The number of program change cycles per day can be increased by purchasing a CS990 system. This digital cassette based software development system, being more versatile, can increase daily program iterations to about ten. Two or three programmers can use a single system comfortably.

The FS990/4 system uses floppy disk storage to further improve flexibility. Daily program iterations can be over 20. Because program turnaround is fast, a single FS990 system is often used by several programmers.

By adding AMPL hardware and software the FS990 system can be upgraded to an AMPL prototyping system. Hardware testing may be performed under program control.

The chart shown in *Figure 2-34* shows the different levels of sophistication of program development systems that can be used with each 9900 CPU.

SELECTED PROCESSOR	TMS9900/ SBP9900A	TMS9980A/ TMS9981	TMS9985	TMS9940
MINIMUM SYSTEM	TM990/100M TM990/101M TM990/302	TM990/189 TM990/180	TM990/185	TM990/140
MEDIUM SYSTEM	CS990/4	CS990/4	CS990/4	TM990/40DS
LARGE SYSTEM	FS990/4 AMPL FS990/10	FS990/4 AMPL FS990/10	FS990/4 AMPL FS990/10	FS990/4 AMPL FS990/10
MAXIMUM CAPABILITY	DS990/10	DS990/10	DS990/10	DS990/10
TIMESHARE				
TMSW101MT	X	X	X	X
GE, NCSS, Tymeshare	X	X	X	X

Figure 2-34. Program Development Systems for Each 9900 Family CPU

Timeshare

Timeshare users approach software development in one of two ways. Either they purchase and install the TMSW101MT cross-support package on an in-house computer, or they lease access to a similar package on a commercial timeshare system such as GE TERMINET, NCSS, and TYMSHARE. Both approaches provide a 9900 cross-assembler compatible with the FS990 prototype development system. Both also provide a simulator and ROM utility. In-house users often interface the ROM utility directly to EPROM programmers. Otherwise several printout formats are available to match standard ROM and PROM order techniques.

The timeshare approach provides high-level development capability at minimum initial cost. It does, however, incur large operating costs, especially when using commercial systems.

Support Software and Firmware

The program development systems and the 9900 Family of components are supported by a full line of software. The chart shown in *Figure 2-35* summarizes the capability of the program development system software.

PDS / PROGRAM STEP		CROSS-SUPPORT TMSW101MT	TM990/302	TM990/40DS	PX990	TX990	FS990 AMPL	DX990 W/AMPL
EDIT		X	X		X	X	X	X
ASSEMBLE		X	X	X	X	X	X	MACRO
LINK		X			X	X	X	X
LOAD		CREATES LOAD MODULE	X	X	X	X	X	X
DEBUG		X	X	X	X	X	X	X
EMULATOR				X			X	X
LOGIC TRACE							X	X
SIMULATOR		X						
READ-ONLY MEMORY PROGRAM-MING	ROM	X				X	X	X
	PROM	X	X	X	X	X	X	X

Figure 2-35. 9900 Family Software Development System Capabilities

Additional software and firmware are as follows:

TM990/401 – TIBUG Monitor

The TMS990/401 TIBUG Monitor is a comprehensive, interactive debug monitor in EPROM included in the basic price of the TM990 CPU modules. TIBUG includes 13 user commands plus six user accessible utilities and operates with 110, 300, 1200 and 2400 baud terminals. The basic TIBUG functions include:

1. Inspect/change the following: CRU, memory locations, program counter, workspace pointer, status register, workspace registers.
2. Execute user programs under breakpoint in single or multiple steps.

TM990/402 – Line-by-Line Assembler (LBLA)

TM990/402 is a line-by-line assembler which is supplied pre-programmed in EPROM for immediate system use. By allowing the entry of instructions in mnemonic form and performing simple address resolution calculations with a displacement range of +254 to −256 bytes, the assembler is an extremely powerful tool for assembly language input of short programs or easy patching of long programs.

POWER BASIC High-Level Language

POWER BASIC, an easy-to-use extension of the original BASIC language, is highly suitable for the majority of industrial control applications. It greatly simplifies the solution of complex system problems and eliminates unnecessary design details.

POWER BASIC can be used for a general system implementation language as well as for information processing. It is also versatile enough to solve problems in real-time control of events while improving programmer efficiency in implementing complex algorithms.

The performance of POWER BASIC is outstanding — 2 to 3 times faster than any existing 8-bit microcomputer-oriented BASIC. In effect, you get minicomputer performance at microcomputer cost.

▸2

Other advantages of POWER BASIC include:

- Full string processing capability
- Multidimensional arrays
- 13-digit arithmetic accuracy
- Automatic minimum memory configuration

POWER BASIC language interpreters are available in economical yet versatile packages shown in *Table 2-5.*

Table 2-5. POWER BASIC Firmware

PART NO.	MEDIA	NAME	DESCRIPTION
TM990/450	EPROM device kit	Evaluation POWER BASIC	Reduced memory version (8K byte) designed to offer evaluation tools for exploring POWER BASIC applications. ROM kit executes standalone on TM990/100M, 101M modules.
TM990/101M-10	*TM990/101M		
TM990/451	EPROM device kit	Development POWER BASIC	Expanded memory version (12K byte) providing capability for design, development, and debug of POWER BASIC programs. Executes on TM990/201 or 302 module interfaced with TM990/100M, 101M CPU modules.
TM990/452	EPROM device kit	Enhancement of Development POWER BASIC Software Package	Provides EPROM programming, dual audio cassette handling, and I/O utilities for TMS990/302.
TMSW201F	FS990 diskette	Configurable POWER BASIC	Fully expanded version including complete diskette file support and a configurator program which reduces the size of POWER BASIC programs for execution.

*Contained in TM990/101M Module

PASCAL High-Level Language

TIPMX Executive Components Library in PASCAL

TIPMX is a configuration of software processes that provides executive functions such as multitask priority scheduling, interrupt servicing, and inter-process communication. It relieves the programmer of the necessity to develop these processes. TIPMX also supports, but is not limited by, PASCAL data structures and program structures.

A tailored TIPMX is configured by selecting desired processes from a library of system and run-time support modules. These processes are link-edited to form a supervisory nucleus which is loaded into EPROM memory to enhance its speed, efficiency and reliability.

PASCAL, FORTRAN or Assembly Language processes then execute under the auspices of this tailored TIPMX executive.

CHAPTER 3

A First Encounter: Getting Your Hands on a 9900

PURPOSE/ WHERE TO BEGIN

PURPOSE

Remember the common saying, "What you've always wanted to know about subject X, but were always afraid to try." The same applies, and probably especially so, to persons who have contact with the world of digital electronics; who have heard about computers and minicomputers and even operated them; who have seen and experienced the advances made in the functional capabilities and low cost of digital integrated circuits by owning and operating handheld calculators; who have worked around and even built electronic equipment; who have heard about microprocessors and their amazing capabilities — but have not tried them.

▶3 If you are one of these people, this chapter is for you, for in it we want to help you try out a microprocessor, work it together, operate it, have success with it. In this way we hope to demonstrate that microprocessor systems are not that difficult to use. That, even though they require an understanding of a new side of electronic system design — "software" — if a base of understanding is established, and if an engineering approach is followed, there is no need to fear getting involved.

So that's the purpose of this first encounter — to get your hands on a 9900 microprocessor system and operate it.

WHERE TO BEGIN

It would be very easy to be satisfied with a paper example for a first encounter, however, it has been demonstrated that a great deal more is learned by actually having the physical equipment and doing something with it. Therefore, this first encounter example requires that specific pieces of equipment be purchased.

However, the purchase is not to be in vain. The first encounter has been chosen so that is may be followed with more extensive applications described in Chapter 9. Applications that will help to bring understanding of the 9900 microprocessor system to the point that actual control applications, akin to automating an assembly line, can be implemented. Outputting control of ac and dc voltage for motors or solenoids and producing controlled logic level signals are examples. In this way, useful outcomes are being accomplished, the equipment is being expanded, and problem solutions are demonstrated. At all times, of course, the base foundation of knowledge about microprocessor systems is growing.

To get underway then, purchase the following items from your industrial electronics distributor that handles Texas Instruments Incorporated products.

Quantity	Part #	Description
1	TM990/100M-1 (Assembly No. 999211-0001) (see *Figure 3-1*)	TMS9900 microcomputer module with TIBUG monitor in two TMS 2708 EPROM's and EIA or TTY serial I/O jumpers option.
1	TM990/301 (see *Figure 3-2*)	Microterminal
1	TIH431121-50 or Amphenol 225-804-50 or Viking 3VH50/9N05 or Elco 00-6064-100-061-001	100 pin, 0.125″ c-c, wire-wrap PCB edge connector (or equivalent solder terminal unit)
1	TIH421121-20 or Viking 3VH20/1JND5	40 pin, 0.1″ c-c, wire-wrap PCB edge connector (or equivalent solder terminal unit)

In addition, some small electronic parts to interconnect the light emitting diode displays that will be used will be needed. These are listed later on so you may want to continue to read further before purchasing the module and microterminal so that all necessary parts can be obtained at the same time.

WHAT YOU HAVE

In *Figure 3-3* is shown a generalized computer system, it has a CPU (central processing unit) which contains an arithmetic and logic unit (ALU), all the control and timing circuits, and interface circuits to the other major parts. It has a memory unit. It has some peripheral units for inputting data such as tape machines, disk memories, terminals and keyboards. It has output units such as printers, CRT screens, tape machines, disk memories.

The TM990/100M-1 microcomputer shown in *Figure 3-1* is a miniature version of this computer system as shown in *Figure 3-4.* It has a CPU centered around the TMS9900 microprocessor, a memory unit — in this case a random access memory (RAM) and a read only memory (ROM). It does not have the input/output units indicated in *Figure 3-3* but it does have circuitry (TMS9901, 9902) for interface to such units. The TMS9901 will handle parallel input/output data and single bit addressed data as will be shown in this first encounter. The TMS9902 handles serial input/output data interface either through a TTY interface. A more complete interconnection of the components of the microcomputer is shown in the block diagram of *Figure 3-5.* The physical position of these units on the board is identified in *Figure 3-1.*

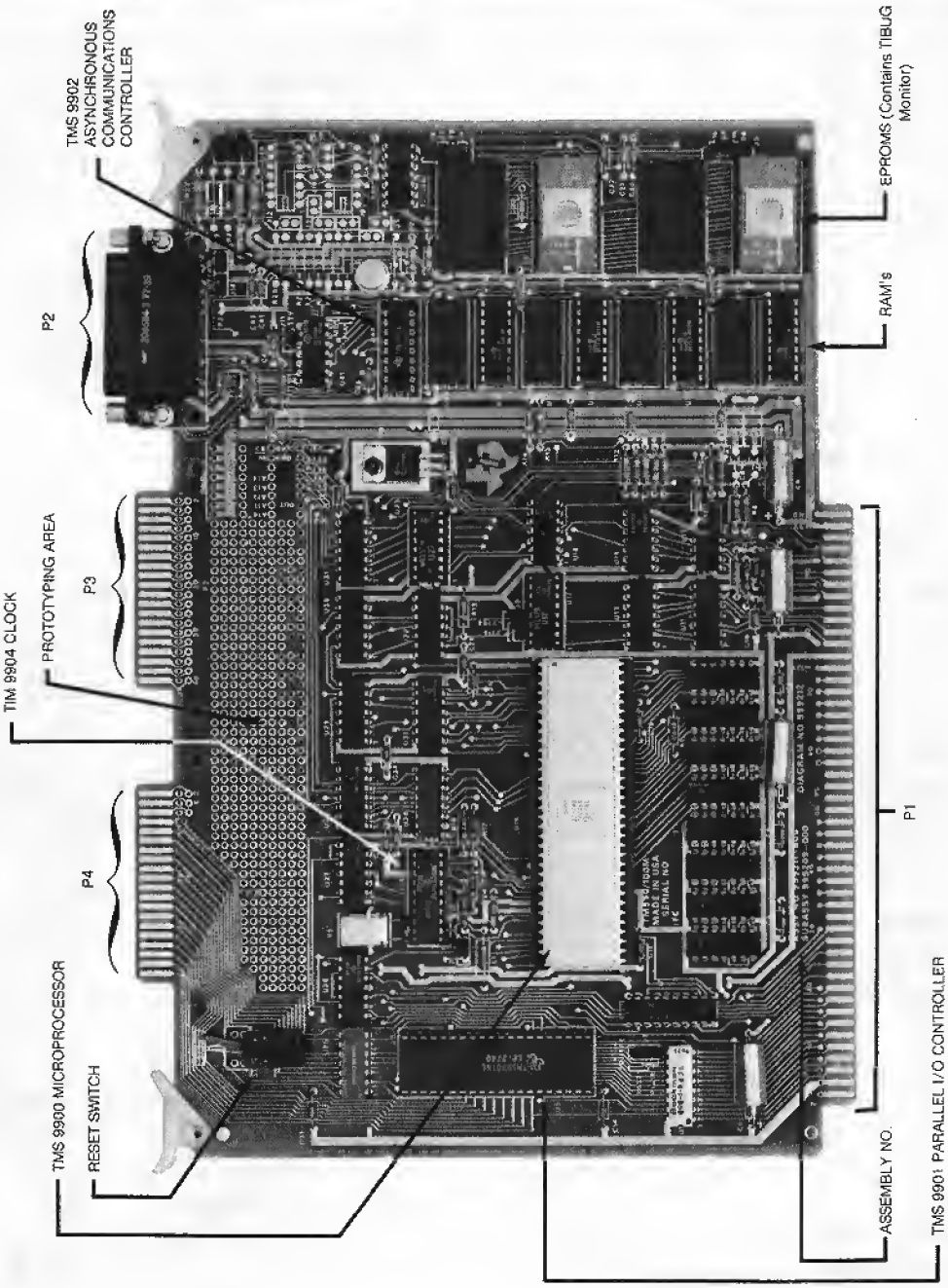


Figure 3-1. TM 990/100M-1 Microcomputer

Figure 3-2. TM 990/301 Microterminal

Just think, a complete microcomputer with: 1) 256 16-bit words of random access memory to hold program steps and program data, expandable to 512 words; 2) 1024 16-bit words of read only memory which contains pre-programmed routines (TIBUG Monitor) that provides the steps necessary for the TM990/100M-1 microcomputer to accept input instructions and data and to provide output data. This ROM capability can be expanded to 4096 words to provide program flexibility; 3) input/output interface that can handle 16 parallel lines expandable to 4096 and an interface for serial characters of 5-8 bits at a programmable data rate; 4) an input terminal to input the sequence of steps to solve a problem — the program.

GETTING IT TOGETHER

Of course, in order to operate the microprocessor system, it must be put together. It must be interconnected.

What function will it perform? The first encounter application is shown in *Figure 3-6.* The microcomputer will be used to provide basic logic level outputs to turn on and off, in sequence, light emitting diode segments of a 7 segment numeric display element, the TIL303. This will demonstrate the "software" techniques used to provide dc logic levels at the I/O interface which through proper drivers can later be used to control solenoids, motors, relays, lights, etc.

In the first encounter application, the microterminal shown in *Figure 3-2* will be used to input the instructions and data required to perform the function.

Recall that a light emitting diode (LED) is made of semiconductor material and emits light when a current is passed through it in the correct direction. Each segment of the 7-segment display is a separate LED. Four segments of the display will turn on in the sequence f, b, e, c at a slow or a fast rate depending on the position of a switch, as shown in *Figure 3-6.* Each segment will first be turned on, then a short delay, then off, then a short delay. The sequence is continued with the next segment; proceeding around through 4 segments and then starting over again. The rate is varied by changing the delay in the sequence. The switch position controls the delay.

A 7-segment display is used because of its ready availability and its dual-in-line package. Only 4 of the segments will be programmed into the sequence although driver capability
▶3 will be provided for 6 segments. This allows flexibility for the person doing the first encounter to experiment on their own to include the remaining 2 segments. A next step would be to provide an additional driver. In this way all 7 segments of the display can be included.

Here's what's required to provide the segment display. *Figure 3-7* shows the integrated circuit driver package for the LED segments, the SN74H05N. The physical package and a schematic are shown. It contains 6 open collector inverters, each capable of "sinking" 20 ma. A 14- or 16-pin dual-in-line socket is required. A wire-wrap one is shown. However, it could be a solder terminal unit just as well.

Figure 3-8 shows the 7-segment display physical package and schematic and a 14- or 16-pin DIP socket for interconnection. 100 ohm resistors for limiting current through the LEDs are also required.

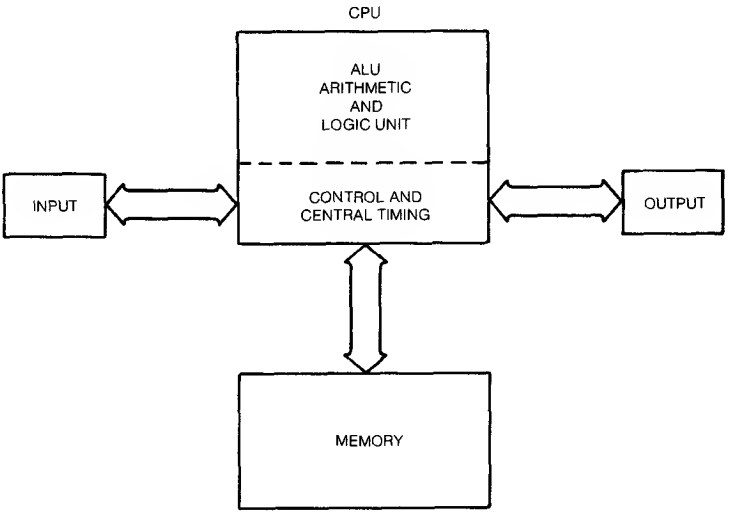


Figure 3-3. *Generalized Computer*

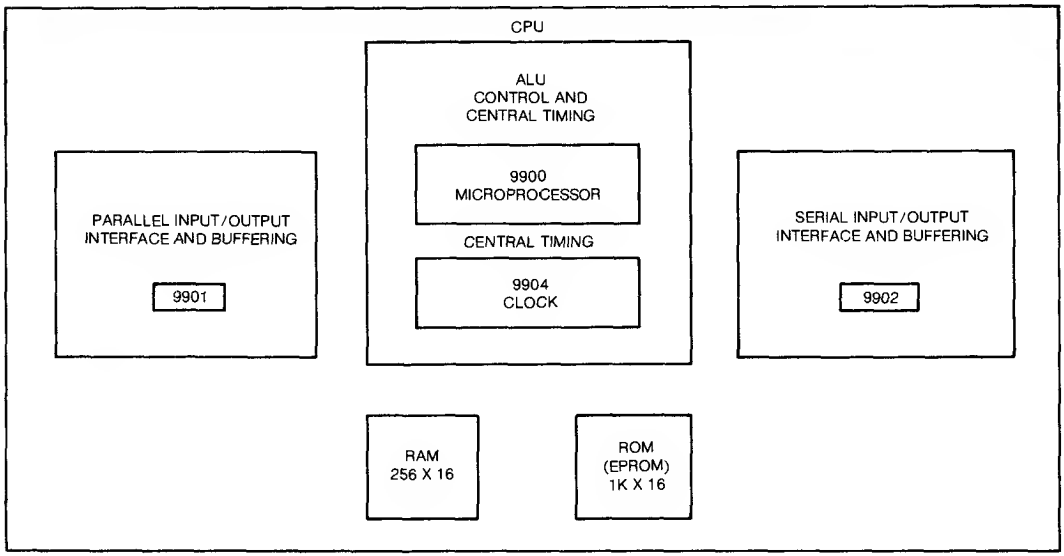


Figure 3-4. Miniature Computer System on TM 990/100M-1 Module

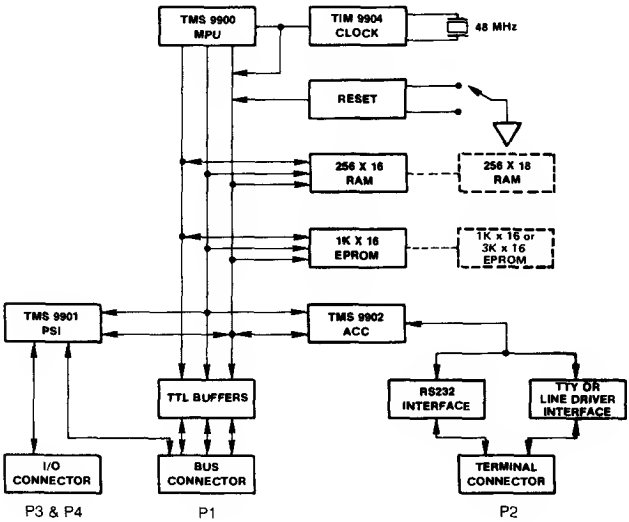


Figure 3-5. TM 990/100M-1 Block Diagram

▶3

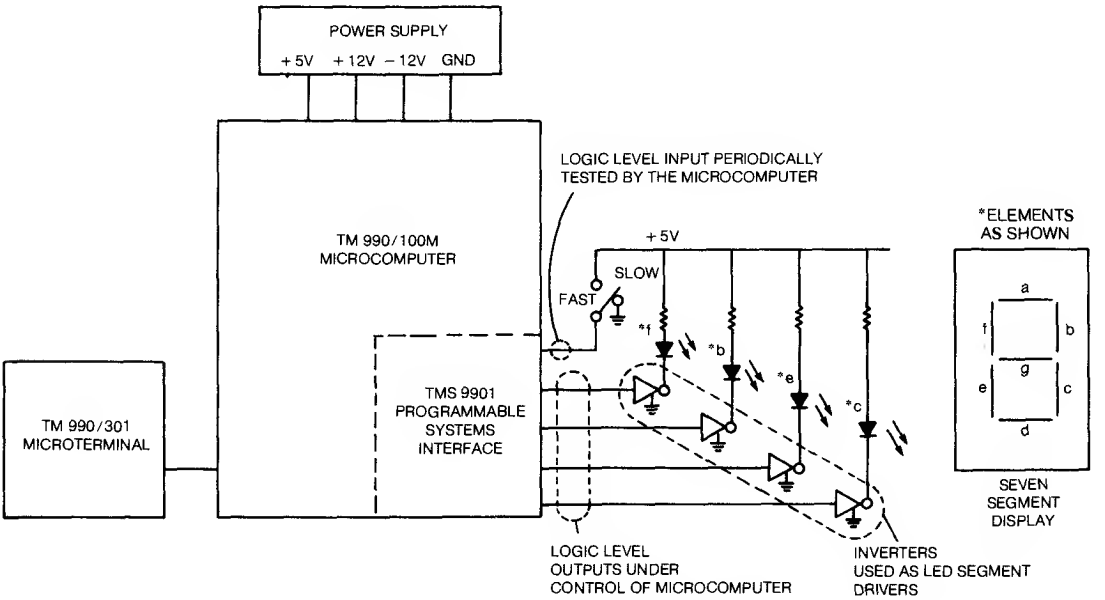
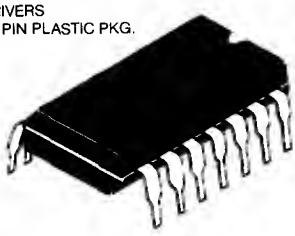
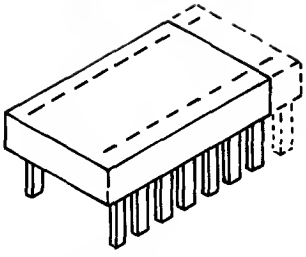


Figure 3-6. The First Encounter Task

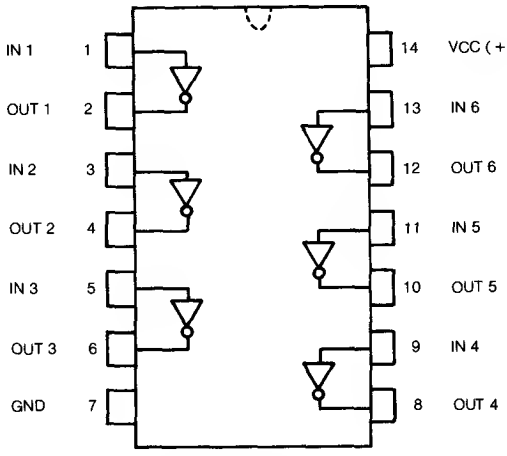
All of the components of *Figure 3-7* and *3-8* are wired together on a separate printed circuit board as shown in *Figure 3-9.* The Radio Shack #276-152 board provides individual plated surfaces around holes to make it easy to anchor components and to interconnect all components with wire-wrap. J4, the 40 pin wire-wrap PCB edge connector accepts the edge connections of P4 on the TM990/100M-1 board shown in *Figure 3-1.* After wiring this connector, put a piece of tape across the top of this connector so that it is correctly oriented before the board is plugged in; or the same can be done here as for P_1 discussed a little later. Note also on *Figure 3-1* that there is an area on the 990/100M-1 board for prototyping. The components of *Figure 3-9* may be wired in this area rather than using a separate printed circuit board. Using a separate board allows this area to be used for more permanent components for a specific dedicated application of the 990/100M module.

A. SN74H05N
SIX INVERTER
DRIVERS
14 PIN PLASTIC PKG.





B. 14—16 PIN DIP SOCKET
(WIRE-WRAP OR SOLDER TERMINALS)



C. SCHEMATIC OF SN74H05N
(TOP VIEW)

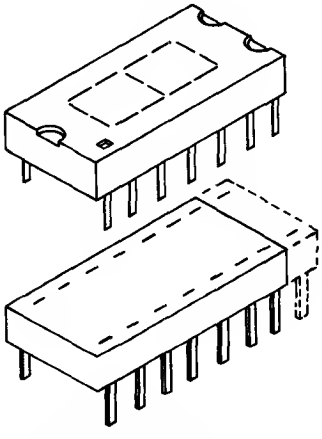
COMPONENT PARTS

1 — SN 74H05N HEX DRIVER
(EACH DRIVER CAPABLE OF SINKING 20 MA.)

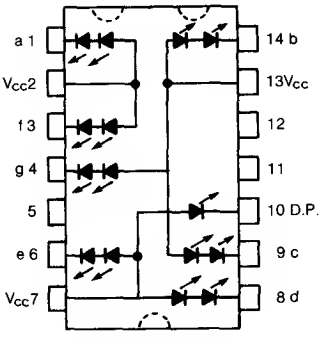
1 — 14 OR 16 PIN DIP SOCKET
(RADIO SHACK #276-1993, 94)
(TI # 811604 M&C — 16 PIN WIRE-WRAP)

Figure 3-7. LED Driver Parts

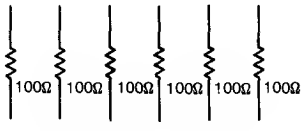
TOP VIEW
A. TIL303 7 SEGMENT NUMERICAL DISPLAY



B. 14 OR 16 PIN DIP SOCKET
(WIRE-WRAP OR SOLDER TERMINALS)



C. SCHEMATIC OF TIL303



D. 100 OHM RESISTORS 1/4 W

COMPONENT PARTS

1 — 7 SEGMENT DISPLAY TIL303

1 — 14 OR 16 PIN DIP PACKAGE
(C-811604 M&C — 16 PIN WIRE WRAP)
(RADIO SHACK — 276 — 1993, 94)

6 — 100 OHM RESISTORS, 1/4 W

Figure 3-8. Segment Display Parts

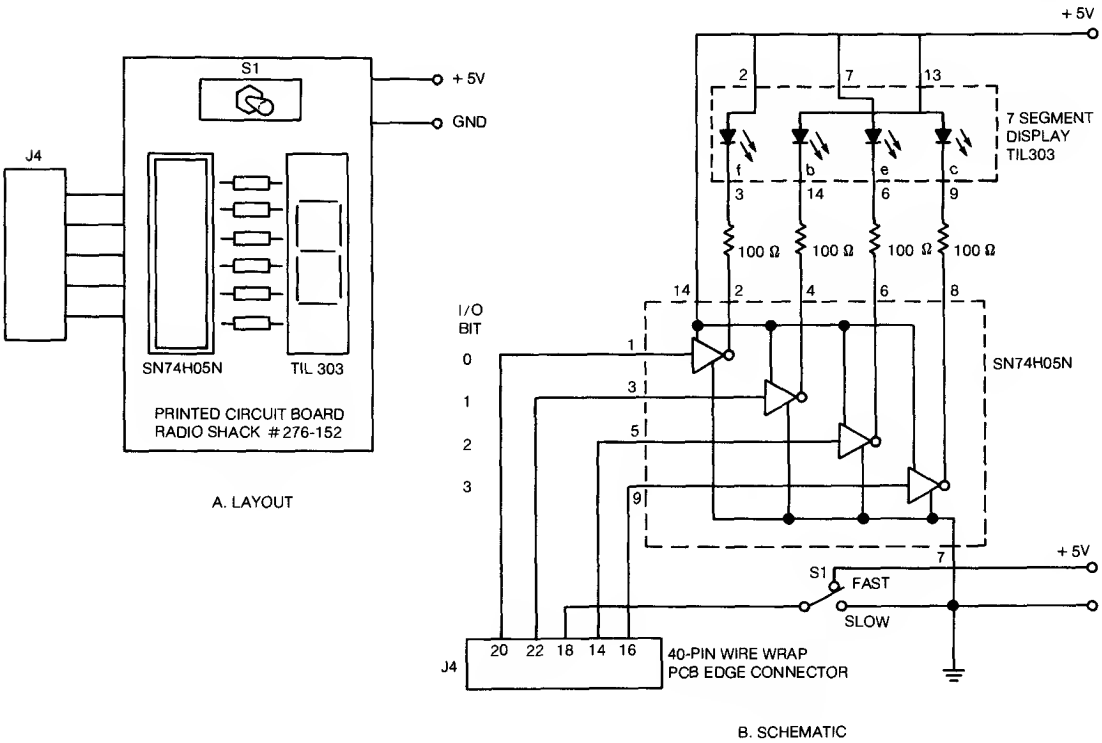


Figure 3-9. The Output Board

Following is a complete list of the parts, tools and supplies required. This is the list that was referred to earlier. Check carefully that all necessary parts are purchased.

PARTS LIST

A. *Microcomputer*

1 — TM990/100M-1	TMS9900 Microcomputer module with TIBUG monitor in two TMS 2708 EPROM's and EIA or TTY serial I/O jumper option.

B. *Terminal*

1 — TM990/301	Microterminal

C. *Output*

1 — Hex LED Driver	SN74H05N
1 — 7 Segment Display	TIL303
2 — 14 or 16 Pin Dip Sockets	TI wire-wrap; 16 Pin — C-811604 M&C; Radio Shack wire-wrap; 14 Pin 276-1993; 16 Pin 276-1994
6 — 100 ohm Resistors, ¼ W	
1 — Switch, Toggle or Slide, SPST or DPST	
1 — J4, 40 pin, 0.1″ c-c, wire-wrap	TIH421121-20
PCB Edge Connector (or equiv. solder terminal unit)	Viking 3VH20/1JND5
1 — Printed Circuit Board	Radio Shack #276-152

D. *Bus Connector* (Use for Power in First Encounter)

1 — J1, 100-pin, 0.125″ c-c, wire-wrap	TIH431121-50
PCB Edge Connector (or equiv. solder terminal unit)	AMPHENOL 225-804-50 Viking 3VH50/9N05 Elco 00-6064-100-061-001

E. *Power Supplies* — Regulated

Voltage	*Regulation*	*Current*
+5V	±3%	1.3A
+12V	±3%	0.2A
−12V	±3%	0.1A

F. *Tools*

Wire-wrap connector tool	Soldering Iron
Wire-wrap disconnecting tool	
Wire stripper (30 G)	Long-nose pliers
	Diagonal cutter
	VOM, DVM, DMM

G. *General Supplies*

Wire (30 G Kynar)
Solder
Plugs and jacks for power supply connections

Note the power supplies required, the voltages, currents, and regulation. Assure that there is a common ground between all units.

(Electronic shops or laboratories might have available individual LEDs, therefore, *Figure 3-10* is provided in case this alternate method of display is chosen. The necessary drivers and resistors are identified. The necessary substitutions can be made on *Figure 3-9.*)

After wiring the output board, what remains is to supply power to the board. This is accomplished through P1 on the 990/100M-1 board. *Figure 3-11* shows how the edge connector is wired to supply power. Be careful to use the correct pins as numbered on P1 on the board; *these pin numbers may not correspond to the number on the particular edge connector used.* Label the top side of the edge connector "TOP" and the bottom "TURN OVER." This will prevent incorrect connection of power to board. Wire the connector pins so that the top and bottom connections on the board are used to supply
▶3 power, e.g., 1 & 2 for ground; 3 & 4 for +5V; 73 & 74 for −12V; and 75 & 76 for +12V. Plugs or jacks may be placed on the end of the power supply wires to make easy interface. With both the P1 and P4 connectors and the output board wired, the total system is ready for interconnection.

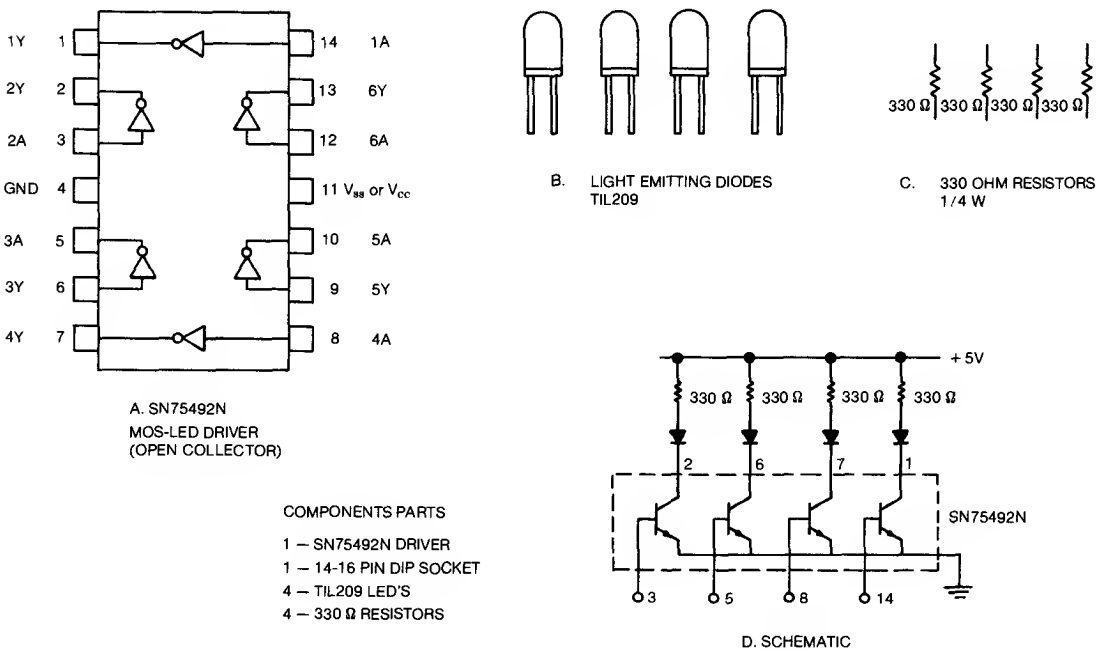
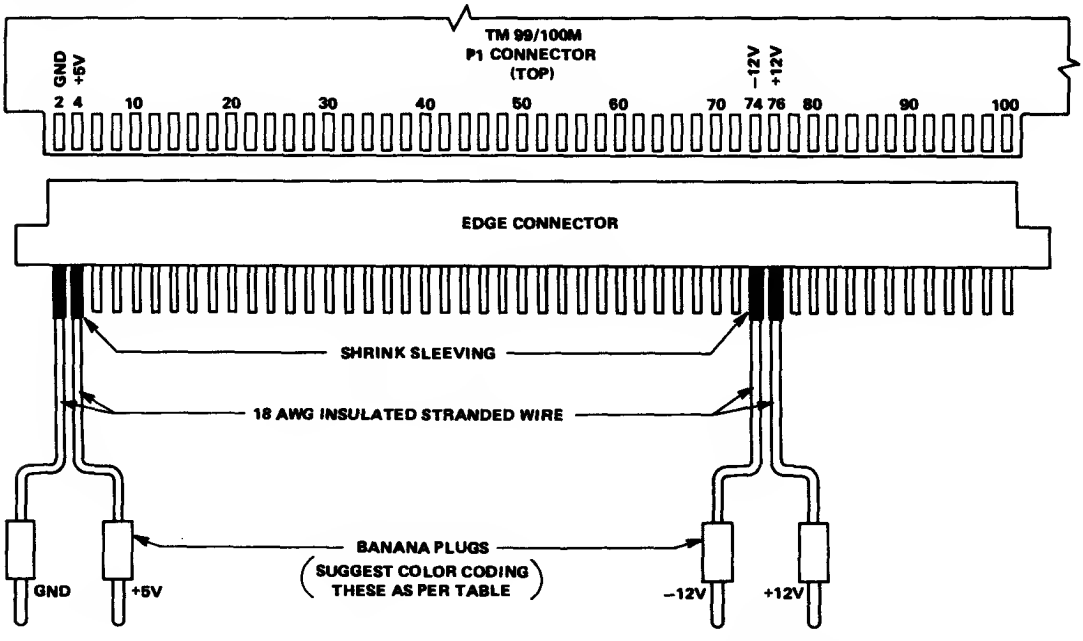


Figure 3-10. Alternative LED Output Display

UNPACKING AND CHECKING THE MICROCOMPUTER (TM990/100M-1)

It is very important to realize that the microcomputer module has MOS (metal-oxide-semiconductor) integrated circuits on it. These circuits are particularly sensitive to static charge and can be damaged permanently if such charge is discharged through their internal circuitry. Therefore, make sure to ground out all body static charge to workbench, table, desk or the like before handling the microcomputer board or any components that go onto it.

After unpacking the TM990/100M-1 module from its carton and examining it for any damage due to shipping, compare it to *Figure 3-12* to determine the correct location of all parts. Additional detail is available in the user's guide shipped with the board. Make sure that EPROM TIBUG Monitor (TM990/401-1) units are in the U42 and U44 positions on the board. Make sure that the RAM integrated circuits are in the U32, 34, 36, and 38 positions.



VOLTAGE	P1 PIN*	SUGGESTED PLUG COLORS
+5V	3, 4	RED
+12V	75, 76	BLUE
−12V	73, 74	GREEN
GND	1, 2	BLACK

*ON BOARD, ODD-NUMBERED PADS ARE DIRECTLY BENEATH EVEN-NUMBERED PADS.

Figure 3-11. Power Supply Hookup for 990/100M-1 Microcomputer

CAUTION: Before connecting the power supply to P1, use a volt-ohmmeter to verify that correct voltages are present as shown in Figure 3-11.

3

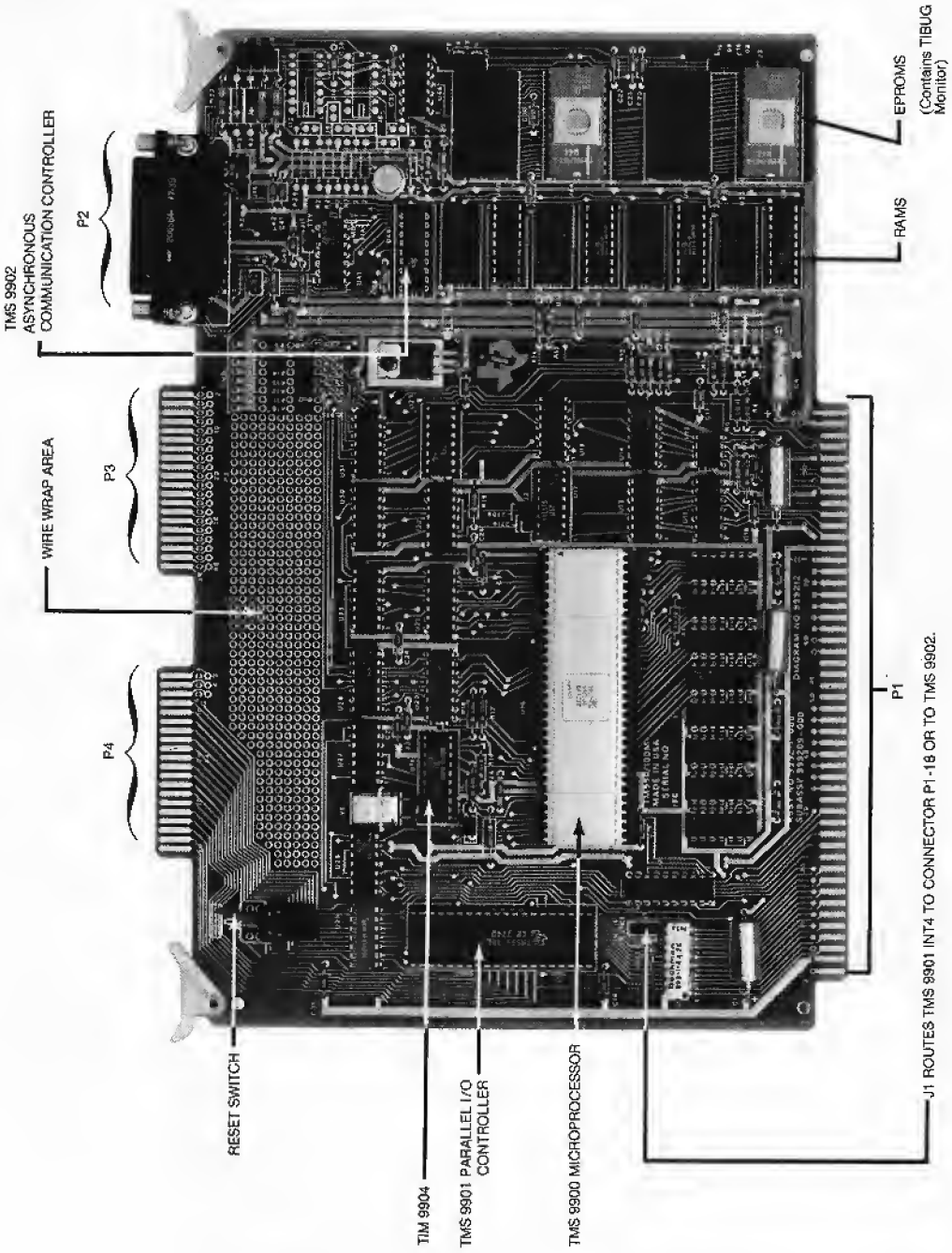


Figure 3-12. TM 990/100M-1 Module as Shipped

CONNECTING THE MICROTERMINAL/ OPERATING THE MICROCOMPUTER

Compare the board to *Figure 3-12 & 3-13.* Make sure that the jumpers are in the following positions:

JUMPER	POSITION	JUMPER	POSITION
J1	P1-18	J4	08, 08
J2	2708	J7	EIA
J3	08, 08	J11	OPEN

They assure that memory locations are identified correctly and that the microterminal interfaces correctly.

CONNECTING THE MICROTERMINAL TM990/301

The microterminal *(Figure 3-2)* should be examined to verify there is no damage due to shipment. It will be connected to the microcomputer through P2 on *Figure 3-12.* Jumpers J13, J14, and J15 must be installed on the TM990/100M-1 board in order to supply power to the microterminal. Using the extra jumpers provided, short pins on the board at J13, J14, and J15 *(Figure 3-13).* Attach the plug on the microterminal cable to the P2 connector on the board.

OPERATING THE MICROCOMPUTER

Check once more that all wiring is correct for the output board *(Figure 3-9)*, the power connector *(Figure 3-11)* and the jumpers, then follow these steps:

Step 1 Begin with connectors to P1 or P4 disconnected

Step 2 Turn on power supplies and verify that all voltages are correct at the connector for P1. Turn off power supplies.

Step 3 Connect the power supply connector to P1. Make sure edge connector has the word "TOP" showing. Turn on −12V supply first, then +12V, then +5V.

Step 4 Verify the voltages of +5V, −12V, and +12V on the board printed wiring connections near the edge of the board between P2 and P3. Adjust power supplies or verify trouble if these are not correct.

Step 5 Verify the voltages of these terminals:

J13	+5V
J14	+12V
J15	−12V

If these are incorrect, correct the problem.

Step 6 Turn off power supplies. With the top edge of connector for P4 in correct position, connect output board to P4, turn on power supplies in same sequence as before, −12V, +12V, +5V.

The total setup should now look like *Figure 3-14* and the microcomputer is now ready to perform the task; all that's required is to tell it what to do.

▶3

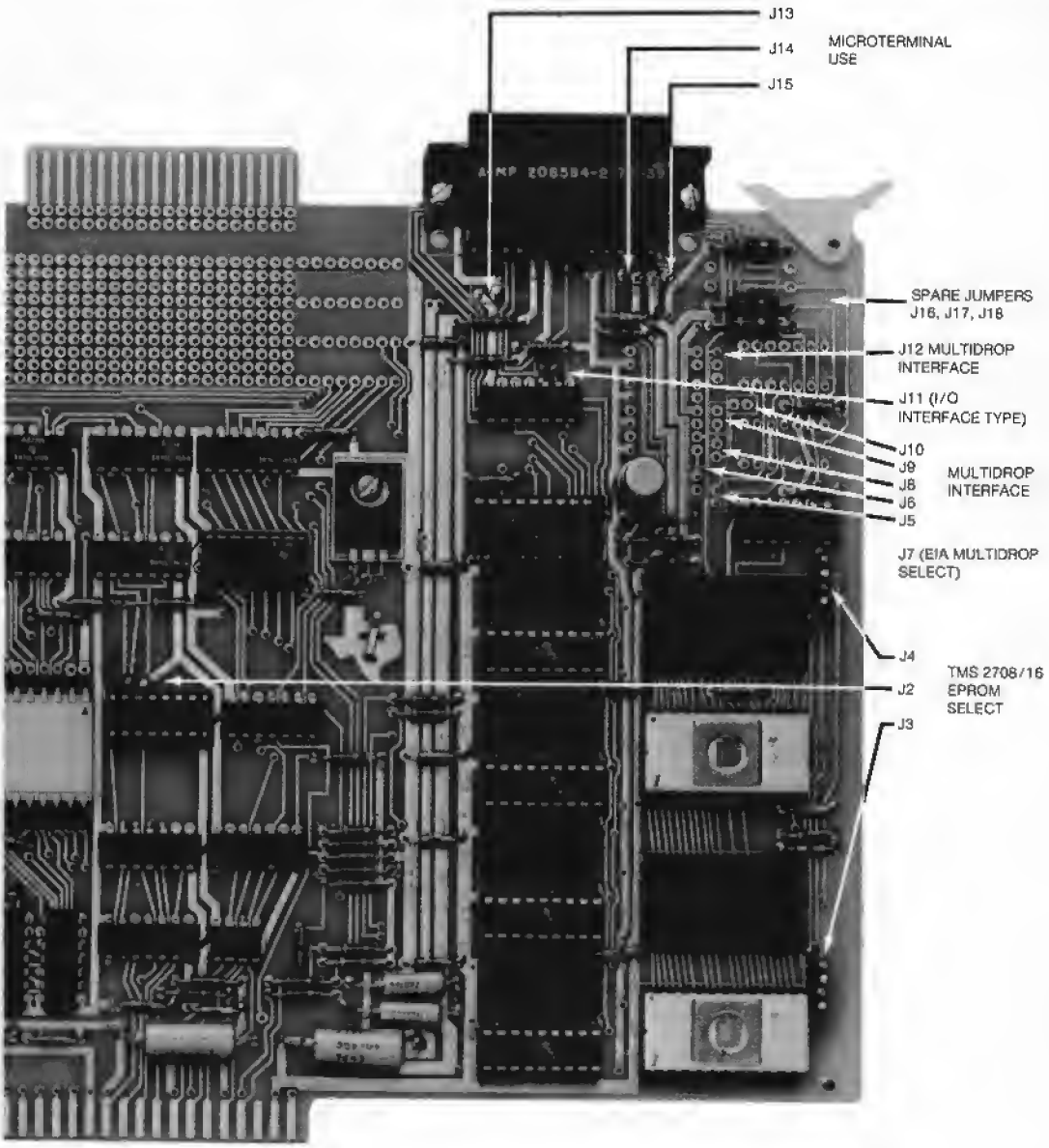


Figure 3-13. Jumpers used on TM 990/100M-1 Board for Option Selection

Figure 3-14. Total System Connected

TELLING THE MICROCOMPUTER WHAT TO DO

The microcomputer is told what to do through the microterminal keyboard. This is shown in *Figure 3-15.* Initial conditions are necessary so *Step 7* starts everything at an initial point.

Step 7 *Figure 3-1* and *Figure 3-12* identify the RESET switch. Switch it all the way to the right (facing the toggle). Now depress the CLR (clear) key on the microterminal. Nothing will be on the display but to verify that it is working, press several of the number keys. The numbers pressed will appear in the display. Now press the CLR key again on the microterminal.

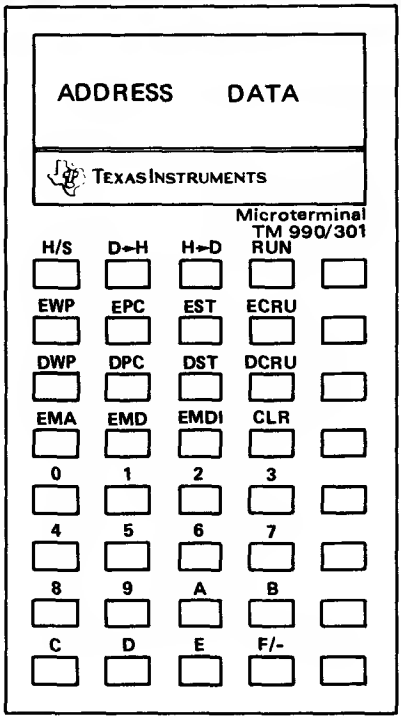
▶3

As we depress selected keys on the microterminal, the microcomputer is being given instructions — a step by step sequence of things to do to perform the first encounter task. The microcomputer is being *programmed* to do a job.

In order for the microcomputer to do its task according to the instructions given, it must also do many things dictated by other instructions that are stored in sequence in the TIBUG Monitor read-only memory (ROM). The program that performs the first encounter task is stored in the random access memory of the microcomputer and used in sequence. As a result, as the microcomputer accomplishes the task for which it is programmed, it performs each of the steps dictated by the "main program" in the RAM and by TIBUG in ROM.

There are only a few keys used on the microterminal for the first encounter. Identify these on *Figure 3-15* and on the microterminal. Three of these are: [EMA] (enter memory address) is used to display a specific memory address and give the user the ability to change the contents of that location. [EMD] (enter memory data) changes the contents of the memory location and [EMD!] (enter memory data and increment) changes the contents of the memory location and advances the address by two.

Note that *Figure 3-15* identifies the information given by the display. There are two banks of 4 digits each that are displayed. The left 4 digits display the address register (memory address) and the right 4 digits display the data in the data register (data to be stored in memory, being read from memory, or being operated on by the microcomputer). It is of no concern at the moment but both of these 4 digit registers are identifying the value of their data in hexadecimal code. Suffice it to say at this time that each hexadecimal digit represents 4 bits of data for a number that has a value represented by 16 bits. Each hexadecimal digit can have at any one time an alphanumeric value of any one of the following: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. The decimal value of these numbers are shown in *Figure 3-16* as they occur in the place value position of the 4 bit display. Hexadecimal numbers will be identified with a subscript of 16 in the text, e.g., $02E0_{16}$ or 0100_{16} whenever there is need to avoid confusion.



The display of the microterminal is divided into two 4 hexadecimal digit banks. The left bank displays address register information and the right bank displays data registers.

Figure 3-15. Microterminal Keyboard and Display

Every program starts at a particular place in the RAM memory. The first encounter program will start at memory location identified by the hexadecimal address FE00. This is a 16 bit address which in machine code looks like this: 1111 1110 0000 0000 (F = 15; E = 14; 0 = 0; 0 = 0) and from *Figure 3-16* has a decimal value of 61,440 + 3584 + 0 + 0 = 65,024. The program starts at memory location 65,024.

To start the sequence of instruction steps for out first encounter, the starting address is entered and the [EMA] (enter memory address) key is depressed on the microterminal. This is program *Step 2* in *Step 8.* To help verify the steps the display data is also recorded.

Step 8

KEYSTROKES	ADDRESS	Display DATA
0. [CLR]	—	—
1. [F/-] [E] [0] [0]		FE00
2. [EMA]	FE00	XXXX (X = Don't care)
3. [0] [2] [E] [0]	FE00	02E0

This keystroke at Step 3 is a hexadecimal code — an instruction — that is telling the microcomputer to load a register with data. The data, however, is at the next address location. Therefore, with the next keystroke [EMDI] (enter memory data and increment), the instruction 02E0 is stored at address location FE00 and the next memory address for an instruction is brought into the display by incrementing (advancing) the FE00 address by 2 (the reason for advancing by 2 will become clear as more is learned about the 9900 microprocessor).

Step 9

	KEYSTROKE	ADDRESS	DATA
▶3	4. [EMDI]	FE02	XXXX

MSB **LSB**

BITS	16^3 0 1 2 3		16^2 4 5 6 7		16^1 8 9 10 11		16^0 12 13 14 15	
	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
	0	0	0	0	0	0	0	0
	1	4 096	1	256	1	16	1	1
	2	8 192	2	512	2	32	2	2
	3	12 288	3	768	3	48	3	3
	4	16 384	4	1 024	4	64	4	4
	5	20 480	5	1 280	5	80	5	5
	6	24 576	6	1 536	6	96	6	6
	7	28 672	7	1 792	7	112	7	7
	8	32 768	8	2 048	8	128	8	8
	9	36 864	9	2 304	9	144	9	9
	A	40 960	A	2 560	A	160	A	10
	B	45 056	B	2 816	B	176	B	11
	C	49 152	C	3 072	C	192	C	12
	D	53 248	D	3 328	D	208	D	13
	E	57 344	E	3 584	E	224	E	14
	F	61 440	F	3 840	F	240	F	15

To convert a number from hexadecimal, add the decimal equivalents for each hexadecimal digit. For example, $7A82_{16}$ would equal in decimal 28,672 + 2,560 + 128 + 2. To convert decimal to hexadecimal find the nearest number in the above table less than or equal to the number being converted. Set down the hexadecimal equivalent then subtract its decimal number from the original decimal number. Using the remainder(s), repeat this process. For example:

$$31{,}362_{10} = 7000_{16} + 2690_{10}$$
$$2{,}690_{10} = A00_{16} + 130_{10}$$
$$130_{10} = 80_{16} + 2_{10}$$
$$2_{10} = 2_{16}$$

$$\begin{array}{r} 7000 \\ A00 \\ 80 \\ 2 \\ \hline 7A82_{16} \end{array}$$

Figure 3-16. Place Value of Hexadecimal Digits in Significant Bit Positions

Program *Step 4* of operating *Step 9* shows this. Memory location identified by address FE02 is now ready for the data that will be put into the register identified by the instruction 02E0 at location FE00. The data is FF20.

5.	[F] [F] [2] [0]	FE02	FF20
6.	[EMDI]	FE04	XXXX

Program *Step 6* has now advanced to the next memory location which is awaiting the next instruction which is keystroked in by program *Step 7.*

7.	[0] [2] [0] [1]	FE04	0201

Step 10

Continue now to program steps through the end of the program. Note how the address memory location advances by 2 each time [EMDI] is pressed. This is how the program will be followed when it is run. The starting address FE00 will be loaded into the program counter. The program counter will then count by 2 and advance the microcomputer through each program step as the instructions are completed.

	KEYSTROKE	ADDRESS	DATA
8.	[EMDI]	FE06	XXXX
9.	[F] [E] [2] [E]	FE06	FE2E
10.	[EMDI]	FE08	XXXX
11.	[0] [2] [0] [C]	FE08	020C
12.	[EMDI]	FE0A	XXXX
13.	[0] [1] [2] [0]	FE0A	0120
14.	[EMDI]	FE0C	XXXX
15.	[1] [D] [0] [0]	FE0C	*1d00
16.	[EMDI]	FE0E	XXXX
17.	[0] [6] [9] [1]	FE0E	0691
18.	[EMDI]	FE10	XXXX
19.	[1] [E] [0] [0]	FE10	1E00
20.	[EMDI]	FE12	XXXX
21.	[0] [6] [9] [1]	FE12	0691
22.	[EMDI]	FE14	XXXX
23.	[1] [D] [0] [1]	FE14	*1d01
24.	[EMDI]	FE16	XXXX
25.	[0] [6] [9] [1]	FE16	0691
26.	[EMDI]	FE18	XXXX
27.	[1] [E] [0] [1]	FE18	1E01
28.	[EMDI]	FE1A	XXXX
29.	[0] [6] [9] [1]	FE1A	0691
30.	[EMDI]	FE1C	XXXX
31.	[1] [D] [0] [2]	FE1C	*1d02

*As displayed on 301 Terminal

Keystroke	Address	Data
32. [EMDI]	FE1E	XXXX
33. [0] [6] [9] [1]	FE1E	0691
34. [EMDI]	FE20	XXXX
35. [1] [E] [0] [2]	FE20	1E02
36. [EMDI]	FE22	XXXX
37. [0] [6] [9] [1]	FE22	0691
38. [EMDI]	FE24	XXXX
39. [1] [D] [0] [3]	FE24	*1d03
40. [EMDI]	FE26	XXXX
▸3 41. [0] [6] [9] [1]	FE26	0691
42. [EMDI]	FE28	XXXX
43. [1] [E] [0] [3]	FE28	1E03
44. [EMDI]	FE2A	XXXX
45. [0] [6] [9] [1]	FE2A	0691
46. [EMDI]	FE2C	XXXX
47. [1] [0] [E] [F]	FE2C	10EF
48. [EMDI]	FE2E	XXXX
49. [1] [F] [0] [4]	FE2E	1F04
50. [EMDI]	FE30	XXXX
51. [1] [3] [0] [5]	FE30	1305
52. [EMDI]	FE32	XXXX
53. [0] [2] [0] [3]	FE32	0203
54. [EMDI]	FE34	XXXX
55. [F] [F] [F] [F]	FE34	FFFF
56. [EMDI]	FE36	XXXX
57. [0] [6] [0] [3]	FE36	0603
58. [EMDI]	FE38	XXXX
59. [1] [6] [F] [E]	FE38	16FE
60. [EMDI]	FE3A	XXXX
61. [0] [4] [5] [B]	FE3A	*045b
62. [EMDI]	FE3C	XXXX
63. [0] [2] [0] [3]	FE3C	0203
64. [EMDI]	FE3E	XXXX
65. [3] [F] [F] [F]	FE3E	3FFF
66. [EMDI]	FE40	XXXX
67. [0] [6] [0] [3]	FE40	0603
68. [EMDI]	FE42	XXXX
69. [1] [6] [F] [E]	FE42	16FE
70. [EMDI]	FE44	XXXX
71. [0] [4] [5] [B]	FE44	*045b
72. [EMDI]	FE46	XXXX

TELLING THE MICROCOMPUTER WHAT TO DO

Step 11

All the program steps are now entered. It remains to run the program, that is, send the microcomputer through its sequenced steps to determine if it will accomplish the task.

Recall, that the system must be set to the initial conditions and to the starting point. This means that the system must start at memory address FE00 because that is where the first instruction is located.

Inside the microcomputer there is a register (a temporary storage location for 16 bits) that always contains the address of an instruction. It was previously noted that as the memory location of instructions was incremented by 2 as the program was entered, so also will the program counter be incremented by 2 by the microcomputer to go to the next instruction. Therefore, the initial conditions are accomplished by loading the program counter with the address location FE00. This is accomplished by an [EPC] key on the microterminal. The [EPC] (enter program counter) key changes the value of the program counter. It will enter into the program counter the value that is in the data register of the microterminal display.

The [DPC] (display program counter) key on the microterminal is depressed to determine if the correct value has been entered into the program counter because it displays the current value of the program counter.

The [RUN] key is depressed to begin execution of the program starting with the address in the program counter.

To run the program, go through *Steps 1* thru *5.*

KEYSTROKE	ADDRESS	DATA
1. [CLR]	—	—
2. [F/-] [E] [0] [0]	—	FE00
3. [EPC]	—	FE00
4. [DPC]	—	FE00
5. [RUN]	—	run

VOILA!

The first encounter task is being accomplished. Switching the toggle switch will change the rate of the segment display.

Under program control output logic levels on a set of output lines have been set to a "1", held for a time, set to a "0", held for a time, etc. in a particular sequence. The delay between "1s" and "0s" also is under program control. Such output levels then have been interfaced to driver circuits to accomplish a given task — in this case lighting LED segments of a display.

Step 12

To stop the program, depress [H/S] . The RESET switch on the microcomputer could also be pressed. (However, in doing so, to return to the program, go through the initial five steps of running the program at the end of operating *Step 10.*) The program may be started again by depressing [RUN] after it was halted by [H/S]

Step 13

If for some reason the first encounter task is not being accomplished after completing *Step 10*, the program can be checked by entering FE00, the beginning address and depress [EMA] . The contents of memory and the instruction at FE00 will be displayed.
▸3 Each memory location can then be examined by depressing [EMDI] and reading the display. In this manner, the program can be examined for an error. When the error is located, the correct data can be entered as it was in the original program and [EMD] is pressed. The program can then be run by returning to the initial sequence of operating *Step 11.*

The program may be entered at any valid address by entering the address and pressing [EMA] and then proceeding step by step with [EMDI]. There is no need to go back to the beginning address each time.

HOW WAS IT DONE?

The question naturally arises — how was this task accomplished by the microcomputer, and more importantly, how was the task taken from idea to the actual program? How does one know what to tell the microcomputer to do?

Of course, this will take a great deal of study of this book and much operation of systems, starting with the TM990/100M-1 microcomputer. The way the *idea* is turned into a *program* for the first encounter is covered in the remaining part of this chapter. This is a good foundation for building knowledge of the 9900 microprocessor, applying the 990/100M microcomputer to many other tasks, and understanding the use of the 9900 in solving other types of problems.

BACK TO BASICS

The process of understanding how the task was taken from idea to instructions for the microcomputer begins by returning to some basic concepts to assure that these are understood.

Recall that *Figure 3-4* identified the functional blocks of our microcomputer. The central processing unit includes the 9900 microprocessor. Examining *Figure 3-5* further and the functional block diagram of *Figure 3-17* shows that the 990/100M microcomputer is bus oriented. Recall that a bus is one or more conductors running in parallel which are used for sending information. The 9900 microprocessor sends an address to memory, to identify data required, on the 15-bit address bus. It receives data from memory on a 16-bit data bus. It should be noted that the same 15-bit address bus goes to the input/output interface units. The address bus is used either to send an address to *memory* or an address to *input/output*, not both at the same time. When the signal $\overline{\text{MEMEN}}$ is a logic low, the address bus is for memory. If the address bus is not for memory then it can be used by I/O. When the address is for I/O, the selection of which lines will be inputs or outputs is under control of the 9900.

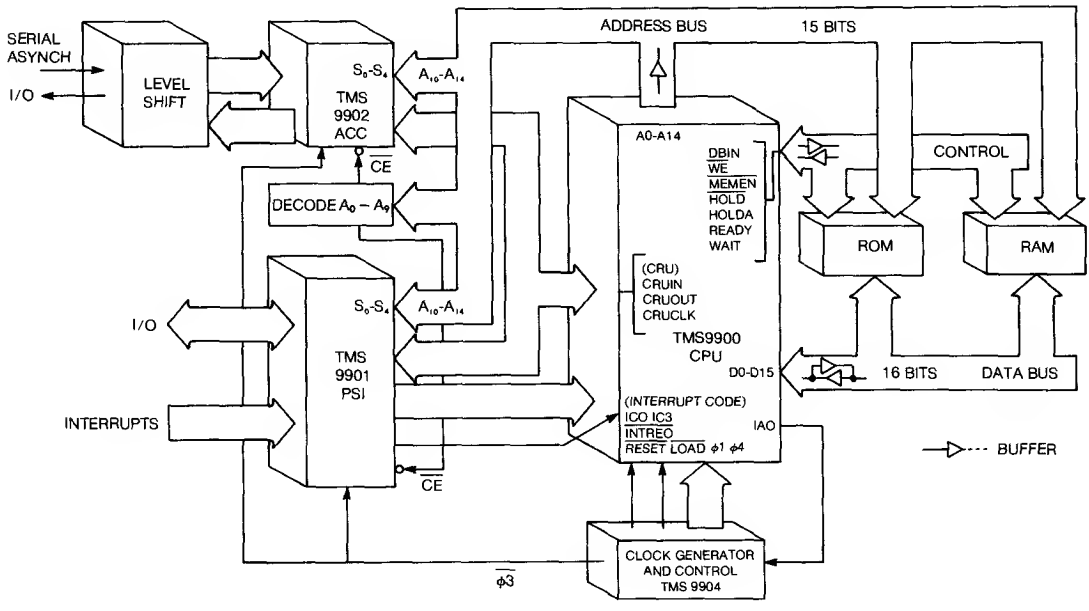


Figure 3-17. Functional Diagram of TM 990/100M-1 Microcomputer

Therefore, lines to accept data as input, or to deliver output data are selected by address bits in the same fashion that address bits locate data in a memory.

Examination of the architecture of the 9900 microcomputer in *Figure 3-18* reveals, as in *Figure 3-17,* the address bus, the data bus, signals for the CRU (the Communications Register Unit is an I/O interface for the 9900 architecture), signals for interrupt, control signals and master timing signals. Each of these are external signals. Further examination of internal parts is required to expand on more basic concepts, with emphasis on the ones that are used for the first encounter task.

REGISTERS

▸3 Recall that a register is a temporary storage unit for digital information. Inside the 9900 there are these types of registers: a memory address register, a source data register (data register), an instruction register, an interrupt register, some auxiliary registers like T_1 and T_2, and the registers that will be most applicable to the first encounter — the program counter, the workspace register, the status register and a shift register used as part of the hardware to select the input and output terminals. Additional parts include: 1) the ALU — it is the arithmetic and logic unit that performs arithmetic functions, logic and comparisons. 2) Multiplexers that direct the data over the correct path as a result of signals from the control ROM and control circuitry. 3) Timing circuits so that all operations are synchronized by the master timing.

Every time a piece of information is required to be stored in memory or retrieved (fetched) from memory, the memory must be told where the data is located or to be located. The memory address register holds the address to be put on the address bus for this purpose.

Data fetched from memory is received either by the source data register and distributed by the 9900 microprocessor as required, or by the instruction register when it is an instruction. The instruction is decoded and transmitted to the control ROM which sequences through microinstructions previously programmed into the control ROM to execute the instruction. The instruction might be "Increment register 1 by two". Instruction steps take the data from register 1 to the ALU which adds "2" and returns the data to register 1.

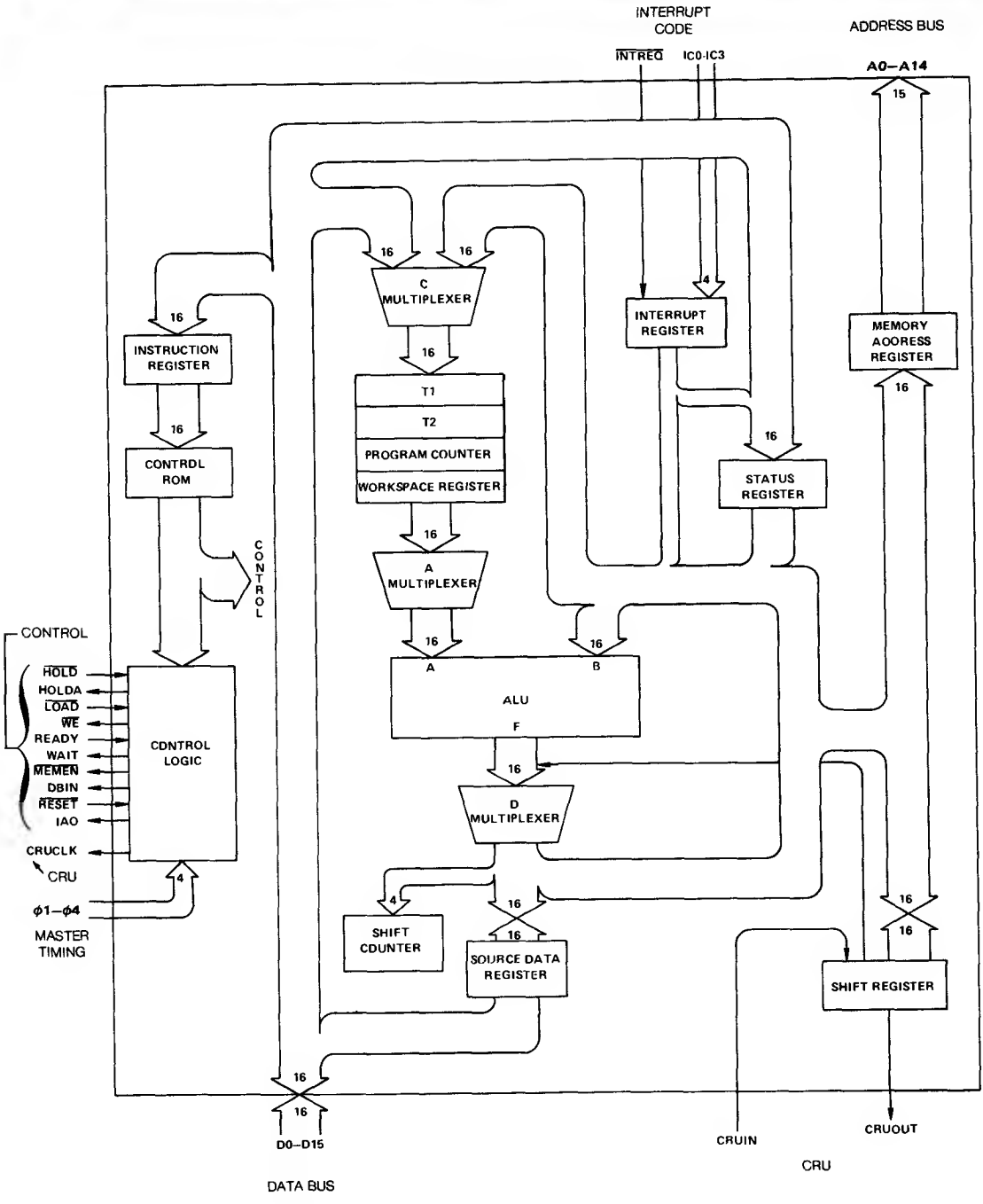


Figure 3-18. Architecture of 9900 Microprocessor

Two registers of significant concern for the first encounter task are the status register and the workspace register. The status register is just what the name implies. The 9900 microprocessor continually checks on how things are going (the status) by following instructions that command it to check various bits of the status register. *Figure 3-19* shows the bits of the status register.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ST0 L>	ST1 A>	ST2 =	ST3 C	ST4 O	ST5 P	ST6 X	not used (=0)					ST12	ST13	ST14	ST15
												Interrupt Mask			

Figure 3-19. Status Register

▶3 Each bit of the first 7 bits is concerned with identifying that a particular operation or event has or has not occurred as shown here.

Bit	Purpose	Bit	Purpose
0	Logical Greater Than	4	Overflow
1	Arithmetic Greater Than	5	Parity
2	Equal	6	XOP
3	Carry	12-15	Interrupt Mask

The last 4 bits are concerned with the interrupt signals and a priority code associated with the interrupts.

The first encounter uses bit 2, the "equals" status bit to change the time delay in the LED sequence.

Workspace

The workspace register is the same as the other registers, but it is used in a special way. As the 9900 microprocessor and the microcomputer step through program instructions, there is a need to have more registers than those available on the 9900. Instead of providing these registers in the 9900, a file of registers is set up in memory and a reference to this file saved in the workspace register. One of the rules in setting up this file is that it will always contain 16 registers in 16 contiguous (one following another in sequence) memory words. The workspace register on the 9900 is called the workspace pointer because, as shown in *Figure 3-20*, it contains the address of the first memory word in the contiguous register file, referred to for the application of the 9900 and in this book as "workspace registers" or just "workspace". The register file can be located anywhere within RAM that seems appropriate. In the total available memory space, there are certain reserved spaces for RAM, others for ROM, and others for special instructions. Therefore, the register file can only be set up in certain portions of memory. So, where 0200_{16} to $021E_{16}$ are the 16 locations shown in *Figure 3-20a,* with the workspace pointer being 0200_{16}, the file could have started at 0300_{16} and extended to $031E_{16}$ as long as these are allowable locations in the overall memory matrix. The workspace pointer would contain 0300_{16} in the second case.

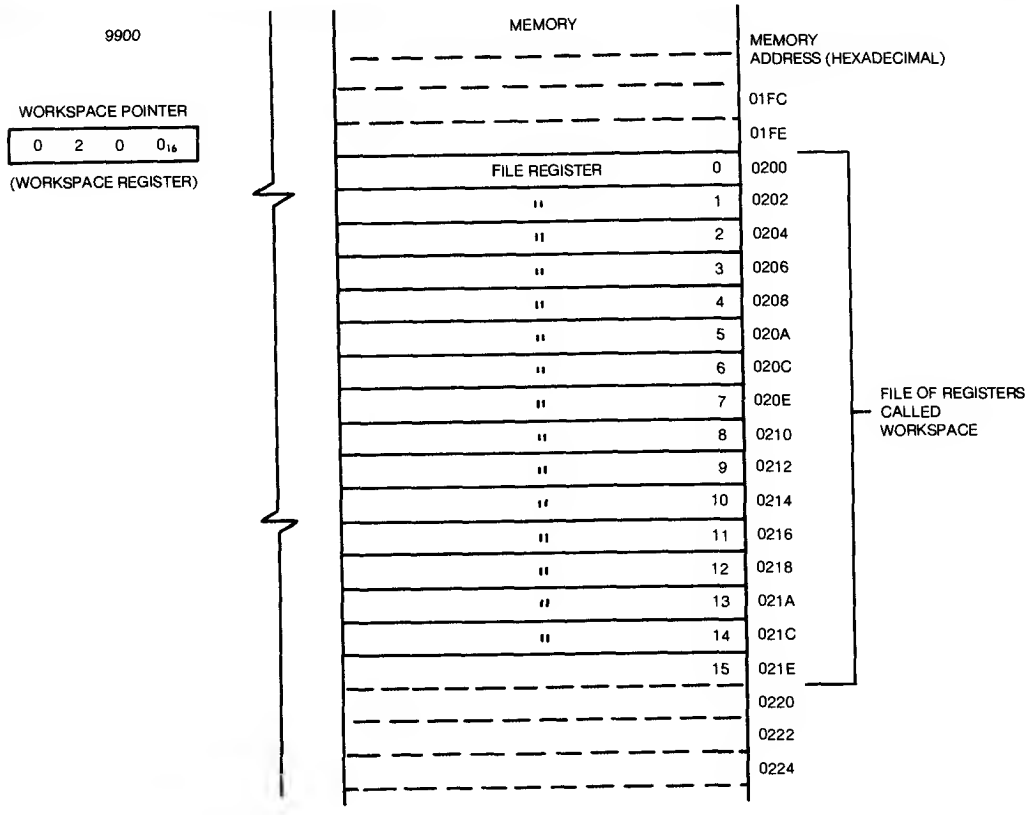


Figure 3-20a. Workspace Registers

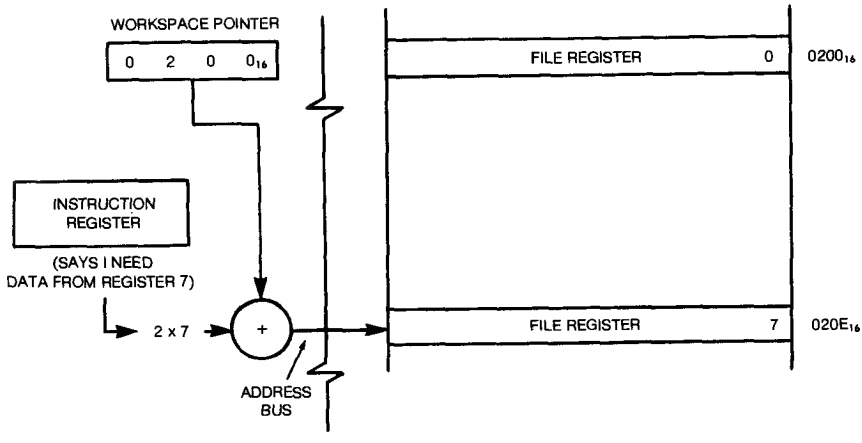


Figure 3-20b. Locating Specific Register

To locate a specific register in the workspace file, the 9900 microprocessor adds the register number to the workspace pointer address to obtain the address of the specific register in the file that is required. (It actually adds 2R, where R is the register number, so that the addresses advance by even numbers. The odd number addresses are used when the word contents are to be processed in 8-bit bytes.) For example, if register 7 contains the information required by the 9900 microprocessor, then the address 020E in *Figure 3-20a* is obtained by adding 14 to the workspace pointer at 0200_{16}. This is shown in *Figure 3-20b.* In like fashion, if the workspace pointer contained 0300_{16}, then adding 14 to 0300_{16} gives $030E_{16}$ the address of register 7, the 7th register down in the file.

▸3 Recall that to accomplish the first encounter task, logic levels on output lines had to be set to a "1" or a "0" in order for the LED drivers to turn on or turn-off the LED segment respectively. Recall, also, that the particular output lines could be selected. To understand how this is done, refer to *Figure 3-21.* This figure is divided into three bounded regions; the TMS 9900, Memory, and the TMS 9901. The output line from the 9900 microprocessor that will do the setting is the line "CRUOUT." It is coupled to the TMS 9901, the programmable systems interface.

The TMS 9901 contains more functional parts to handle the interrupt code and interrupt input signals but for now the part that is important is that shown in *Figure 3-21.* The portion shown is a demultiplexer. The data appearing at CRUOUT is strobed by CRUCLK into latches feeding the output pins. The particular latch and the particular output line is selected by the code that exists on the select bit lines S_0, S_1, S_2, S_3, and S_4, which, as shown in *Figure 3-21*, are the address lines A_{10} through A_{14}. The code on S_0 through S_4, and the CRU logic selects the output latch and line that is to be set. The "1" or "0" on CRUOUT does the setting. The latching occurs when CRUCLK strobes the data in.

SBZ AND SBO INSTRUCTIONS

Enough basics have now been covered to begin understanding several of the important instructions for the first encounter task. *Figure 3-21* will again be used and will be followed from left to right and top to bottom starting with the upper left corner. At a particular step in the program, controlled by the program counter, the instruction address (the bit contents of the program counter) is sent to memory over the address bus to obtain the instruction. Memory is read and the instruction is received by the 9900 on the data bus and placed in the instruction register. Via the control ROM and the control logic, the instruction is interpreted as an SBO instruction — "set CRU bit to one." The 9900 is designed so that it generates the correct S_0-S_4 address for the TMS9901 that selects the output line to be set to a "1" by the instruction. However, as indicated in *Figure 3-21,* first an ALU operation must occur before the correct address is obtained. The ALU adds the contents of one of the registers in the file, workspace register 12 (WR12), to a portion of the instruction, SBO. This portion of the SBO instruction is identified as DISP

(meaning displacement) in *Figure 3-21.* It identifies the specific line to be used in the 9901 for the output. Eight bits are used for the signed displacement (7 and a sign). Bits 3 through 14 are used from the workspace register 12.

After the ALU operation, the address is sent out on the address bus. Because the $\overline{\text{MEMEN}}$ line is not active, this tells the 9901 that the address is for I/O. All 15 address bits are there; however, only A_3 through A_{14} are used for the effective CRU address. A_{10} through A_{14} provide S_0 through S_4 for the 9901, while bits A_0 through A_9 are used for decoding additional I/O as shown in *Figure 3-17.* A_0, A_1, and A_2 are set to zero for all CRU data transfer operations.

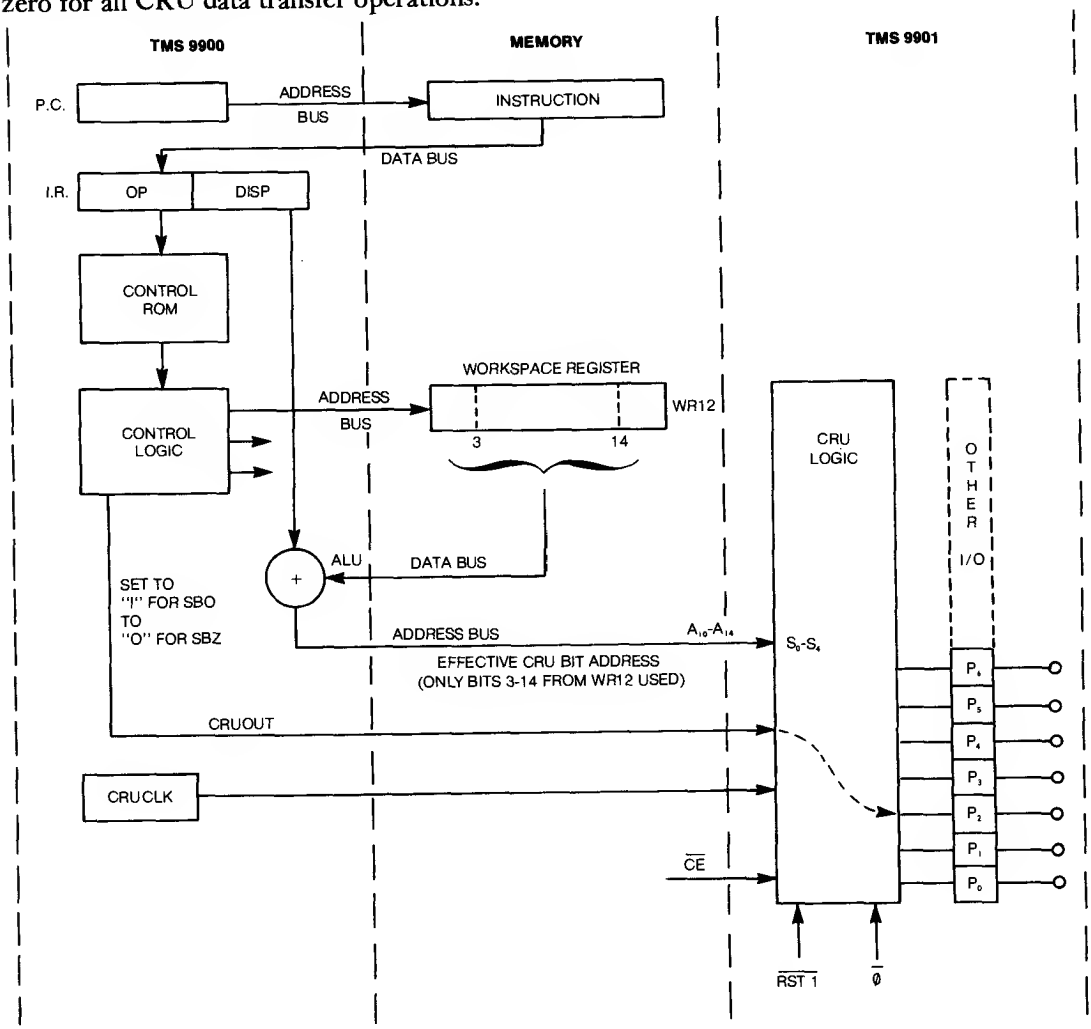


Figure 3-21. CRU Concept — Single Bit Output SBO or SBZ on CRUOUT.

I/O Selection

The codes required on S_0 through S_4 that select a specific output or input in the 9901 are shown in *Figure 3-22.* To make it convenient, the P_0 line will be used for bit zero of the output or input, P_1 for bit one, P_2 for bit 2, P_3 for bit 3 and P_4 for bit 4. Therefore, to select the line for P_0, the code on S_0 through S_4 must be 1 0000. Rather than starting at select bit 0, the output sequence is started at select bit 16. Adding 32_{10} (base ten) to the contents of the file register 12 accomplishes this. This is 10_{16} in Hex code times two, to shift it into bits 3 through 14.

What do the contents of register 12 indicate? They identify the particular 9901 used. Referring back to *Figure 3-17*, it is noted that several I/O units are connected to the address bus of the microcomputer TM990/100M-1. In order for the decoder to activate the correct $\overline{CE}$ signal to enable the right I/O, a base address is assigned to each I/O unit. The software base address for the 9901 on the microcomputer is 0100_{16}. The hardware base address is 0080_{16}.

Figure 3-23 summarizes the ALU operation. Workspace register 12 contains the software base address of the 9901 on board the microcomputer. The signed displacement of $+10_{16}$ is located in the instruction register as part of the SBO instruction. These two pieces are added together by the ALU and the result placed in the address register. Note that the ALU uses only the bits from 3 to 14 of the software base address to get the hardware base address, adds the displacement, and that the effective CRU address is bits 0 through 14. Bit 15 becomes a "don't care" bit.

SELECT BIT	S_0	S_1	S_2	S_3	S_4	INPUTS	OUTPUTS
0	0	0	0	0	0		
⋮	⋮	⋮	⋮	⋮	⋮		
15	0	1	1	1	1		
16	1	0	0	0	0	P_0IN	P_0OUT
17	1	0	0	0	1	P_1IN	P_1OUT
18	1	0	0	1	0	P_2IN	P_2OUT
19	1	0	0	1	1	P_3IN	P_3OUT
20	1	0	1	0	0	P_4IN	P_4OUT
21	1	0	1	0	1	P_5IN	P_5OUT
22	1	0	1	1	0	P_6IN	P_6OUT
23	1	0	1	1	1	P_7IN	P_7OUT
24	1	1	0	0	0	P_8IN	P_8OUT
25	1	1	0	0	1	P_9IN	P_9OUT
26	1	1	0	1	0	P_{10}IN	P_{10}OUT
27	1	1	0	1	1	P_{11}IN	P_{11}OUT
28	1	1	1	0	0	P_{12}IN	P_{12}OUT
29	1	1	1	0	1	P_{13}IN	P_{13}OUT
30	1	1	1	1	0	P_{14}IN	P_{14}OUT
31	1	1	1	1	1	P_{15}IN	P_{15}OUT

Figure 3-22. I/O Selection in TMS 9901

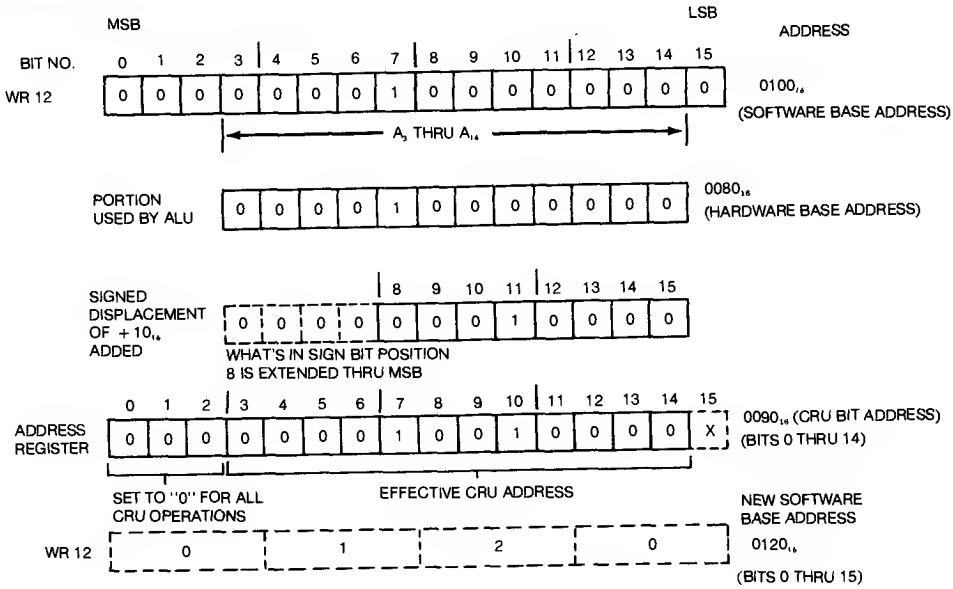


Figure 3-23. Generating the Output Line Address for the 9901

Recall that the software base address assigned to the 9901 is 0100_{16} but this is to be changed by the added displacement of 10_{16}. If all of WR 12 were used, the sum would be 0110_{16}. Because the signed displacement addition occurs with bit 15 neglected, the effective CRU address sent to the 9901 for bits A_0 through A_{14} is 0090_{16}, the hardware base address of 0080_{16} plus the 10_{16} displacement. Had bit 15 not been neglected, the sum would appear as shifted over one bit position or 0120_{16}.

Additional displacement will have to be added to the 10_{16} displacement to obtain the correct code for P_1, P_2 and P_3 shown in *Figure 3-22.* To be able to add a displacement of "0" for the zero bit, 1 for the one bit, 2 for the two bit, 120_{16} is used as the software base address in workspace register 12 right from the start.

From the past discussion, it should be quite clear now that one of the 32 outputs or inputs can be selected by including this information with the SBO instruction, and that a particular 9901 (if there were more than one) is selected by programming the correct base address into the workspace register 12.

Referring back to *Figure 3-21,* a SBZ — "Set CRU bit to Zero" — instruction is the same as the SBO instruction except that the output latch is now set to a "0" rather than a "1". Note in particular in both of these instructions that only one bit is set at a time. An instruction must be included for each bit to be set when using SBO and SBZ.

TB INSTRUCTION

Besides setting logic levels on output pins, an additional system requirement for the first encounter task is to receive an input on an input line. One way of accomplishing this is to have the 9900 microprocessor look at a selected input line, bring the information present at a specified time into the 9900 and then examine the information, or test it, to determine if the information was a "1" or a "0". The TB instruction, "Test CRU Bit", accomplishes bringing the information into the 9900. Subsequent instructions are added to determine if the information was a "1" or a "0".

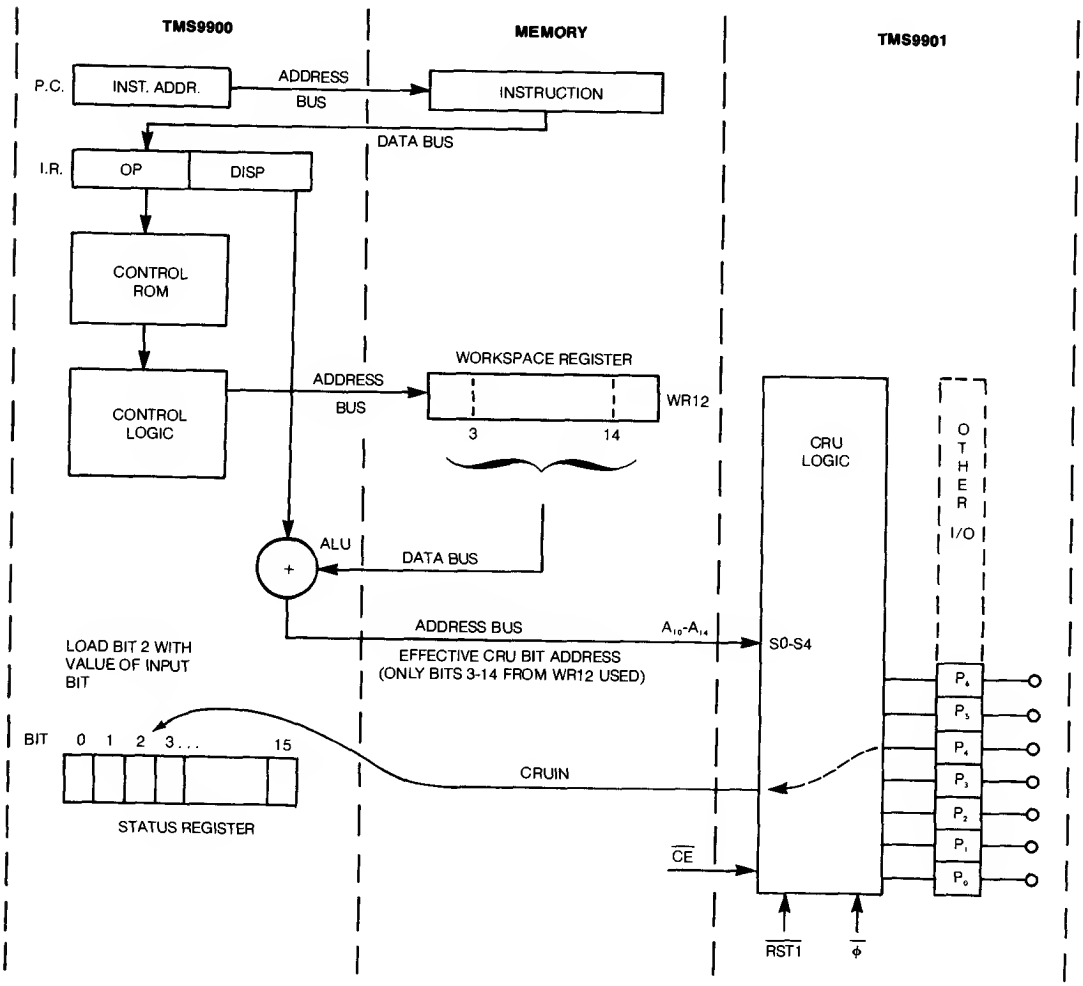


Figure 3-24. CRU Concept-Single Bit TB Input on CRUIN

The selection of the particular line in the I/O unit is the first concern. *Figure 3-24* shows that this is done in the same way as just explained for the SBO and SBZ instructions. The same portions of the 9901 are used as for the SBO or SBZ instructions except now these portions are a data selector. Data is selected from one of multiple input lines and sent to the 9900 microprocessor along the CRUIN line. The value of the information on the line is placed in bit 2 position of the status register. As discussed previously for the status register, instructions must follow the TB instruction that will examine bit 2 of the status register to determine what to do if this bit is a "1" and what to do if it is a "0". Conditional jump instructions are used to make the decision based on the value of the data. Note again that this is done one bit at a time.

Accomplishing a TB instruction requires that a base address be given for the particular input or output line desired. This hardware base address adjusted to a software base address is placed in workspace register 12. With the TB instruction, a displacement is given that identifies the particular line which needs to be sampled. This again is the same as for SBO. The line selected provides data straight through to the CRUIN line — there are no latches, as with the output data.

Thus, the basic concepts studied have shown the means of getting data to the output and bringing data in from an input — one bit at a time. They have shown how data is located, read, transferred, stored, and operated on arithmetically. With this, it should be possible now to get the first encounter idea into a sequence of steps — a program for the microcomputer to follow.

IDEA TO FLOWCHART

Bringing the idea from concept to program begins with a concept level diagram as shown in *Figure 3-25*. It has been decided that the microcomputer is to do the first encounter task; turn on and off 4 lights in sequence, with a time delay between each light activation.

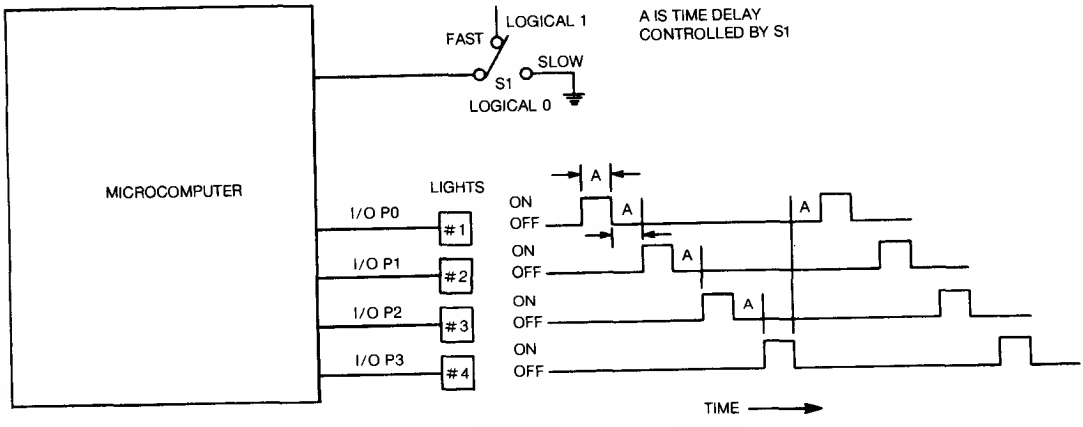


Figure 3-25. Concept Level Diagram

The time delay is to be under control of an external switch.

Understanding the basic concepts of the microcomputer led to the discussion that output lines could be selected and set to standard TTL logic levels to control drivers that would light the lights. In like fashion, a standard TTL logic level signal could be brought to the microcomputer as an input and examined. With this information, a decision could be made to vary the time delay. If the input is a "1," the lights would go on and off at a fast sequence. If the input is a "0," the sequence rate would be slow.

Obviously, other mechanical decisions also were made, such as:

1) The lights would be segments of a 7 segment light emitting diode numerical display
▶3 because of the compatible packaging and ease of availability.

2) The microcomputer output pins, I/O identification and light number to 7 segment display segment were set as follows:

990/100M	*9901 I/O*	*Light No.*	*Display Segment*	*Note*
P_4 Connector				
20	P_0	1	f	
22	P_1	2	b	
14	P_2	3	e	
16	P_3	4	c	
18	P_4			to S_1

(These pin identifications are obtained from the schematics in the TM990/100M User's Guide and data sheet information on the TIL303.)

The microterminal TM990/301 was selected as the unit to use for communication with the microcomputer because of its low cost and ease of use. Terminals such as a TTY and a 743 KSR can be used and an application shown in Chapter 9 takes up this type interface.

FLOWCHARTS

The problem solution proceeds from concept to program by constructing a well defined flowchart to follow in an organized fashion while generating the sequence of steps required for the microcomputer to complete the task. *Figure 3-26* is such a flowchart of the first encounter task.

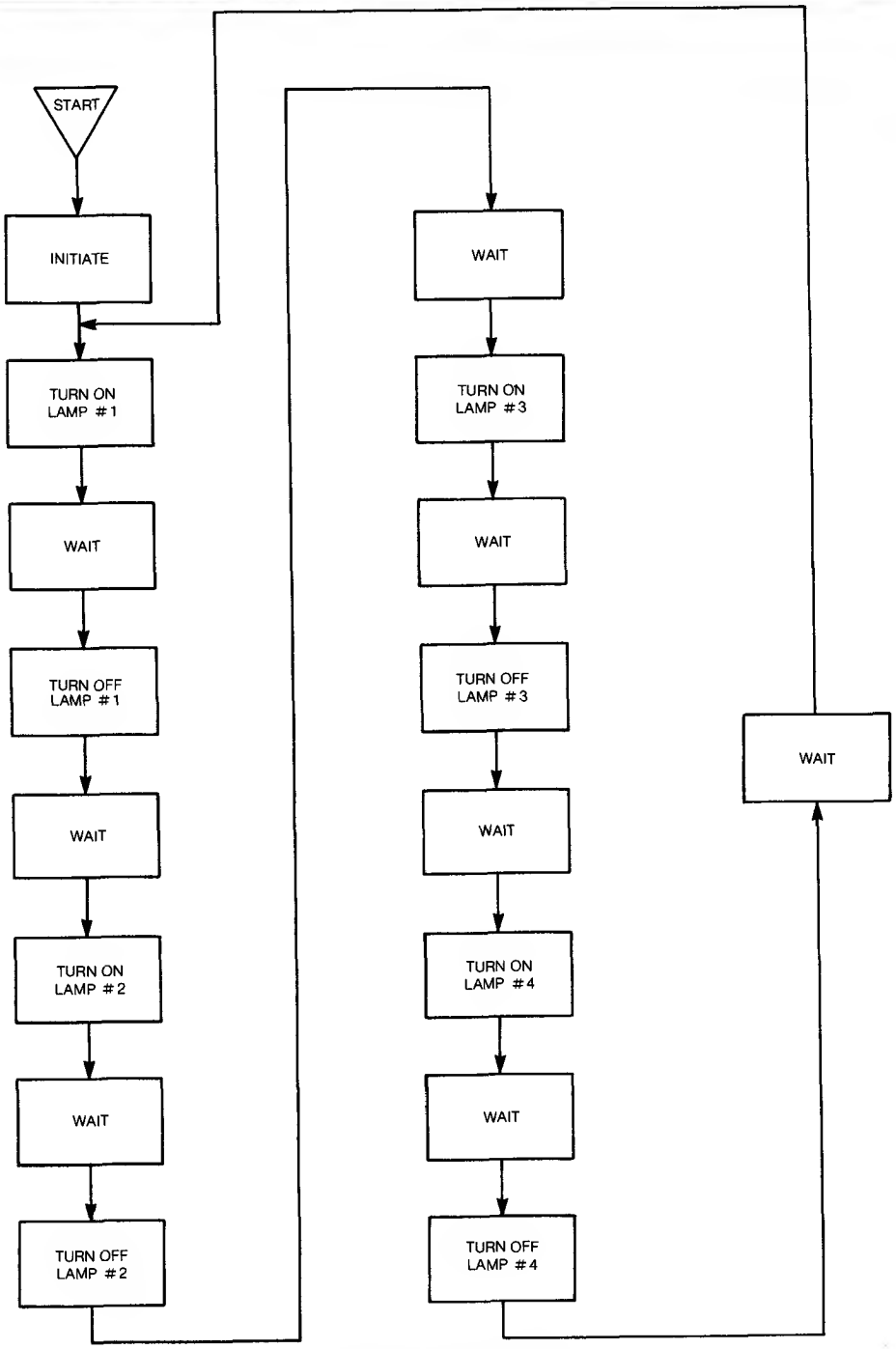


Figure 3-26. Flowchart

From START, which requires initial conditions, and a signal to begin — INITIATE — the task is diagrammed. Each light is turned on, the time delay occurs, the light is turned off, the time delay occurs, the next light is turned on, etc. The sequence continues until all lights have been turned on and off and the program begins again.

WAIT SUBROUTINE

Note the time delay is identified as WAIT and it occurs over and over again in the sequence. Because of this, separate steps will be written for this sequence only one time rather than repeating it over and over in the program. In this manner, the main sequence of steps, the main program, can be directed to this identified set of steps, called a subroutine, by an instruction. The main program is then said to branch to the subroutine until it completes the steps in the subroutine, then it returns to the main program.

In simpler terms, the WAIT block of the flowchart requires a given number of program steps, say X. WAIT occurs 8 times in the flowchart. Instead of rewriting the X steps 8 times in the program, the X steps are written once, given the name WAIT, and referred to 8 times.

Because WAIT is a subroutine, a separate flowchart *(Figure 3-27)* is generated for it. In addition, the time delay is to be varied by the switch S_1, therefore, different steps are followed if the switch is "on" with a value of a logical "1" or "off" with a value of a logical "0". Note that when the subroutine WAIT is encountered, the first thing that occurs is to find out the position of the switch. Is it a logical "1" or a logical "0?" A decision is made on the basis of what is found. "Yes, the switch is on," (logical "1") makes the time delay short and the sequence fast. "No, the switch is off," (logical "0") makes the time delay long and the sequence slow.

There are a number of ways to provide a time delay. This flowchart uses one of the simplest — load a register with a number, keep subtracting one (decrementing) from the number until the number is zero. The number of cycles it takes to get the number to zero times the time for each cycle is the time delay. Larger numbers, longer counts, provide longer delays.

Each arm of the flowchart contains the same type of sequence, loading the number; decrementing; checking for zero; if not zero, jumping back and decrementing again; if zero, returning to the main program. Note that in the flowchart there is a branch decision and a branch decision with a jump back or a loop. The program runs in this loop until it comes to a condition where it can get out of the loop or "exit from the loop."

Subroutine Jump

Special things happen when a subroutine such as WAIT is encountered in the main program. *Figure 3-28* diagrams the steps. The main program has executed from *Step 1* to *Step 5.* At *Step 6,* the computer encounters the instruction telling it to branch to subroutine A and do subroutine A. Therefore, in order to return to the correct location in the main program after executing the subroutine, the branch instruction at *Step 6* also tells the computer to remember the address of *Step 7.*

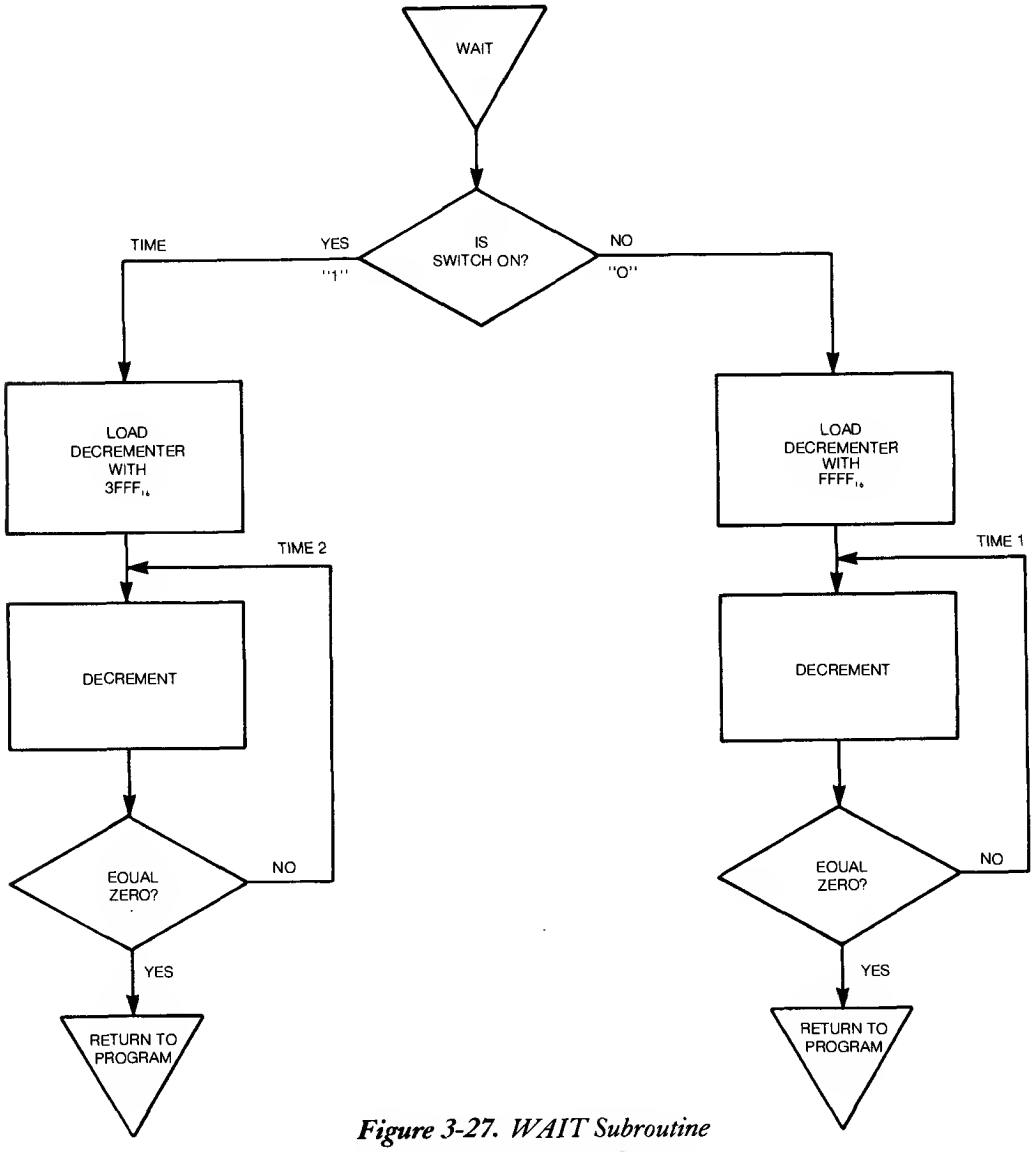


Figure 3-27. WAIT Subroutine

The subroutine is executed through *Step A-8.* Whereupon the computer encounters an instruction at *Step A-9* that tells it to return to the *Step 7* address which it remembered at *Step 6.* In this fashion, each subroutine can be executed and program control returned to the main program. Of course, there are branches that can occur from a subroutine to another subroutine but the principle is the same.

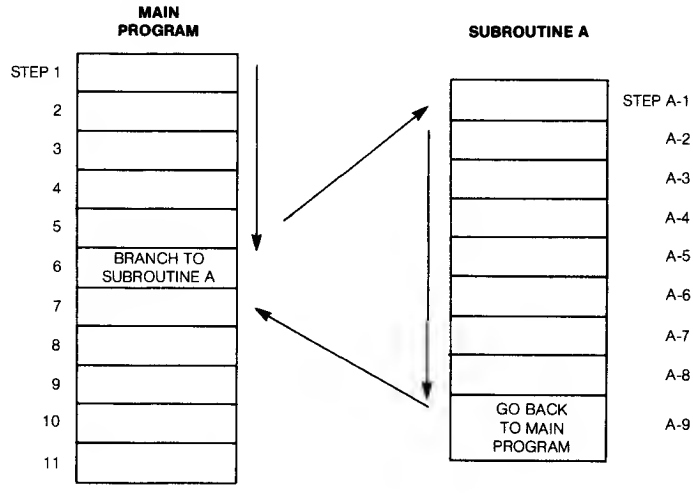


Figure 3-28. Branch to Subroutine

The instruction from the TM990/100M microcomputer instruction set that accomplishes the branch to a subroutine is called *Branch* and *Link.* This is called a "subroutine jump" instruction and will be identified by the letters BL and some additional information that tells the location of the address of the first instruction of the subroutine. In addition, recall that a register file is to be set up for general registers. Well, register 11 of this file (WR 11) is the storage place used to remember the main program address that is returned to after executing the subroutine.

The return instruction from the subroutine used is called an unconditional branch instruction. It is identified by *Branch.* Since the contents of register 11 must be returned to the program counter to return from a subroutine, this instruction will be identified as B*11. Note that the file register 11 must be reserved for this use by the programmer, otherwise its contents are likely to be changed at the wrong time and the computer misled into a wrong sequence.

A Loop Within the WAIT Subroutine

Within the WAIT subroutine is another common reoccurring concept — a loop. However, before examining this program sequence further, it would be beneficial to clearly understand the meaning of the blocks in the flow charts. The general meaning of the most commonly used blocks is shown in *Figure 3-29.* There is a symbol for the entry to or exit from a program (or for an off-page connection). This is identified with an appropriate symbol or label — START and STOP in this example. Rectangles identify operations. Inside the rectangle is an appropriate abbreviated statement to describe the operation. Decisions are identified with a diamond. Since programmed logic occurs in sequence, these blocks are relatively simple. A two-state decision answers a question of yes or no, true or false, etc. A three-state decision answers a comparison question of greater than, equal to, or less than (of course, there could be further mixtures of these). So decision blocks have appropriate questions identifying them.

In the WAIT subroutine of *Figure 3-27,* the first decision is "Is the switch ON?," and the consequences have already been discussed. The second decision has the question "The quantity examined — is it equal to zero?" Within this program sequence, if the quantity is not equal to zero, then the program goes through the same path again.

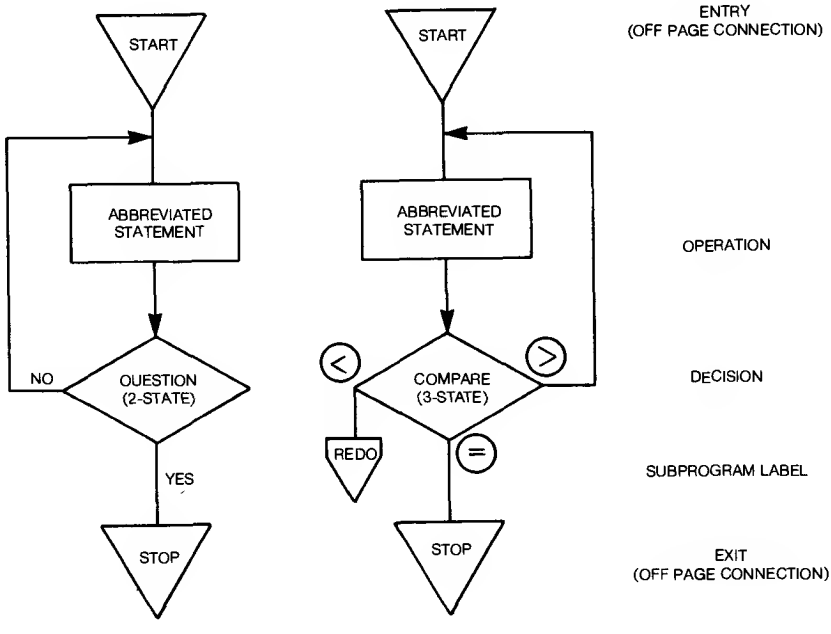


Figure 3-29. Common Flow Chart Blocks

The program loop is accomplished by a branch instruction from the instruction set called a conditional jump instruction. The conditional jump causes the microcomputer program to branch to a specified program step depending on the condition of certain bits in the status register. Recall in *Figure 3-19* that the status bits were identified and that the "equals bit" — bit 2 — was going to be used to change the time delay sequence. Therefore, the decision block in the program is really asking, "Is bit 2 of the status register set to a "1"?"

The status bit 2 is set to 1 by the program step before the decision block in *Figure 3-27* — the decrement step. An instruction *Decrement (by One)* causes a named file register to have one subtracted from its contents, comparison of the result to zero and the setting of the appropriate status bits (0-4) of the status register. When the register contents are equal to zero, the "equals" status bit (2) will be set to a "1".

▶3

When the status bit 2 is not set to a "1", the program must return to the decrement instruction and subtract one again from the register. JNE (label) is the conditional jump instruction that will be used to accomplish the loop. It is activated by the "equals bit" being "0". The program will jump to a point ahead of the decrement step which will be identified with an appropriate label. In the program this label must be included with the JNE *(Jump if not equal)* instruction.

A similar type of conditional jump instruction is used to answer the question of the switch in the first decision block of the WAIT subroutine. However, in this case, *Jump if Equal* (JEQ (label)), with the appropriate label will be used. Now the conditional jump will occur if the equal bit is set to a "1". Recall, this is the type instruction previously referred to that must follow the TB instruction so that the status bit can be examined and a decision made.

The number of steps in the decrement block is now the only remaining portion of the subroutine which has not been discussed.

LOADING A REGISTER FOR THE TIME DELAY

Assume that the switch is "ON" in the WAIT routine. A logical "1" is the input to the microcomputer. The TB instruction identifies the logical "1" and it sets the equals bit 2 of the status register to a "1" as previously described. The JEQ instruction jumps to a selected (labeled) instruction which loads a selected file register with a number, $3FFF_{16}$. As a 16 bit binary number, it is 0011 1111 1111 1111. No jump occurs in the program if the switch is inputting a logical "0". The program just proceeds to the next step.

Well, how does the data get loaded into the selected file register? Simply enough with a load instruction which is one of the data transfer instructions. *Load Immediate* (file register number), $3FFF_{16}$ will tell the microcomputer to load the hexadecimal number $3FFF_{16}$ into the selected register. What actually happens is that two memory words must be used for this instruction. The first word provides the operation code and register number and the second word the operand or data to be operated on. For the addressing mode used for the *Load Immediate* instruction, the word following the instruction LI 3, will contain the data to be put into register 3, $3FFF_{16}$. The programmer must remember that a memory word location (PC + 2) is used for the $3FFF_{16}$ data when the instruction is located at PC.

Following on then, new data is placed into the same register by a new *Load Immediate* instruction. For example, for a longer time delay, the file register R3 is loaded with $FFFF_{16}$. The instruction LI 3, $FFFF_{16}$ accomplishes this.

WHERE DOES THE PROGRAM START?

Most of the information is now in hand to write the program. The question is, "Where does the program start"? Recall that when the program was entered into the microcomputer through the microterminal, $FE00_{16}$ was chosen as the starting memory location. How was this decided?

The first step in the decision is to determine what words are available in memory — what addresses can be used.

Figure 3-30 is reproduced from the TM 990/100M Users Guide. There are address locations from 0000_{16} to $FFFE_{16}$ for 65,536 bytes (8-bit pieces), or 32,768 16-bit word locations. This is commonly called the address space. Word address locations move by an increment of 2, byte locations by 1. The incrementing of the program counter by 2 was previously noted. This is the reason.

Recall that the TM 990/100M microcomputer has 256 16-bit words of RAM into which the program is going to be placed and it also has 1024 16-bit words of ROM, or EPROM in this case. The EPROM is the TIBUG monitor that provides the necessary pre-programmed instructions that were referred to for accepting input and output data.

The 256 words of RAM occupy address space from $FE00_{16}$ to $FFFE_{16}$ as shown in *Figure 3-30.* The EPROM address space is from 0000_{16} through $07FE_{16}$ which is address space that is dedicated for this purpose and not available for change by the first encounter program. Notice that within this space are interrupt and XOP vectors. These are of no concern at this time.

Since not all the available memory sockets are filled, address space from 0800_{16} through $FDFE_{16}$ does not have memory cells — it is unpopulated.

It would seem that all the address spaces in RAM from $FE00_{16}$ to $FFFE_{16}$ are available. However, as shown in *Figure 3-30,* 40 words of RAM must be reserved for use by the TIBUG monitor and additional space is necessary for interrupts. Thus, the available space is from $FE00_{16}$ to $FF66_{16}$.

Obviously, some analysis of the possible length of the program in number of steps must be made, as well as some estimate of the number of file register blocks of 16 (workspaces) that will be used. This will determine whether adequate address space is available or whether additional memory space must be populated.

The first encounter assumptions are as follows:

▶3

1. The program will be less than 96 steps long — 96 words or 192 bytes.
2. Only one workspace will be required. (16 contiguous words)

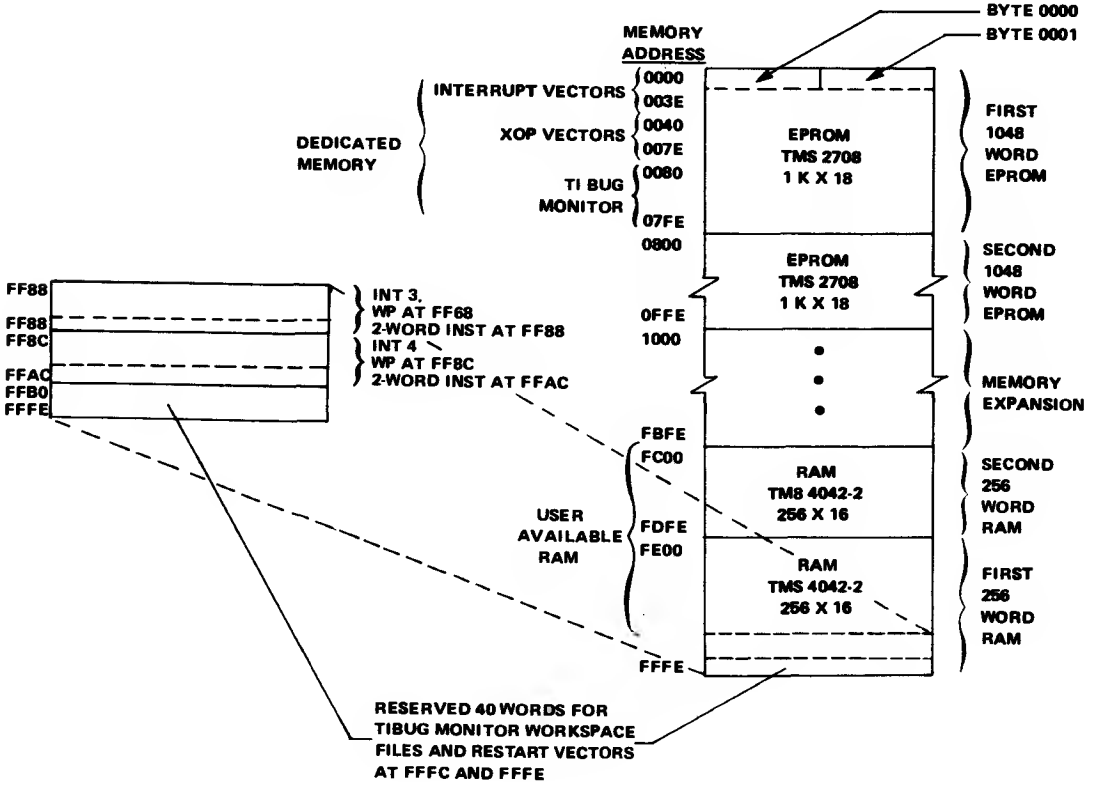


Figure 3-30. Memory Map

On this basis, the starting address of the program is chosen as $FE00_{16}$. The workspace file register could have been chosen to start at 16 words away from $FF66_{16}$. However, since there is plenty of space, it is placed at $FF20_{16}$, leaving the room from $FE00_{16}$ to $FF1E_{16}$ as the space for the program (144 words):

WRITING THE PROGRAM

Refer now to the flowchart in *Figure 3-26* as the basis for writing the program. To help in the organization of the program, a form shown in *Table 3-1* will be used. Note that it has a column for addresses, for machine code, for a label, for the assembly language statement and for comments. Each of these columns will be filled in as needed as the program is developed. Not all columns will have an entry when the program is complete. The machine code will be the last column completed. Of particular importance, especially for later references, or reference by another programmer, will be the comments column. Keep referring to *Table 3-1* after each program step to note the comments and see the program develop.

Figure 3-26 indicates that the first step in the program is to be an initializing statement. The location of the file register (workspace) used must be identified by loading the workspace pointer with the address $FF20_{16}$. The program must at all times know where the file registers are in memory for it will use these registers for obtaining data or addresses.

Reference to Chapter 5 and 6 shows there is a load instruction for the workspace pointer, LWPI, *Load Workspace Pointer Immediate.* Recall that the immediate addressing requires two words. Therefore, *Step 1* of the program at address $FE00_{16}$ is shown as:

Step	*A*	*MC*	*L*	*ASSY LANG.*
1	FE00			LWPI >FF20

and *Step 2* has the operand to be loaded. The greater than (>) sign identifies the data as hexadecimal.

The program must be able to branch to the subroutine WAIT when that routine is called by the program. Therefore, the starting address of the WAIT subroutine must be loaded into a file register which then will be referenced when the address is needed. *Step 3* of the program accomplishes this with a *Load Immediate* instruction and register 1 is chosen to hold the address. Note that the program address is incrementing by two. *Step 3* is:

Step	*A*	*MC*	*L*	*ASSY. LANG*
3	FE04			LI 1,>XXXX

Note that the specific address cannot be put in at this time — not until the location is known. *Step 4* is the step for loading the operand.

Recall that a reference needs to be established for the particular 9901 I/O interface unit to be used by the microcomputer. This was referred to as the CRU base address for the chosen 9901. Register 12 of the file register is the one that must contain the CRU base address, therefore, it must be loaded with 0120_{16}, the software base address of the 9901 in the TM990/100M microcomputer. Step 5 of the program is for this purpose.

Step	*A*	*MC*	*L*	*Assy. Lang.*
5	FE08			LI 12, >0120

▶3

Again *Step 6* must be added because of the immediate addressing.

All initial conditions are now complete and the flowchart now moves to the start of the light sequence. Light #1 must be turned on. Recall from *Figure 3-25* that light #1 is connected to I/O output 0 (P_0). Therefore, I/O-0 on the 9901 must be set to a "1". This is accomplished with the SBO instruction of *Step 7.* Recall, this instruction was previously discussed in detail. *Step 7* looks like this:

Step	*A*	*MC*	*L*	*Assy. Lang.*
7	FE0C		BEGIN	SBO 0

Note that this instruction is labeled BEGIN. This is done because the program will jump back to this address location after the complete sequence of the first encounter task is completed. The label BEGIN provides an easy reference to this location.

WAIT SUBROUTINE CALL

The first encounter task as defined now requires the light #1 be held on for the time delay represented by the subroutine WAIT. Therefore, the program must be directed to the first address of the subroutine. This first address is contained in the file register 1 (workspace register 1) because *Step 3* and *Step 4* accomplished this.

Recall the discussion on the WAIT subroutine (*Figure 3-28*). The main program must be directed to the subroutine (the main program "calls" the subroutine) but it must also remember where it is in the main program so it can return to the correct location. The *Branch and Link* to register 1 of *Step 8* accomplishes this.

Step	*A*	*MC*	*L*	*Assy. Lang.*
8	FE0E			BL *1

At the same time the address of the next step in the program, *Step 9* is being saved in register 11.

However, note that there is a new symbol in the assembly language instruction. The asterisk (*) means that an indirect addressing mode is used. That means that file register 1 (WR1) does not contain operand information but contains the *address* of an operand to be used for further processing. That is exactly what has been put into register 1 — the address of the first instruction of the WAIT subroutine. Therefore, an indirect addressing mode is used.

Why is that important? When the machine code for an instruction is constructed a little later (this will be done by hand but normally it would be done by a computer under control of a program called an assembler), an identifying code for the addressing mode must be used in the format for each instruction.

Figure 3-31 shows how the 16 bits of the machine code are arranged for the various types of instructions. Much more discussion of these formats is contained in Chapters 5 and 6. For the purpose here, format 6 is the one of particular interest for the *Branch and Link* instruction. Note that for format 6 the first 10 bits are for the operation code, bits 10 and 11 are a T_s field, and bits 12 thru 15 are an S field for identifying the address of the source information. Note that the code for T_s defines the addressing mode for the instruction. 01 will be entered in this field for bits 10 and 11 for the *Branch and Link* instruction because this is the code for indirect addressing. 0001 will be the code for the S field because register 1 contains the source address.

RETURN FROM WAIT SUBROUTINE

The end of the subroutine will return the microcomputer to the main program at *Step 9* because this is the address saved in register 11. *Step 9,* according to the flowchart of *Figure 3-26*, must now turn light # 1 off. The instruction is:

Step	*A*	*MC*	*L*	*Assy. Lang.*
9	FE10			SBZ 0

Since I/O port 0 was set to a "1" in order to turn the light on, now it is set to a "0" to turn the light off.

Time delay subroutine WAIT is called for again for the next step and again the *Branch and Link* instruction is used. Thus, *Step 10 is:*

Step	*A*	*MC*	*L*	*Assy. Lang.*
10	FE12			BL *1

Upon return from the WAIT subroutine light #2 is turned on, the WAIT routine occurs, light #2 is turned off, the WAIT routine occurs and the process continues until light #4 is turned off and the time delay is complete. These steps are shown in *Table 3-1* and carry the program through *Step 22.*

The program will return to *Step 23* after the time delay. The flowchart indicates a return to the beginning of the sequence. Recall that this was labeled BEGIN. Therefore, *Step 23* is a jump instruction that jumps the program back to the address of the instruction labeled BEGIN. The assembly language instruction is simple enough:

▶3

Step	*A*	*MC*	*L*	*Assy. Lang.*
23	FE2C			JMP BEGIN

This instruction is called an unconditional jump instruction because there are no decisions involved — just the direction to "go to" a specified place. There is no return instruction address saved in register 11 and no testing of status bits.

All the program steps in the flowchart of *Figure 3-26* are now complete. What remains is to define the steps in the subroutine WAIT. *Figure 3-27* is used for this purpose.

WAIT SUBROUTINE

The address at *Step 24,* $FE2E_{16}$, is the one that must be loaded into register 1 at *Step 3* because it is the first instruction of the subroutine. The flowchart identifies this step as a decision block. Is the switch on for a logical "1" or is it off for a logical "0"? The input line must be tested to determine this. A TB instruction, examining I/O pin P_4, is used for this purpose. This instruction is *Step 24:*

Step	*A*	*MC*	*L*	*Assy. Lang.*
24	FE2E			TB 4

This is the *Test Bit* instruction discussed previously. Recall that when the input line is tested by the instruction it sets the "equals" bit, bit 2 of the status register to the value of the input.

In order to make the decision called for in the flowchart, an instruction that examines bit 2 of the status register must follow. This will be a conditional jump instruction because if the status bit is a "1", the time delay is to be the shortest and the sequence fast. Correspondingly, the sequence would be slow and the time delay long for a status bit 2 of "0". Chapters 5 and 6 identify the jump instructions. JEQ is the one selected which says that the program will jump to a new location if the "equals" bit is set to a "1", otherwise, the program will continue on to the next step. The instruction is:

Step	*A*	*MC*	*L*	*Assy. Lang.*
25	FE30			JEQ TIME

Convenient labels have been placed on the flowchart of *Figure 3-27*. The branch jumped to in *Step 25* is labeled TIME. This branch will be executed in a moment. For now, assume that the "equals" bit is set to "0" and the program continues. The next step is to load a register so that it can be decremented to produce the time delay. In this branch, this must be the largest value for the longest delay and the slowest sequence. Another file register must be selected. Register 3 is chosen and the load instruction is as follows:

Step	*A*	*MC*	*L*	*Assy. Lang.*
26	FE32			LI 3, >FFFF

3◀

This is the same as previous *Load Immediate* instructions and another word must be allowed for the value to be loaded. Thus, *Step 27* at FE34.

One must now be subtracted from the value. There is an instruction called *Decrement (by one)* and, of course, it must tell what value to be decremented. In this case, the contents of R3. Thus, *Step 28* is:

Step	*A*	*MC*	*L*	*Assy. Lang.*
28	FE3		TIME1	DEC 3

The flowchart shows the decrement as an operation. In addition, as mentioned previously, *the value in register 3 is compared to zero* and the greater than, equal, carry or overflow status bits are set accordingly. This is found in the discussion on the instructions in Chapter 5 and 6.

The decision that follows is made on the basis again of examining the "equals" bit. The flow chart shows that if the "equals" bit is not set, the program will loop back and be decremented again as previously discussed. Therefore, a label, TIME 1, is placed on the instruction at FE36 to tell the program the location of the jump.

The jump occurs this time if the "equals" bit is not set, using the instruction *Jump if Not Equal,* and looks like:

Step	*A*	*MC*	*L*	*Assy. Lang.*
29	FE38			JNE TIME1

When the file register has been decremented to zero, the equals bit will be set and the program is ready to return to the main program. Recall that register 11 contains the address (location) for the return. The branch instruction used for the return is Branch and *Step 30* is:

Step	*A*	*MC*	*L*	*Assy. Lang.*
30	FE3A			B *11

Note this again is an indirect addressing mode.

TIME BRANCH

▶3

The only remaining portion of the flowchart that must be programmed is the TIME branch.

In this branch, the time delay is shorter to make the sequence faster. R3, the same register, is loaded with a smaller value, $3FFF_{16}$. Again a *Load Immediate* instruction shown in *Step 31* is used.

Step	*A*	*MC*	*L*	*Assy. Lang.*
31	FE3C		TIME	LI3, >3FFF

This step is labeled with TIME, and will be the location jumped to from *Step 25. Step 32* is the extra word required.

The register must again be decremented, therefore, the instruction is the same type as *Step 28.* However, the label for the location to jump to is now TIME2. *Step 33* is:

Step	*A*	*MC*	*L*	*Assy. Lang.*
33	FE40		TIME2	DEC 3

The same jump instruction is used in this branch as for *Step 29* except the label is now TIME 2. Therefore, *Step 34* is:

Step	*A*	*MC*	*L*	*Assy. Lang.*
34	FE42			JNE TIME2

When the equals bit is set, the program must return to the main program as with the other branch. The same return instruction as *Step 30* is used, as shown in *Step 35.*

Step	*A*	*MC*	*L*	*Assy. Lang.*
35	FE44			B *11

The total program is now complete in assembly language. It is shown in *Table 3-1.*

TABLE 3-1
ASSEMBLY LANGUAGE PROGRAM

Table 3-1. Assembly Language Program.

(Source Code Statements)

Step	Hex Address	Hex Machine Code	Label	Op Code	Operand	Comments.
1.	FE00			LWPI	>FF20	Load workspace pointer
2.	FE02					with $FF20_{16}$
3.	FE04			LI	1, >FF2E	Load R1
4.	FE06					with 1st Address of WAIT
5.	FE08			LI	12, >0102	Load R12
6.	FE0A					with base address of 9901, 0120_{16}
7.	FE0C		BEGIN	SBO	0	Set I/O P_0 (segment f) equal to one
8.	FE0E			BL	*1	Branch to address in R1 (saves next address in R11)
9.	FE10			SBZ	0	Set I/O P_0 (segment f) equal to zero
10.	FE12			BL	*1	Branch to address in R1 (saves next address in R11)
11.	FE14			SBO	1	Set I/O P_1 (segment b) equal to one
12.	FE16			BL	*1	Branch to address in R1
13.	FE18			SBZ	1	Set I/O P_1 equal to zero
14.	FE1A			BL	*1	Branch to address in R1
15.	FE1C			SBO	2	Set I/O P_2 (segment e) equal to one
16.	FE1E			BL	*1	Branch to address in R1
17.	FE20			SBZ	2	Set I/O P_2 equal to zero
18.	FE22			BL	*1	Branch to address in R1
19.	FE24			SBO	3	Set I/O P_3 (segment c) equal to one
20.	FE26			BL	*1	Branch to address in R1
21.	FE28			SBZ	3	Set I/O P_3 to equal to zero
22.	FE2A			BL	*1	Branch to address in R1
23.	FE2C			JMP	BEGIN	Jump to BEGIN
24.	FE2E		WAIT	TB	4	Test I/O P_4 for a "1" or a "0"
25.	FE30			JEQ	TIME	If equals bit is set ("1"), jump to TIME
26.	FE32			LI	3, >FFFF	Load R3
27.	FE34					with $FFFF_{16}$
28.	FE36		TIME1	DEC	3	Decrement R3
29.	FE38			JNE	TIME1	Jump to TIME 1 if equals bit is not set
30.	FE3A			B	*11	Return to main program (by way of R11)
31.	FE3C		TIME	LI	3, >3FFF	Load R3
32.	FE3E					with $3FFF_{16}$
33.	FE40		TIME2	DEC	3	Decrement R3
34.	FE42			JNE	TIME2	Jump to TIME 2 if equals bit is not set
35.	FE44			B	*11	Return to main program (by way of R11)

WRITING THE MACHINE CODE

Normally the next step in programming (shown in *Table 3-2*) would be done by a computer as mentioned previously. However, in order to demonstrate what an Assembler Program would do and because the program input to the TM990/100M microcomputer is through the microterminal, which requires the machine code, it will be a good exercise to demonstrate how to develop the machine code. If this is of no interest, this portion of the discussion can be bypassed and a jump made to the summary.

As mentioned previously in *Figure 3-31,* there is a set format for the 16 bits of machine code that must be generated for each instruction. The formats used for the first encounter task are shown in *Figure 3-32* for reference. Each instruction has an operation code (OP CODE) and then additional information is required in the various fields of the format. A complete discussion of the format for each instruction can be found in Chapter 6. *Figure 3-33* lists the instructions used in the first encounter.

The same programming form will be used as before which is summarized to this point in *Table 3-1.* The machine code will be filled in and several other changes made and the result will be the final program of *Table 3-2.* As before, continue to refer to *Table 3-2* as the machine code is developed.

Immediate Instructions

The coding begins at *Step 1.* LWPI is an immediate instruction. Therefore, the format 8 of *Figure 3-32* is used. There are two words to this instruction; the second one containing the immediate value to be loaded. In the first word, the op code occupies bits 0 through 10; register numbers, where the immediate value is going to be placed, occupy bits 12 thru 15. Bit 11 is not used. The op code is obtained from *Figure 3-33* for the LWPI instruction. The filled out instruction would look like this.

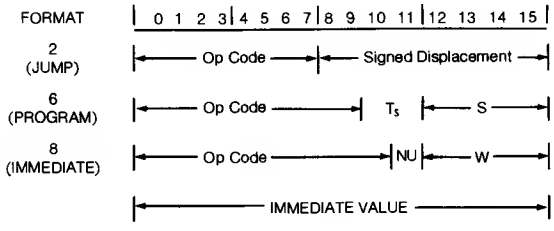
	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Binary —	0 0 0 0	0 0 1 0	1 1 1 0	0 0 0 0
Op Code —	0	2	E	0
Machine Code —	0	2	E	0

LWPI is a special case of format 8. Bits 11-15 are not used and as such could contain anything. They are don't care conditions. Therefore, the machine code is 02E0. This is entered into *Table 3-2* on the same line as LWPI as *Step 1. Step 2* is the immediate value FF20, therefore, the machine code is $FF20_{16}$.

FORMAT (USE)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1 (ARITH)	OP CODE			B	T_D		D				T_S		S			
2 (JUMP)	OP CODE								SIGNED DISPLACEMENT*							
3 (LOGICAL)	OP CODE						D				T_S		S			
4 (CRU)	OP CODE						C				T_S		S			
5 (SHIFT)	OP CODE								C				W			
6 (PROGRAM)	OP CODE										T_S		S			
7 (CONTROL)	OP CODE												NOT USED			
8 (IMMEDIATE)	OP CODE											NU	W			
	IMMEDIATE VALUE															
9 (MPY, DIV,XOP)	OP CODE						D				T_S		S			

KEY
B = BYTE INDICATOR
(1 = BYTE, 0 = WORD)
T_D = D ADDR. MODIFICATION
D = DESTINATION ADDR.
T_S = S ADDR. MODIFICATION
S = SOURCE ADDR.
C = XFR OR SHIFT LENGTH (COUNT)
W = WORKSPACE REGISTER NO.
* = SIGNED DISPLACEMENT OF − 128 TO + 127 WORDS
NU = NOT USED

Figure 3-31. Instruction Formats



NOTES:

T_S = SOURCE ADDRESS MODIFICATION
S = SOURCE ADDRESS
W = WORKSPACE (FILE) REGISTER NO.
NU = NOT USED

SIGNED DISPLACEMENT CAN BE
− 128 TO + 127 WORDS

CODES FOR T_S FIELD	ADDRESSING MODE
00	REGISTER
01	INDIRECT
10	INDEXED (S OR D ≠ 0)
10	SYMBOLIC (DIRECT, S OR D = 0)
11	INDIRECT WITH AUTO INCREMENT

Figure 3-32. Formats used for First Encounter

In like fashion, the instructions at *Step 3* and *Step 5* are immediate instructions, use the same format, and are coded with the appropriate register numbers. *Step 4* and *Step 6* are the immediate values to be loaded.

Note, however, that when the program was first prepared, the first address of the WAIT subroutine was not known. Now, it is known. It is substituted for the XXXX in *Table 3-1* at Step 3. Thus, the address of *Step 24,* FE2E is placed after the "greater than" symbol.

The op code for LI is 0200_{16} and since register 1 is used for *Step 3,* the machine code is 0201_{16} while for *Step 5* it is 020C because register 12 is being loaded. The machine code
▶3 for *Step 4* is the value $FE2E_{16}$ and for *Step 6* it is 0120_{16}.

INSTRUCTIONS SBO, SBZ

The instruction SBO at *Step 7* uses a different format. This is format 2 in *Figure 3-32.* It has the op code in bits 0 through 7 and the signed displacement that was discussed previously when the 9901 I/O unit program was examined. Recall that the CRU base address was arranged so that the bit number is the value that is put in for the signed displacement.

The op code for SBO from *Figure 3-33* is $1D00_{16}$ and with the first bit being zero, the machine code is:

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Binary —	0 0 0 1	1 1 0 1	0 0 0 0	0 0 0 0
Op Code —	1	D	0	0
Machine Code —	1	D	0	0

MNEMONIC	HEX OP CODE	FORMAT	RESULT COMPARE TO ZERO	INSTRUCTION
LWPI	02E0	8	N	LOAD IMMEDIATE TO WORKSPACE POINTER
LI	0200	8	N	LOAD IMMEDIATE
BL	0680	6	N	BRANCH AND LINK (WR11)
B	0440	6	N	BRANCH
DEC	0600	6	Y	DECREMENT (BY ONE)
SBO	1D00	2	N	SET CRU BIT TO ONE
SBZ	1E00	2	N	SET CRU BIT TO ZERO
TB	1F00	2	N	TEST CRU BIT
JEO	1300	2	N	JUMP EOUAL (ST2 = 1)
JMP	1000	2	N	JUMP UNCONDITIONAL
JNE	1600	2	N	JUMP NOT EOUAL (ST2 = 0)

Figure 3-33. Instructions used for First Encounter.

The other SBO instructions can be machine coded accordingly using the appropriate bit number. Therefore, *Step 11* is $1D01_{16}$, *Step 15* is $1D02_{16}$ and *Step 19* is $1D03_{16}$.

Similarly, using the op code of $1E00_{16}$ for the SBZ instructions and the appropriate bit number, *Step 9* is $1E00_{16}$, *Step 13* is $1E01_{16}$, *Step 17* is $1E02_{16}$, and *Step 21* is $1E03_{16}$.

Instruction BL

Now *Step 8* brings in another new format. For the BL instruction, it is format 6. Bits 0 thru 9 contain the op code. Bits 12 through 15 are the address of the source data. Ts is a field that modifies the source address and it contains the two bits that code the addressing mode that is being used. Recall BL *1 uses indirect addressing. Therefore, from *Figure 3-32* Ts would be 01 for these 2 bits. It's important to remember that this modifies the op code into a different number for the machine code as shown below.

		0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Op Code	—	0	6	8	0
Binary	—	0 0 0 0	0 1 1 0	1 0 0 0	0 0 0 0
T_S	—			0 1	
S	—				0 0 0 1
Machine Code—(Binary)	—	0 0 0 0	0 1 1 0	1 0 0 1	0 0 0 1
Machine Code—(Hex)	—	0	6	9	1

Thus, the machine code is 0691_{16} and can be placed in *Step 8, 10, 12, 14, 16, 18, 20* and *22,* since register 1 is used in each case.

Miscellaneous Instructions

Because the jump instructions fall into a class that needs special discussion, the remaining instructions will be coded first.

Step 26 and *Step 31* are LI instructions like Step 3 and Step 5 — the code is 0203_{16} in this case because register 3 is being used. Don't forget the values of $FFFF_{16}$ for *Step 27* and $3FFF_{16}$ for *Step 32.*

The TB instruction has an op code of $1F00_{16}$ and a format 2. It is just like the SBO and SBZ so that the bit must be used for the displacement. Bit 4 causes a displacement of 4, therefore, the machine code is $1F04_{16}$. This is *Step 24.*

A branch instruction similar to BL, but does not save the next address in register 11, is the instruction B. It is using the contents of register 11 for a return to the main program. The op code for B is 0440_{16}. It uses an indirect addressing mode so $T_S = 01$ and S is 1011 for register 11. The machine code results as follows:

		0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Op Code	—	0	4	4	0
		0 0 0 0	0 1 0 0	0 1 0 0	0 0 0 0
T_S	—			0 1	
S	—				1 0 1 1
Machine Code (Binary)	—	0 0 0 0	0 1 0 0	0 1 0 1	1 0 1 1
Machine Code (Hex)	—	0	4	5	B

▶3

It is entered at *Step 30* and *35*.

The only remaining instruction besides the jump instructions is the decrement instruction DEC. From *Figure 3-33* the op code is 0600_{16} and the format is 6. Register 3 is being used, therefore, S is 0011. The addressing mode is a register mode so T_S is 00 and there is no modification of the op code. The machine code is then 0603_{16} for *Step 28* and *33*.

Jump Instructions

Jump instructions use format 2 of *Figure 3-32* which has an op code for bits 0 through 7 and a signed displacement in bits 8 through 15. The signed displacement means the number of program addresses that the program must move to arrive at the required address. For example, let

A_J = present address of jump instruction
A_D = destination address of jump instruction

then,

1.) $A_J + 2\ DISP = A_D$

since the program moves by increments of 2.

However, for the 9900 microprocessor in the TM990/100M microcomputer, the jump instruction signed displacement must be calculated from the address following the address of the jump instruction or $A_J + 2$. Therefore, equation (1) becomes,

2.) $(A_J + 2) + 2\ DISP = A_D$

Solving for DISP, gives

3.) $\frac{A_D - (A_J + 2)}{2} = DISP$

Recall that in preparing the program of *Table 3-1* labels were used for instructions so that easy reference could be made to the desired destination address for a jump instruction. *Step 23* at $FE2C_{16}$ is the first jump instruction. The destination is the label BEGIN which is located at address $FE0C_{16}$. Applying equation (3) gives (in Hex)

4.) $$DISP = \frac{FE0C - (FE2C + 2)}{2}$$

5.) $$DISP = \frac{FE0C - FE2E}{2}$$

Now,

$$\begin{array}{r} FE0C \\ -\underline{FE2E} \\ -0022_{16} \end{array}$$

Therefore,

6.) $$DISP = \frac{-22}{2} = -11_{16}$$

This means that in the jump instruction the program moves back 11_{16} steps or 17 decimal steps.

Now, since this is a negative number, a two's complement must be used for the code, thus

	−0011
COMPLEMENT	FFEE
ADD ONE	+0001
2'S COMPLEMENT	FFEF

Now, only the 8 least significant bits are used along with the op code of *Figure 3-33.* JMP of *Step 23* has an op code of 1000_{16}. Therefore, the machine code is:

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Op Code —	1	0	0	0
Displacement —			E	F
Machine Code —	1	0	E	F

This machine code is entered at *Step 23.*

Step 25 has a JEQ instruction. A_J is $FE30_{16}$. The instruction says to jump to TIME which has an address of FE3C at *Step 31,* therefore, A_D = FE3C. Applying equation **(3)** gives, again in hexadecimal;

$$\text{DISP} = \frac{\text{FE3C} - \text{FE32}}{2} = \frac{10}{2} = 5_{16}$$

▶3

JEQ has an op code of 1300 and the machine code then becomes:

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Op Code —	1	3	0	0
Displacement —			0	5
Machine Code—	1	3	0	5

Step 25 then has 1305 as the machine code.

The remaining jump instructions, JNE at *Steps 29* and *34* have an op code of 1600_{16}. Calculating the displacement from *Step 29* to *Step 28* and from *Step 34* to *Step 33,* obviously is -02_{16}. The complement of -02 is FFFD and the twos complement is FFFE. Thus the machine code is:

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Op Code —	1	6	0	0
Displacement —			F	E
Machine Code—	1	6	F	E

Even though the labels jumped to for *Steps 29* and *34* are different, the displacement is the same and, therefore, the machine code entered at these steps is the same, $16FE_{16}$.

TABLE 3-2

Every step is now coded and the program is complete. This is the program that was entered into the microcomputer via the terminal to accomplish the first encounter task.

Only one comment remains. If the *Table 3-1* program were run on a computer under the direction of an assembler program, certain symbols used for directives to the assembler would have to be used. The $ symbol could have been used to indicate the fact that a displacement is to be made from the jump instruction address which was identified in equation **(3)** as A_J. The instruction then would contain the $ symbol followed by the necessary displacement in hexadecimal. For this reason the instructions at *Step 23, 25, 29,* and *34* would have looked as follows:

Step	*A*	*MC*	*L*	*Assy. Lang.*
23	FE2C	10EF		JMP $–17
25	FE30	1305		JEQ $+5
29	FE38	16FE		JNE $–2
34	FE42	16FE		JNE $–2

SUMMARY

It has been a long discussion. However, a great deal of material has been covered and many basic concepts developed. The facts and procedures presented should provide a solid foundation for expanding an understanding of the 9900 Family of microprocessors and microcomputer component peripherals and the microcomputers which use it. Hopefully, enough examples have been presented with the first encounter task that with a minimum of effort, new real applications of the TM990/100M board can be implemented. A few simple ones that can be implemented immediately with the present setup would be:

A. Wire-up the necessary drivers and resistors to drive all seven-segments of the display and write a new program to make the numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 come up in sequence.

B. Write a program that uses the 7 segment display numbers so that they spell a word when read up-side down.

Maybe more memory will be required, but that is easy to add to the TM990/100M.

The next step is to implement the logic levels at the output pins into real applications of controlling dc and ac voltages for control applications. An extended application in Chapter 9 using this same TM990/100M board setup shows how this can be done. Persons interested can follow right into this application example to gain more insight into the details of the 9900 family of components explained in detail in the following chapters.

TABLE 3-2: ASSEMBLY LANGUAGE PROGRAM

Table 3-2. Assembly Language Program.

(With Machine Code)

Step	Hex Address	Hex Machine Code	Label	Op Code	Operand	Comments.
1.	FE00	02E0		LWPI	>FF20	Load workspace pointer
2.	FE02	FF20				with $FF20_{16}$
3.	FE04	0201		LI	1, >FE2E	Load R1
4.	FE06	FE2E				with 1st address of WAIT
5.	FE08	020C		LI	12, >0102	Load 12
6.	FE0A	0120				Will base address of 9901, 0120_{16}
▶3 7.	FE0C	1D00	BEGIN	SBO	0	Set I/O P_0 (segment f) equal to one
8.	FE0E	0691		BL	*1	Branch to address in R1, (saves next address in R11)
9.	FE10	1E00		SBZ	0	Set I/O P_0 (segment f) equal to zero
10.	FE12	0691		BL	*1	Branch to address in R1 (saves next address in R11)
11.	FE14	1D01		SBO	1	Set I/O P_1 (segment b) equal to one
12.	FE16	0691		BL	*1	Branch to address in R1
13.	FE18	1E01		SBZ	1	Set I/O P_1 equal to zero
14.	FE1A	0691		BL	*1	Branch to address in R1
15.	FE1C	1D02		SBO	2	Set I/O P_2 (segment e) to one
16.	FE1E	0691		BL	*1	Branch to address in R1
17.	FE20	1E02		SBZ	2	Set I/O P_2 equal to zero
18.	FE22	0691		BL	*1	Branch to address in R1
19	FE24	1D03		SBO	3	Set I/O P_3 (segment c) equal to one
20.	FE26	0691		BL	*1	Branch to address in R1
21.	FE28	1E03		SBZ	3	Set I/O P_3 equal to zero
22.	FE2A	0691		BL	*1	Branch to address in R1
23.	FE2C	10EF		JMP	BEGIN	Jump to BEGIN
24.	FE2E	1F04	WAIT	TB	4	Test I/O P_4 for a "1" or a "0"
25.	FE30	1305		JEQ	TIME	If equals bit is set ("1"), jump to TIME
26.	FE32	0203		LI	3, >FFFF	Load R3
27.	FE34	FFFF				with $FFFF_{16}$
28.	FE36	0603	TIME1	DEC	3	Decrement R3
29.	FE38	16FE		JNE	TIME1	Jump to TIME1 if equals bit is not set
30.	FE3A	045B		B	*11	Return to main program (by way of 11)
31.	FE3C	0203	TIME	LI	3, >3FFF	Load R3
32.	FE3E	3FFF				with $3FFF_{16}$
33.	FE40	0603	TIME2	DEC	3	Decrement R3
34.	FE42	16FE		JNE	TIME2	Jump to TIME2 if equals bit is not set
35.	FE44	045B		B	*11	Return to main program (by way of R11)

CHAPTER 4

Hardware Design: Architecture and Interfacing Techniques

4

INTRODUCTION

Describing the 9900 system from a hardware standpoint clearly requires detailed descriptions of a large number of design features as well as the interaction between the 9900 and peripheral circuits. In this chapter, material is arranged to develop a 9900 system from the viewpoint of the 9900 microprocessor chip. In the architecture section, the concepts of instruction fetch and decode, the memory-to-microprocessor bus structures, and memory partitioning (the use of volatile and non-volatile memories) are explained. Other topics include descriptions of the registers on the microprocessor chip and the working registers, the concept of memory-to-memory architecture, timing and descriptions of interface signals.

A special section covers memory in detail, especially the controls and timing, multichip memory structure, static and dynamic RAM, and DMA (direct memory access).

Following the architecture and memory sections are sections devoted to the instruction set, design considerations for input/output techniques especially in CRU development, the interrupt structure and electrical requirements.

A special section devoted to the unique features of the single chip microcomputer, the TMS9940, is included at the end of the chapter.

Information in this chapter flows from the most basic fundamentals to an understanding of the more complex design features of the 9900 and the chip family. When very specific and detailed information regarding pin assignments and speed is given, the TMS9900 device specifications are used. These examples will give direction and illustration for interpreting the data sheet information found in Chapter 8.

The 9900 family of 16-bit microprocessors includes several device types each aimed at a specific market segment. The same basic architecture and instruction set are maintained throughout. Consider first the single-chip microprocessor which consists of an ALU (arithmetic and logic unit), a few registers, and instruction handling circuitry (*Figure 4-1*). There is no memory on the chip for instructions and data so it must be interfaced to memory devices, usually RAM for data (and instructions which must be modified) and ROM, PROM, or EPROM for instructions (*Figure 4-2*). It is often desirable to store instructions in a non-volatile memory to eliminate the requirement for loading the program into memory immediately following application of power. This is especially important in dedicated applications where the program is fixed and power off-on cycles are common occurrences.

The microprocessor is connected to memory devices and external input/output (I/O) devices via sets of signals or busses (*Figure 4-2*). An address bus selects a word of memory. The contents of this word will be transferred to or from the microprocessor via the data bus. Control signals required to effect the transfer of information between the microprocessor and the memory are grouped into a control bus.

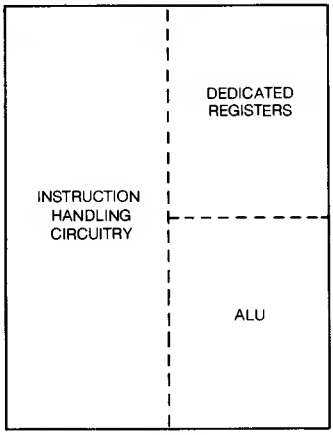


Figure 4-1. The 9900 Microprocessor

4◀

The interface to external devices (I/O) may be accomplished by using the address, data and control busses. This technique is known as parallel I/O or memory mapped I/O because data is transferred in parallel and the I/O devices occupy locations in the memory address space.

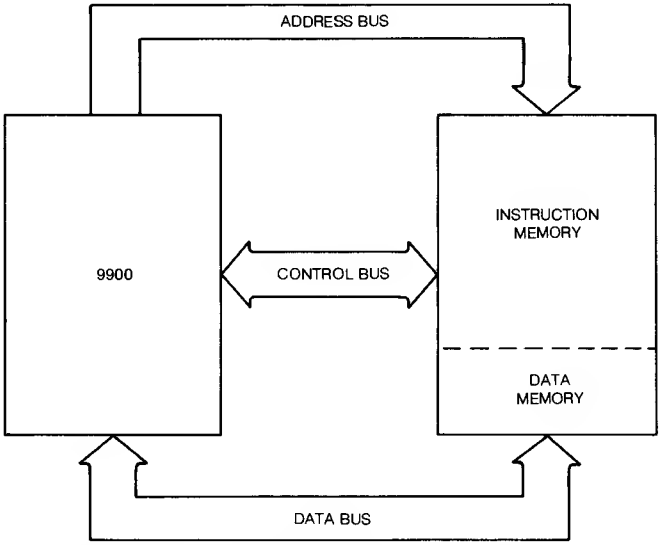


Figure 4-2. 9900 Microprocessor and Memory

The extension of parallel I/O is direct memory access (DMA). External hardware is employed to act as a separate special purpose processor for transferring large blocks of contiguous memory words to or from an external device (such as a disc memory). Once such a transfer is set up (via a string of instructions in the program), the DMA controller automatically synchronizes the transfer of data between the external device and memory, sharing the buses timewise with the microprocessor.

The 9900 architecture includes one other important I/O technique. Designed primarily for single bit I/O transfers, the communications register unit (CRU) provides a powerful alternative to parallel, memory mapped I/O (*Figure 4-3*). The address bus is used to select one of 4096 individual input or output bits in the CRU address space. During the execution of one of the single bit CRU instructions, the processor transfers one bit in or out. Multiple bit instructions are also available which provide for transfer of up to sixteen bits via a single CRU operation.

▶4 While this chapter describes primarily the basic TMS9900 16-bit microprocessor, all of the 9900 family CPU's are covered in detail in the *Product Data* chapter.

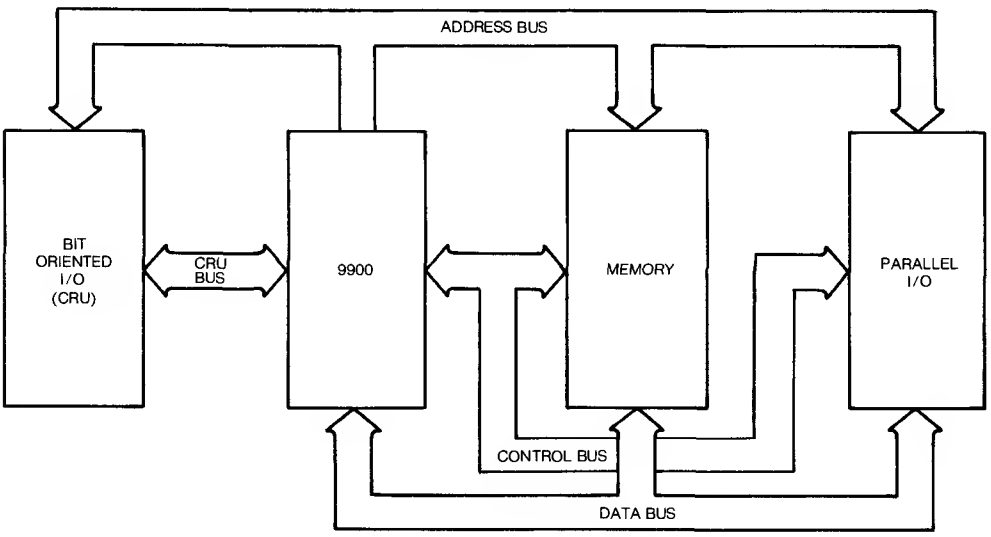


Figure 4-3. *9900 Bus Architecture*

An overview is given here to establish design paths for microprocessor systems. Listed below are the processors in the 9900 family.

Device	*Technology*	*Description*
TMS 9900	N-MOS	16-bit CPU 3 MHz
TMS 9900-40	N-MOS	16-bit CPU 4 MHz
SBP 9900A	I^2L	16-bit CPU −55° to 125°C
TMS 9980A/81	N-MOS	16-bit CPU 40-pin package
TMS 9985	N-MOS	16-bit CPU 40-pin package
TMS 9940	N-MOS	16-bit CPU with 2 k on-chip ROM

General purpose applications are designed around the TMS9900 device. The same is true for systems with severe environmental specs; however, a transition to the SBP9900A is made after the design is complete and the software completely debugged. The TMS9980A/81 and the TMS9985 are used where the 40-pin package is advantageous and a slightly slower speed is acceptable. The TMS9940 is a single-chip microcomputer for small special purpose controllers.

At the end of this chapter and in the *Product Data* chapter there is detailed design data for application of the LSI (large scale integration) peripheral support circuits in the 9900 family which are available for use in 9900 microprocessor-based systems. But in order to read and understand the data presented in this chapter and in this book, an understanding of the basic fundamentals of microprocessors is needed.

ARCHITECTURE

Basic Microprocessor Chip

The 9900 is an advanced 16-bit LSI microprocessor with minicomputer-like architecture and instructions. It is easy to understand and easy to use. Consider first the microprocessor device itself (*Figure 4-4*). Operations are carried out with a set of dedicated registers, an ALU, and instruction handling circuits. As clock signals are applied, the processor will fetch an instruction word from a memory (external to the chip), will execute it, fetch another instruction, execute it and so on. In each case the instruction is saved in an instruction register (IR) on the chip. The decode circuit sets up the appropriate controls based on the content of the instruction register for a multi-step execution phase. A memory address register (MAR) is used to hold address information on the address bus. The ALU and the other registers perform their specified functions during the execute phase of the instruction cycle.

Microprocessor Registers

There are three registers on the 9900 chip which are the key architectural features of the microprocessor (*Figure 4-5*). They are the workspace pointer (WP), the program counter (PC), and the status register (ST).

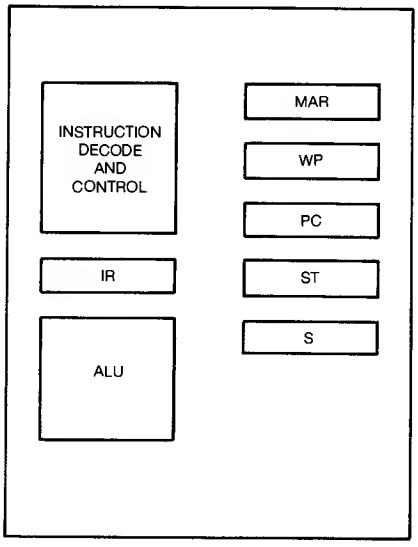


Figure 4-4. 9900 Functional Elements

▶4

Workspace Pointer

The general purpose registers for the 9900 are implemented as blocks of memory called workspaces. A workspace consists of 16 contiguous words of memory, but are general registers to the user. The workspace pointer on the 9900 chip holds the address of the first word in the workspace. After initializing the content of the WP at the beginning of a program (or subprogram), the programmer may concentrate on writing a program using the registers to hold data words or to address data elsewhere in memory.

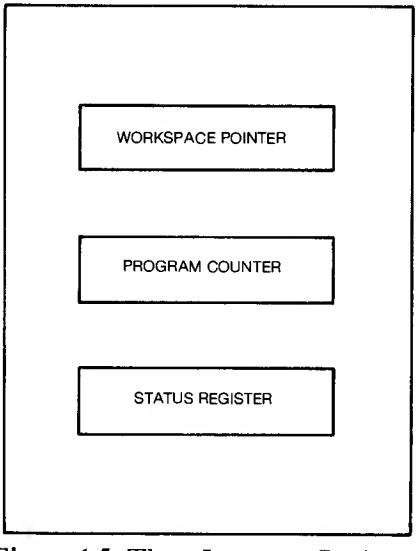


Figure 4-5. Three Important Registers

Program Counter

The program counter (PC) in the 9900 is used in the conventional way to locate the next instruction to be executed. As each instruction is executed, the program counter is incremented to the next consecutive word address. Because word addresses are even numbers in the 9900, the program counter is incremented by two in order to address sequential instructions. If the instruction to be executed occupies two or three memory words, the program counter will be incremented to generate sequential (even) addresses to access the required number of words. At the end of execution the PC is incremented to the next even address which is the location of the next instruction. If the instruction to be executed is a jump or branch instruction, the program counter is loaded with a new address and program execution continues starting with the instruction at that location in memory.

Figure 4-6 shows the program counter pointing to (addressing) instruction words in the program. Starting with location (x) the instructions are performed in sequence until a jump is encountered at (y). Processing resumes sequentially starting at location (z) which was the address specified by the jump instruction to be placed in the program counter.

Status Register

The status register (ST) is the basis for decision making during program execution. Individual bits of the ST are set as flags as the result of instructions. They may thereafter be tested in the execution of conditional jump instructions. *Figure 4-7* shows the status register and its flag bits.

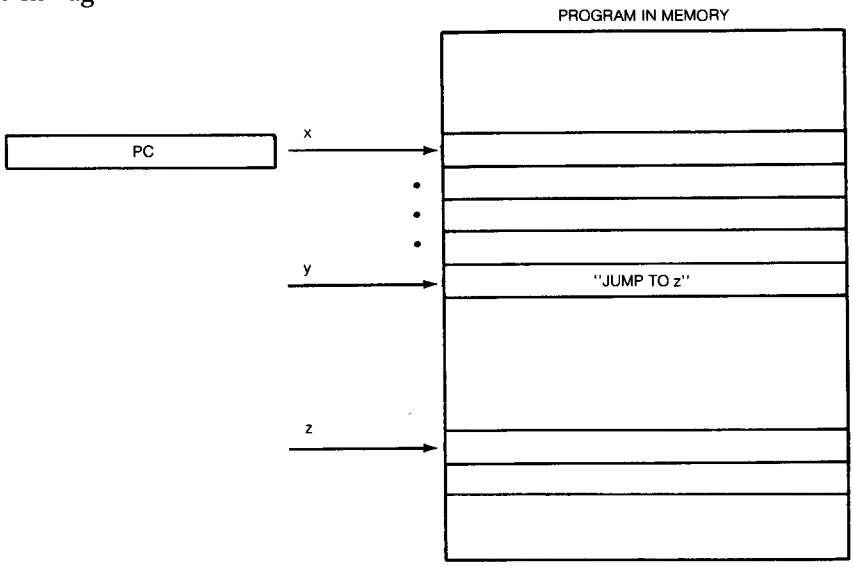


***Figure 4-6.** Program Counter Operation*

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L>	A>	=	C	O	P	X	RESERVED					INTERRUPT MASK			

Bit	Function
0	Logical "Greater Than"
1	Arithmetic "Greater Than"
2	Equal
3	Carry
4	Overflow
5	Parity
6	XOP Instruction Being Executed
12-15	Interrupt Mask

Figure 4-7. Status Register

▶4

The first three bits are set as a result of comparisons. Some instructions identify two operands (numbers) to be compared. If the first is greater than the second, the "greater than" bit should be set. In the 9900 there are two such conditions. First, the logical-greater-than bit considers 16-bit words as positive integers and the comparison is made accordingly. Second, the arithmetic-greater-than bit is set as the result of a comparison of two numbers which are considered in two's complement form. For example: consider the numbers A and B below as the numbers in the compare instruction C A, B:

A 1000 1110 1100 0101

B 0110 1010 1100 1101

If they are 16-bit positive integers, it is clear from the most significant bits (MSB) that A is greater than B, and the logical-greater-than bit of the status register should be set to one. But as two's complement numbers, A is negative (MSB = 1) and B is positive. Therefore the arithmetic-greater-than bit must be made zero (A is not greater than B). Since the processor has no way of knowing how the designer has used the memory words for data (integers or two's complement), two status bits must be provided for decision making. The designer can select the appropriate conditional jump instruction (testing status bit 0 or 1) because he knows what the data format is.

Status bit 2, the equal bit, is set if the two words compared are equal.

In many instructions, only one number is involved or a new number is determined as the result of an arithmetic operation. For these instructions status bits 0, 1 and 2 are set as the result of comparisons against zero; that is, if the single number or answer obtained is greater than zero or equal to zero.

Memory-to-Memory Architecture

The 9900 family of processors employs memory-to-memory architecture in the execution of instructions. Memory-to-memory architecture is that computer organization and instruction set which enables direct modification of memory data via a single instruction. That is, a single instruction can fetch one or two operands from memory, perform an arithmetic or logical operation, and also store the result in memory. In doing so, some of the on-chip registers are used as temporary buffers in much the same manner as an accumulator is used in other systems. But instructions to load an accumulator and store the accumulator are rarely necessary in memory-to-memory architecture. A single 9900 instruction (arithmetic or logical) does the work of two or more instructions in other systems.

Figure 4-8 describes the technique used by the 9900 to locate words in memory as "registers" in the workspace. Additional information is included for reference purposes. Registers 1-15 may be used for indexing (see the description of this addressing mode in Chapter 5 and 6). Register 0 may be used for a shift count. Registers 11 and 13-15 are used for subroutine techniques. Register 12 is a base value for CRU instructions. These special uses of the workspace registers are stated here as an initial evaluation of the register set. Program control and CRU instructions make use of the contents of registers 11-15; therefore, programmers and systems designers must be aware that while use of these registers is not restricted to their special functions, they should be used with caution in performing other functions.

The use of these workspaces in an actual application is best described in the Software Design chapter. But the step-by-step execution of the instructions is of concern in hardware design because of the execution speed and the techniques for handling interrupts.

Instruction cycles in the 9900 require memory access not only for the instruction words but also for operand addresses and actual operands (or numbers to be operated upon.) A simple add instruction requires at least four memory cycles: one to fetch the instruction, two to access the two numbers to be added, and one to store the result. As will be explained in detail later in this chapter, the execution of an add instruction may require as many as eight memory cycles (because of the addressing mode.) The execution steps are not the same for all instructions. There is, in fact, substantial variation of execution steps within any one instruction due to addressing. Tables and charts are provided in this chapter to explain the execution time of each instruction.

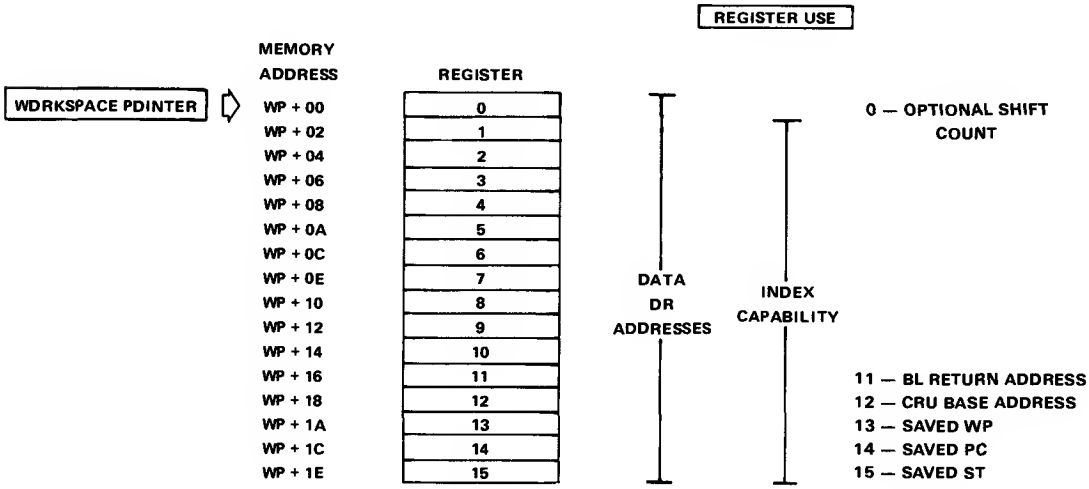


Figure 4-8. 9900 Workspace Registers

There is one additional concept regarding microprocessor and memory interfacing to be introduced at this time: it is the way in which data is stored in the memory. *Figure 4-9* shows the bit numbering for a general 16-bit data word or instruction. Instructions and 16-bit data words are always located at even addresses. Since the memory is byte addressable, even and odd bytes are the left and right half words in the 16-bit memory organization and have even or odd addresses respectively. Memories for the TMS9900 and SBP9900A contain 16 bits per word, while the other processors in the family use 8-bit memory structures. But all use the same addressing concept: a 16-bit address describing a 64k-byte address space.

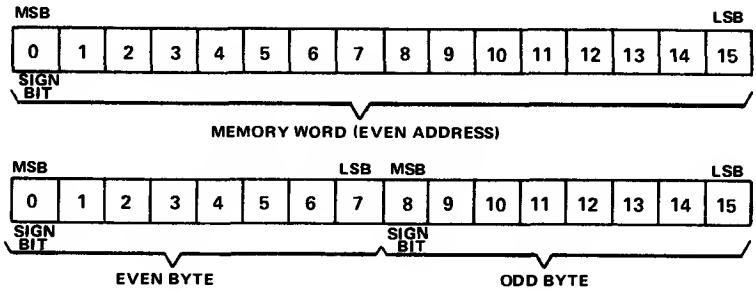


Figure 4-9. Word and Byte Formats

Context Switching

One of the more important advantages of the workspace architecture of the 9900 is the fact that "register save and restore" operations are greatly simplified. In any interrupt processing system, provisions must be made to perform an orderly transition into a new program segment in response to an interrupt. In other microprocessor systems, the first few instructions of an interrupt service routine perform the steps of saving register contents in memory, and then loading new values into the registers.

In the 9900, an interrupt cycle starts with a hardware operation to save the contents of the three key registers, the WP, PC and ST. In addition, the WP and PC must be loaded with new numbers. *Figures 4-10* and *4-11* show an example of the technique. Prior to the interrupt, the WP locates the workspace (pointing to 0800), the PC locates the current instruction (pointing to 0100), and the ST contains the status as a result of the execution of the current instruction (e.g., 4000). At the end of execution, the processor tests for an interrupt condition and finding it, performs a *context switch* as follows.

Step 1. The new WP value is fetched from the appropriate interrupt vector location in the first 32 words of memory. This identifies the location of the workspace assigned to the interrupt service routine.

Step 2. The current values of the WP, PC and ST registers are stored in the new workspace — ST in R15, PC in R14, WP in R13 in that order. After this, the new PC value is fetched from memory (the second location of the two-word interrupt vector) and loaded into the PC.

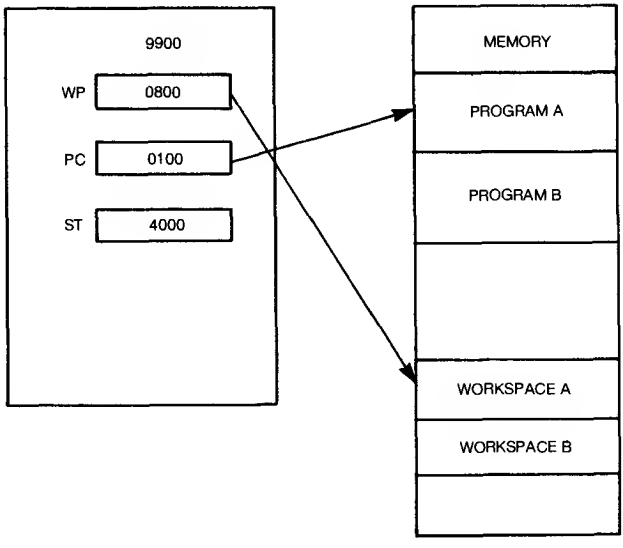
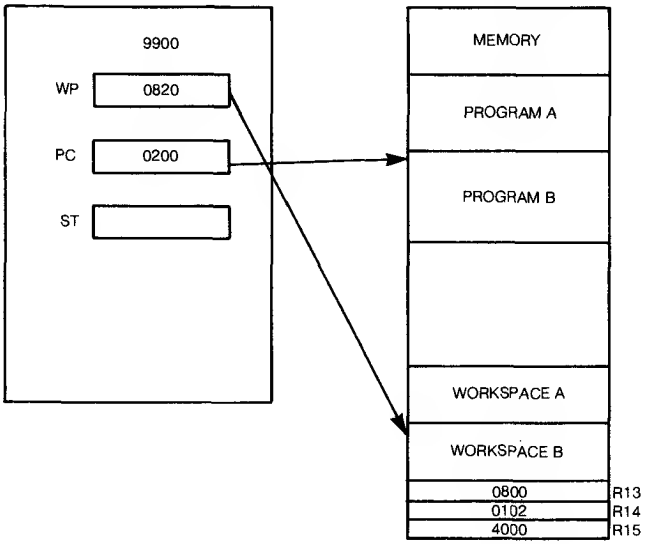


***Figure 4-10.** Before Context Switch*



▶4

Figure 4-11. After Context Switch

Step 3. With the context switch completed, processing resumes with the first instruction in the interrupt service routine.

Processing continues in this mode until, at the end of the interrupt routine, an RTWP instruction is encountered. A "reverse" context switch now occurs to return to the previous program. Since R13, 14 and 15 contain the control register contents for the previous program, they are now transferred to the CPU which loads them into the WP, PC and ST. Processing resumes from the point at which the interrupt occurred.

The obvious advantage of context switching is the reduced register-save register-restore operations required by microprocessors in an interrupt environment. The context switch is also used as a subroutine technique. This is described in Chapters 5 and 6, but the important fact is that context switching is, to the designer, a single step, when in fact several steps are performed by the microprocessor.

MEMORY

The 9900 is easily interfaced to any of the standard types of semiconductor memory devices. Texas Instruments provides masked ROMs, field-programmable ROMs (PROMs), and erasable PROMs (EPROMs) for non-volatile program and data storage. RAMs are available in sizes from a 64 x 8 static RAM to the 64K dynamic RAMs for use as a temporary program and data storage. 9900-compatible memory devices are listed in Chapter 2.

Memory Organization

The 9900 instructions build a 16-bit address word which describes a 64K x 8 bit address space. A memory map for the 9900 is shown in *Figure 4-12.*

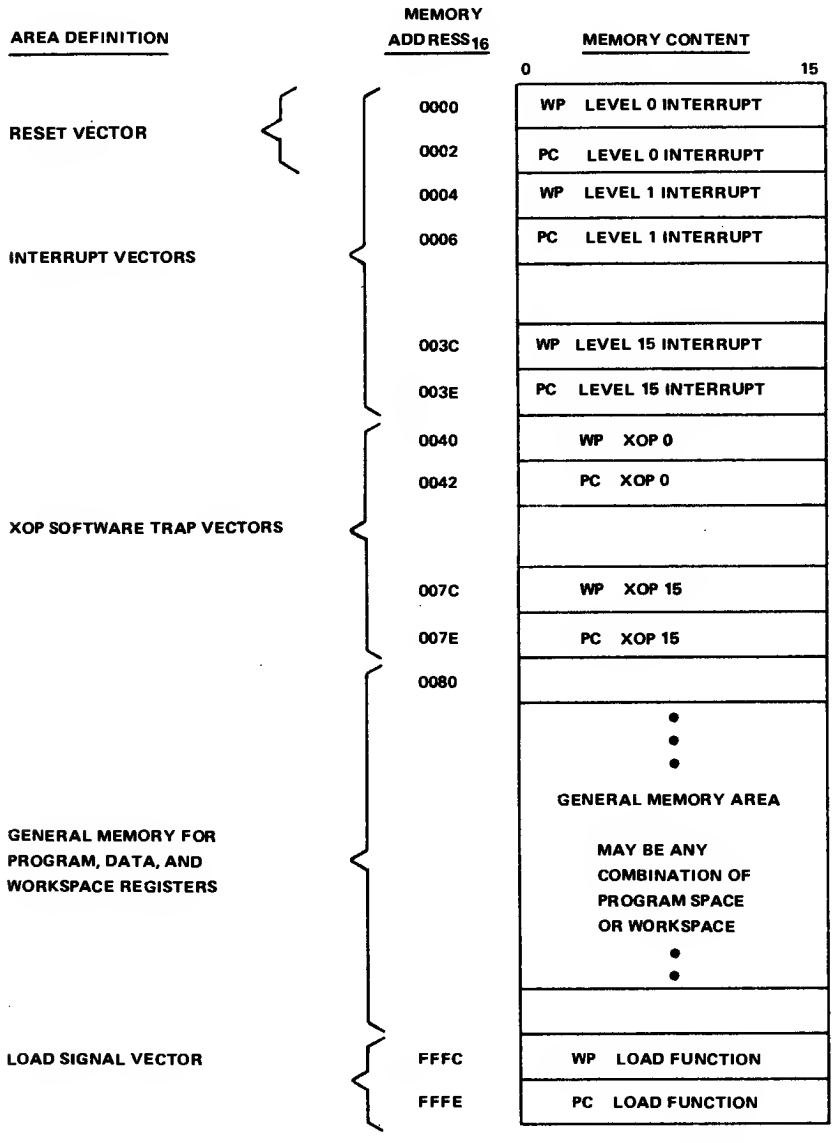


Figure 4-12. TMS 9900 Dedicated Memory Addresses

RESET Vector

The first two memory words are reserved for storage of the RESET vector. The RESET vector is used to load the new WP and PC whenever the CPU RESET signal occurs. The first word contains the new WP, which is the starting address of the RESET workspace. The second word contains the new PC, which is the starting address of the RESET service routine.

Interrupt Vectors

The next thirty memory words, 0004_{16} through $003E_{16}$ are reserved for storage of the interrupt transfer vectors for levels 1 through 15. Each interrupt level uses a word for the workspace pointer (WP) and a word for the starting address of the service routine (PC). If an interrupt level is not used within a system, then the corresponding two memory words can be used for program or data storage.

Software Trap Vectors

▶4 The next thirty-two memory words, 0040_{16} through $007E_{16}$, are used for extended-operation software trap vectors. When the CPU executes one of the 16 extended operations (XOPs), the program traps through the corresponding vector. Two words are reserved for each trap vector, with one word for the WP and one word for the PC. If an XOP instruction is not used, the corresponding vector words can be used for program or data storage.

LOAD Vector

The last two memory words $FFFC_{16}$ and $FFFE_{16}$ are reserved for the LOAD vector, with one word for the WP and one word for the PC. The LOAD vector is used whenever the CPU LOAD signal is active (low).

Transfer Vector Storage

The transfer vectors can be stored either in ROM or RAM, but either the RESET or LOAD vector should be in non-volatile memory and should point to a program in non-volatile storage to ensure proper system start-up. The restart routine should initialize any vector which is in RAM. The program can then manipulate the RAM-based vectors to alter workspace assignments or service routine entry points, while ROM-based vectors are fixed and cannot be altered.

Memory Control Signals

The 9900 uses three signals to control the use of the data bus and address bus during memory read or write cycles. Memory enable ($\overline{MEMEN}$) is active low during all memory cycles.

Data bus in (DBIN) is active high during memory read cycles and indicates that the CPU has disabled the output data buffers.

Write enable ($\overline{WE}$) is active low during memory write cycles and has timing compatible with the read/write (R/$\overline{W}$) control signal for many standard RAMs.

Memory Read Cycle

Figure 4-13 illustrates the timing for a memory read machine cycle with no wait states. At the beginning of the machine cycle, $\overline{MEMEN}$ and DBIN become active and the valid address is output on A0-A14. D0-D15 output drivers are disabled to avoid conflicts with input data. $\overline{WE}$ remains high for the entire machine cycle. The READY input is sampled on Φ1 of clock cycle 1, and must be high if no wait states are desired. Data is sampled on Φ1 of clock cycle 2, and set-up and hold timing requirements must be observed. A memory-read cycle is never followed by a memory-write cycle, and D0-D15 output drivers remain disabled for at least one additional clock cycle.

Memory Write Cycle

Figure 4-14 illustrates the timing for a memory write machine cycle with no wait states. $\overline{MEMEN}$ becomes active, and valid address and data are output at the beginning of the machine cycle. DBIN remains inactive for the complete cycle. $\overline{WE}$ goes low on Φ1 of clock cycle 1 and goes high on Φ1 of clock cycle 2, meeting the address and data set-up and hold timing requirements for the static RAMs listed in Chapter 2. For no wait states, READY must be high during Φ1 of clock cycle 1.

Read/Write Control with DBIN

In some memory systems, particularly with dynamic RAMs, it may be desirable to have READ and WRITE control signals active during the full memory cycle. *Figure 4-15* shows how the WRITE signal can be generated. Note that DBIN is high *only* for READ cycles; therefore, $\overline{MEMEN}$ can be NORed with DBIN to yield a WRITE signal which is high only during memory write operations.

Slow Memory Control

Although most memories operate with the 9900 at the full system speed, some memories cannot properly respond within the minimum access time determined by the system clock. The system clock could be slowed down in order to lengthen the access time but the system through-put would be adversely affected since non-memory and other memory reference cycles would be unnecessarily longer. The READY and WAIT signals are used instead to synchronize the CPU with slow memories. The timing for memory-read and memory-write cycles with wait states is shown in *Figures 4-16* and *4-17*.

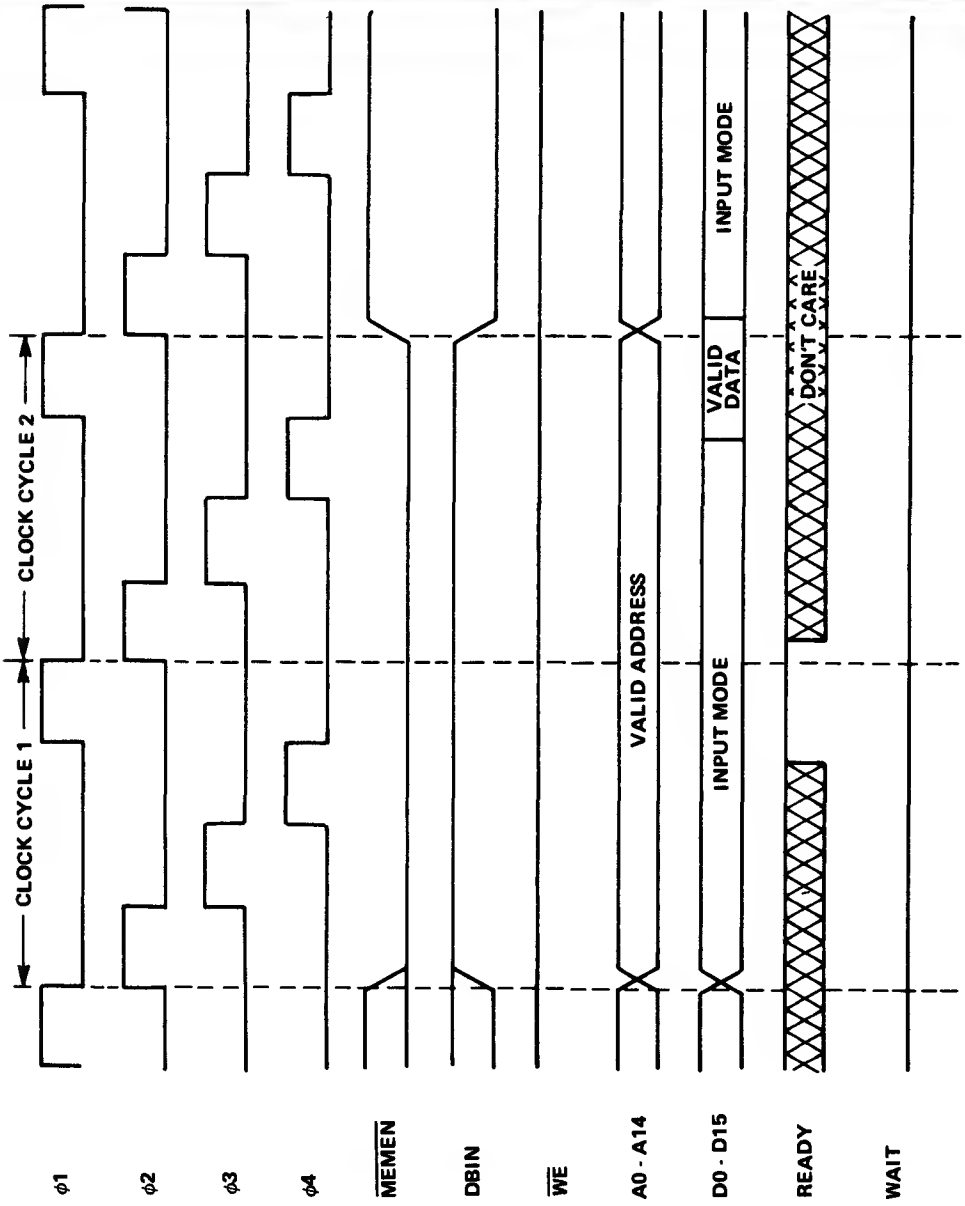


Figure 4-13. Memory-Read Cycle Timing

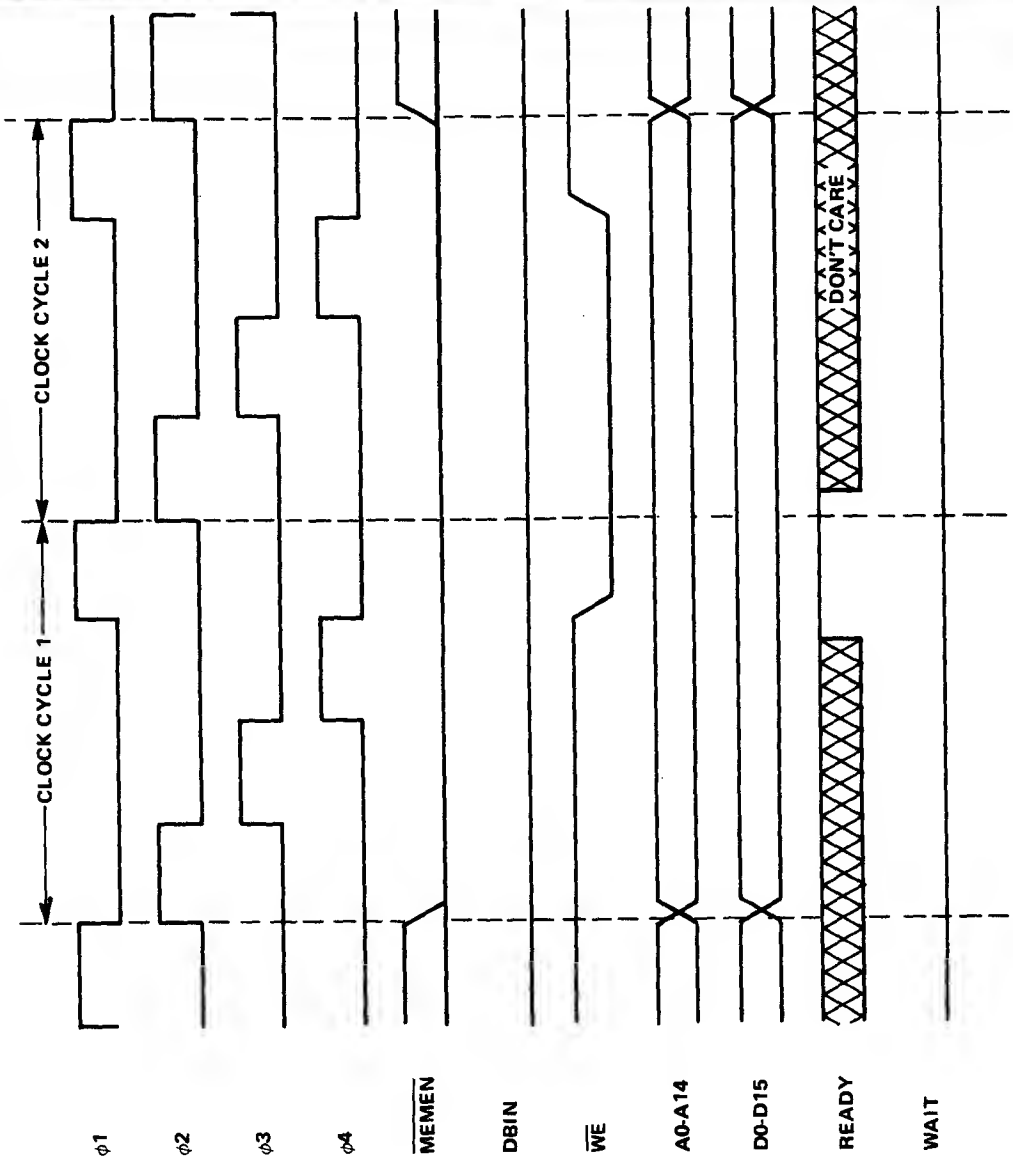


Figure 4-14. Memory-Write Cycle Timing

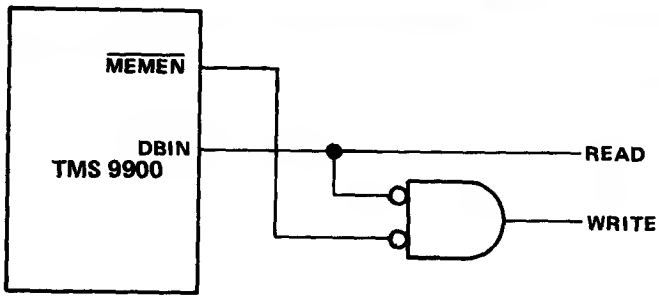


Figure 4-15. Read/Write Control Using $\overline{MEMEN}$ and DBIN

The READY input is tested on $\Phi 1$ of clock cycle 1 of memory-read and memory-write cycles. If READY = 1, no wait states are used and the data transfer is completed on the next clock cycle. If READY = 0, the processor enters the wait state on the next clock cycle and all memory control, address, and data signals maintain their current levels. The WAIT output goes high on $\Phi 3$ to indicate that a wait state has been entered. While in the wait state, the processor continues to sample READY on $\Phi 1$, and remains in the wait state until READY = 1. When READY = 1 the processor progresses to clock cycle 2 and the data transfer is completed. WAIT goes low on $\Phi 3$. It is important to note that READY is only tested during $\Phi 1$, of clock cycle 1 of memory-read and memory-write cycles and wait states, and the specified set-up and hold timing requirements must be met; at any other time the READY input may assume any value. The effect of inserting wait states into memory access cycles is to extend the minimum allowable access time by one clock period for each wait state.

Wait State Control

Figure 4-18 illustrates the connection of the WAIT output to the READY input to generate one wait state for a selected memory segment. The address decode circuity generates an active low signal ($\overline{SLOMEM} = 0$) whenever the slow memory is addressed. For example, if memory addresses 8000_{16} – $FFFE_{16}$ select slow memory, $\overline{SLOMEM}$ = A0. If one wait state is required for all memory, WAIT may be connected directly to READY, causing one wait state to be generated on each memory-read or memory-write machine cycle. Referring again to *Figures 4-16* and *4-17* note that the WAIT output satisfies all of the timing requirements for the READY input for a single wait state. The address decode signal is active only when a particular set of memory locations has been addressed. *Figure 4-19* illustrates the generation of two wait states for selected memory by simply delaying propagation of the WAIT output to the READY input one clock cycle with a D-type flip-flop. The rising edge of $\Phi 2$TTL is assumed to be coincident with the falling edge of the $\Phi 2$ clock input to the TMS 9900.

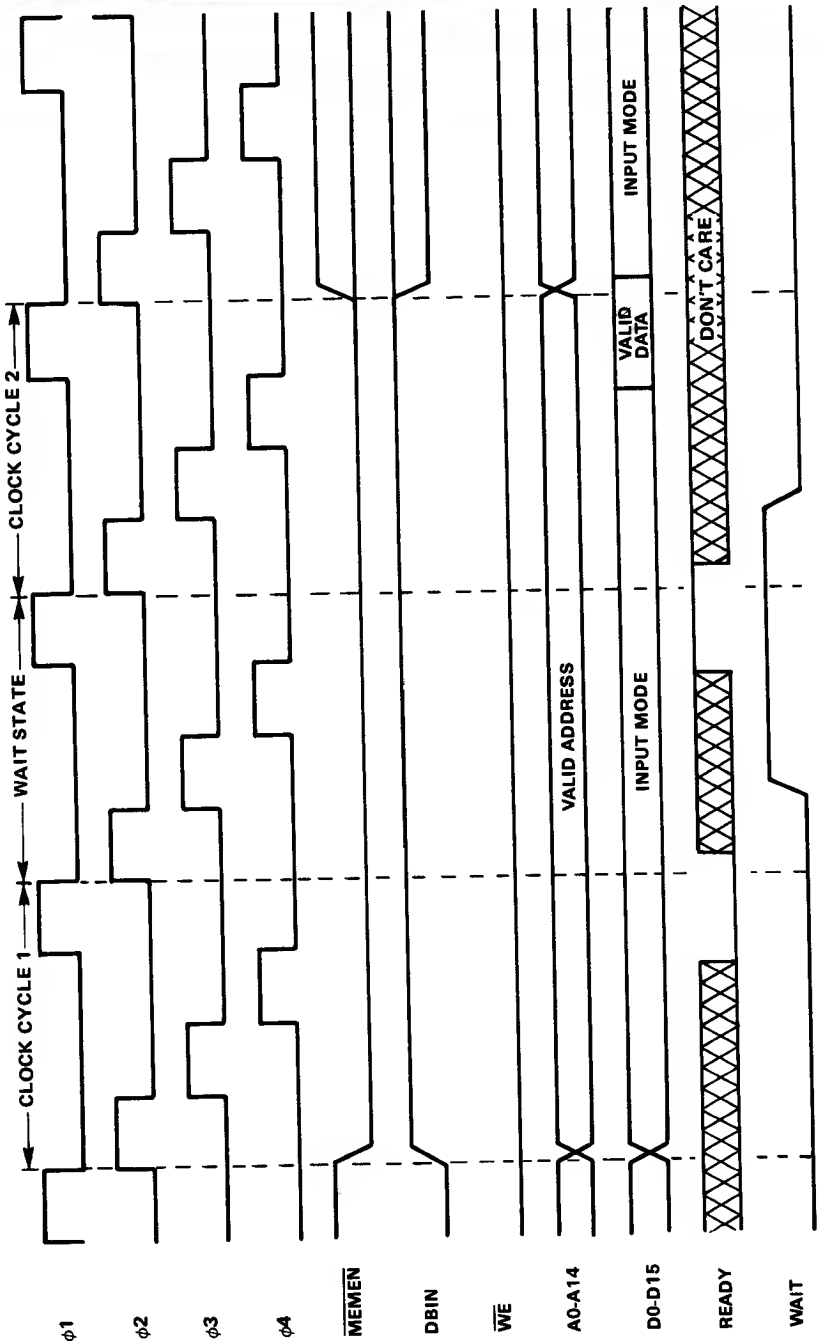


Figure 4-16. Memory-Read Cycle With One Wait State

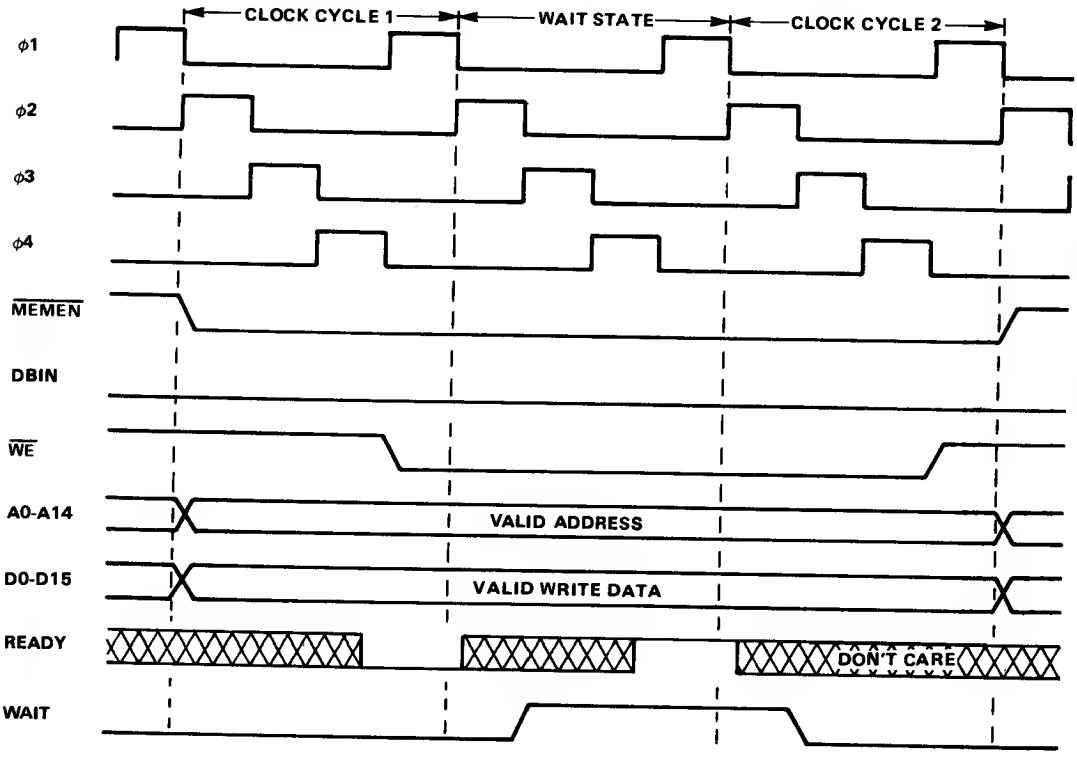


Figure 4-17. Memory-Write Cycle With One Wait State

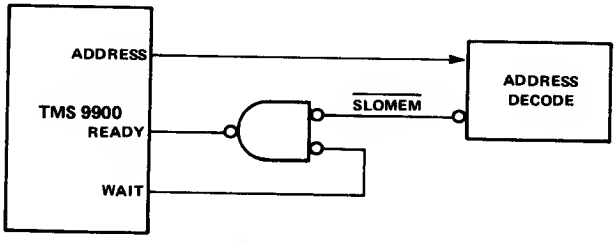


Figure 4-18. Single Wait State for Slow Memory

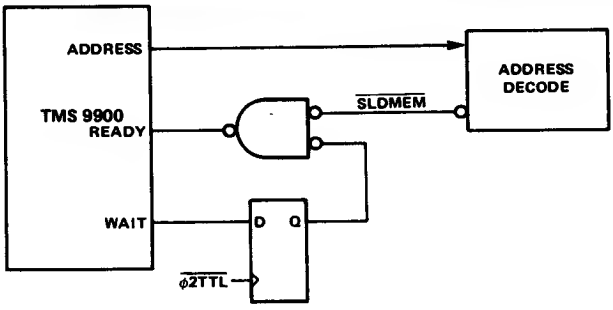


Figure 4-19. Double Wait States for Slow Memory

Memory Access Time Calculation

4

Maximum allowable memory access time for the TMS 9900 can be determined with the aid of *Figure 4-20.* Memory control and address signals are output on $\Phi 2$ of clock cycle 1, and are stable 20 ns (t_{PLH}, t_{PHL}) afterwards. Data from memory must be valid 40 ns (t_{su}) before the leading edge of $\Phi 1$ during clock cycle 2. Therefore, memory access time may be expressed by the equation:

$$t_{acc} \leq (1.75 + n)\, t_{cy} - t_{PLH} - t_r - t_{su}$$

where n equals the number of wait states in the memory-read cycle. Assigning worst-case specified values for t_{PLH} (20ns), t_r (12ns), and t_{su} (40 ns), and assuming 3 MHz operation:

$$t_{acc} \leq \frac{(1.75 + n)}{0.003} - 72 \text{ ns}$$

Access time is further reduced by address decoding, control signal gating, and address and data bus buffering, when used. Thus, for a known access time for a given device, the number of required wait states can be determined.

For example, a TMS 4042-2 RAM has a 450 nanosecond access time and does not require any wait states. A TMS 4042 has a 1000 nanosecond access time and requires two wait states. Propagation delays caused by address or data buffers should be added to the nominal device access time in order to determine the effective access time.

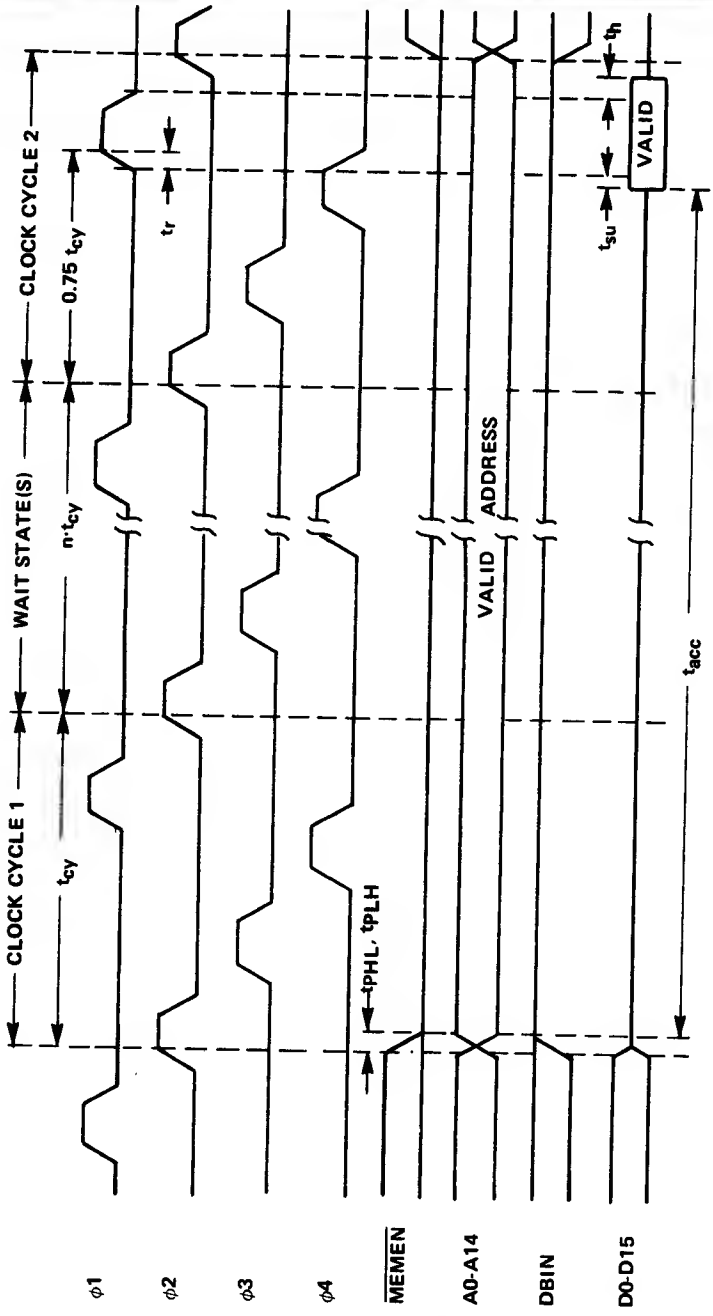


Figure 4-20. Memory Access Timing Calculation

Static Memory

Static RAMs and PROMs are easily interfaced to the 9900. A 9900 memory system using the TMS 4042-2 256 X 4 static RAM and the TMS 2708 1K X 8 EPROM is shown in *Figure 4-21.*

Address

The most-significant address bit, A0, is used to select either the EPROMs or the RAMs during memory cycles. When A0 is low, the EPROMs are selected, and when A0 is high, the RAMs are selected. Address lines A1 through A4 are not used since the full address space of the TMS 9900 is not required in the example. The lower address bits select internal RAM or EPROM cells. Other memory systems can fully decode the address word for maximum memory expansion.

Control Signals

Since DBIN is also used to select the EPROMs during memory-write cycles, the EPROMs cannot inadvertently be selected and placed into output mode while the CPU is also in the output mode on the data bus. $\overline{\text{MEMEN}}$ is used to select the RAMs during either read or write cycles, and $\overline{\text{WE}}$ is used to select the read/write mode. DBIN is also used to control the RAM output bus drivers.

The 9900 outputs $\overline{\text{WE}}$ three clock phases after the address, data, and $\overline{\text{MEMEN}}$ are output. As a result, the address, data, and enable-hold times are easily met. $\overline{\text{WE}}$ is enabled for one clock cycle and satisfies the minimum write pulse width requirement of 300 nanoseconds. Finally, $\overline{\text{WE}}$ is disabled one clock phase before the address, data, and other control signals and meets the TMS 4042-2 50-nanosecond minimum data and address hold time.

Loading

The loads on the CPU and memory outputs are well below the maximum rated loads. As a result no buffering is required for the memory system in *Figure 4-21.* The TMS 4042-2 and the TMS 2708 access times are low enough to eliminate the need for wait states, and the CPU READY input is connected to V_{CC}.

The minimum high-level input voltage of the TMS 2708 is 3 volts while the maximum high-output voltage for the TMS 9900 is 2.4 volts at the maximum specified loading. For the system in *Figure 4-21,* the loads on the CPU and memory outputs are well below the maximum rated load. At this loading, the TMS9900 output voltage exceeds 3 volts, so pull-up resisters are not needed.

There are many other Texas Instruments static memories compatible with the TMS 9900. Most memory devices do not require wait states when used with the TMS 9900 at 3 MHz.

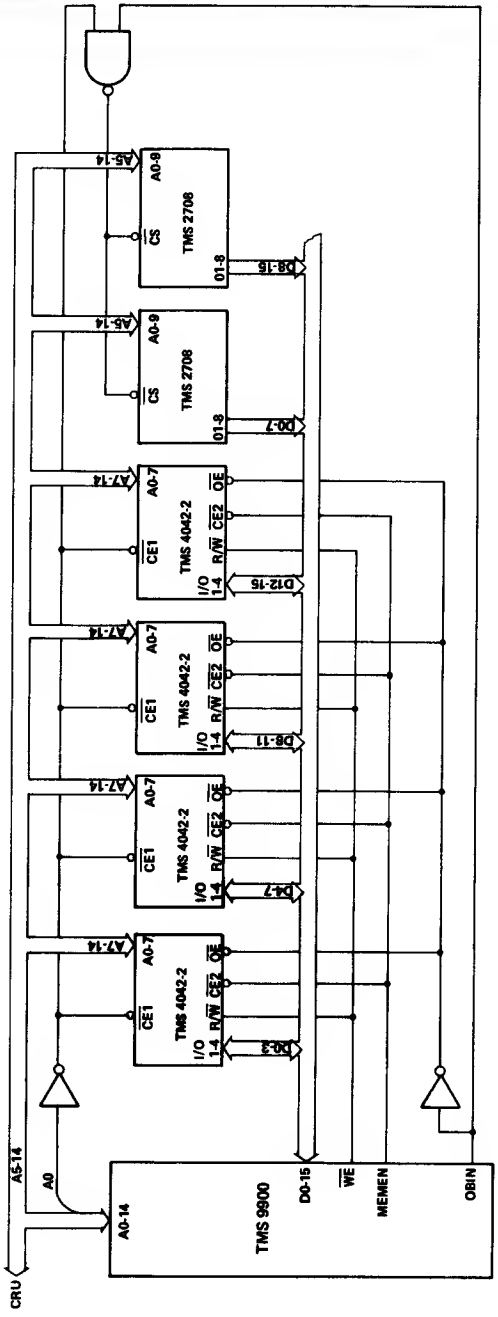


Figure 4-21. TMS 9900 Static Memory System

Dynamic Memory

Memory applications requiring large bit storage can use 4K, 16K or 64K dynamic memories for low cost, low power consumption, and high bit density. TMS 9900 systems requiring 4K words or more of RAM, can economically use the 4096-bit TMS 4051, the 16,384-bit TMS 4116, or any of the other dynamic RAMs covered in Chapter 2.

Refresh

The dynamic RAMs must be refreshed periodically to avoid the loss of stored data. The RAM data cells are organized into a matrix of rows and columns with on-chip gating to select the addressed bit. Refresh of the 4K RAM cell matrix is accomplished by performing a memory cycle of each of the 64 row addresses every 2 milliseconds or less. The 16K RAM has 128 row addresses. Performing a memory cycle at any cell on a row refreshes all cells in the row, thus allowing the use of arbitrary column address during refresh.

Refresh Modes

There are several dynamic memory refresh techniques which can be used for a TMS 9900 system. If the system periodically accesses at least one cell of each row every 2 milliseconds, then no additional refresh circuitry is required. A CRT controller which periodically refreshes the display, illustrates this concept.

Refresh control logic is included wherever the system cannot otherwise ensure that all rows are refreshed every 2 milliseconds. The dynamic memory in such TMS 9900 systems can be refreshed in the block, cycle stealing, or transparent mode.

Block Refresh.

The block mode of refresh halts the CPU every 2 milliseconds and sequentially refreshes each of the rows. The block technique halts execution for a 128 (4K) or 256 (16K) clock cycle periods every 2 milliseconds. Some TMS 9900 systems cannot use this technique because of the possibility of slow response to priority interrupts or because of the effect of the delay during critical timing or I/O routines.

Cycle Stealing.

The cycle stealing mode of refresh "steals" a cycle from the system periodically to refresh one row. The refresh interval is determined by the maximum refresh time and the number of rows to be refreshed. The 4K dynamic RAMs have 64 rows to be refreshed every 2 milliseconds and thus require a maximum cycle stealing interval of 31.2 microseconds.

A cycle stealing refresh controller for the TMS 4051 4K dynamic RAM is shown in *Figure 4-22.* The refresh timer generates the refresh signal (RFPLS) every 30 microseconds. The refresh request signal (RFREQ) is true until the refresh cycle is completed. The refresh grant signal (RFGNT) goes high during the next CPU clock cycle in which the CPU is not accessing the dynamic memory. The refresh memory cycle takes two clock cycles to complete after RFGNT is true. During the second clock cycle, however, the CPU can attempt to access the dynamic memory since the CPU is not synchronized to the refresh controller. If the CPU does access memory during the last clock cycle of the refresh memory cycle, the refresh controller makes the memory not-ready for the remainder of the refresh memory cycle, and the CPU enters a wait state during this interval. The dynamic memory row address during the refresh memory cycle is the output of a modulo-64 counter. The counter is incremented each refresh cycle in order to refresh the rows sequentially.

The dynamic memory timing controller generates the proper chip enable timing for both CPU and refresh initiated memory cycles. The timing controller can be easily modified to operate with other dynamic RAMs.

Since the TMS 9900 performs no more than three consecutive memory cycles, the refresh request will be granted in a maximum of three memory cycles. Some systems may have block DMA, which uses $\overline{\text{HOLD}}$. RFREQ can be used in such systems to disable HOLDA temporarily in order to perform a refresh memory cycle if the DMA block transfer is relatively long (greater than 30 microseconds). The cycle stealing mode "steals" clock cycles only when the CPU attempts to access the dynamic memory during the last half of the refresh cycle. Even if this interference occurs during each refresh cycle, a maximum of 64 clock cycles are "stolen" for refresh every 2 milliseconds.

Transparent Refresh.

The transparent refresh mode eliminates this interference by synchronizing the refresh cycle to the CPU memory cycle. The rising edge of $\overline{\text{MEMEN}}$ marks the end of a memory cycle immediately preceding a non-memory cycle. The $\overline{\text{MEMEN}}$ rising edge can initiate a refresh cycle with no interference with memory cycles. The refresh requirement does not interfere with the system throughput since only non-memory cycles are used for the refresh cycles. The worst-case TMS 9900 instruction execution sequence (all divides) will guarantee the complete refresh of a 4K or 16K dynamic RAM within 2 milliseconds.

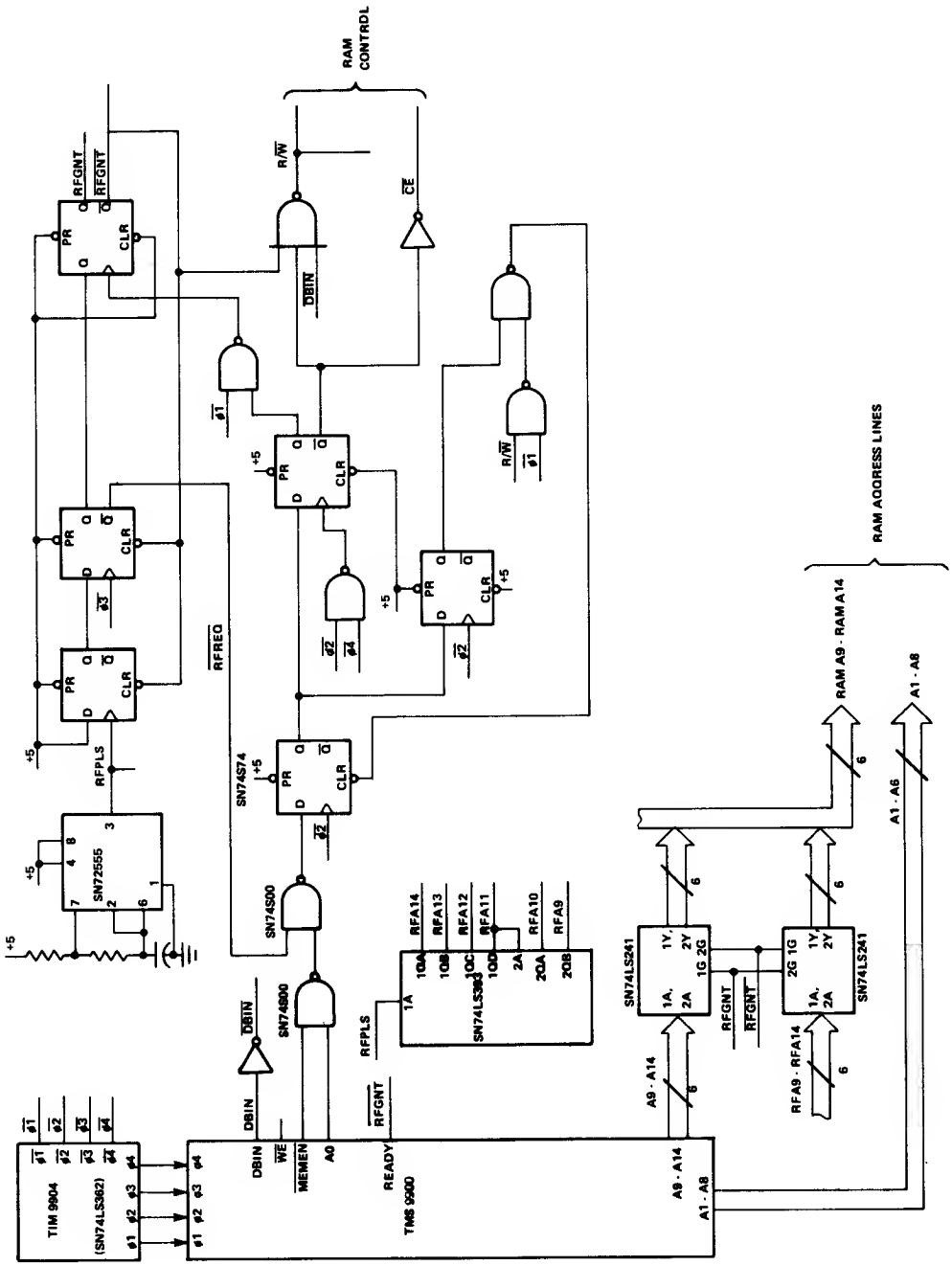


Figure 4-22. Cycle-Stealing Dynamic RAM Refresh for TMS 4051

While the transparent refresh mode eliminates refresh-related system performance degradation, the system power consumption can be higher since the RAMs are refreshed more often than required. As many as one-half of the CPU machine cycles can be refresh cycles, resulting in multiple refresh cycles for each row during the refresh interval. This situation can be corrected by adding a timer to determine the start of the refresh interval and an overflow detector for the refresh row counter. When every row has been refreshed during an interval, the refresh circuit is disabled until the beginning of the next interval. Since each row is refreshed only once, the system power consumption is reduced to a minimum.

Direct memory access using $\overline{\text{HOLD}}$ should guarantee that sufficient non-memory cycles are available for refresh during large block transfers. An additional refresh timer can be used to block HOLDA in order to provide periodic refresh cycles.

Buffered Memory

▶4 The TMS 9900 outputs can drive approximately two standard TTL inputs and 200 picofarads. Higher capacitive loads may be driven, but with increased rise and fall times. Many small memory systems can thus be directly connected to the CPU without buffer circuits. Larger memory systems, however, may require external bipolar buffers to drive the address or data buses because of increased loading. Texas Instruments manufactures a number of buffer circuits compatible with the TMS 9900. The SN74LS241 noninverting-octal buffer with three-state outputs is an example of a buffer circuit.

A TMS 9900 memory system with address and data bus buffering is shown in *Figure 4-23.* The system consists of sets of four 256 X 4 memory devices in parallel to provide the 16-bit data word. The four sets of four devices provide a total of 1024 words of memory. The memory devices can be the TMS 4042-2 NMOS static RAM.

The SN74S412 octal buffer/latch is designed to provide a minimum high-level output voltage of 3.65 V. Buffered TMS 9900 memory systems containing the TMS 4700 ROM or the TMS 2708 EPROM, for example, require input voltages in excess of the output voltages of many buffer circuits. The SN74S412 can be used to buffer the memories without the pull-up resistors needed for buffers.

Memory Parity

Parity or other error detection/correction schemes are often used to minimize the effects of memory errors. Error detection schemes such as parity are used to indicate the presence of bad data, while error correction schemes correct single or multiple errors.

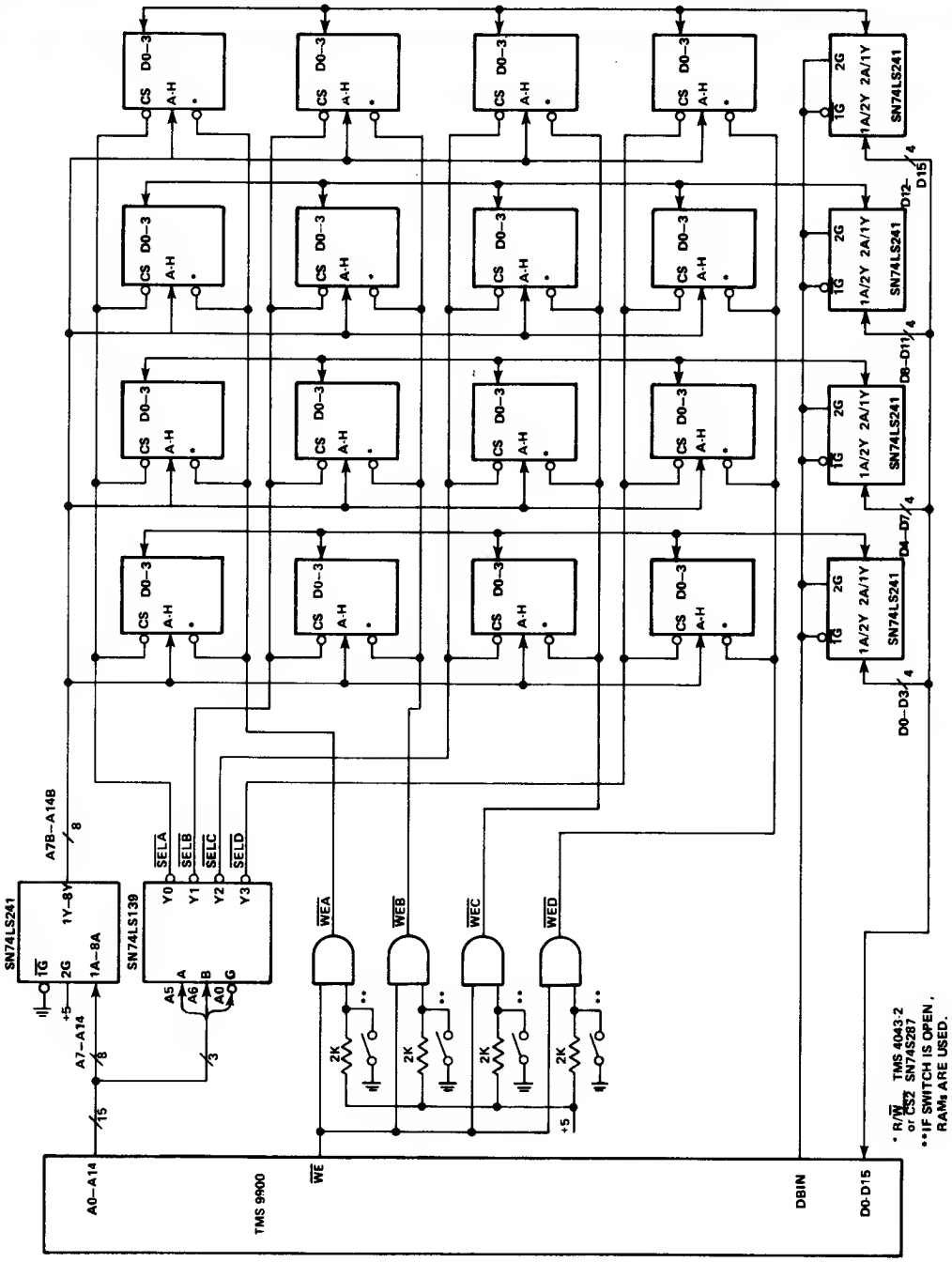


Figure 4-23. Buffered Memory with Mixed PROM/ROM

The SN74LS280 parity generator/checker can be used to implement memory parity in a TMS 9900 system. The system in *Figure 4-24* uses two SN74LS280 circuits to generate and to check the odd-memory parity. During memory write cycles, the generated parity bit is output to bit D16 of the memory. During memory read cycles, the parity is checked and an interrupt, $\overline{\text{PARERR}}$, is generated if the parity is even.

It should be noted that the faulty memory word will have already been used by the CPU as an op code, address, or data before the interrupt is generated. This can cause trouble in determining the exact location of the error. For example, an error in bit 8 of the CLR op code will cause the CPU to branch unconditionally. When the interrupt is serviced, there would then be no linkage to the part of the program at which the error occurred. A diagnostic routine can often isolate such errors by scanning the memory and checking parity under program control. Such a parity error in the diagnostic itself can be extremely difficult to isolate.

▶4 An external address latch clocked at IAQ can be used to retain program linkage under the above circumstances. When the parity error is detected, the address latch is frozen, thus pointing to the address of the instruction during which the parity error occurred.

Memory Layout

It is generally advantageous to lay out memory devices as arrays in the system. The advantages are twofold. First, positioning the devices in an orderly fashion simplifies identification of a particular memory element when troubleshooting. Second, and most important, layout of memory arrays simplifies layout, shortens interconnections, and generally allows a more compact and efficient utilization of board space. Crosstalk between adjacent lines in memory arrays is minimized by running address and data lines parallel to each other, and by running chip enable signals perpendicular to the address lines.

Memory devices, particularly dynamic RAMs generally require substantially greater supply currents when addressed than otherwise. It is therefore important that all power and ground paths be as wide as possible to memory arrays. Furthermore, in order to avoid spikes in supply voltages, it is advisable to decouple supply voltages with capacitors as close as possible to the pins of the memory devices. As an example, a system containing a 4K x 16-bit array of TMS 4051s should contain one 15 μF and one 0.05 μF capacitor for each set of four memory devices; with the large capacitors decoupling V_{DD}, and the small capacitors decoupling V_{BB}.

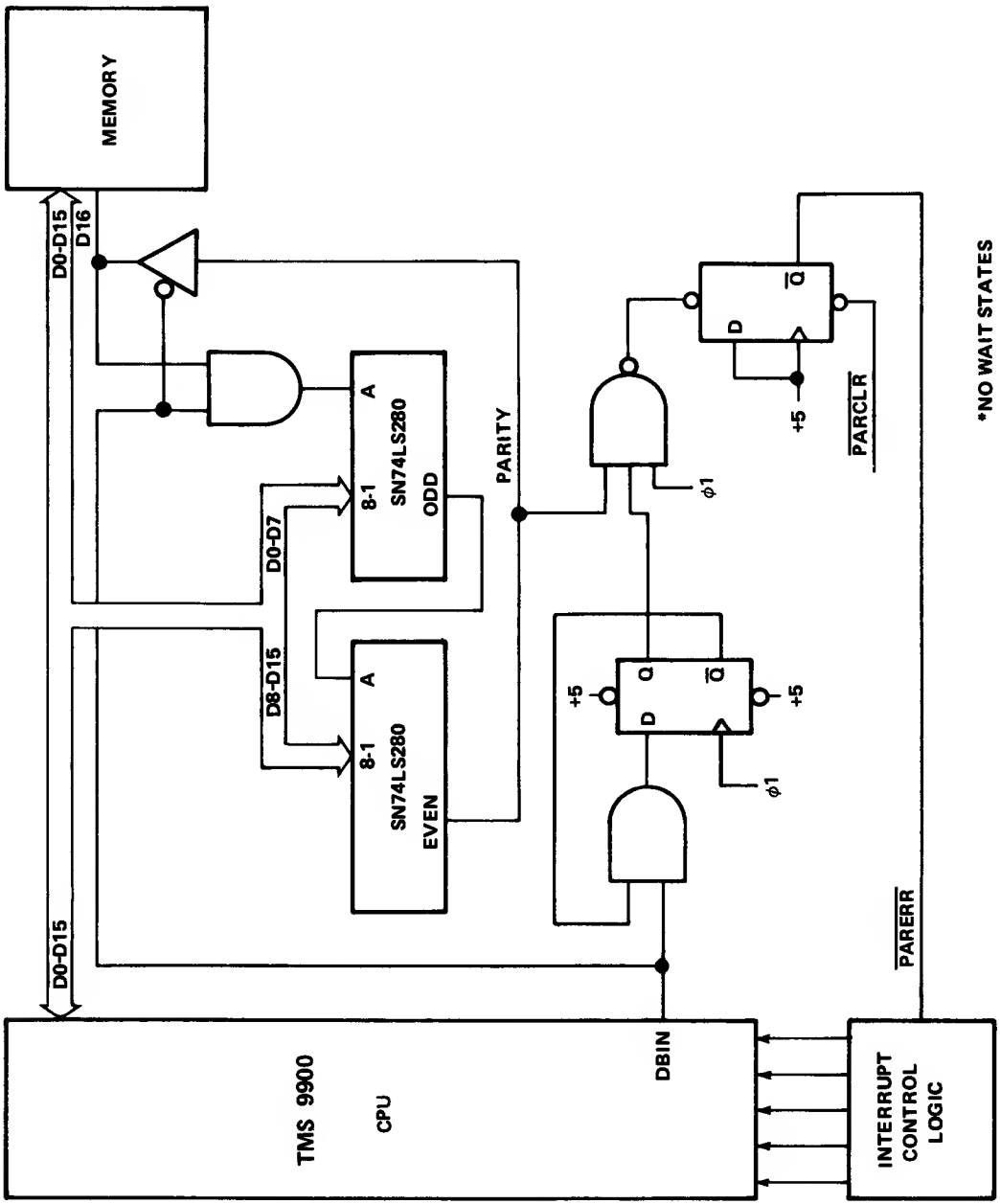


Figure 4-24. Memory Parity Generator Checker

INSTRUCTION EXECUTION

Execution time for an instruction is a function of the clock frequency, the number of clock cycles, the number of memory accesses and the number of wait states if required for slower memories. The following tables list the number of clock cycles required to execute each instruction if no wait states are required. The number of memory accesses is also given so that the extra clock cycles can be calculated for the number of wait states required. A wait state is entered when the ready signal from the memory does not go high within one clock period after initiation of a memory cycle. For example: The clock frequency for the TMS 9900 is 3 MHz. From the calculation of maximum access time for no wait states, the memory access time must be less than 512 ns. One wait state (of 333 ns duration) will be required for memories with access times between 512 ns and 845 ns, two wait states will be required if the access time is between 845 ns and 1.178 μ sec, and so on.

▶4 TIMING

From *Figure 4-25,* the first execution time table, an add instruction (A) using direct register addressing for both operands requires 14 clock cycles if there are no wait states required. For other addressing modes, the number of clock cycles increases to a maximum of 30. If the memory requires one wait state per access, an additional four clock periods will be required since there are four memory cycles in the execution of an add instruction. For the TMS 9900 running at 3 MHz, 14 clock periods will take 4.667 microseconds; 30 clock periods will take 10.0 microseconds. The number of memory cycles is from 4 up to 8 depending upon addressing mode (3 to 7 for compare, C). Use the tables in the following manner. Assuming one wait state, a clock frequency of 3 MHz, and an instruction with complex addressing, the tables can be used to determine the execution time for the instruction

A *R1, @ LIST

is 26 clock cycles for fetch and execution and 6 clock cycles for wait states, or 32 x .333 microseconds which is 10.667 microseconds.

Figures 4-26, 27 and *28* give the rest of the execution time data, always by number of clock cycles (assuming no wait states) and memory cycles. Execution times for instructions which do not use the five general addressing modes may be found in Chapter 8 in the *CPU* section.

INSTRUCTIONS A, C†, S, SOC, SZC, MOV

	Destination Address	Source Address				
		R	*R	*R +	@LIST	@TABLE (R)
Clock Cycles	R	14	18	22	22	22
	*R	18	22	26	26	26
	*R +	22	26	30	30	30
	@LIST	22	26	30	30	30
	@TABLE (R)	22	26	30	30	30
Memory Cycles	R	4	5	6	5	6
	*R	5	6	7	6	7
	*R +	6	7	8	7	8
	@LIST	5	6	7	6	7
	@TABLE (R)	6	7	8	7	8
†Memory Cycles for C instr.	R	3	4	5	4	5
	*R	4	5	6	5	6
	*R +	5	6	7	6	7
	@LIST	4	5	6	5	6
	@TABLE (R)	5	5	6	7	6

Figure 4-25.

INSTRUCTIONS: AB, CB††, SB. SOCB, SZCB, MOVB

	Destination Address	Source Address				
		R	*R	*R +	@LIST	@TABLE (R)
Clock Cycles	R	14	18	20	22	22
	*R	18	22	24	26	26
	*R +	20	24	26	28	28
	@LIST	22	26	28	28	28
	@TABLE (R)	22	26	28	28	28
Memory Cycles	R	4	5	6	5	6
	*R	5	6	7	6	7
	*R +	6	7	8	7	8
	@LIST	5	6	7	6	7
	@TABLE (R)	6	7	8	7	8
††Memory Cycles for CB instr.	R	3	4	5	4	5
	*R	4	5	6	5	6
	*R +	5	6	7	6	7
	@LIST	4	5	6	5	6
	@TABLE (R)	5	6	7	6	7

Figure 4-26.

INSTRUCTIONS LDCR, STCR

LDCR

	Addressing Mode	Bit Count, C															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
Clock Cycles	R	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50	52
	*R	26	28	30	32	34	36	38	40	42	44	46	48	50	52	54	56
	*R +	28	30	32	34	36	38	40	42	46	48	50	52	54	56	58	60
	@LIST	30	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60
	@TABLE (R)	30	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60
Memory Cycles	R	3								3							
	*R	4								4							
	*R +	5								5							
	@LIST	4								4							
	@TABLE (R)	5								5							

STCR

▶4

	Addressing Mode	Bit Count, C															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
Clock Cycles	R	42	42	42	42	42	42	42	44	58	58	58	58	58	58	58	60
	*R	46	46	46	46	46	46	46	48	62	62	62	62	62	62	62	64
	*R +	48	48	48	48	48	48	48	50	66	66	66	66	66	66	66	68
	@LIST	50	50	50	50	50	50	50	52	66	66	66	66	66	66	66	68
	@TABLE (R)	50	50	50	50	50	50	50	52	66	66	66	66	66	66	66	68
Memory Cycles	R	4								4							
	*R	5								5							
	*R..	6								6							
	@LIST	5								5							
	@TABLE (R)	6								6							

Figure 4-27.

	Clock Cycles					Memory Cycles				
Instruction	R	*R	*R +	@LIST,	@TABLE (R)	R	*R	*R +	@LIST	@TABLE (R)
ABS MSB = 0	12	16	20	20	20	2	3	4	3	4
MSB = 1	14	18	22	22	22	3	4	5	4	5
B	8	12	16	16	16	2	3	4	3	4
BL	12	16	20	20	20	3	4	5	4	5
BLWP	26	30	34	34	34	6	7	8	7	8
CLR	10	14	18	18	18	3	4	5	4	5
DEC	10	14	18	18	18	3	4	5	4	5
DECT	10	14	18	18	18	3	4	5	4	5
INC	10	14	18	18	18	3	4	5	4	5
INCT	10	14	18	18	18	3	4	5	4	5
INV	10	14	18	18	18	3	4	5	4	5
NEG	12	16	20	20	20	3	4	5	4	5
SETO	10	14	18	18	18	3	4	5	4	5
SWPB	10	14	18	18	18	3	4	5	4	5
XOP	36	40	44	44	44	8	9	8	9	8
XOR	14	18	22	22	22	4	5	6	5	6

4◀

Figure 4-28.

Cyclic Operation

An example of a machine cycle sequence is illustrated in *Figure 4-29.* For an add instruction the machine cycles alternate between memory cycles and ALU cycles. The first cycle is always a memory read cycle to fetch the instruction and the second is always an ALU cycle to decode the instruction. Each machine cycle requires two clock cycles, thus the 7 machine cycles shown for the add instruction require 14 clock cycles.

A R1, R2

1	Memory Read	Instruction Fetch
2	ALU	Decode Opcode
3	Memory Read	Fetch (WR1)
4	ALU	Set Up
5	Memory Read	Fetch (WR2)
6	ALU	Addition
7	Memory Write	Store Result in WR2 and Increment PC

Figure 4-29. Machine Cycles for an Add Instruction

The 9900 performs its functions under control of a 4-phase clock and, fundamentally, performs instruction fetch and execution cycles. *Figure 4-30* illustrates the step-by-step procedure the 9900 uses to execute an add instruction. From previous cycles, the workspace pointer has been loaded with the number 0800, and the program counter contains the number 0100.

Step 1. The first step in any instruction cycle is to fetch the instruction. This is accomplished by gating the content of the program counter into the memory address register. The output of the memory address register is the address bus which is connected to the memory. In this case, word number 0100 is read from the memory and placed in the instruction register on the 9900 chip. From this point, the ones and zeros of the instruction register control the sequence of microcode stored in the microcontrol read only memory on the 9900 chip. These microsteps become the execution phase of the instruction.

▶4 *Step 2.* At this point, the microcontrol shifts to the execution of an add instruction; the first operand must be obtained from memory. In order to do this, the workspace pointer and a portion of the instruction word (the source operand register number) are added together via the ALU and placed in the memory address register.

Step 3. The address 0802 is the result (in this example), and being supplied to the memory produces on the data bus the content of memory word 0802 which is the binary equivalent of 25. This number must be stored in a temporary register on the 9900 chip, in this case the T1 register.

Step 4. Now a second operand must be fetched. Again the workspace pointer is added to the content of that portion of the instruction word which is the destination register identifier. The sum of these two is 0804 for register two, and this number is placed in memory address register and goes out on the address bus.

Step 5. Memory word 0804 is read and the number 10 is brought into the 9900 chip. The register which stores the second operand is called the source data register or S register.

Step 6. At this point the two operands have been loaded into registers on the 9900 chip and may be added by the ALU to produce the result. Register T1 containing 25 is added to the register S which contains 10 and the sum, 35, replaces the 10 in the S register and is placed on the data bus via the S register.

Step 7. The address bus still contains the number 0804 which was the address of the second operand and is the location in memory where the result is to be stored. So at this point in the cycle, a memory write cycle is initiated and the binary equivalent of 35 is stored in memory location 0804. At the conclusion of this memory cycle the program counter is incremented by two to point to the next sequential memory word, which is the instruction to be executed next.

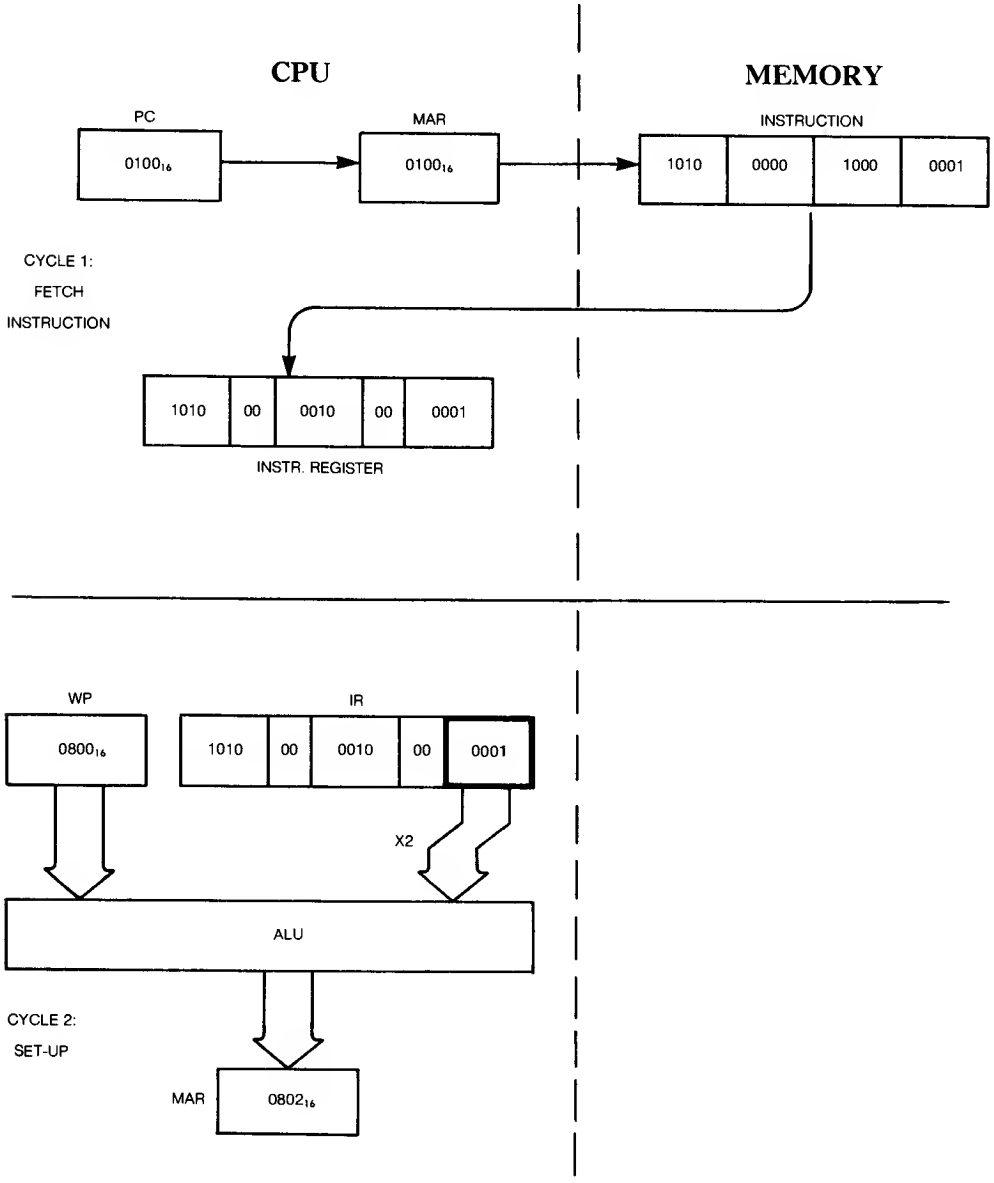


Figure 4-30a. Add Instruction Cycle

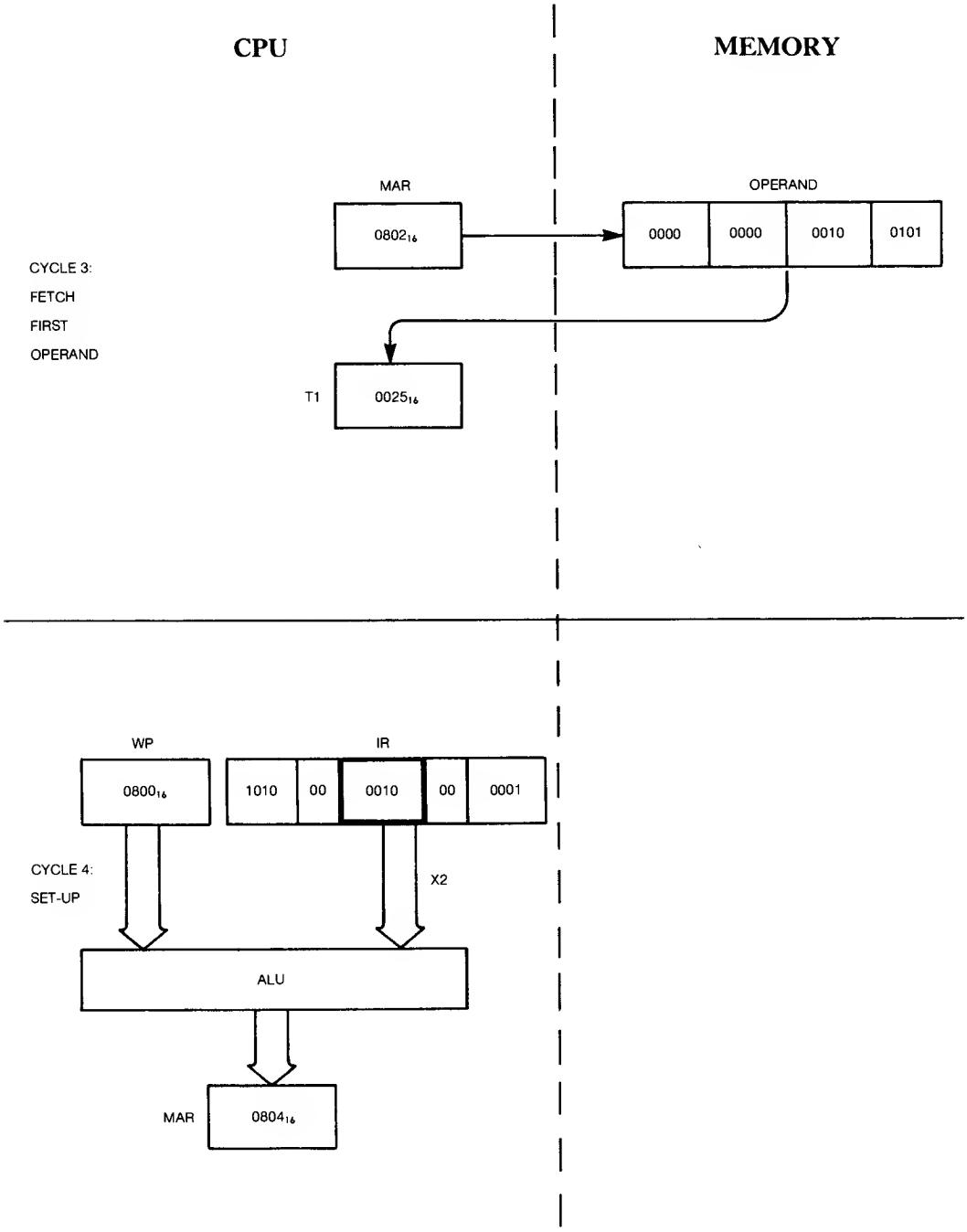


Figure 4-30b. Add Instruction Cycle

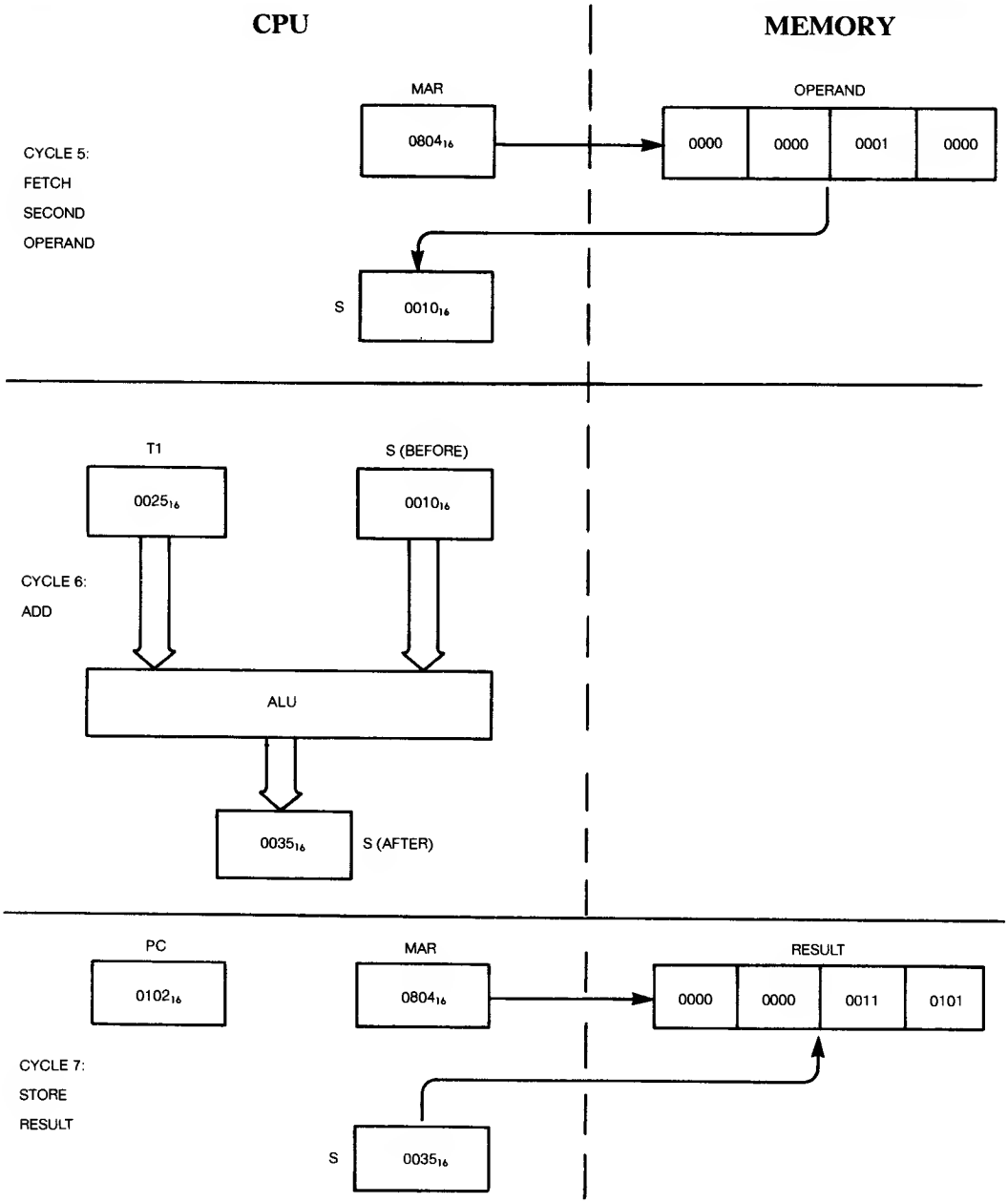


Figure 4-30c. Add Instruction Cycle

After all steps have been done, the processor checks to see if there is any pending interrupt operations to be performed and, if not, fetches the next instruction and the cycle continues. In the event that an interrupt signal were present, the processor would proceed to the appropriate interrupt service routine and continue execution from that point. Interrupts are described in detail in a special section of this chapter.

Each operation performed by the 9900 consists of a sequence of machine cycles. In each machine cycle the processor performs a data transfer with memory or with CRU and/or an arithmetic or logical operation internally with the ALU. A detailed discussion of the machine cycles for each instruction is included at the end of the chapter.

Each ALU machine cycle is two clock cycles long. In an ALU cycle no external data transfer occurs, but the ALU performs an arithmetic or logical operation on two operands contained internally. The function of the memory read cycle is to transfer a word of data contained in the memory to the processor. An ALU operation may be performed during the memory read cycle. Memory read cycles are a minimum of two clock cycles long. The memory write cycle is identical to the memory read cycle except that data is written rather than read from memory.

Each CRU output machine cycle is two clock cycles long. In addition to outputting a bit of CRU data, an ALU operation may also be performed internally. The CRU input cycle is identical to the CRU output cycle except that one bit of data is input rather than output.

Machine Cycle Limits

Table 4-1 lists information which will be useful for system design. The maximum number of consecutive memory-read cycles is used to calculate the maximum latency for the TMS 9900 to enter the hold state since the hold state is only entered from ALU, CRU input, or CRU output machine cycles. The minimum frequency of consecutive memory/non-memory cycle sequences occurs when the DIV instruction is executed. This number is used to ensure that the refresh rate meets specifications when the transparent-refresh mode described in the memory section is used since memory is refreshed in this mode each time an ALU or CRU cycle follows a memory cycle. *Figure 4-31* shows the logic to generate a pulse for each memory access cycle. Consecutive cycle timing is shown in *Figure 4-32.*

Table 4-1. Machine Cycle Limits

	MINIMUM	MAXIMUM
Consecutive Memory Read Cycles	1	3
Consecutive Memory Write Cycles	1	1
Consecutive ALU Cycles	1	51
Consecutive CRU Cycles	1	16
Frequency of Consecutive memory/non-memory cycle pairs (used for transparent refresh)	5 pairs (64 machine cycles during DIV.)	

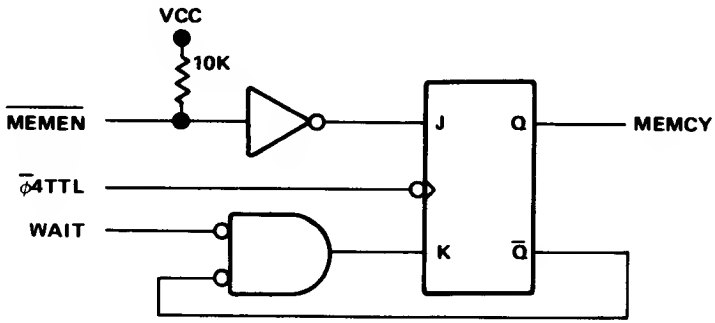


Figure 4-31. Memory Cycle Pulse Generation

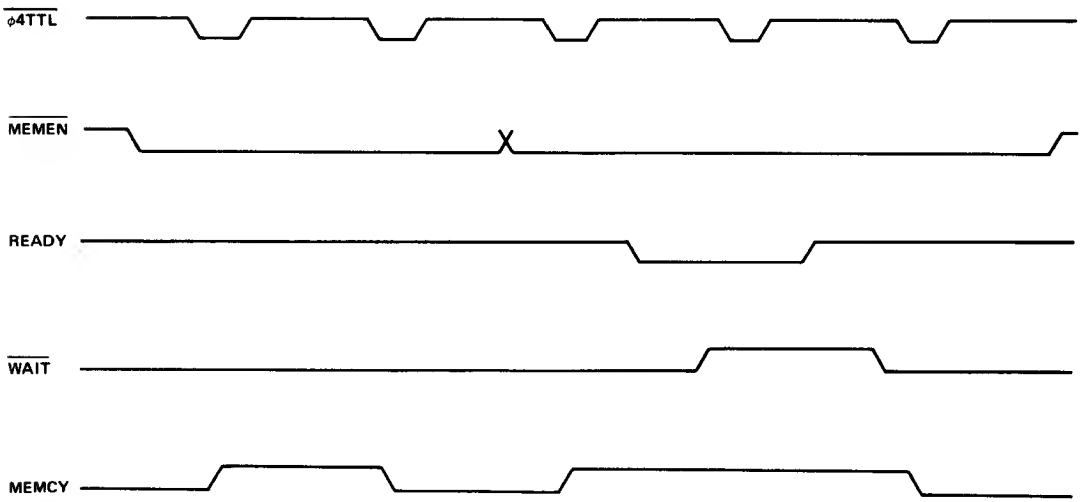
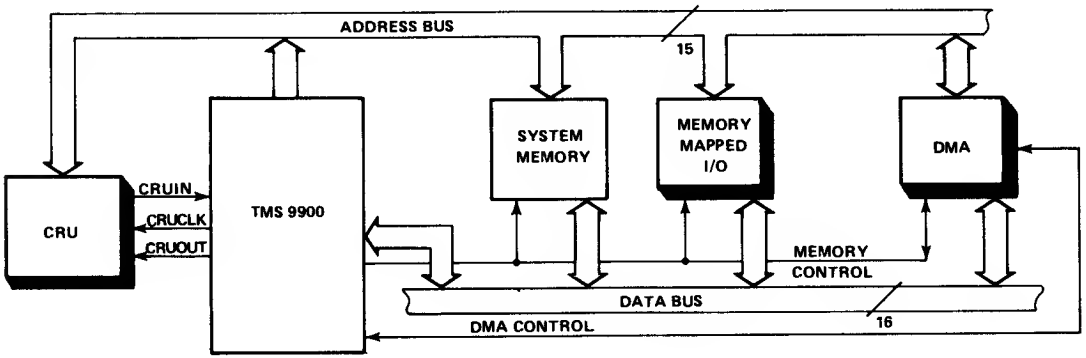


Figure 4-32. Memory Cycle Pulse Timing

INPUT/OUTPUT

The 9900 has three I/O modes: direct memory access (DMA), memory mapped, and communications register unit (CRU). This multi-mode capability enables the designer to optimize a 9900 I/O system to match a specific application. One or all modes can be used, as shown in *Figure 4-33.*



- COMMUNICATIONS REGISTER UNIT - CRU
- MEMORY MAPPED I/O
- DIRECT MEMORY ACCESS - DMA

Figure 4-33. 9900 I/O Capability

Direct Memory Access

DMA is used for high-speed block data transfer when CPU interaction is undesirable or not required. The DMA control circuitry can be relatively complex and expensive when compared to other I/O methods. However, a special interface device, the TMS 9911, is available for DMA control. This device is described in Chapter 8.

The 9900 controls CRU-based I/O transfers between the memory and peripheral devices. Data must pass through the CPU during these program-driven I/O transfers, and the CPU may need to be synchronized with the I/O device by interrupts or status-bit polling.

Some I/O devices, such as disk units, transfer large amounts of data to or from memory. Program driven I/O can require relatively large response times, high program overhead, or complex programming techniques. Consequently, direct memory access (DMA) is used to permit the I/O device to transfer data to or from memory without CPU intervention. DMA can result in a high I/O response time and system throughput, especially for block data transfers. The DMA control circuitry is somewhat more expensive and complex than the economical CRU I/O circuitry and should therefore be used only when required.

The 9900-based DMA can occur in the same modes as dynamic memory refresh: block, cycle stealing, or transparent. The transparent DMA mode is implemented similar to the refresh mode and must be synchronized with memory refresh cycles if dynamic memory is used. The block and cycle stealing modes, however, use the CPU $\overline{\text{HOLD}}$ capability and are more commonly used. The I/O device holds $\overline{\text{HOLD}}$ active (low) when a DMA transfer needs to occur. At the beginning of the next available non-memory cycle, the CPU enters the hold state and raises HOLDA to acknowledge the $\overline{\text{HOLD}}$ request. The maximum latency time between the hold request and the hold acknowledge is equal to three clock cycles plus three memory cycles. The minimum latency time is equal to one clock cycle. A 3-megahertz system with no wait cycles has a maximum hold latency of nine clock cycles or 3 microseconds and a minimum hold latency of one clock cycle or 0.3 microseconds.

When HOLDA goes high, the CPU address bus, data bus, DBIN, $\overline{\text{MEMEN}}$, and $\overline{\text{WE}}$ are in the high-impedance state to allow the I/O device to use the memory bus. The I/O device must then generate the proper address, data, and control signals and timing to transfer data to or from the memory as shown in *Figure 4-34.* Thus the DMA device has control of the memory bus when the TMS 9900 enters the hold state (HOLDA = 1), and may perform memory accesses without intervention by the microprocessor. Since DMA operations, in effect remove the 9900 from control while memory accesses are being performed, no further discussion is provided in this manual. Because the lines shown in *Figure 4-34* go into high impedance when HOLDA = 1, the DMA controller must force these signals to the proper levels. The I/O device can use the memory bus for one transfer (cycle-stealing mode) or for multiple transfers (block mode). At the end of the DMA transfer, the I/O device releases $\overline{\text{HOLD}}$ and normal CPU operation proceeds. The 9900 $\overline{\text{HOLD}}$ and HOLDA timing are shown in *Figure 4-35.*

4

Memory Mapped I/O

Memory mapped I/O permits I/O data to be addressed as memory with parallel data transfer through the system data bus. Memory mapped I/O requires a memory bus compatible interface; that is, the device is addressed in the same manner as a memory, thus the interface is identical to that of memory. *Figure 4-36* shows a memory mapped I/O interface with eight latched outputs and eight buffered inputs. In using memory mapped I/O for output only, care must be taken in developing the output device strobe to ensure it is not enabled during the initial read of the memory address, since the 9900 family of processors first reads, then writes data to a memory location in write operations. This can be effectively accomplished by using the processor write control signal $\overline{\text{WE}}$ in decoding the output address.

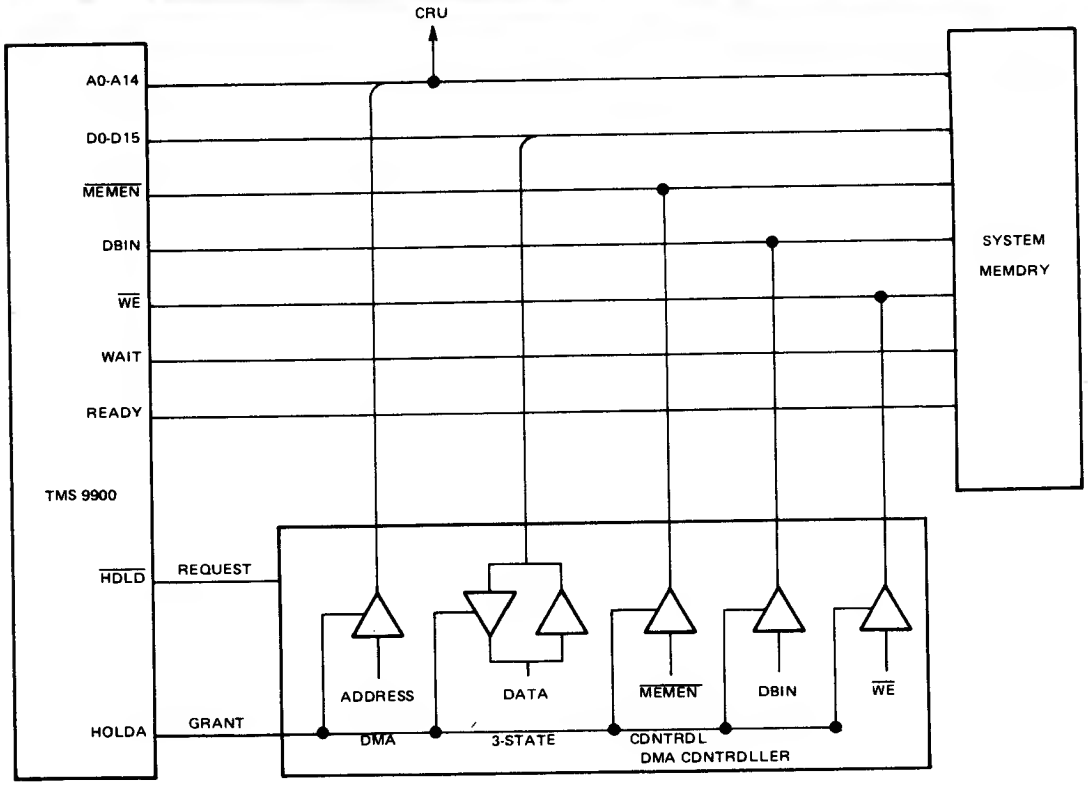


Figure 4-34. DMA Bus Control

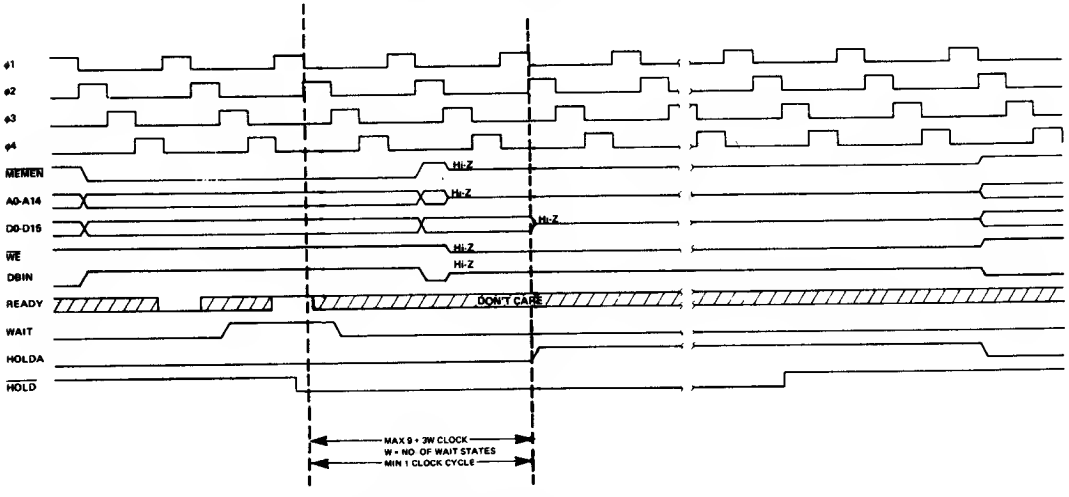


Figure 4-35. $\overline{HOLD}$ and HOLDA Timing

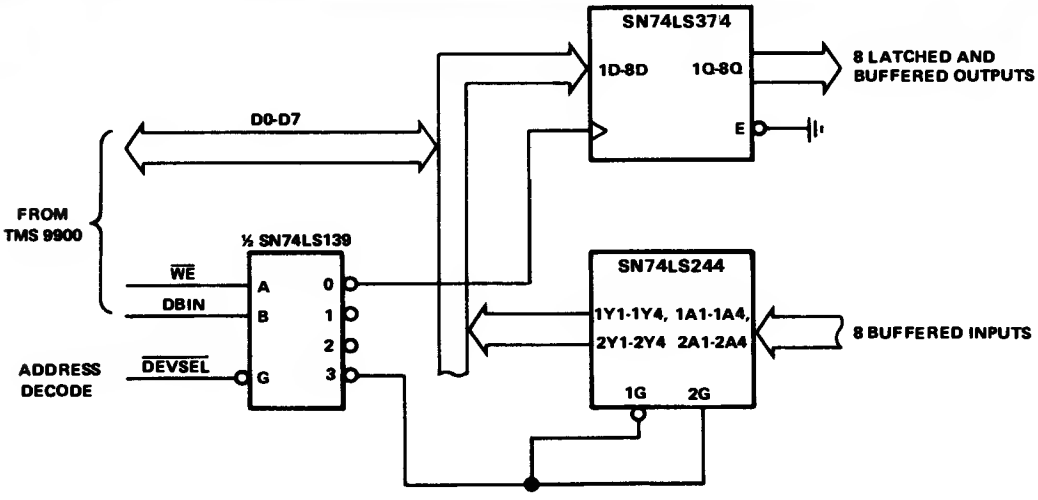


Figure 4-36. 8-Bit Memory Mapped I/O Interface

Communication Register Unit (CRU)

CRU I/O uses a dedicated bit addressable interface for I/O. The CRU instructions permit transfer of one to sixteen bits. The CRU interface requires fewer interface signals than the memory interface and can be expanded without affecting the memory system. In the majority of applications, CRU I/O is superior to memory mapped I/O as a result of the powerful bit manipulation capability, flexible field lengths, and simple bus structure.

The CRU bit manipulation instructions eliminate the masking instructions required to isolate a bit in memory mapped I/O. The CRU multiple-bit instructions allow the use of I/O fields not identical to the memory word size, thus permitting optimal use of the I/O interface. Therefore, the CRU minimizes the size and complexity of the I/O control programs, while increasing system throughput.

The CRU does not utilize the memory data bus. This can reduce the complexity of printed circuit board layouts for most systems. The standard 16-pin CRU I/O devices are less expensive and easier to insert than larger, specially designed, memory mapped I/O devices. The smaller I/O devices are possible as a result of the bit addressable CRU bus which eliminates the need for multiple pins dedicated to a parallel-data bus with multiple control lines. System costs are lower because of simplified circuit layouts, increased density, and lower component costs.

CRU Interface

The interface between the 9900 and CRU devices consists of address bus lines A0-A14, and the three control lines, CRUIN, CRUOUT, and CRUCLK as shown in *Figure 4-33.* A0-A2 indicate whether data is to be transferred and A3-A14 contain the address of the selected bit for data transfers; therefore, up to 2^{12} or 4,096 bits of input and 4,096 bits of output may be individually addressed. CRU operations and memory-data transfers both use A0-A14; however, these operations are performed independently, thus no conflict arises. The $\overline{\text{MEMEN}}$ line may be used to distinguish between CRU and memory cycles.

CRU Interface Logic

CRU based I/O interfaces are easily implemented using either CRU peripheral devices such as the TMS 9901 or the TMS 9902, or TTL multiplexers and addressable latches, such as the TIM 9905 (SN74LS251) and the TIM 9906 (SN74LS259). These I/O circuits can be easily cascaded with the addition of simple address decoding logic.

▶4 *TTL Outputs.* The TIM 9906 (SN74LS259) octal-addressable latch can be used for CRU outputs. The latch outputs are stable and are altered only when the CRUCLK is pulsed during a CRU output transfer. Each addressable latch is enabled only when addressed as determined by the upper address bits. The least-significant address bits (A12-A14) determine which of the eight outputs of the selected latch is to be set equal to CRUOUT during CRUCLK, and shown in *Figure 4-37.*

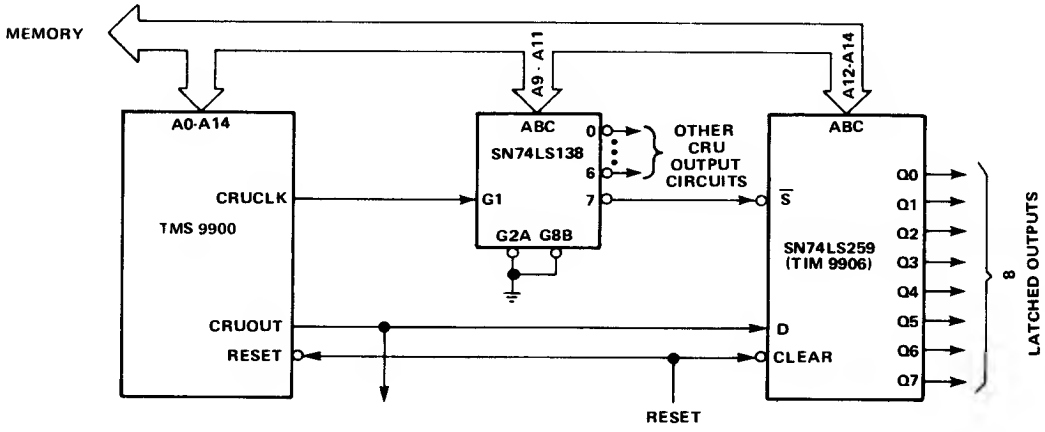


Figure 4-37. Latched CRU Interface

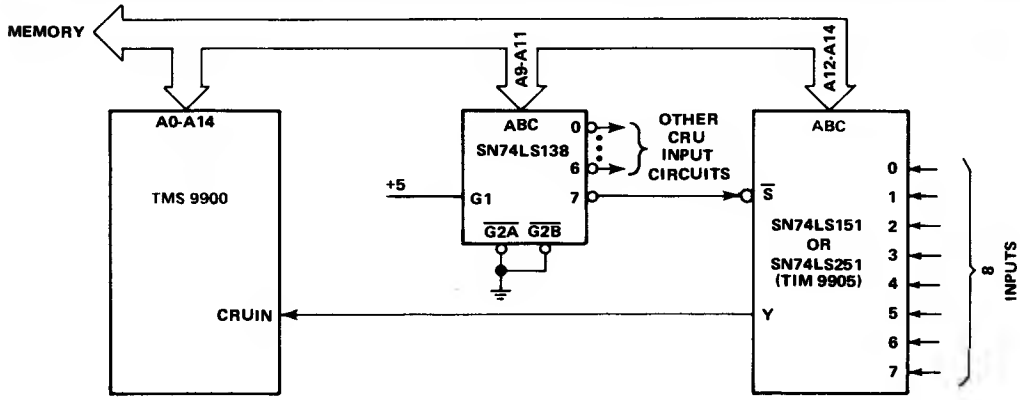


Figure 4-38. Multiplexer CRU Interface

4◀

TTL Inputs. The SN74LS151 and TIM 9905 (SN74LS251) octal multiplexers are used for CRU inputs as shown in *Figure 4-38.* The multiplexers are continuously enabled with CRUIN equal to the addressed input. The TIM 9905 should be used for larger systems since its three-state outputs permit simple "wire-ORing" of parallel-input multiplexers.

Expanding CRU I/O

A CRU interface with eight inputs and eight outputs is shown in *Figure 4-39* using the TMS 9901. An expanded interface with 16 inputs and 16 outputs is shown in *Figure 4-40* using TTL devices. The CRU inputs and outputs can be expanded up to 4096 inputs and 4096 outputs by decoding the complete CRU address. Larger I/O requirements can be satisfied by using memory mapped I/O or by using a CRU bank switch, which is set and reset under program control. When reset, the lower CRU I/O bank is selected, and when set, the upper CRU I/O bank is selected. In actual system applications, however, only the exact number of interface bits required need to be implemented. It is not necessary to have a 16-bit CRU output register to interface a 10-bit device.

CRU Machine Cycles

Each CRU operation consists of one or more CRU output or CRU input machine cycles, each of which is two clock cycles long. As shown in *Table 4-2,* five instructions (LDCR, STCR, SBO, SBZ, TB) transfer data to or from the 9900 with CRU machine cycles, and five external control instructions (IDLE, RSET, CKOF, CKON, LREX) generate control signals with CRU output machine cycles.

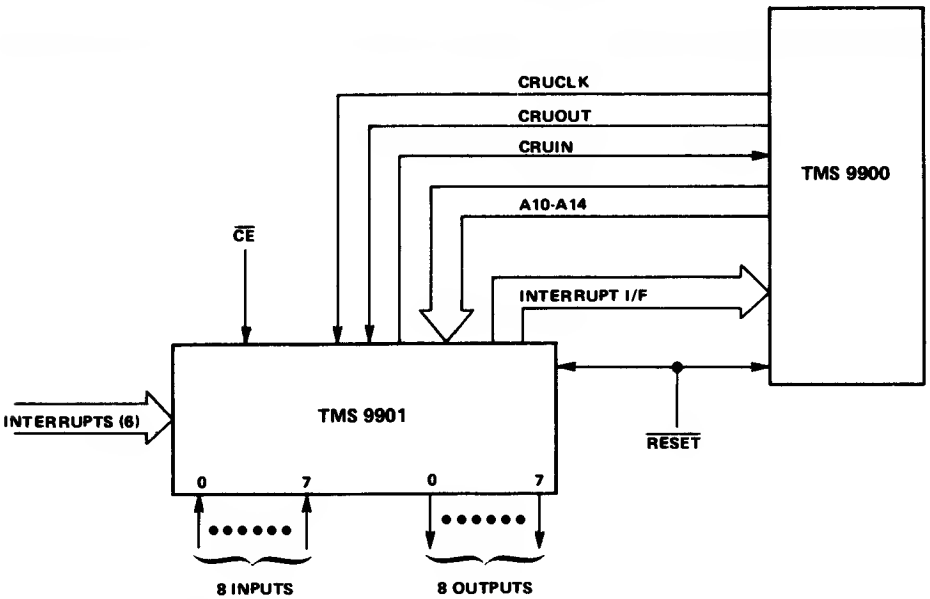


Figure 4-39. 8-Bit CRU Interface

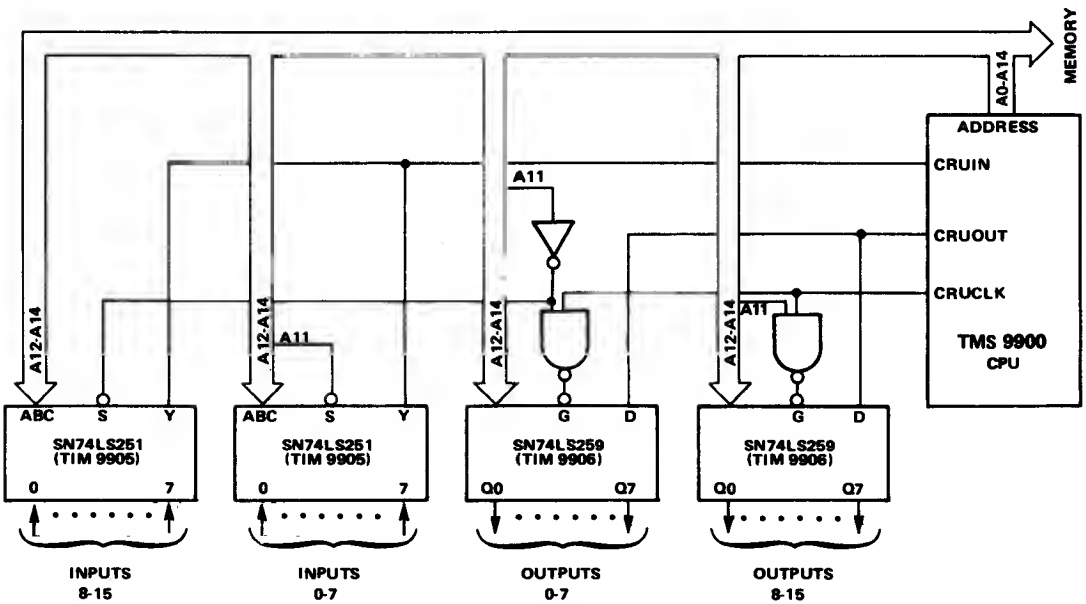


Figure 4-40. 16-Bit CRU Interface

Table 4-2. Instructions Generating CRU Cycles

INSTRUCTION	NUMBER OF CRU CYCLES	TYPE OF CRU CYCLES	A0-A2	DATA TRANSFER
LDCR	1-16	Output	0 0 0	Yes
STCR	1-16	Input	0 0 0	Yes
SBO	1	Output	0 0 0	Yes
SBZ	1	Output	0 0 0	Yes
TB	1	Input	0 0 0	Yes
IDLE	1	Output	0 1 0	No
RSET	1	Output	0 1 1	No
CKOF	1	Output	1 0 1	No
CKON	1	Output	1 1 0	No
LREX	1	Ouput	1 1 1	No

Figure 4-41 shows the timing for CRU *output* machine cycles. Address (A0-A14) and data (CRUOUT) are output on $\phi 2$ of clock cycle 1. One clock cycle later, the 9900 outputs a pulse on CRUCLK for ½ clock cycle. Thus, CRUCLK can be used as a strobe, since address and data are stable during the pulse. Referring again to *Table 4-2,* it is important to note that output data is transferred only when A0-A2 = 000. Otherwise, no data transfer should occur, and A0-A2 should be decoded to determine which external control instruction is being executed. These external control instructions may be used to perform simple control operations. The generation of control strobes for external instructions and a data transfer strobe (OUTCLK) is illustrated in *Figure 4-42.* If none of the external control instructions is used, A0-A2 need not be decoded for data transfer since they will always equal 000.

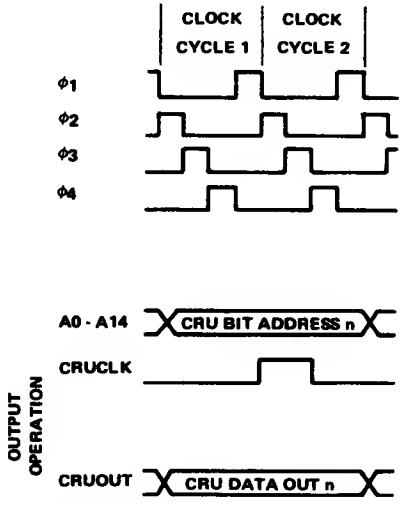


Figure 4-41. CRU Output Machine Cycle Timing

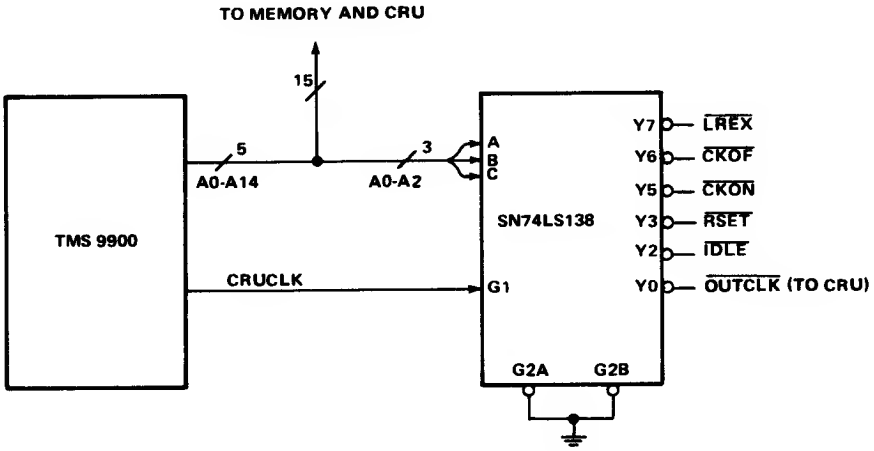


Figure 4-42. CRU Control Strobe Generation

The timing for CRU *input* machine cycles is shown in *Figure 4-43.* The address is output at the beginning of the first clock cycle. The CRUIN data input is sampled on $\phi 1$ of clock cycle 2. Thus, CRU input is accomplished by simply multiplexing the addressed bit onto the CRUIN input. A0-A2 will always be 000, and may be ignored. CRU input machine cycles cannot be differentiated from ALU cycles by external logic, thus no operations (such as clearing interrupts) other than CRU input should be performed during CRU input machine cycles.

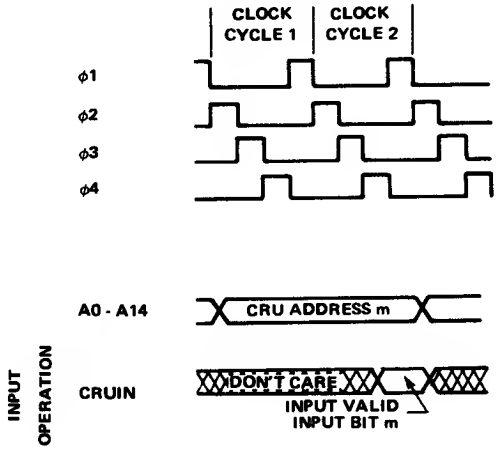


Figure 4-43. CRU Input Machine Cycle Timing

CRU Data Transfer

In order to transfer data from a memory location to an external latch in the Communications Register Unit, or to transfer data from a CRU multiplexer to memory, special instructions must be used. The CRU instructions are:

SBO	Set bit to one (output)
SBZ	Set bit to zero (output)
TB	Test bit (input)
LDCR	Load n bits to CRU (output)
STCR	Receive n bits from CRU (input)

These instructions always use the address bus to identify the bit or bits to be transferred, but they make the actual transfer of data over the dedicated CRU lines, CRUIN and CRUOUT. Addressing of the CRU bits is accomplished by adding a portion of the instruction word to a CRU base address register. The use of such a base address technique allows one program segment to service any number of identical I/O devices. For example: five TMS 9902's each with its own assigned base address can be operated from a single program, provided the base address register is properly set at the beginning. In the 9900, workspace register 12 is the CRU software base address register. All CRU instructions use the contents of this register in addressing individual CRU bits.

The CRU hardware base address is defined by bits 3-14 of the current WR12 when CRU data transfer is performed. Bits 0-2 and bit 15 of WR12 are ignored for CRU address determination.

For single-bit CRU instructions (SBO, SBZ, TB), the address of the CRU bit to or from which data is transferred is determined as shown in *Figure 4-44.* Bits 8-15 of the machine code instruction contain a signed displacement. This signed displacement is added to the CRU hardware base address (bits 3-14 of WR12). The result of this addition is output on A3-A14 during the CRU output or the CRU input machine cycle.

For example, assume the instruction "SBO 9" is executed when WR12 contains a value of 1040_{16}. The machine code for "SBO 9" is $1D09_{16}$ and the signed displacement is 0009_{16}. The CRU hardware base address is 0820_{16} (bits 0-2 and bit 15 are ignored). Thus, the effective CRU bit address is $0820_{16} + 0009_{16} = 0829_{16}$, and this value is output on A0-A14 during the CRU output machine cycle.

As a second example, assume that the instruction TB −32 is executed when WR12 = 100_{16}. The effective CRU address is 80_{16}. (CRU hardware base) + $FFEO_{16}$ (signed displacement) = 60_{16}. Thus, the TB −32 instruction in this example causes the value of the CRU input bit at address 60_{16} to be transferred to bit 2 of the status register. This bit is tested in the execution of the JEQ or JNE instructions; if it is a one, the PC will be loaded with a new value (JEQ instruction).

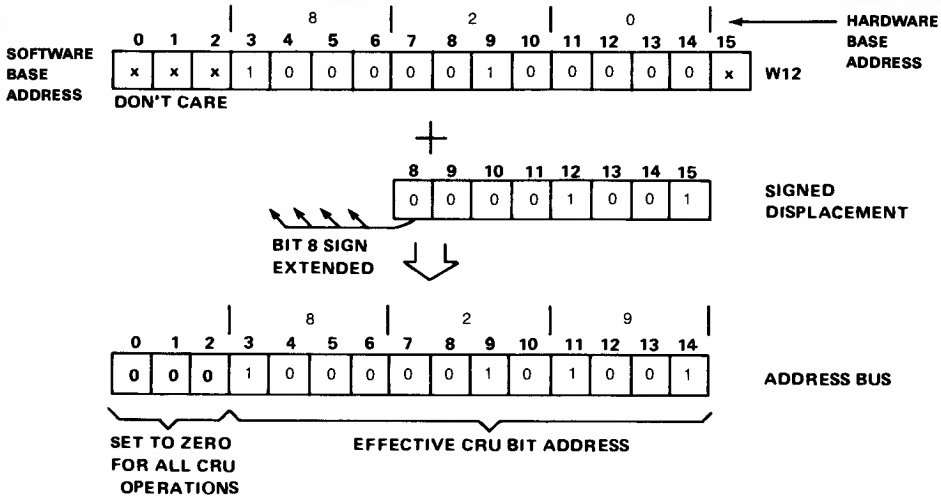


Figure 4-44. TMS 9900 Single-Bit CRU Address Development

▶4

LDCR Instruction

The LDCR may transfer from 1 to 16 fits of output data with each instruction. Output of each bit is performed by a CRU output machine cycle; thus, the number of CRU output machine cycles performed by an LDCR instruction is equal to the number of bits to be transferred.

As an example, assume that the instruction "LDCR @600,10" is executed, and that WR12 = 800_{16} and the memory word at address 600 contains the bit pattern shown in *Figure 4-45.* In the first CRU output machine cycle the least significant bit of the operand (a) is output on CRUOUT. In each successive machine cycle the address is incremented by one and the next least-significant bit of the operand is output on CRUOUT, until 10 bits have been output. It is important to note that the CRU base address is unaltered by the LDCR instruction, even though the address is incremented as each successive bit is output.

STCR Instruction

The STCR instruction causes from 1 to 16 bits of CRU data to be transferred into memory. Each bit is input by a CRU input machine cycle.

Consider the circuit shown in *Figure 4-46.* The CRU interface logic multiplexes input signals m-t onto the CRUIN line for addresses 200_{16}-207_{16}. If WR12 = 400_{16} when the instruction "STCR @ 602,6" is executed, the operation is performed as shown in *Figure 4-47.* At the end of the instruction, the six LSBs of memory byte 602 are loaded with m-r. The upper bits of the operand are forced to zero.

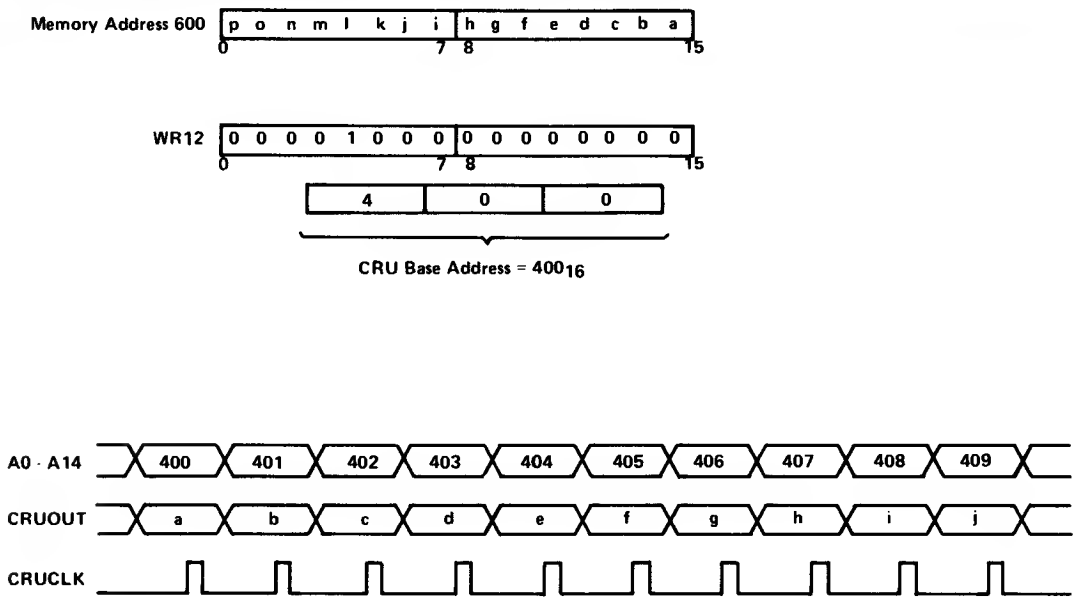


Figure 4-45. Multiple-Bit CRU Output

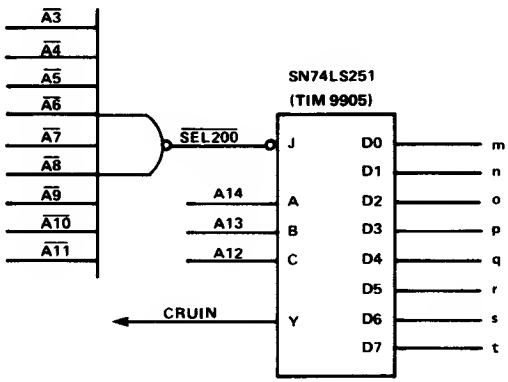


Figure 4-46. Example CRU Input Circuit

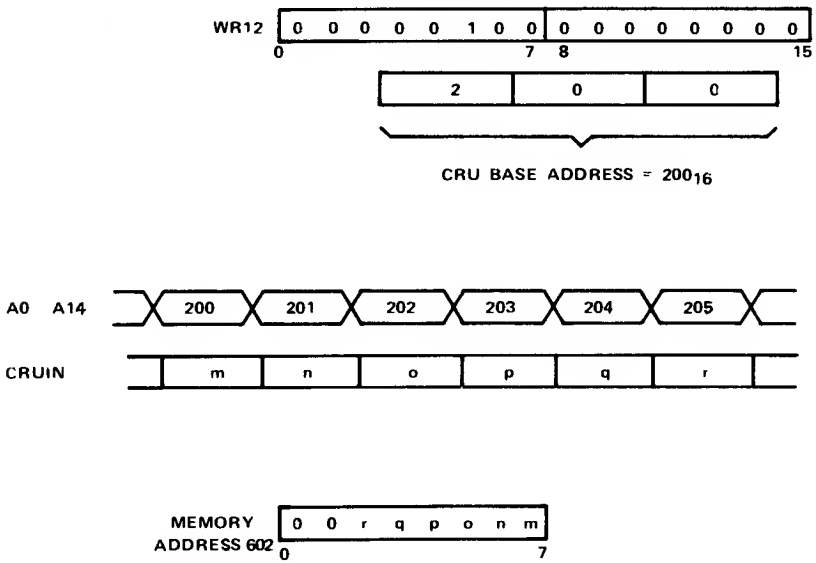


Figure 4-47. Multiple-Bit CRU Input

CRU Paper Tape Reader Interface

CRU interface circuits are used to interface data and control lines from external devices to the 9900. This section describes an example interface from a paper tape reader. The paper tape reader is assumed to have the following characteristics:

1. It generates a TTL-level active-high signal (SPROCKET HOLE) on detection of a sprocket hole on the paper tape.
2. It generates an 8-bit TTL active-low data which stays valid during SPROCKET HOLE = 1.
3. It responds to a TTL-level active-high command (Paper Tape RUN) signal by turning on when PTRUN = 1 and turning off when PTRUN = 0.

Figure 4-48 illustrates the circuitry to interface the reader to the CRU. The interface is selected when $\overline{PTRSEL}$ = 0; $\overline{PTRSEL}$ is decoded from the A0-A11 address outputs from the 9900. Thus, the output of the SN74LS251 is active only when $\overline{PTRSEL}$ = 0; otherwise, the output is in high impedance and other devices may drive CRUIN. The data inputs are selected by A12-A14 and inverted, resulting in active high data input on CRUIN. The positive transition of SPROCKET HOLE causes $\overline{PTRINT}$ to go low. $\overline{PTRINT}$ is the active low interrupt from the interface. $\overline{PTRINT}$ is set high, clearing the interrupt, whenever a CRU output machine cycle is executed and the address causes $\overline{PTRSEL}$ to be active. When a one is output, $\overline{PTRUN}$ is set, enabling the reader, and the reader is disabled when a zero is output to the device. Thus, any time $\overline{PTRUN}$ is set or reset, the interrupt is automatically cleared.

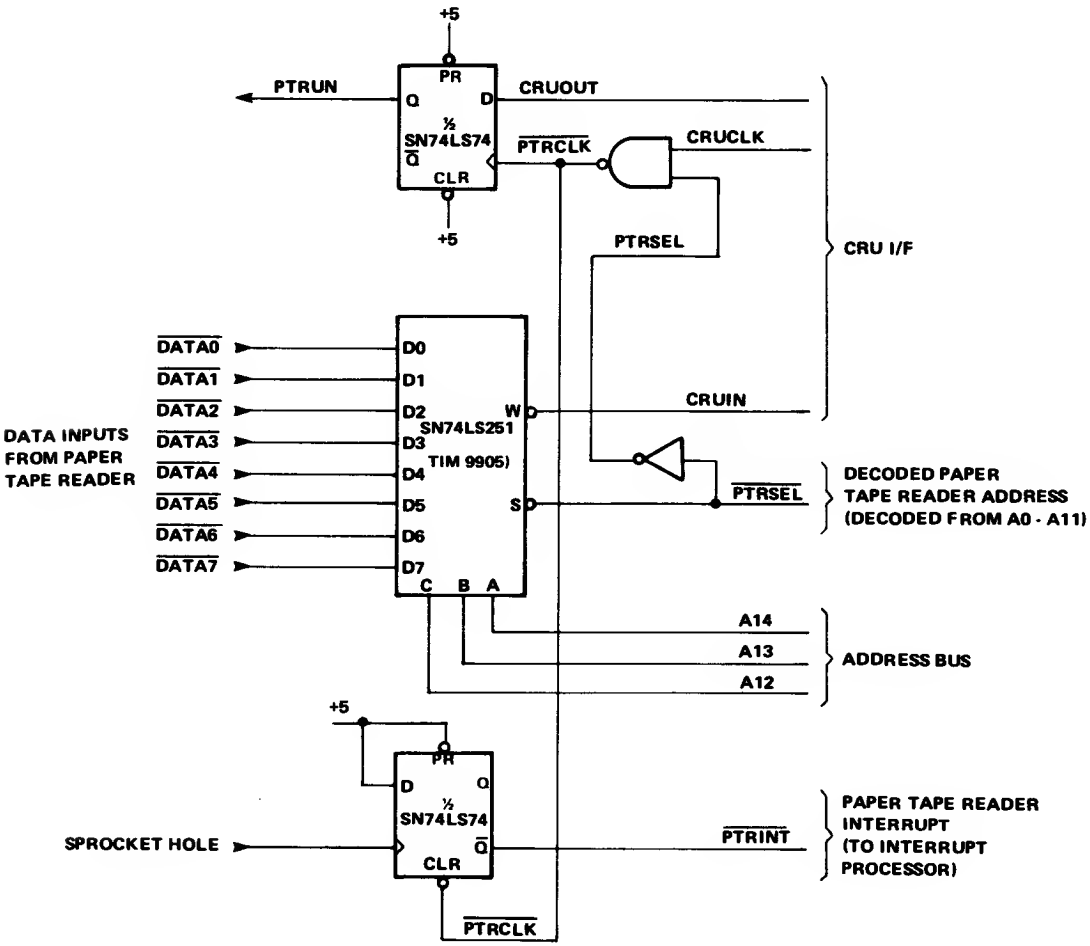


Figure 4-48. Paper Tape Reader Interface

The software routine in *Figure 4-49* controls the paper-tape reader interface described above. It is a re-entrant procedure that can be shared by several readers. The assumptions are that:

1. Each reader has its own workspace which is set up on the trap location for that reader's interrupt.
2. The workspace registers are allocated as shown in *Figure 4-50.*
3. The CRU input bits 0-7 (relative to CRU base) are reader data. CRU output bit 0 controls PTRUN and clears the interrupt.
4. The most significant byte of R9 = End of File Code.
5. R10 = Overflow Count
6. R11 = Data Table Pointer Address.

The procedure has two entry points. It is entered by a calling routine at PTRBEG to
▶4 start the reader and it returns control to that routine. It is entered at PTRINT via interrupt to read a character. The return in this case is to the interrupted program.

The control program may be used by any number of paper-tape reader interfaces, as long as each interface has a separate interrupt level and workspace. As each reader issues an interrupt, the 9900 will process the interrupt beginning at location PTRINT. However, the workspace unique to the interrupting device is used. The organization of memory to control two paper tape readers is shown in *Figure 4-50.* The interrupt-transfer vector causes the appropriate WP value to be loaded. In both cases PTRINT, the entry point for the control program, is loaded into the PC.

```
PTRINT      STCR     *R11,8
            CB       *R11+,R9
            JEQ      PTREND
            DEC      R10
            JEQ      PTREND
PTRBEG      SBO      PTRUN
            RTWP
PTREND      SBZ      PTRUN
            LI       R10,MAXCOUNT
            RTWP
```

Figure 4-49. Paper Tape Reader Control Program

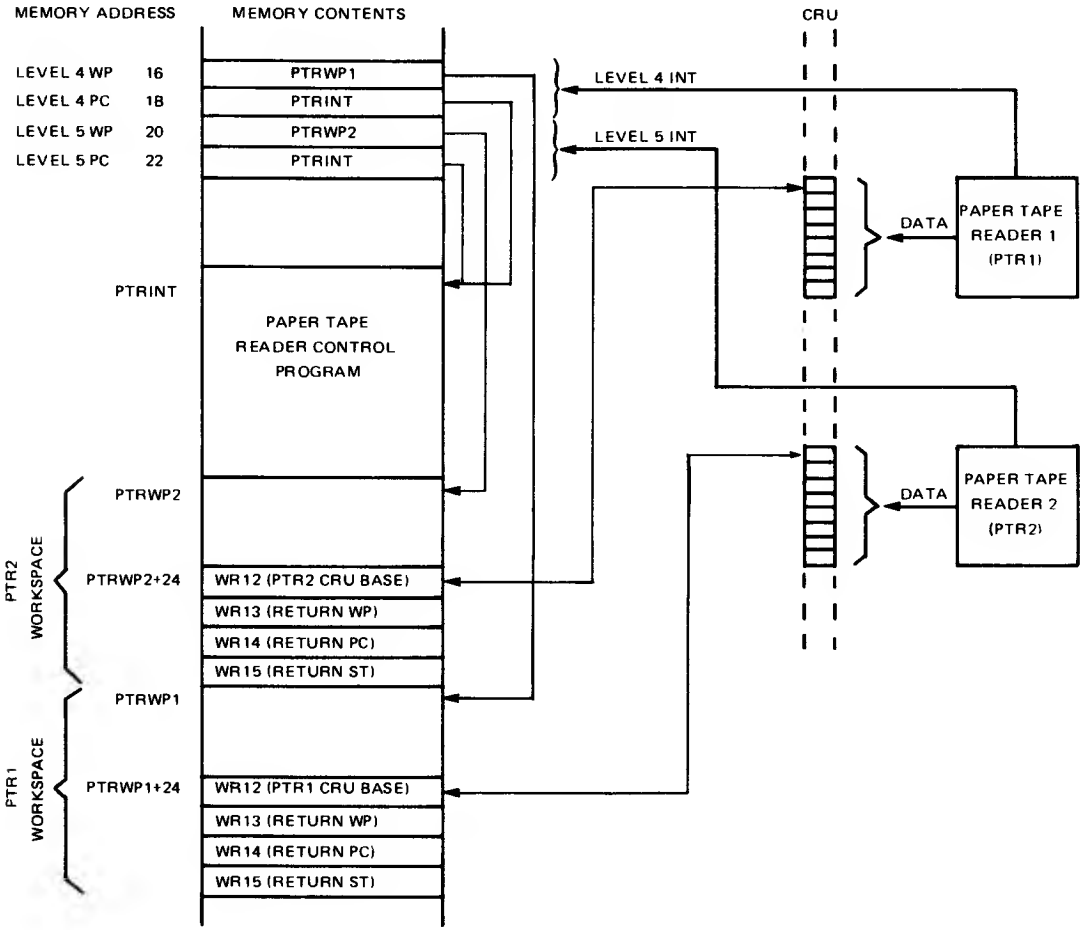


Figure 4-50. Software Configuration for Two Paper Tape Readers with Common Control Program

Burroughs SELF-SCAN Display Interface

This section describes a TMS9900 CRU interface to a Burroughs SELF-SCAN® panel display model SS30132-0070. The display panel has a 32-position, single-row character array with a repertoire of 128 characters.

The panel display operates in a serial-shift mode in which characters are shifted into the panel one at a time. Characters are shifted in right-to-left and can be shifted or backspaced left-to-right. A clear pulse erases the display.

The CRU display interface is shown in *Figure 4-51* and a display control subroutine is shown in *Figure 4-52*. The subroutine is called by one of two XOP instructions, XOP0 and XOP1. The calling routine passes the address and length of the output string in registers 8 and 9 of its workspace. The two XOP subroutines share the same workspace and perform the same function except that XOP1 clears the panel display first. The backspace feature is not used. The panel display is blanked during character entry.

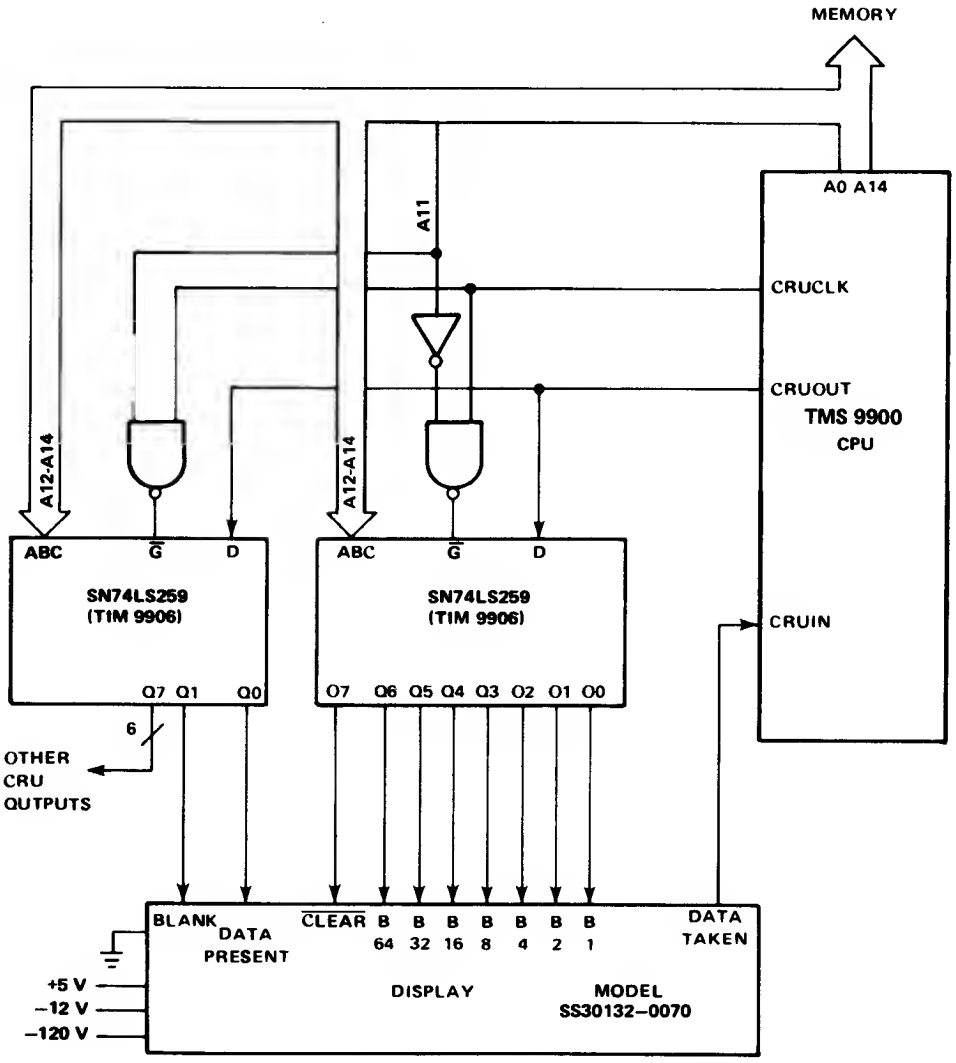


Figure 4-51. Display Control Interface

```
EIGHT    EQU    16
NINE     EQU    18

RXOP1    SBZ    7                 Clear Panel
         LI     R1,11

LOOP1    DEC    R1                Delay >67µsec
         JNE    LOOP1
         SBO    7
RXOP2    SBO    9                 Blank Panel
         MOV    @EIGHT (13),1     Load Address (Old R8→R1)
         MOV    @NINE (13),2      Load Length (Old R9→R2)

LOOP2    LDCR   *1+,7             Output Char
         SBO    8                 Data Present

WAIT     TB     0                 Wait for Data Taken
         JEQ    Wait
         SBZ    8
         DEC    2                 Decrement Count
         JNE    LOOP2             Loop Until Through
         SBZ    9                 Unblank Panel
         RTWP                     Return
```

Figure 4-52. Burroughs SELF-SCAN® Display Control Program

INTERRUPTS

The TMS 9900 provides fifteen maskable interrupt levels in addition to the $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ functions. The CPU has a priority ranking system to resolve conflicts between simultaneous interrupts and a level mask to disable lower priority interrupts. Once an interrupt is recognized, the CPU performs a vectored context switch to the interrupt service routine. The $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ functions are initiated by external input signals.

Reset

The $\overline{\text{RESET}}$ signal is normally used to initialize the CPU following a power-up. When active (low), the $\overline{\text{RESET}}$ signal inhibits $\overline{\text{WE}}$ and CRUCLK, places the CPU memory bus and control signals in a high-impedance state, and resets the CPU. When the $\overline{\text{RESET}}$ signal is released, the CPU fetches the restart vector from locations 0000 and 0002, stores the old WP, PC, and ST into the new workspace, resets all status bits to zero and starts execution at the new PC. The $\overline{\text{RESET}}$ signal must be held active for a minimum of three clock cycles. The $\overline{\text{RESET}}$ machine cycle sequence is shown in *Figure 4-53.*

A convenient method of generating the $\overline{\text{RESET}}$ signal is to use the Schmitt-triggered D-input of the TIM9904 clock generator. An RC network connected to the D-input maintains an active $\overline{\text{RESET}}$ signal for a short time immediately following the power-on, as shown in *Figure 4-54.*

▸4

CYCLE	TYPE	FUNCTION
*	*	Loop While Reset is Active
1	ALU	Set Up
2	ALU	Set Up
3	Memory	Fetch New WP, Move Status To T Reg, Clear Status
4	ALU	Set Up
5	Memory	Store Status
6	ALU	Set Up
7	Memory	Store PC
8	ALU	Set Up
9	Memory	Store WP
10	ALU	Set Up
11	Memory	Fetch New PC
12	ALU	Set Up MAR for Next Instruction

Figure 4-53. RESET Machine Cycles

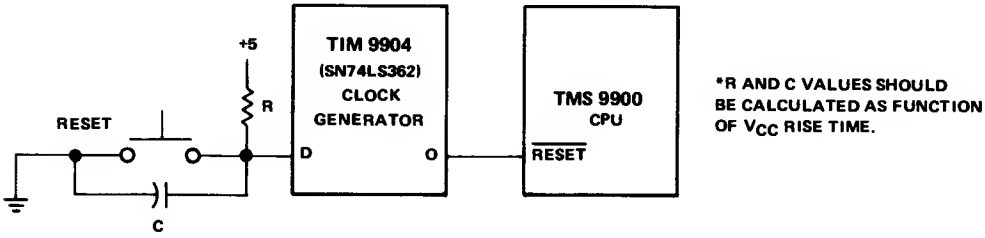


Figure 4-54. $\overline{RESET}$ Generation

4◀

LOAD

The $\overline{LOAD}$ signal is normally used to implement a restart ROM loader or front panel functions. When active (low), the $\overline{LOAD}$ signal causes the CPU to perform a non-maskable interrupt. The $\overline{LOAD}$ signal can be used to terminate a CPU idle state.

The $\overline{LOAD}$ signal should be active for one instruction period. Since there is no standard TMS 9900 instruction period, IAQ should be used to determine instruction boundaries. If the $\overline{LOAD}$ signal is active during the time that the $\overline{RESET}$ signal is released, the CPU will perform the $\overline{LOAD}$ function immediately after the $\overline{RESET}$ function is completed. The CPU performs the $\overline{LOAD}$ function by fetching the $\overline{LOAD}$ vector from addresses $FFFC_{16}$ and $FFFE_{16}$, storing the old WP, PC, and ST in the new workspace, and starting the $\overline{LOAD}$ service routine at the new PC, as shown in *Figure 4-55.*

An example of the use of the $\overline{LOAD}$ signal is a bootstrap ROM loader. When the $\overline{LOAD}$ signal is enabled, the CPU enters the service routine, transfers a program from peripheral storage to RAM, and then transfers control to the loaded program.

Figure 4-56 illustrates the generation of the $\overline{LOAD}$ signal for one instruction period.

CYCLE	TYPE	FUNCTION
1	ALU	Set Up
2	Memory Read	Fetch New WP
3	ALU	Set Up
4	Memory Write	Store Status
5	ALU	Set Up
6	Memory Write	Store PC
7	ALU	Set Up
8	Memory Write	Store WP
9	ALU	Set Up
10	Memory Read	Fetch New PC
11	ALU	Set UP MAR for Next Instruction

Figure 4-55. $\overline{LOAD}$ Machine Cycle Sequence

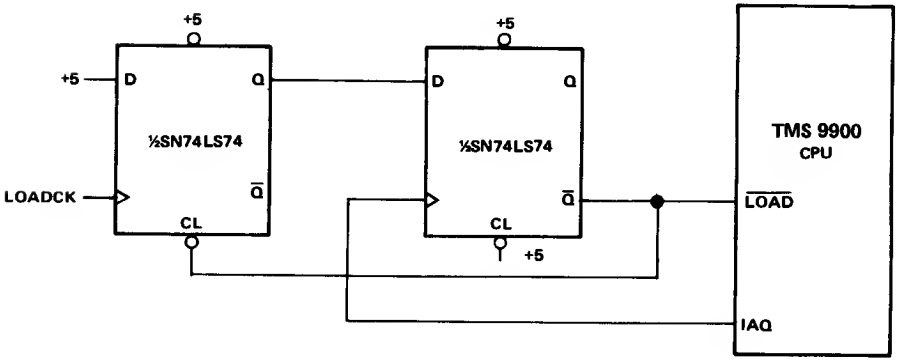


Figure 4-56. $\overline{LOAD}$ Generation

Basic Machine Cycle

The interrelationship between the $\overline{LOAD}$ and $\overline{RESET}$ signals and the general operation of the 9900 and execution of instructions may best be shown by the flow diagram in *Figure 4-57.* An orderly starting procedure involves the holding of the $\overline{RESET}$ line low when power is applied to the chip. After application of power and after the clock has begun to run, the internal instruction control circuitry checks to see if the $\overline{RESET}$ line is held low, and, if the answer is "yes", will stay in a loop as shown in the diagram. When the $\overline{RESET}$ line goes high, it is no longer active and a level zero interrupt is taken in which the $\overline{RESET}$ vector, the numbers to fill the workspace pointer and program counter registers, are fetched from memory locations zero and two. Furthermore, the previous values of the workspace pointer, program counter and status register are stored in the new workspace, although these values are random numbers immediately following power up. Following this, the interrupt mask is set to zero to mask all other interrupts.

The next decision is regarding the $\overline{LOAD}$ line. If this particular line is active, or low, then immediately there will be another context switch in which the $\overline{LOAD}$ vector will be brought in from the last two locations in memory, $FFFC_{16}$ and $FFFE_{16}$, and loaded into the workspace pointer and program counter respectively. If the $\overline{LOAD}$ is not active, the 9900 proceeds directly to an instruction acquisition cycle. In either case, the very next step is to fetch the instruction from the memory and execute it.

Following this, the program counter is updated and a sequence of checks made regarding the $\overline{LOAD}$, XOP, and interrupt conditions. First is the check for the $\overline{LOAD}$ line. If this is active, the $\overline{LOAD}$ context switch will occur. If not, there will be a test to see if the instruction just executed was an XOP or BLWP. If not, the interrupt request line will be checked. If there is not an interrupt request, and the last instruction was not an idle instruction, the machine may proceed to fetch the next instruction and continue.

In the event that the last instruction executed was an XOP or BLWP, the 9900 will ignore the interrupt request line and will proceed to fetch a new instruction. This insures that at least one instruction of a subprogram that is entered via a context switch will be executed before another context switch may occur, such as an interrupt. In the event that the interrupt request line is active following the execution of a normal instruction, a test is made to determine that the interrupt is valid, that is to say, "Is the interrupt mask set to allow this interrupt." If the interrupt is not allowed, the processor proceeds to fetch the next instruction. In the event that it is allowed, a context switch will be made and the interrupt vector from the appropriate locations in the first 32 words of memory will be fetched and the workspace pointer and program counter will be loaded with the new numbers. As a part of this context switch, the interrupt mask is set to a level one less than the interrupt just taken. This is to insure that no lower priority interrupt may occur during the servicing of the current interrupt cycle. Notice further that in this diagram that the logic is such that at least one instruction of any subprogram will be

executed immediately following a context switch. The only exception to this is the simultaneous presence of $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ signals. Finally, the idle instruction will suspend instruction execution in the 9900 until an interrupt, $\overline{\text{RESET}}$ or $\overline{\text{LOAD}}$ signal occurs.

Maskable Interrupts

The TMS 9900 has 16 interrupt levels with the lower 15 priority levels used for maskable interrupts. The maskable interrupts are prioritized and have transfer vectors similar to the $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ vectors.

Interrupt Service

A pending interrupt of unmasked priority level is serviced at the end of the current instruction cycle with two exceptions. The first instruction of a $\overline{\text{RESET}}$, $\overline{\text{LOAD}}$, or interrupt service routine is executed before the CPU tests the $\overline{\text{INTREQ}}$ signal. The interrupt is also inhibited for one instruction if the current instruction is a branch and load workspace pointer instruction (BLWP) or an extended operation (XOP). The one instruction delay permits one instruction to be completed before an interrupt context switch can occur. A LIMI instruction can be used as the first instruction in a routine to lock out higher priority maskable interrupts.

The pending interrupt request should remain active until recognized by the CPU during the service routine. The interrupt request should then be cleared under program control. The CRU bit manipulation instructions can be used to recognize and clear the interrupt request.

The interrupt context switch causes the interrupt vector to be fetched, the old WP, PC, and ST to be saved in the new workspace, and the new WP and PC to be loaded. Bits 12-15 of ST are loaded with a value of one less than the level of the interrupt being serviced. The old WP, PC, and ST are stored in the new workspace registers 13, 14, and 15. When the return instruction is executed, the old WP, PC, and ST are restored to the CPU. Since the ST contains the interrupt mask, the old interrupt level is also restored. Consequently, all interrupt service routines should terminate with the return instruction in order to restore the CPU to its state before the interrupt.

The linkage between two interrupt service routines is shown in *Figure 4-58* and the interrupt machine cycle sequence is shown in *Figure 4-59*.

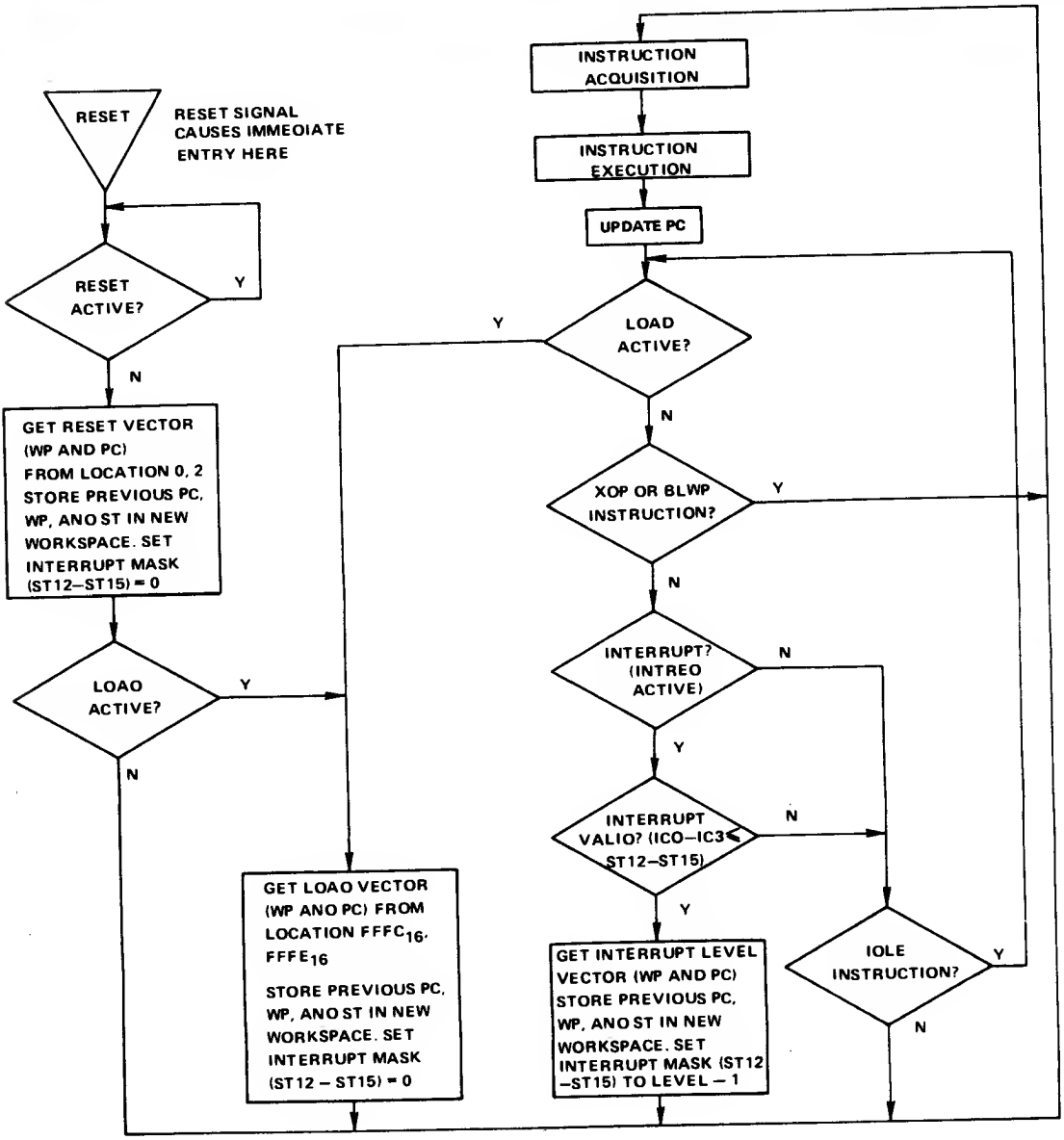


Figure 4-57. TMS 9900 CPU Flow Chart

Interrupt Signals

The TMS 9900 has five inputs that control maskable interrupts. The $\overline{\text{INTREQ}}$ signal is active (low) when a maskable interrupt is pending. If $\overline{\text{INTREQ}}$ is active at the end of the instruction cycle, the CPU compares the priority code on IC0 through IC3 to the interrupt mask (ST12-ST15). If the interrupt code of the pending interrupt is equal to or less than the current interrupt mask, the CPU executes a vectored interrupt; otherwise, the interrupt request is ignored. The interrupt priority codes are shown in *Table 4-3.* Note that the level-0 interrupt code should not be used for external interrupts since level 0 is reserved for RESET.

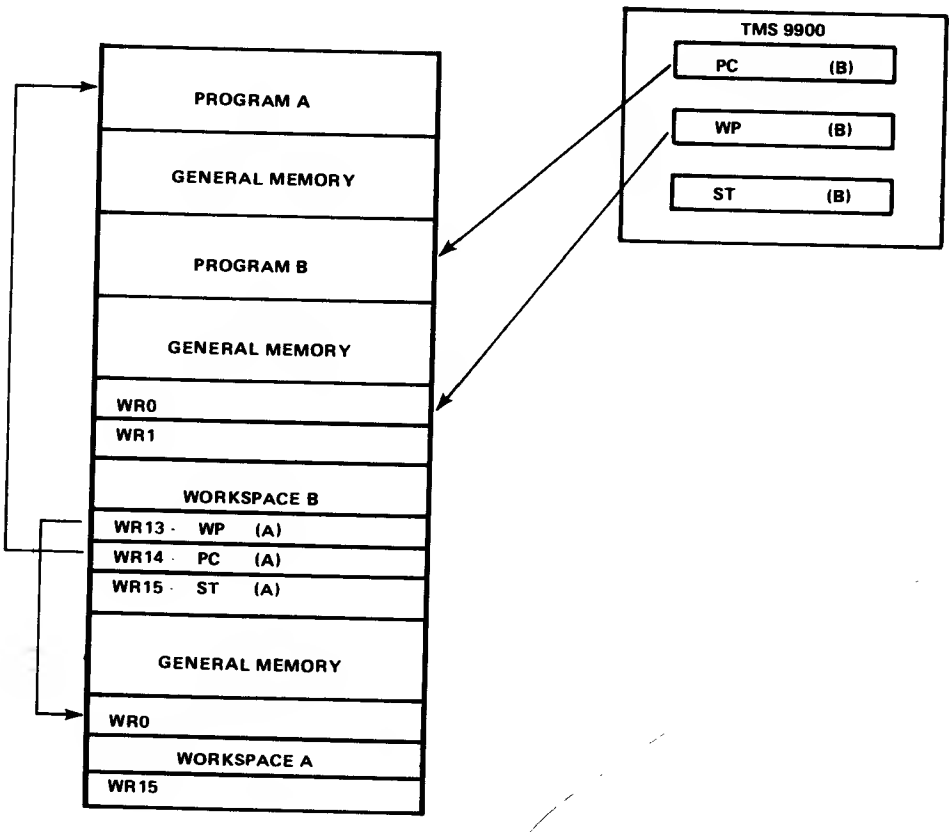


Figure 4-58. Interrupt Linkage

CYCLE	TYPE	FUNCTION
1	ALU	Set Up
2	Memory Read	Fetch New WP
3	ALU	Set Up
4	Memory Write	Store Status
5	ALU	Set Up
6	Memory Write	Store PC
7	ALU	Set Up
8	Memory Write	Store WP
9	ALU	Set Up
10	Memory Read	Fetch New PC
11	ALU	Set Up MAR for Next Instruction

Figure 4-59. Interrupt Processing Machine Cycle Sequence

Figure 4-60 illustrates the use of the TMS 9901 programmable system interface for generation of the interrupt code from individual interrupt input lines. The TMS 9901 provides six dedicated and nine programmable latched, synchronized, and prioritized interrupts, complete with individual enabling/disabling masks. Synchronization prevents transition of ICO-IC3 while the code is being read. A single-interrupt system with an arbitrarily chosen level-7 code is shown in *Figure 4-61.* The single-interrupt input does not need to be synchronized since the hardwired interrupt code is always stable.

Interrupt Masking

The TMS 9900 uses a four-bit field in the status register, ST12 through ST15, to determine the current interrupt priority level. The interrupt mask is automatically loaded with a value of one less than the level of the maskable interrupt being serviced. The interrupt mask is also affected by the load interrupt mask instruction (LIMI).

Since the interrupt mask is compared to the external interrupt code before an interrupt is recognized, an interrupt service routine will not be halted due to another interrupt of lower or equal priority unless a LIMI instruction is used to alter the interrupt mask. The LIMI instruction can be used to alter the interrupt-mask level in order to disable intervening interrupt levels. At the end of the service routine, a return (RTWP) restores the interrupt mask to its value before the current interrupt occurred.

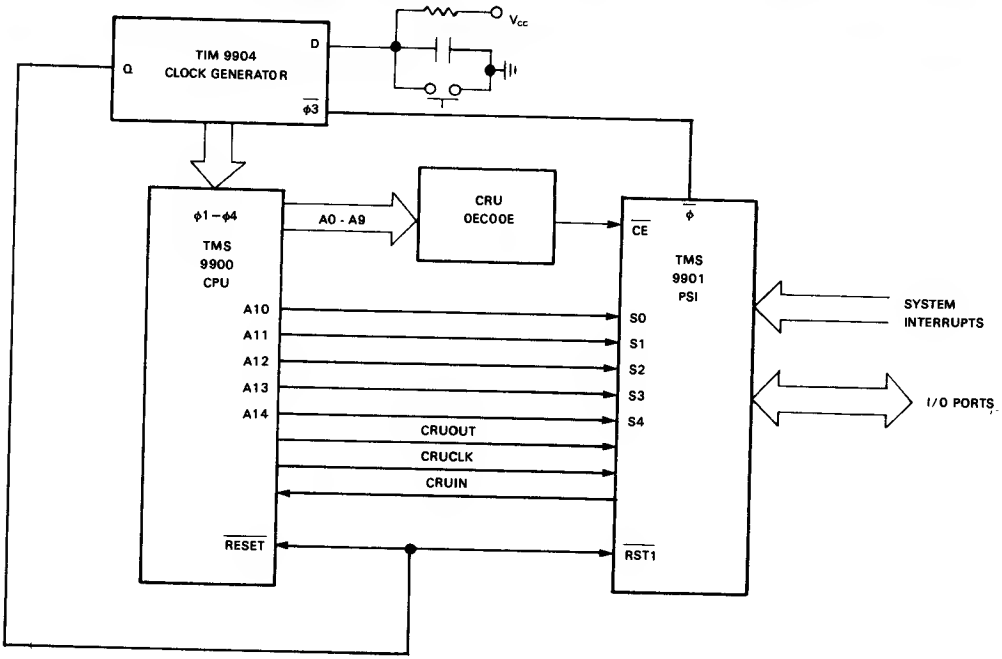


Figure 4-60. System With 15 External Interrupts

Table 4-3. Interrupt Priority Codes

Interrupt Level		Vector Location (Memory Address In Hex)	Device Assignment	Interrupt Mask Values To Enable Respective Interrupts (ST12 thru ST15)	Interrupt Codes IC0 thru IC3
(Highest priority)	0	00	Reset	0 through F*	0000
	1	04	External device	1 through F	0001
	2	08		2 through F	0010
	3	0C		3 through F	0011
	4	10		4 through F	0100
	5	14		5 through F	0101
	6	18		6 through F	0110
	7	1C		7 through F	0111
	8	20		8 through F	1000
	9	24		9 through F	1001
	10	28		A through F	1010
	11	2C		B through F	1011
	12	30		C through F	1100
	13	34		D through F	1101
	14	38		E and F	1110
(Lowest priority)	15	3C	External device	F only	1111

* Level 0 can not be disabled.

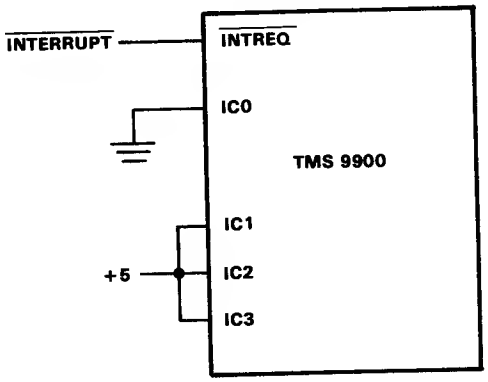


Figure 4-61. Single-Interrupt System

Note that the TMS 9900 actually generates the interrupt vector address using IC0-IC3 five clock cycles after it has sampled $\overline{\text{INTREQ}}$ and four clock cycles after it has compared the interrupt code to the interrupt mask in the status register. Thus, interrupt sources which have individual masking capability can cause erroneous operation if a command to the device to mask the interrupt occurs at a time when the interrupt is active and just after the TMS 9900 has sampled $\overline{\text{INTREQ}}$ but before the vector address has been generated using IC0-IC3.

The individual interrupt masking operation can be easily allowed if the masking instruction is placed in a short subroutine which masks all interrupts with a LIMI 0 instruction before individually masking the interrupt at the device, as shown in *Figure 4-62.*

```
INCORRECT

        XXX

        SBO     0               SET MASK (INTERRUPT CAN OCCUR
                                DURING SBO CAUSING ERRONEOUS
        YYY                     OPERATION)

CORRECT

        XXX

        BLWP    9               (WR9)  =  ADDRESS OF SBW
                                (WR10) =  ADDRESS OF SB1
        XXXX

SB1     LIMI    0               CLEAR STATUS MASK TO INHIBIT INTERRUPTS
        MOV     @24(13), 12     MOVE CRU BASE ADDRESS TO WR12
        SBO     0               SET MASK
        RTWP                    RETURN

SBW     BSS     32              SUBROUTINE WORKSPACE
```

Figure 4-62. External Interrupt Clearing Routine

Interrupt Processing Example

The routine in *Figure 4-63* illustrates the use of the LIMI instruction as a privileged or non-interruptable instruction. The level-5 routine sets a CRU bit and then loops until a corresponding CRU bit is true. The first instruction in the routine is completed before a higher priority interrupt can be recognized. The LIMI instruction, however, raises the CPU priority level to level 0 in order to disable all other maskable interrupts. Consequently, the level-5 routine will run to completion unless a RESET signal or a $\overline{\text{LOAD}}$ signal is generated. At the end of the routine, the RTWP instruction restores the CPU to its state before the level-5 interrupt occurred.

```
Level 5   LIMI   0      Disable Maskable INTREQs
          SBO    ACK    Set CRU Output Bit
Loop      TB     RDY    Test CRU Input Bit
          JNE    LOOP   Loop Until Input True
          RTWP          Return
```

Figure 4-63. LIMI Instruction Routine

ELECTRICAL REQUIREMENTS

Understanding the Electrical Specifications

A description of the interface to the 9900 would be incomplete without a set of specifications for the electrical signals which perform the functions described in the previous sections. Each pin of the 9900 may be characterized with a set of minimum and maximum voltage and current levels. In many cases, the switching characteristics, the rate of transition from the high state to the low state is also important. The detailed electrical specifications for each of the processors in the 9900 family are given in the *Product Data* chapter. A brief statement about the basic concepts of device characterization and data sheet specification is of value to designers with limited exposure to microprocessor and semiconductor memory products.

Specifications are given in two ways. First, absolute maximum ratings are given which simply define the limits of stress which the chip can withstand without damage. (*Figure 4-64* shows the absolute maximum ratings for the TMS 9900.) The normal design specification is the recommended operating conditions table *(Figure 4-65)* which specifies power supply limits, signal voltage levels, and the operating temperature range. In reading these two tables it is necessary to read the explanatory notes, one of which points out that the absolute maximum power supply voltages are specified with respect to the chip substrate or V_{BB} (pin 1). In the normal operating conditions, all voltages are specified with respect to the V_{SS} or ground (pins 26, 40). The four voltages given, V_{BB}, V_{CC}, V_{DD}, and V_{SS} are not actually four power supplies, but three power supplies: +5V, −5V, and +12V, with V_{SS} being the ground or reference point.

4◀

ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Supply voltage, V_{CC} (see Note 1)	−0.3 to 20 V
Supply voltage, V_{DD} (see Note 1)	−0.3 to 20 V
Supply voltage, V_{SS} (see Note 1)	−0.3 to 20 V
All input voltages (see Note 1)	−0.3 to 20 V
Output voltage (with respect to V_{SS})	−2 V to 7 V
Continuous power dissipation	1.2 W
Operating free-air temperature range	0°C to 70°C
Storage temperature range	−55°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: Under absolute maximum ratings voltage values are with respect to the most negative supply, V_{BB} (substrate), unless otherwise noted. Throughout the remainder of this section, voltage values are with respect to V_{SS}.

Figure 4-64. Absolute Maximum Ratings

RECOMMENDED OPERATING CONDITIONS

	MIN	NOM	MAX	UNIT
Supply voltage, V_{BB}	−5.25	−5	−4.75	V
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply voltage, V_{DD}	11.4	12	12.6	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH} (all inputs except clocks)	2.2	2.4	$V_{CC}+1$	V
High-level clock input voltage, $V_{IH(\phi)}$	$V_{DD}-2$		V_{DD}	V
Low-level input voltage, V_{IL} (all inputs except clocks)	−1	0.4	0.8	V
Low-level clock input voltage, $V_{IL(\phi)}$	−0.3	0.3	0.6	V
Operating free-air temperature, T_A	0		70	C

Figure 4-65. Recommended Operating Conditions

Input signals should be in the range from 2.2V to 6V (assuming V_{CC} is 5V) for the high level, the nominal design point being at 2.4V. Low level input voltage should be below 0.6V (but not less than −0.3V.) These specifications are not the same as the standard TTL specifications as far as the "worst case" design criteria are concerned. Care should be exercised when interfacing the 9900 with TTL circuits that loading of the TTL devices does not produce input voltages to the 9900 which are outside the specified range.

The clock signal voltages are substantially different from the TTL standard; however, the TMS 9904 is available to provide these signals.

The electrical characteristics specification, *Figure 4-66.* defines the current into or out of the 9900 chip at the operating voltage levels. The input current, I_I, is specified for four groups of input signals over a range of input voltages. For example, the input current for any input on the data bus (when reading data from the memory) is nominally ±50 microamps over the input voltage range from 0V to 5V (when V_{CC} is 5V). The current is negative (flowing out of the 9900) for low levels, and positive (into the 9900) for high levels. For "worst case" design the maximum values should be used.

Voltage specifications on the output pins show how the 9900 output devices drive external circuits. For the high level, V_{OH}, the voltage will be at least 2.4V but may go as high as 5V (V_{CC}) under the condition of output current of 0.4 mA. (Currents flowing out of the chip are shown as negative values.) When an output signal is at the low state, the output voltage, V_{OL}, will be no greater than 0.65V when the current flowing into the chip is 3.2 mA. Although the I-V characteristic of the output circuit is nonlinear, a second data point is given: if the current is 2 mA, the voltage will be no greater than 0.50V. These numbers tell the designer what the output drive circuit current sinking capability is. Two standard TTL loads (1.6 mA each) can be accommodated, but the V_{OL} level, as specified, may be as high as at 0.65V (the standard TTL specification for outputs is V_{OL} 0.4V.)

ELECTRICAL CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (UNLESS OTHERWISE NOTED)

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
I_I	Input current	Data bus during DBIN	$V_I = V_{SS}$ to V_{CC}		±50	±100	μA
		$\overline{WE}$, $\overline{MEMEN}$, DBIN, Address bus, Data bus during HOLDA	$V_I = V_{SS}$ to V_{CC}		±50	±100	
		Clock *	$V_I = -0.3$ to 12.6 V		±25	±75	
		Any other inputs	$V_I = V_{SS}$ to V_{CC}		±1	±10	
V_{OH}	High-level output voltage		$I_O = -0.4$ mA	2.4		V_{CC}	V
V_{OL}	Low-level output voltage		$I_O = 3.2$ mA			0.65	V
			$I_O = 2$ mA			0.50	
I_{BB}	Supply current from V_{BB}				0.1	1	mA
I_{CC}	Supply current from V_{CC}				50	75	mA
I_{DD}	Supply current from V_{DD}				25	45	mA
C_i	Input capacitance (any inputs except clock and data bus)		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		10	15	pF
$C_{i(\phi 1)}$	Clock-1 input capacitance		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		100	150	pF
$C_{i(\phi 2)}$	Clock-2 input capacitance		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		150	200	pF
$C_{i(\phi 3)}$	Clock-3 input capacitance		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		100	150	pF
$C_{i(\phi 4)}$	Clock-4 input capacitance		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		100	150	pF
C_{DB}	Data bus capacitance		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		15	25	pF
C_o	Output capacitance (any output except data bus)		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		10	15	pF

† All typical values are at $T_A = 25°C$ and nominal voltages.
*O.C. Component of Operating Clock

Figure 4-66. Electrical Characteristics

The timing of the various signals on the TMS 9900 chip is shown in *Figure 4-67*. The fundamental propagation time from a clock phase pulse (leading edge) to the specified output is given as t_p and is typically 20 ns but is never more than 40 ns (worst case). The parameters t_{pLH} and t_{pHL} are the propagation delays from the appropriate clock signal to the low-to-high transition of the output (t_{pLH}) or the high-to-low transition of the output (t_{pHL}). For example, the WE signal makes its high-to-low transition 20 ns after $\phi 1$ clock, and makes a low-to-high transition 20 ns after the next $\phi 1$ clock. Most of the output signals make transitions 20 ns after the $\phi 2$ clock, and remain valid until the next 02 clock.

Additional information regarding design constraints based on the electrical specifications is given in the next section.

SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH} or t_{PHL} Propagation delay time, clocks to outputs	C_L = 200 pF		20		ns

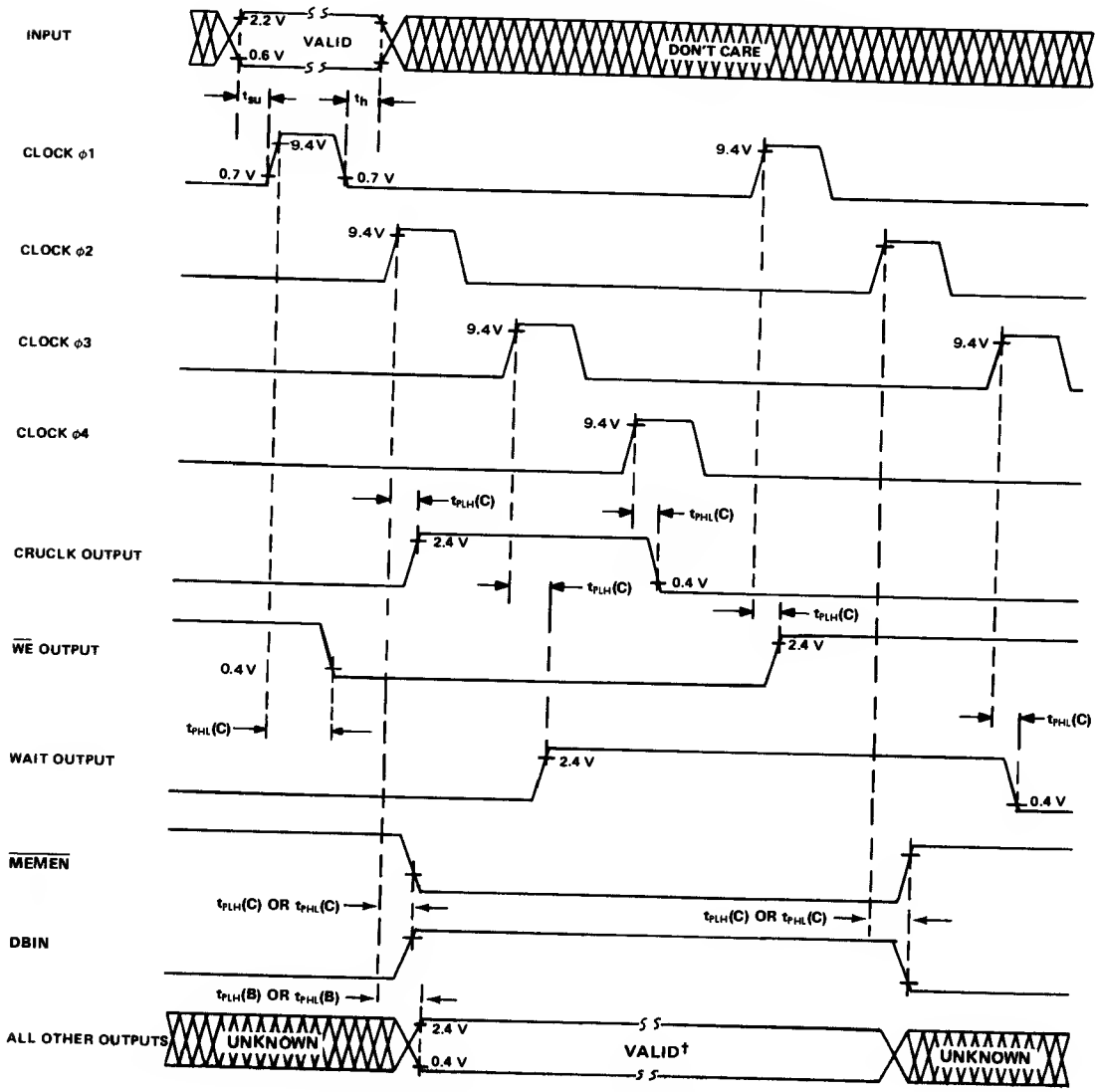


Figure 4-67. Switching Characteristics

Detailed Electrical Interface Specifications (TMS 9900)

This section reviews the TMS 9900 electrical requirements, including the system clock generation and interface signal characteristics. The "TMS9900 Data Manual" (Chapter 8) should be used for minimum and maximum values.

TMS 9900 Clock Generation

The TMS 9900 requires a non-overlapping four-phase clock system with high-level MOS drivers. Additional TTL outputs are typically required for external signal synchronization or for dynamic memory controllers. A single-chip clock driver, the TIM 9904, can be used to produce these clock signals. An alternative clock generator uses standard TTL logic circuits and discrete components.

The TMS 9900 requires four non-overlapping 12V clocks. The clock frequency can vary from 2 to 3 Megahertz. The clock rise and fall times must not exceed 100 nanoseconds and must be 10 to 15 nanoseconds for higher frequencies in order to satisfy clock pulse width requirements. While the clocks must not overlap, the delay time between clocks must not exceed 50 microseconds at lower frequencies. The typical clock timing for 3 MHz is illustrated in *Figure 4-68.*

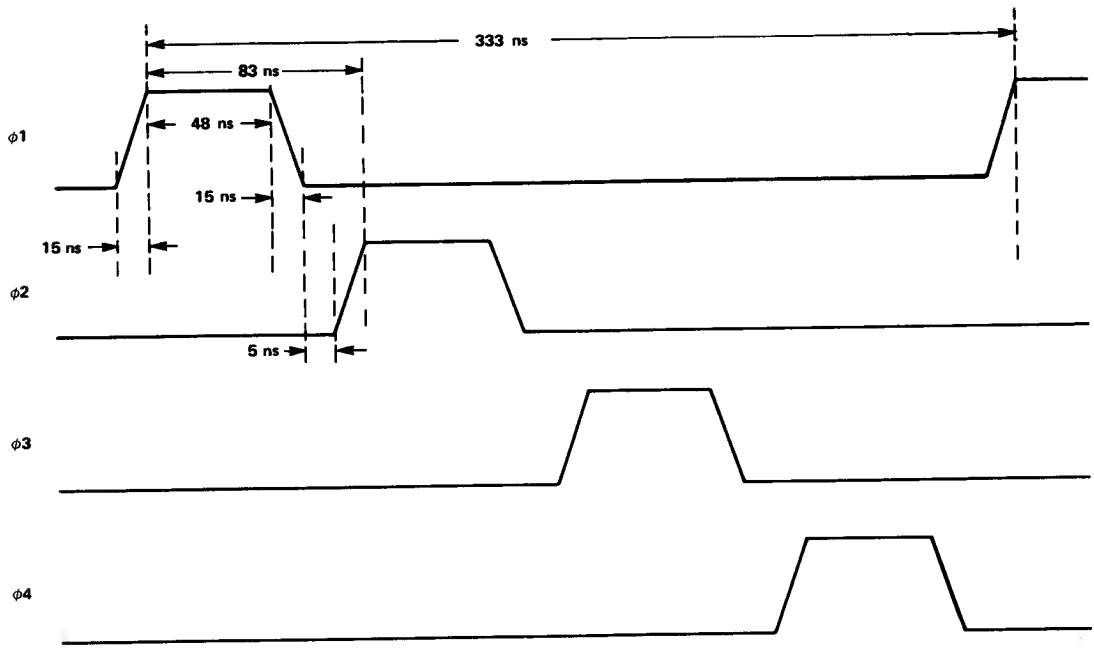


Figure 4-68. TMS 9900 Typical Clock Timing

TIM 9904 Clock Generator

The TIM 9904 (SN74LS362) is a single-chip clock generator and driver for use with the TMS 9900. The TIM 9904 contains a crystal-controlled oscillator, waveshaping circuitry, a synchronizing flip-flop, and quad MOS/TTL drivers as shown in *Figure 4-69*.

The clock frequency is selected by either an external crystal or by an external TTL-level oscillator input. Crystal operation requires a 16X input crystal frequency since the TIM 9904 divides the input frequency for waveshaping. For 3-megahertz operation, a 48-megahertz crystal is required. The LC tank inputs permit the use of overtone crystals. The LC network values are determined by the network resonant frequency:

$$f = \frac{1}{2\pi\sqrt{LC}}$$

▶4

For less precise frequency control, a capacitor can be used instead of the crystal.

The external-oscillator input can be used instead of the crystal input. The oscillator input frequency is 4X the output frequency. A 12-megahertz input oscillator frequency is required for a 3-megahertz output frequency. A 4X TTL-compatible oscillator output (OSCOUT) is provided in order to permit the derivation of other system timing signals from the crystal or oscillator frequency source.

The oscillator frequency is divided by four to provide the proper frequency for each of the 4-clock phases. A high-level MOS output and an inverted TTL-compatible output is provided by each clock phase. The MOS-level clocks are used for the TMS 9900 CPU while the TTL clocks are used for system timing.

The D-type flip-flop is clocked by $\phi 3$ and can be used to synchronize external signals such as a $\overline{\text{RESET}}$. The Schmitt-triggered input permits the use of an external RC network for power-on $\overline{\text{RESET}}$ generation. The RC values are dependent on the power supply rise time and should hold $\overline{\text{RESET}}$ low for at least three clock cycles after the supply voltages reach the minimum voltages.

All TIM 9904 TTL-compatible outputs have standard short circuit protection. The high-level MOS clock outputs, however, do not have short circuit protection.

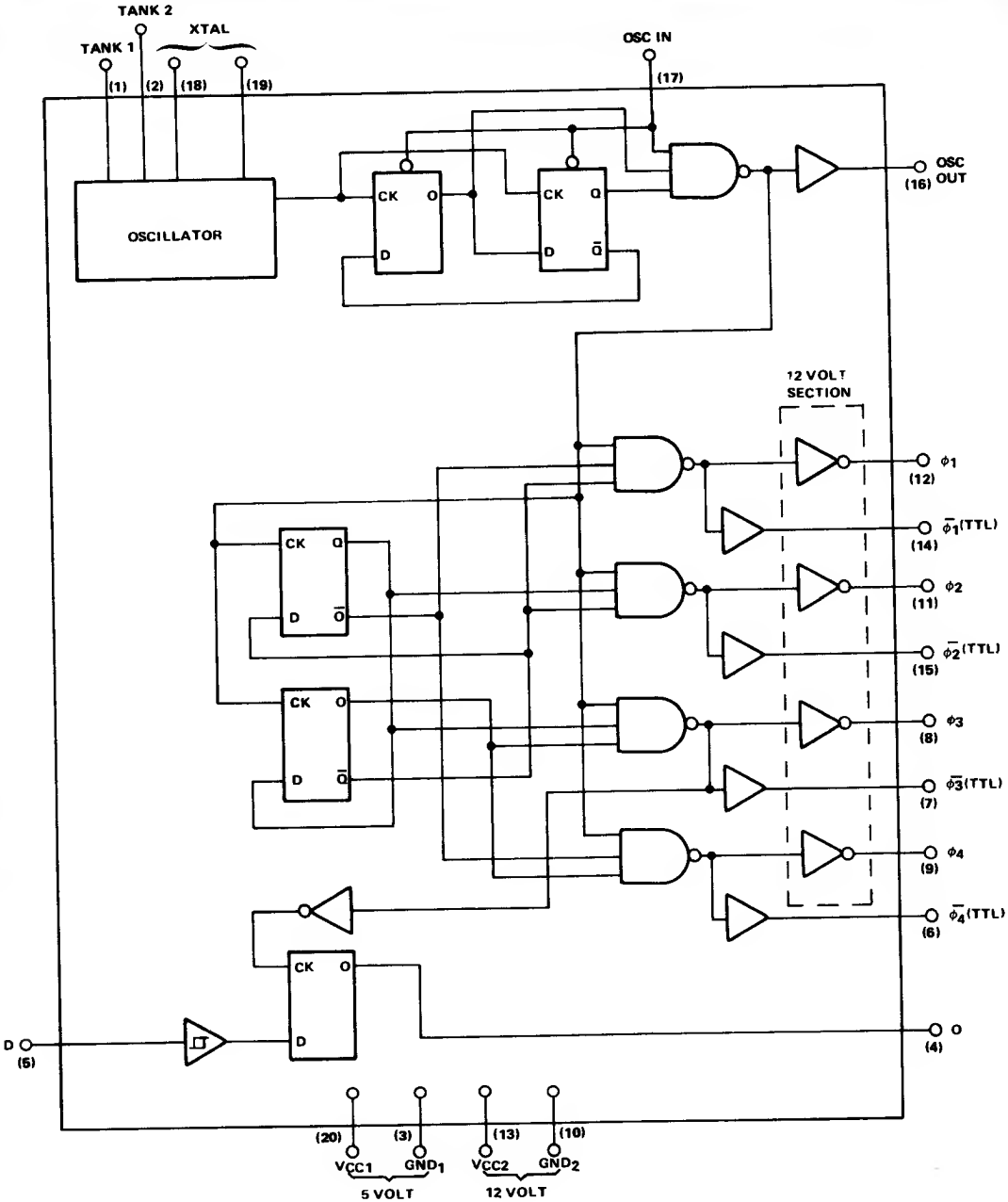


Figure 4-69. TIM 9904 Clock Generator

This driver uses inexpensive 2N3703s and 2N3704s and broad tolerance passive components. Resistor tolerances can be 10% with capacitor variations as much as 20% without affecting its performance noticeably. It shows very little sensitivity to transistor variations and its propagation times are largely unaffected by output capacitive loading. It produces rise times in the 10-12 ns region with fall times from 8-10 ns, driving 200 pF capacitive loads. Propagation times for this driver are such that it produces an output pulse that is wider than its input pulse. This driver can easily be used at 3 megahertz without special selection of components. It does have the disadvantage of taking nine discrete components per driver, but if assembly costs are prohibitive, these can be reduced by using two Q2T2222 and two Q2T2905 transistor packs. The Q2T2222 is basically four NPN transistors of the 2N2222 type while the Q2T2905 has four PNP, 2N2905 type transistors in single 14-pin dual-in-line packages. Thus, all four drivers can be built using two packages each of these quad packs.

TMS 9900 Signal Interfacing

The non-clock CPU inputs and outputs are TTL compatible and can be used with bipolar circuits without external pull-up resistors or level shifters. The TMS 9900 inputs are high impendance to minimize loading on peripheral circuits. The TMS 9900 outputs can drive approximately two TTL loads, thus eliminating the need for buffer circuits in many systems.

Switching Levels

The TMS 9900 input switch levels are compatible with most MOS and TTL circuits and do not require pull-up resistors to reach the required high-level input switching voltage. The TMS 9900 output levels can drive most MOS and bipolar inputs. Some typical switching levels are shown in *Table 4-4.*

Table 4-4. *Switch Levels*

SWITCHING LEVEL (V)	TMS 9900	TMS 2708	TMS 4042-2	SN 74XX	SN 74LSXX
V_{IH} min	2.2	3.0	2.2	2.0	2.0
V_{IL} max	0.6	0.65	0.65	0.8	0.7
V_{OH}* min	2.4	3.7	2.2	2.4	2.7
V_{OL} max	0.5	0.45	0.45	0.5	0.5

*V_{OH} exceeds 2.4 V as shown in Figure 4-70.

It should be noted that some MOS circuits such as the TMS 4700 ROM and the TMS 2708 EPROM have a minimum high-level input voltage of 3 V to 3.3 V, which exceeds the TMS 9900 minimum high-level output voltage of 2.4 V. The TMS 9900 high-level output voltage exceeds 3.3 V; however, longer transition times as shown in *Figure 4-70* are required.

Loading

The TMS 9900 has high-impedance inputs to minimize loading on the system buses. The CPU data bus presents a *maximum* current load of ± 100 μA when DBIN is high. $\overline{WE}$, $\overline{MEMEN}$, and DBIN cause a *maximum* current load of ± 100 μA during HOLDA. Otherwise, the TMS 9900 inputs present a current load of only ± 10 μA. The data bus inputs have a 25-picofarad input capacitance, and all other non-clock inputs have a 15-picofarad input capacitance.

The TMS 9900 outputs can drive approximately two standard TTL loads. Since most memory devices have high-impedance inputs, the CPU can drive small memory systems without address or data buffers. If the bus load exceeds the equivalent of two TTL unit loads, external buffers are required.

The TMS 9900 output switching characteristics are determined for approximately 200 picofarads. Higher capacitive loads can be driven with degraded switching characteristics as shown in *Figure 4-71.*

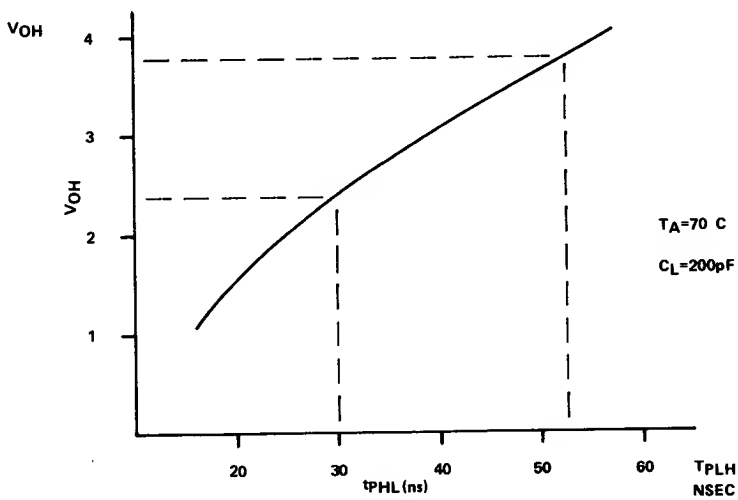


Figure 4-70. *t_{PLH} vs V_{OH} Typical Output Levels*

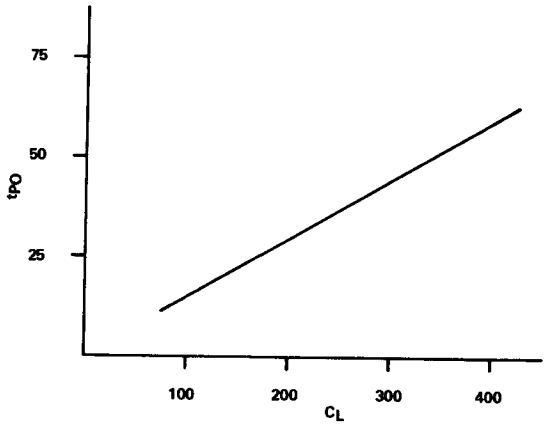


Figure 4-71. t_{PO} *vs Load Capacitance (Typical)*

▶4

Recommended Interface Logic

The TMS 9900 is compatible with the logic from any of the common TTL logic families. The Texas Instruments low-power Schottky logic circuits are, however, recommended for use in microprocessor systems. The SN74LSXX circuits have higher impedance inputs than standard TTL, allowing more circuits to be used without buffering. The SN74LSXX gates also consume less power at similar switching speeds. Texas Instruments has a wide assortment of bipolar support circuits which can be used with the TMS 9900, as shown in *Table 4-5.* Note that five circuits which are particularly useful in many applications have been dual symbolized with TIM 99XX numbers for easy reference.

There are a number of buffer circuits available for use in TMS 9900 systems. The SN74S241 and SN74LS241 non-inverting octal buffers with three-state outputs can be used as memory address drivers or as bidirectional data transceivers. The SN74S240 and SN74LS240 are similar, but with inverted outputs. The SN74LS241 can be used as either a memory-address buffer or as a transceiver for bidirectional data transfers. The use of a single circuit type for both functions can result in a lower inventory and parts cost. The buffer switching times can be derated for higher capacitive loading as required.

System Layout

The pin assignments of the TMS 9900 are such that sets of signals (data bus, address bus, interrupt port, etc.) are grouped together. The layout of a printed circuit board can be simplified by taking advantage of these groups by locating associated circuitry (address buffers, interrupt processing hardware, etc.) as close as possible to the TMS 9900 interface. Shortened conductor runs result in minimal noise and compact and efficient utilization of printed circuit board area.

It is particularly important that the drivers for $\phi 1$—$\phi 4$ be located as close as possible to the inputs of the TMS 9900, since these signals have fast rise and fall times while driving fairly high capacitance over a wide voltage range. The 12 volt supply to the clock drivers should be decoupled with both high (15μF) and low (0.05μF) value capacitors in order to filter out high and lower frequency variations in supply voltage.

Table 4-5. TMS 9900 Bipolar Support Circuits

DEVICE	FUNCTION	PACKAGE
	BUFFERS (3-STATE)	
SN74125	QUAD Inverting Buffer	14
SN74126	QUAD Inverting Buffer	14
SN74LS240	OCTAL Inverting Buffer/Transceiver	20
SN74LS241	OCTAL Noninverting Buffer/Transceiver	20
SN74LS242	OCTAL Inverting Transceiver	14
SN74LS243	OCTAL Noninverting Transceiver	14
SN74S240	OCTAL Inverting Buffer/Transceiver	20
SN74S241	OCTAL Noninverting Buffer/Transceiver	20
SN74365	Hex Noninverting Buffer	16
SN74366	Hex Inverting Buffer	16
SN74367	Hex Noninverting Buffer	16
SN74368	Hex Inverting Buffer	16
	LATCHES	
SN74LS259 (TIM9906)	OCTAL Addressable Latch	16
SN74LS373	OCTAL Transparent Latch (3-state)	20
SN74LS412	OCTAL I/O Port (3-state)	24
	DATA MULTIPLEXERS	
SN74LS151	OCTAL Multiplexer	16
SN74LS251 (TIM9905)	OCTAL Multiplexer (3-state)	16
	OTHER SUPPORT CIRCUITS	
SN74148 (TIM 9907)	Priority Encoder	16
SN74LS348 (TIM9908)	Priority Encoder	16
SN74LS74	Dual D-type flip-flop	14
SN74LS174	Hex D-type flip-flop	16
SN74LS175	Qual D-type flip-flop	16
SN74LS37	QUAD 2-Input nand Buffers	14
SN74LS362 (TIM9904)	Clock Generator	20

All voltage inputs to the TMS 9900 should be decoupled at the device. Particular attention should be paid to the +5 volt supply. All data and address lines are switched simultaneously. The worst-case condition occurs when all data and address signals switch to a low level simultaneously and they are each sinking 3.2 mA. It is thus possible for the supply current to vary nearly 100 mA over a 20 ns interval. Careful attention must be paid by the designer to avoid supply voltage spiking. The exact values for capacitors should be determined empirically, based on actual system layout and drive requirements.

TMS 9940 MICROCOMPUTER

The TMS9940 is a microcomputer chip in a 40-pin package which includes all of the elements of a computer, that is, memory, I/O and utilities in addition to ALU and control. Useful in a wide variety of dedicated control functions, it contains a 2k × 8 EPROM program memory and a 128 × 8 RAM for data, a 14 bit interval timer, and a multiprocessor system interface. Although the memory organization on chip is in 8 bit
▶4 bytes, the instructions are the same 16-bit instructions of the 9900 family.

While most of the instructions are identical to the instruction set of the 9900, there are 68 instructions in the 9940 set (as opposed to 69 in the TMS9900) including three new ones. The differences in the instruction set are illustrated by the following list of instructions.

DCA	Decimal Correct for BCD add	Added Instructions
DCS	Decimal Correct for BCD subtract	Added Instructions
LIIM	Load Interrupt mask	Added Instructions
RSET	(external instructions in 9900)	Deleted Instructions
CKOF	(external instructions in 9900)	Deleted Instructions
CKON	(external instructions in 9900)	Deleted Instructions
LREX	(external instructions in 9900)	Deleted Instructions
IDLE	Put processor into the idle state	Hardware in the 9940

The first three of the instructions in the above list are new instructions and are unique to the 9940 microcomputer. The DCA and DCS instructions perform decimal correct for BCD arithmetic. The LIIM instruction is a single word instruction to load the interrupt mask. (This instruction should be contrasted with the LIMI instruction of the 9900 set which performs the same function but occupies two memory words.) The idle instruction, an external instruction in the 9900 set, is now implemented in hardware. Four instructions in the list are not implemented in the 9940; they are external instructions in the 9900 set.

Pin Assignments and Functional Control

One of the most extraordinary features of the TMS9940 is the I/O structure in which 32 pins of the 40-pin package are software assignable. That is, they do not perform single, predefined, hard-wired functions, but instead are under the control of the programmer in structuring input/output functions. *Table 4-6* lists the functions of four specific bits in the CRU which are called configuration bits. Because these four bits are assigned specific locations in the CRU output field and are therefore addressable and accessible via CRU output instructions, the pins of the package may be dynamically reassigned during program execution.

Table 4-6. Configuration Bit Functions

Configuration Bit	*Function*
0	External CRU expansion
1	Multiprocessor
2	Clock output for sync
3	Power down

Table 4-7 describes the way these four configuration bits assign the individual general-purpose I/O pins to specific functions. In effect, each of the pins may serve two or three functions, as described in the table. *Table 4-8* defines the functions of addressable CRU locations. The first 256 locations, addresses 000 through 0FF, are for external expansion of the general I/O to an additional 245 (256 less 11 used for expansion). It is important to note here that these 256 bits are two fields of 256 bits each, one for input and one for output. Addresses 100 through 17F are not used, and address 180 through 1DF are used internally.

Notice in *Table 4-8* that CRU addresses 183, 184, 185 and 186 locate the four configuration bits. It is via the setting or resetting of these individual bits that the I/O configuration is established.

Four other significant features should be pointed out.
One: The interrupt structure includes four levels of interrupt as opposed to the 16-level interrupt capability of the general 9900 microprocessor group.
Two: There is an on-chip timer, or event counter.
Three: 32 bits of CRU I/O are implemented on the chip. (Addresses 1E0-1EFF)
Four: A multiprocessor system interface is constructed as part of the CRU I/O.

Interrupts

The four interrupt levels are shown in the table below.

Level 0	Reset	*Level 2*	Decrementer
Level 1	General Interrupt 1	*Level 3*	General Interrupt 2

Table 4-7. Configuration Bit Effects by Pin

Pin	*Configuration bit 0 (CRU Expansion)*	
	0	1
23	P0 (general I/O	A1
24	P1	A2
25	P2	A3
26	P3	A4
27	P4	A5
28	P5	A6
29	P6	A7
30	P7	A8
18	P8	CRUIN
17	P9	CRUOUT
16	P0	CRUCLK

Pin	*Configuration bit 1 (Multiprocessor)*	
	0	1
14	P11	$\overline{TC}$ (Clock)
11	P12	TD (Data)

Pin	*Configuration bit 2 (Sync)*	
	0	1
15	P13	ϕ (Clock)

Pin	*Configuration bit 3 (Power Down)*	
	0	1
10	P14	$\overline{HLD}$
9	P15	$\overline{HLDA}$
8	P16	$\overline{IDLE}$

Table 4-8. Functions of CRU Address

CRU Addresses	*Contents of R12*	*Input*	*Output*
000-0FF	000-1FE	CRU Expansion	CRU Expansion
100-17F	200-2FE	NA	NA
180	300	Test for Interrupt 1	
181	302	Test for Decrementer Interrupt	Clear Decrementer
182	304	Test for Interrupt 2	
183	306		Set Configuration Bit 0
184	308		Set Configuration Bit 1
185	30A		Set Configuration Bit 2
186	30C		Set Configuration Bit 3
187-18F	30E-31E	NA	NA
190-19D	320-33A	Read Decrementer Value	Load Decrementer Value
19E	33C		TE (Timer/Event Cntr)
19F	33E		
1A0-1AF	340-35E	Read MPSI Value	Load MPSI Value
1B0-1BF	360-37E	Read Flag Register	Set Flag Register
1C0-1DF	380-3BE		Set I/O Direction for P0-P31
1E0-1FF	3C0-3FE	P0-P31 Input Data	P0-P31 Output Data

The 9940 implements interrupts using the same context switch concept of the 9900. Thus, the interrupt vectors for the four interrupt levels must be stored in the first 16 words of the 9940's program memory. As is described in a subsequent paragraph, the decrementer acts like a counter in an external piece of hardware in that after the contents of the circuit have been decremented to zero an interrupt signals the processor to perform a context switch and perform whatever function was programmed as the service routine for the decrementer. The reset, INT 1, and INT 2 interrupt signals are available to external hardware.

Since there is no $\overline{\text{INTREQ}}$ (interrupt request) signal input for the 9940, an interrupt input must be set and remain set until acknowledged. In fact, the acknowledgement of an interrupt must include instructions to reset holding flip-flops (if used) via CRU operations.

In the 9940, the interrupt input may be masked (as in all 9900 processors) but there are specific CRU bits which, if tested, will reveal pending interrupts which are not being serviced. Thus, the programmer may wish to mask interrupts but still be aware (via TB instructions to CRU locations 180, 181 and 182 as shown in *Table 4-8*) of the interrupt input status.

Decrementer

A timer/event counter is implemented on the 9940 chip to introduce interrupts after a predefined time period or number of events. A set of dedicated CRU addresses define the location of decrementer input and output registers. A value may be loaded into the decrementer via an LDCR instruction which loads CRU locations $190_{16} - 19D_{16}$. Likewise, the current value of the decrementer may be read via an STCR instruction identifying the same CRU field.

When the decrementer contents count down to zero, an interrupt is issued. The context switch thus activated automatically clears the interrupt request.

As a timer, the decrementer counts down at the rate of 1/30 of the oscillator frequency. With a clock frequency of 5 MHz, the time interval for counting is six microseconds.

As an event counter, the decrementer is first loaded with a value and it then counts down (one bit for each positive transition on pin 7) until it reaches zero. An interrupt is then issued.

CRU Implementation

One of the most important features of the 9940 is the manner in which the CRU is used to perform pin assignments and functional control as well as input and output. The major impact is that the external devices and some of the internal devices are under direct control of the programmer via CRU instructions. The major emphasis (see *Table 4-8*) is as follows.

32 bits of input — on-chip multiplexer
32 bits of output — on-chip flip flops
32 bit register defining signal direction (in or out) for the assignable pins
16 bit flag register — may be written or read
14 bit "clock" register—for loading the decrementer
14 bit "read" register—for reading the decrementer
16 bit shift register for receiving instructions in a multiprocessor application, or used for sending 16-bit information over the MPSI data line to other processors
14 bit decrementer (used as a timer or counter)
256 bit CRU expansion (input and output)

Pin assignments may be explained by showing the basic application concept, that of using the 32 bits of internal CRU. Here the only decision is one of signal direction. It is possible to set the configuration once during initialization and never change it. But this limits the total number of I/O signals to 32. It is permissible to change the signal direction of each pin as needed, thus obtaining full utilization of the 32 inputs and 32 outputs. The pins themselves (labelled P0-P31 in *Table 4-9*) serve as a dynamically configurable bidirectional CRU port. Data is addressed in the CRU address field 1E0 to 1FF. Direction control is established by writing a logical one for output or zero for input to the appropriate address(es) in the CRU field 1C0-1DF. Reading the addresses assigned for output is permissible and allows the program to interrogate or determine the status of the on-chip CRU output flip flops.

Functional assignments of the first 18 I/O signals may be accomplished as a "configuring" of the pins. As shown in *Table 4-9,* eighteen additional signals may pass through the pins corresponding to P0-P17. By setting configuration bit 0 for example, signals P0-P10 are no longer available to external hardware. Instead, the CRU expansion signals, A1-A8 and CRU controls, are available. Configuring may be accomplished by the following code.

LI R12,>200	Set CRU hardware base address at 100_{16}
SBO >83	Add 83_{16} to set CRU bit 183_{16}

(The LI instruction must set R12 to two times the hardware base address because the LSB is ignored. See the CRU explanation in the TMS9940 section of Chapter 8.)

Multiprocessor System Interface (MPSI)

A two-wire communication technique is provided so that the 9940 may exchange 16-bit data and/or instructions with other CPU's in a multiprocessor application. This capability allows the RAM to be used as an instruction memory for short subprograms downloaded from another processor. Since the technique is based on the CRU concept, the 9940 will easily interface with the processors in the 9900 family. In order to use this feature, configuration bit 1 must first be set via

```
LI R12, >200
SBO >84
```

Then the information flows in from an external processor and is clocked by the external processor so that this operation is completely transparent to the CPU. The sender must interrupt the receiver to cause reading of the input word via

LI R12, > 340	Address the MPSI register
STCR @BUFF, 0	Store 16 bits in memory location BUFF

Refer to *Table 4-8* for CRU addresses of this and other functions.

To send data out over the MPSI the 9940 must first have configuration bit 1 set, and then it simply executes

LDCR @BUFF, 0

to send out 16 bits of data from memory location BUFF. The switch into and out of "send" status is automatic.

Table 4-9. TMS 9940 Configurable Pins

Pin Number	*General I/O*	*CRU Address Data I/O*	*CRU Address for Direction Control*	*Alternate Function*	*Configuration Bit*	*CRU Address of Config. Bit*
23	P0	1E0	1C0	A1	0	183
24	P1	1E1	1C1	A2	0	183
25	P2	1E2	1C2	A3	0	183
26	P3	1E3	1C3	A4	0	183
27	P4	1E4	1C4	A5	0	183
28	P5	1E5	1C5	A6	0	183
29	P6	1E6	1C6	A7	0	183
30	P7	1E7	1C7	A8	0	183
18	P8	1E8	1C8	CRUIN	0	183
17	P9	1E9	1C9	CRUOUT	0	183
16	P10	1EA	1CA	CRUCLK	0	183
14	P11	1EB	1CB	$\overline{TC}$	1	184
11	P12	1EC	1CC	TD	1	184
15	P13	1ED	1CD	$\overline{\phi}$	2	185
10	P14	1EE	1CE	$\overline{HLD}$	3	186
9	P15	1EF	1CF	$\overline{HLDA}$	3	186
8	P16	1F0	1D0	$\overline{IDLE}$	3	186
7	P17	1F1	1D1	EC	–	19E
6	P18	1F2	1D2			
5	P19	1F3	1D3			
4	P20	1F4	1D4			
3	P21	1F5	1D5			
2	P22	1F6	1D6			
1	P23	1F7	1D7			
31	P24	1F8	1D8			
32	P25	1F9	1D9			
33	P26	1FA	1DA			
34	P27	1FB	1DB			
35	P28	1FC	1DC			
36	P29	1FD	1DD			
38	P30	1FE	1DE			
39	P31	1FF	1DF			

Summary

The 9940 is a powerful member of the 9900 family with execution techniques which are actually faster than the TMS9900. In fact, because of its higher speed clock (5 MHz) and a fast on-chip execution microcycle for register location, the average throughput is 20% faster than the standard 9900 devices. The assignability of the package pins via software adds a new dimension to microprocessor technology for improved flexibility and performance.

For detailed information on this part, see the 9940 section of Chapter 8.

COMPLETE LISTING OF MACHINE CYCLES

In order to complete the description of instruction execution, the individual instruction execution cycles are given in this section. Each machine cycle consists of two or more clock cycles (depending upon addressing mode) as defined herein. (*Note:* These machine cycles apply equally to the TMS 9980A/81 microprocessor, with the exception of the memory cycle as detailed below.) The 9900 family machine cycles are divided into three categories described in the following paragraphs.

Machine Cycles

ALU Cycle

The ALU cycle performs an internal operation of the microprocessor. The memory interface control signals and CRU interface control signals are not affected by the execution of an ALU cycle, which takes two clock cycles to execute.

Memory Cycle

The memory cycle primarily performs a data transfer between the microprocessor and the external memory device. Appropriate memory bus control signals are generated by the microprocessor as a result of a memory cycle execution. The memory cycle takes 2 + W (where W is the number of wait states) clock cycles to execute.

In the TMS 9980A/81, which has an 8-bit data bus, the memory cycle is composed of two data transfers to move a complete 16-bit word. The TMS 9980A/81 memory cycle takes 4 + 2W (where W is the number of wait states) clock cycles to execute. For the TMS 9980A/81 the following machine cycle sequences replace the memory sequences used in the instruction discussion.

CYCLE			
1	Memory read/write	AB =	Address of most significant byte (A13 = 0)
		DB =	Most significant byte
2	Memory read/write	AB =	Address of least significant byte (A13 = 1)
		DB =	Least significant byte

CRU Cycle

The CRU cycle performs a bit transfer between the microprocessor and I/O devices. It takes two clock cycles to execute. The address of the CRU bit is set up during the first clock cycle. For an input operation the CRUIN line is sampled by the microprocessor during the second clock cycle. For an output operation the data bit is set up on the CRUOUT line at the same time the address is set up. The CRUCLK line is pulsed during the second clock cycle of the CRU output cycle. Please refer to the specific 99XX microprocessor data manual for timing diagrams.

The 9900 executes its operations under the control of a microprogrammed control ROM. Each microinstruction specifies a machine cycle. A microprogram specifies a sequence of machine cycles. The 9900 executes a specific sequence of machine cycles for a specific operation. These sequences are detailed on the following pages. The information can be used by the systems designers to determine the bus contents and other interface behavior at various instants during a certain 9900 operation. This description is maintained at the address bus (AD) and data bus (DB) levels.

9900 Machine Cycle Sequences

Most 9900 instructions execution consists of two parts: 1) the data derivation and 2) operation execution. The data derivation sequence depends on the addressing mode for the data. Since the addressing modes are common to all instructions, the data derivation sequence is the same for the same addressing mode, regardless of the instruction. Therefore, the data derivation sequences are described first. These are then referred to in appropriate sequence in the instruction execution description.

▶4

Terms and Definitions

The following terms are used in describing the instructions of the 9900:

TERM	DEFINITION
B	Byte Indicator (1 = byte, 0 = word)
C	Bit count
D	Destination address register
DA	Destination address
IOP	Immediate operand
PC	Program counter
Result	Result of operation performed by instruction
S	Source address register
SA	Source address
ST	Status register
STn	Bit n of status register
SD	Source data register internal to the TMS 9900 microprocessor*
W	Workspace register
SRn	Workspace register n
(n)	Contents of n
Ns	Number of machine cycles to derive source operand
Nd	Number of machine cycles to derive destination operand
AB	Address Bus of the TMS 9900
DB	Data Bus of the TMS 9900
NC	No change from previous cycle

**Note:* The contents of the SD register remain latched at the last value written by the processor unless changed by the ALU. Therefore, during all memory read or ALU machine cycles the SD register and hence the data bus will contain the operand last written to the data bus by the CPU or the results of the last ALU cycle to have loaded the SD register.

Data Derivation Sequence

Workspace Register

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = Workspace register address DB = Operand

Workspace Register Indirect

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = Workspace register address DB = Workspace register contents
2	ALU	AB = NC DB = SD
3	Memory read	AB = Workspace register content DB = Operand

Workspace Register Indirect Auto-Increment (Byte-Operand)

4◀

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = Workspace register address DB = Workspace register contents
2	ALU	AB = NC DB = SD
3	Memory write	AB = Workspace register address DB = (WRn) + 1
4	Memory read	AB = Workspace register contents DB = Operand

Workspace Register Indirect Auto-Increment (Word Operand)

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = Workspace register address DB = Workspace register contents
2	ALU	AB = NC DB = SD
3	ALU	AB = NC DB = SD
4	Memory write	AB = Workspace register address DB = (WRn) + 2
5	Memory read	AB = Workspace register contents DB = Operand

Symbolic

CYCLE	TYPE	DESCRIPTION
1	ALU	AB = NC DB = SD
2	ALU	AB = NC DB = SD
3	Memory read	AB = PC + 2 DB = Symbolic address
4	ALU	AB = NC DB = 0000_{16}
5	Memory read	AB = Symbolic address DB = Operand

Indexed

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = Workspace register address DB = Workspace register contents
2	ALU	AB = NC DB = SD
3	Memory read	AB = PC + 2 DB = Symbolic address
4	ALU	AB = PC + 2 DB = Workspace register contents
5	Memory read	AB = Symbolic address + (WRn) DB = Operand

▶4

Instruction Execution Sequence

A, AB, C, CB, S, SB, SOC, SOCB, SZC, SZCB, MOV, MOVB, COC, CZC, XOR

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
Ns	Insert appropriate sequence for source data addressing mode, from the data derivation sequences	(Note 1)
3 + Ns	ALU	AB = NC DB = SD
Nd	Insert appropriate sequence for destination data addressing mode from the data derivation sequences	(Note 2, 3)
4 + Ns + Nd	ALU	AB = NC DB = SD
5 + Ns + Nd	Memory write	AB = DA (Note 4) DB = Result

NOTES:

1) Since the memory operations of the 9900 microprocessor family fetch or store 16-bit words, the source and the destination data fetched for byte operations are 16-bit words. The ALU operates on the specified bytes of these words and modifies the appropriate byte in the destination word. The adjacent byte in the destination word remains unaltered. At the completion of the instruction, the destination word, consisting of the modified byte and the adjacent unmodified byte, is stored in a single-memory write operation.
2) For MOVB instruction the destination data word (16 bits) is fetched. The specified byte in the destination word is replaced with the specified byte of the source-data word. The resultant destination word is then stored at the destination address.
3) For MOV instruction the destination data word (16 bits) is fetched although not used.
4) For C, CB, COC, CZC instructions cycle $5 + N_s + N_d$ above is an ALU cycle with AB = DA and DB = SD.

MPY (Multiply)

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
Ns	Insert appropriate data derivation sequence according to the source data (multiplier) addressing mode	
3 + Ns	ALU	AB = NC DB = SD
4 + Ns	Memory read	AB = Workspace register address DB = Workspace register contents
5 + Ns	ALU	AB = NC DB = SD
6 + Ns	ALU	AB = NC DB = Multiplier
7 + Ns		Multiply the two operands
	16 ALU	AB = NC DB = MSH of partial product
24 + Ns	Memory write	AB = Workspace register address DB = MSH of the product
25 + Ns	ALU	AB = DA + 2 DB = MSH of product
26 + Ns	Memory write	AB = DA + 2 DB = LSH of the product

DIV (Divide)

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
Ns	Insert appropriate data derivation sequence according to the source data (divisor) addressing mode	
3 + Ns	ALU	AB = NC DB = SD
4 + Ns	Memory read	AB = Address of workspace register DB = Contents of workspace register
5 + Ns	ALU	(Check overflow) AB = NC DB = Divisor
6 + Ns	ALU	(Skip if overflow to next instruction fetch) AB = NC DB = SD
7 + Ns	Memory read	AB = DA + 2 DB = Contents of DA + 2
8 + Ns	ALU	AB = NC DB = SD
9 + Ns	ALU	AB = NC DB = SD
	Divide sequence consisting of Ni cycles where $48 \leq Ni \leq 32$. Ni is data dependent	AB = NC DB = SD
10 + Ns + Ni	ALU	AB = NC DB = SD
11 + Ns + Ni	Memory write	AB = Workspace register address DB = Quotient
12 + Ns + Ni	ALU	AB = DA + 2 DB = Quotient
13 + Ns + Ni	Memory write	AB = DA + 2 DB = Remainder

XOP

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	Instruction decode AB = NC DB = SD
Ns	Insert appropriate data derivation sequence according to the source data addressing mode	
3 + Ns	ALU	AB = NC DB = SD
4 + Ns	ALU	AB = NC DB = SA
5 + Ns	ALU	AB = NC DB = SD
6 + Ns	Memory read	AB = 40_{16} + 4 x D DB = New workspace pointer

CYCLE	TYPE	DESCRIPTION
7+Ns	ALU	AB = NC DB = SA
8+Ns	Memory write	AB = Address of WR11 DB = SA
9+Ns	ALU	AB = Address of WR15 DB = SA
10+Ns	Memory write	AB = Address of workspace register 15 DB = Status register contents
11+Ns	ALU	AB = NC DB = PC+2
12+Ns	Memory write	AB = Address of workspace register 14 DB = PC+2
13+Ns	ALU	AB = Address of WR13 DB = SD
14+Ns	Memory write	AB = Address of workspace register 13 DB = WP
15+Ns	ALU	AB = NC DB = SD
16+Ns	Memory read	AB = 42_{16}+4 x D DB = New PC
17+Ns	ALU	AB = NC DB = SD

4◀

CLR, SETO, INV, NEG, INC, INCT, DEC, DECT, SWPB

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
Ns	Insert appropriate data derivation sequence according to the source data addressing mode	
3+Ns	ALU	AB = NC DB = SD
4+Ns	Memory write	AB = Source data address DB = Modified source data

Note: The operand is fetched for CLR and SETO although not used.

ABS

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
Ns	Insert appropriate data derivation sequence according to the source data addressing mode	
3+Ns	ALU	Test source data AB = NC DB = SD
4+Ns	ALU	Jump to 5′+Ns if data positive AB = NC DB = SD

CYCLE	TYPE	DESCRIPTION
5 + ns	ALU	Negate source AB = NC DB = SD
6 + Ns	Memory write	AB = Source data address DB = Modified source data
5′ + Ns	ALU	AB = NC DB = SD

X

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
Ns	Insert the appropriate data derivation sequence according to the source data addressing mode	
▶4 3 + Ns	ALU	AB = NC DB = SD

Note: Add sequence for the instruction specified by the operand.

B

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
Ns	Insert appropriate data derivation sequence according to the source data addressing mode	
3 + Ns	ALU	AB = NC DB = SD

Note: The source data is fetched, although it is not used.

BL

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
Ns	Insert appropriate data derivation sequence according to the source data addressing mode	
3 + Ns	ALU	AB = NC DB = SD
4 + Ns	ALU	AB = Address of WR11 DB = SD
5 + Ns	Memory write	AB = Address of WR11 DB = PC + 2

Note: The source data is fetched although it is not used.

BLWP

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
Ns	Insert appropriate data derivation sequence according to the source data addressing mode	
3+Ns	ALU	AB = NC DB = SD
4+Ns	ALU	AB = Address of WR15 DB = NC
5+Ns	Memory write	AB = Address of workspace register 15 DB = Status register contents
6+Ns	ALU	AB = NC DB = PC+2
7+Ns	Memory write	AB = Address of workspace register 14 DB = PC+2
8+Ns	ALU	AB = Address or workspace register 13 DB = SD
9+Ns	Memory write	AB = Address of workspace register 13 DB = WP
10+Ns	ALU	AB = NC DB = SD
11+Ns	Memory read	AB = Address of new PC DB = New PC
12+Ns	ALU	AB = NC DB = SD

LDCR

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
Ns	Insert appropriate data derivation sequence	
3+Ns	ALU	AB = NC DB = SD
4+Ns	ALU	AB = NC DB = SD
5+Ns	ALU	AB = Address of WR12 DB = SD
6+Ns	ALU	AB = Address of WR12 DB = SD
7+Ns	Memory read	AB = Address of WR12 DB = Contents of WR12
8+Ns	ALU	AB = NC DB = SD
C	Shift next bit onto CRUOUT line. Enable CRUCLK. Increment CRU bit address on AB. Iterate this sequence C times, where C is number of bits to be transferred.	AB = Address +2 Increments C Times DB = SD
9+Ns+C	ALU	AB = NC DB = SD

STCR

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
Ns	Insert appropriate data derivation sequence according to the source data addressing mode	
3+Ns	ALU	AB = NC DB = SD
4+Ns	Memory read	AB = Address of WR12 DB = Contents of WR12
5+Ns	ALU	AB = NC DB = SD
6+Ns	ALU	AB = NC DB = SD
C	Input selected CRU bit. Increment CRU bit address. Iterate this sequence C times where C is the number of CRU bits to be input.	AB = Address + 2 C times DB = SD
7+Ns+C	ALU	AB = NC DB = SD
8+Ns+C	ALU	AB = NC DB = SD
C′	Right adjust (with zero fill) byte (if C<8) or word (if 8 < C < 16).	AB = NC DB = SD
C′	= 8-C-1 if C ≤ 8 = 16-C if 8 < C ≤ 16	
9+Ns+C+C′	ALU	AB = NC DB = SD
10+Ns+C+C′	ALU	AB = NC DB = SD
11+Ns+C+C′	ALU	AB = Source address DB = SD
12+Ns+C+C′	Memory write	AB = Source address DB = I/O data

Note: For STCR instruction the 16-bit word at the source address is fetched. If the number of CRU bits to be transferred is ≤ 8, the CRU data is right justified (with zero fill) in the specified byte of the source word and source data word thus modified is then stored back in memory. If the bits to be transferred is > 8 then the source data fetched is not used. The CRU data in this case is right justified in 16-bit word which is then stored at the source address.

SBZ, SBO

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
3	ALU	AB = NC DB = SD
4	Memory read	AB = Address of WR12 DB = Contents of WR12
5	ALU	AB = NC DB = SD
6	CRU	Set CRUOUT = 0 for SBZ = 1 for SBO AB = CRU Bit Address Enable CRUCLK

TB

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
3	ALU	AB = NC DB = SD
4	Memory read	AB = Address of WR12 DB = Contents of WR12
5	ALU	AB = NC DB = SD
6	CRU	Set ST(2) = CRUIN AB = Address of CRU bit DB = SD

4◀

JEQ, JGT, JH, JHE, JL, JLE, JLT, JMP, JNC, JNE, JNO, JOC, JOP

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
3	ALU	Skip to cycle #5 if TMS 9900 status satisfies the specified jump condition AB = NC DB = SD
4	ALU	AB = NC DB = Displacement value
5	ALU	AB = NC DB = SD

SRA, SLA, SRL, SRC

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
3	Memory read	AB = Address of the workspace register DB = Contents of the workspace register
4	ALU	Skip to cycle #9 if C ≠ 0 C = Shift count AB = NC DB = SD
5	ALU	AB = NC DB = SD
6	Memory read	AB = Address of WR0 DB = Contents of WR0
7	ALU	AB = Source address DB = SD
▶4 8	ALU	AB = NC DB = SD
9		AB = NC DB = SD
C	Shift the contents of the specified workspace register in the specified direction by the specified number of bits. Set appropriate status bits.	
9+C	Memory write	AB = Address of the workspace register DB = Result
10+C	ALU	Increment PC AB = NC DB = SD

AI, ANDI, ORI

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
3	ALU	AB = NC DB = SD
4	Memory read	AB = Address of workspace register DB = Contents of workspace register
5	Memory read	AB = PC+2 DB = Immediate operand
6	ALU	AB = NC DB = SD
7	Memory write	AB = Address of workspace register DB = Result of instruction

CI

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = NC
3	Memory read	AB = Address of workspace register DB = Contents of workspace register
4	ALU	AB = NC DB = SD
5	Memory read	AB = PC+2 DB = Immediate operand
6	ALU	AB = NC DB = SD
7	ALU	AB = NC DB = SD

LI

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
3	ALU	AB = NC DB = SD
4	Memory read	AB = PC+2 DB = Immediate operand
5	ALU	AB = Address of workspace register DB = SD
6	Memory write	AB = Address of workspace register DB = Immediate operand

LWPI

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
3	ALU	AB = NC DB = SD
4	Memory read	AB = PC+2 DB = Immediate operand
5	ALU	AB = NC DB = SD

LIMI

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
3	ALU	AB = NC DB = SD
4	Memory read	AB = PC+2 DB = Immediate data

CYCLE	TYPE	DESCRIPTION
5	ALU	AB = NC DB = SD
6	ALU	AB = NC DB = SD
7	ALU	AB = NC DB = SD

STWP, STST

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
3	ALU	AB = Address of workspace register DB = SD
4	Memory write	AB = Address of the workspace register DB = TMS 9900 internal register contents (WP or ST)

▸4

CKON, CKOF, LREX, RSET

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
3	ALU	AB = NC DB = SD
4	CRU	Enable CRUCLK AB = External instruction code DB = SD
5	ALU	AB = NC DB = SD
6	ALU	AB = NC DB = SD

IDLE

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
3	ALU	AB = NC DB = SD
4	CRU	Enable CRUCLK AB = Idle code DB = SD
5	ALU	AB = NC DB = SD
6	ALU	AB = NC DB = NC

RTWP

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC DB = Instruction
2	ALU	AB = NC DB = SD
3	ALU	WP+30
4	Memory read	AB = Address of WR15 DB = $Status_{OLD}$
5	Memory read	AB = Address of WR14 DB = PC_{OLD}
6	Memory read	AB = Address of WR13 DB = WP_{OLD}
7	ALU	AB = NC DB = SD

Machine-Cycle Sequence in Response to External Stimuli

RESET

4◀

CYCLE	TYPE	DESCRIPTION
1*	ALU	AB = NC DB = SD
2	ALU	AB = NC DB = SD
3	ALU	AB = 0 DB = 0
4	Memory read	AB = 0 DB = Workspace pointer
5	ALU	AB = NC DB = Status
6	Memory write	AB = Address of WR15 DB = Contents of Status register
7	ALU	AB = NC DB = PC
8	Memory write	AB = Address of workspace register 14 DB = PC+2
9	ALU	AB = Address of WR13 DB = SD
10	Memory write	AB = Address of workspace register 13 DB = WP
11	ALU	AB = NC DB = SD
12	Memory read	AB = 2 DB = New PC
13	ALU	AB = NC DB = SD

*Occurs immediately after $\overline{RESET}$ is released following a minimum 3 cycle $\overline{RESET}$

LOAD

CYCLE	TYPE	DESCRIPTION
1*	ALU	AB = NC DB = SD
2	Memory read	AB = $FFFC_{16}$ DB = Contents of $FFFC_{16}$
3	ALU	AB = NC DB = Status
4	Memory write	AB = Address of WR15 DB = Contents of status register
5	ALU	AB = NC DB = PC
6	Memory write	AB = Address of WR14 DB = PC + 2
7	ALU	AB = Address of WR13 DB = SD
8	Memory write	AB = Address of workspace register 13 DB = WP
▶4 9	ALU	AB = NC DB = SD
10	Memory read	AB = FFFE DB = New PC
11	ALU	AB = NC DB = SD

*Occurs immediately after last clock cycle of preceding instruction.

Interrupts

CYCLE	TYPE	DESCRIPTION
1*	ALU	AB = NC DB = SD
2	Memory read	AB = Address of interrupt vector DB = WP
3	ALU	AB = NC DB = Status
4	Memory write	AB = Address of WR15 DB = Status
5	ALU	AB = NC DB = PC
6	Memory write	AB = Address of WR 14 DB = PC+2
7	ALU	AB = Address of WR13 DB = SD
8	Memory write	AB = Address of WR 13 DB = WP
9	ALU	AB = NC DB = SD
10	Memory read	AB = Address of second word of interrupt vector DB = New PC
11	ALU	AB = NC DB = SD

*Occurs immediately after last clock cycle of preceding instruction

4◀

Timing

The timing of the ALU, CRU, and memory cycles is shown in *Figures 4-77, 78* and *79*. *Figure 4-80* shows the TMS9980A/81 memory cycle.

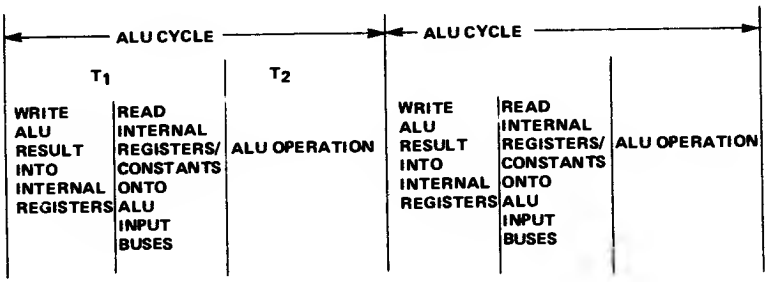


Figure 4-77. ALU Cycle

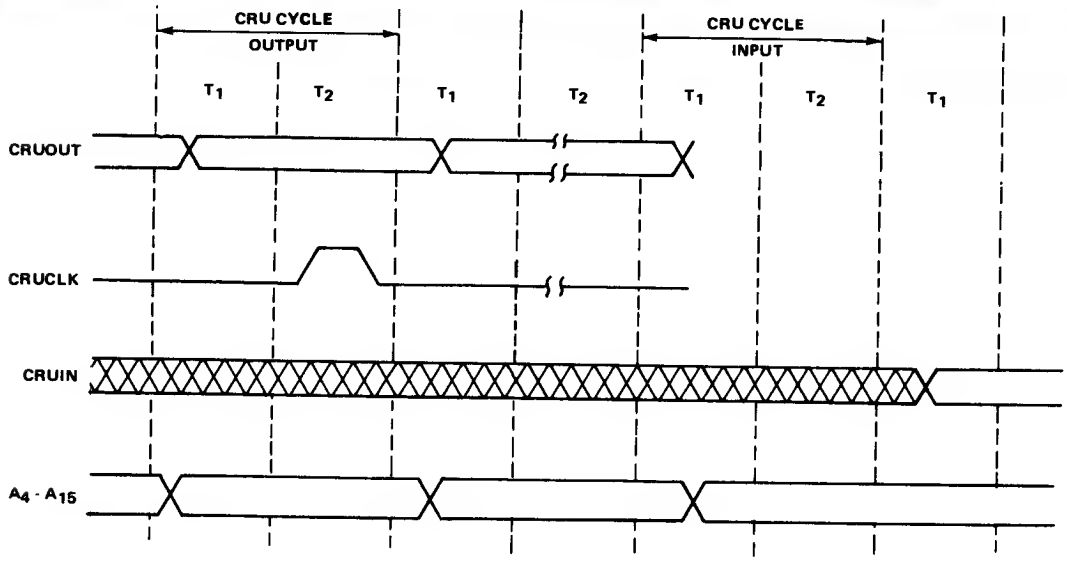


Figure 4-78. CRU Cycle.

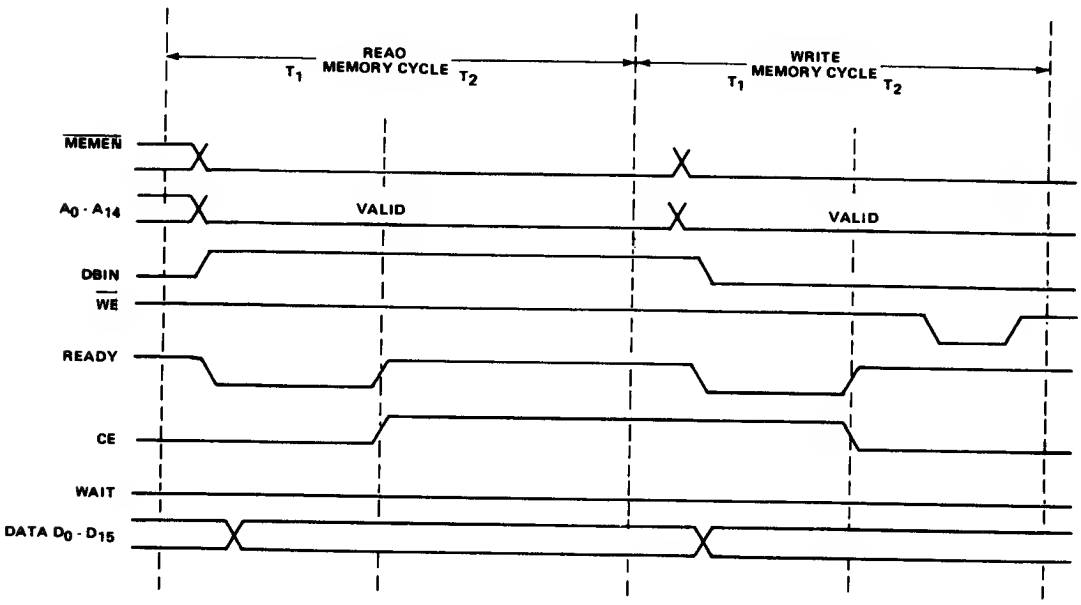


Figure 4-79. TMS 9900 Memory Cycle (No Wait States)

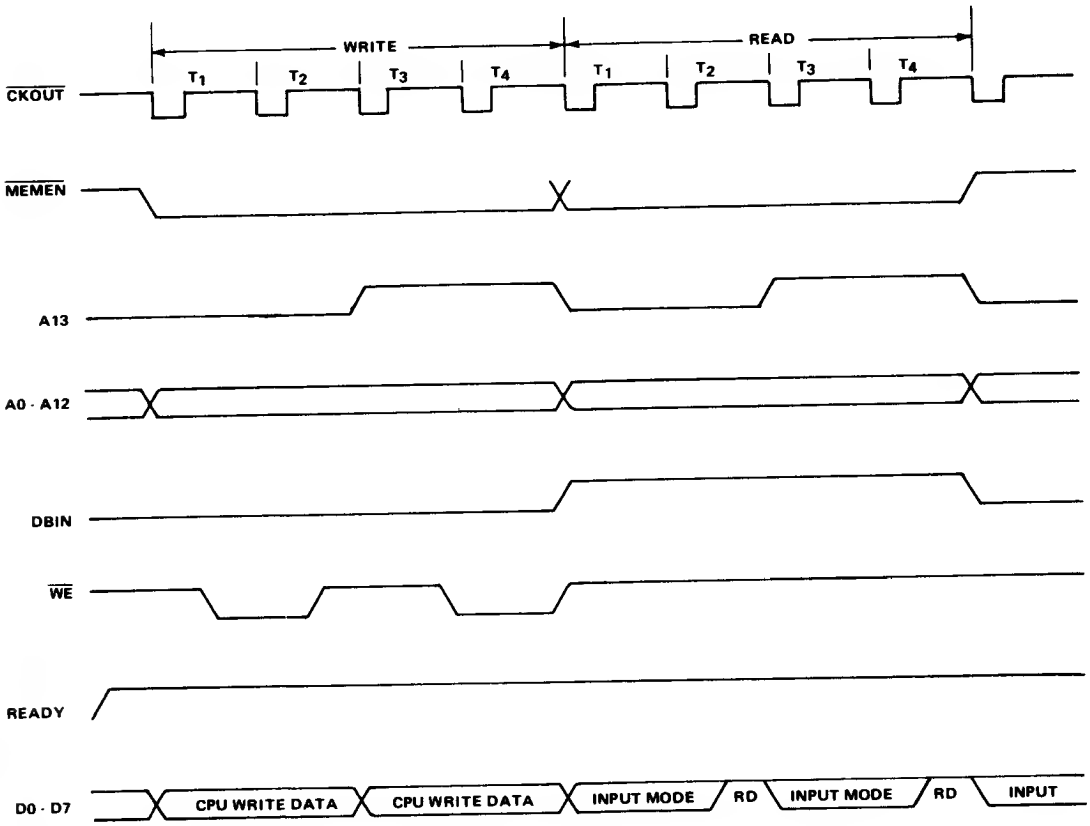


Figure 4-80. TMS 9980A/81 Memory Cycle (No Wait States)

CHAPTER 5

Software Design: Programming Methods and Techniques

5

9900 ARCHITECTURE

The 9900 system is illustrated in *Figure 5-1.* The major subsystems are the 9900 processor, the memory for program and data storage, and input and output devices for external communication and control. The processor controls the fetching of data and instructions from memory or input devices and the transferring of data from one location to another. The data and instructions are transferred 16 bits at a time in groups called words. These words are addressed or located by signals on the 15 address lines A_0 through A_{14} (called the address bus). A 15 binary bit address will select one of 32,768 memory words.

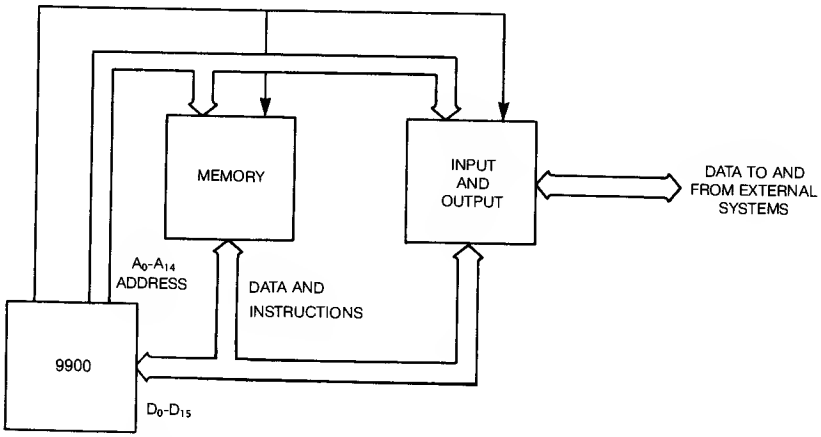


Figure 5-1. General 9900 System Structure

Internally, the processor generates a 16 bit address but the least significant bit, A_{15}, is not sent to the memory. Each word is further broken down into two 8 bit groups called bytes as shown in *Figure 5-2.* The first 8 bit byte of a word is located at an even address ($A_{15} = 0$). The second 8 bit byte is located at an odd address ($A_{15} = 1$). The byte selection is done internally in the processor once the full 16 bit data word is obtained from one of the 32,768 word locations in memory. Byte addressing is used only on instructions that perform byte operations; most 9900 instructions are word operations.

The processor contains certain basic elements as shown in *Figure 5-3.* The timing and control section is of primary interest to the hardware designer who must make certain that all system events occur in the correct order and at the correct time. The software designer is interested in what operations the ALU provides and the registers that determine the instruction and data addresses. These registers are the program counter, the status register, and the workspace pointer. In addition, the instruction register is of interest in understanding the basic instruction cycle of the processor. The 9900 contains other registers such as data address registers, ALU scratchpad registers, and so on. The processor also provides hardware to decode instructions, control the ALU operation, and to control the CRU input and outputs. These components all work together to provide the basic instruction fetch and execution cycle of the processor.

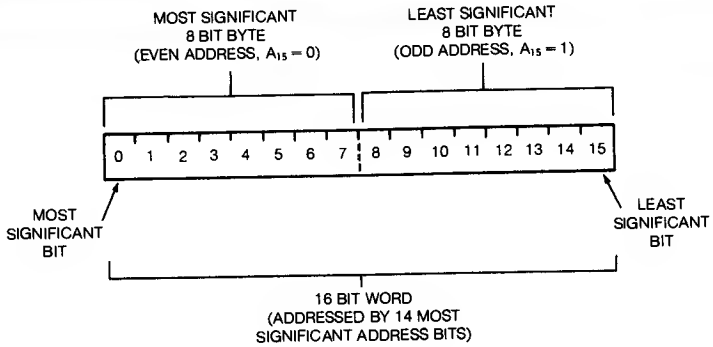


Figure 5-2. 9900 Words and Bytes

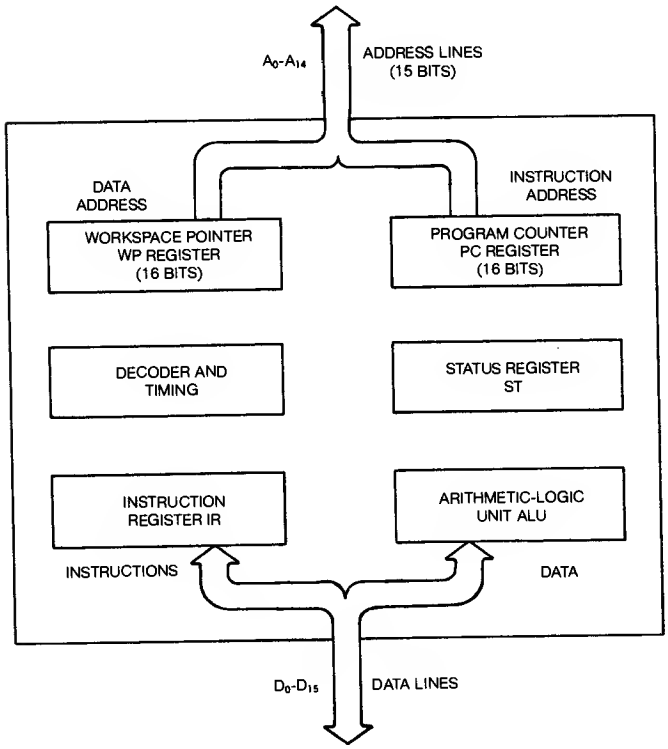


Figure 5-3. Basic 9900 Elements.

Instruction Register and Cycle

The instruction cycle that is performed over and over again by the processor consists of the following basic operations:

1) *Instruction Fetch* — the contents of the program counter are sent out on the address lines and a memory read is performed. The 16 bit instruction operation code word is sent from the memory along the data lines D_0 through D_{15} and is latched in the processor instruction register.

2) *Instruction Execution* — The instruction is decoded and executed. Usually, the address of the data to be operated on (source data) is generated and a memory read cycle is performed to get the data into the processor. Then a destination address is generated and a memory write cycle is performed to store the result of the operation at a desired destination memory location.

3) The contents of the program counter are changed to indicate the address of the next instruction and the processor returns to the instruction fetch operation.

This sequence is repeated continually as long as power is supplied to the processor.

The number of memory references required in the instruction operation depends on the format that is used for the instruction. Instructions can have one of 9 such formats as illustrated in *Figure 5-4.* The instruction code indicates to the processor how many memory references are required to get all the information needed by the instruction. The first memory read obtains the instruction code which determines which operation is to be performed and how the data is located. A second and possibly a third memory read may be required to obtain values or addresses for the data to be used in this operation. An immediate instruction (format 8) consists of two successive memory words: the first for the instruction code and a second word that contains the data constant to be used. Other instruction formats contain a T_s and/or a T_d field to indicate the existence of data addresses as part of the instruction. If a T_s or T_d two bit field contains a 10_2, the address of the source or destination locations or both will be contained in the one or two memory locations immediately following the instruction code word as illustrated in *Figure 5-5.* In these cases, one or two additional memory reads are required to fetch these addresses for use by the instruction to locate data in memory. Obviously, the more memory references required to get all of the instruction, the longer the execution time for that instruction. The programmer also needs to be aware of the number of words of memory required for each instruction in order to estimate program memory requirements.

Instruction Format	*Instruction Coding Fields**					
1 (Arithmetic)	CODE	B	T_d	D	T_s	S
2 (Jump/CRU)	CODE	DISPLACEMENT				
3 (Logical)	CODE	D	T_s	S		
4 (CRU)	CODE	C	T_s	S		
5 (Shift)	CODE	C	W			
6 (Program)	CODE	T_s	S			
7 (Control)	CODE	0000				
8 (Immediate)	CODE	00	W			
9 (Multiply, Divide, & Extended Operation)	CODE	D	T_s	S		

*The Fields are defined as follows:
CODE — Indicates the bits defining the operation code
B — Byte/Word Indicator (Single bit)
D — Workspace Register of the destination code (4 bits)
T_d — Addressing mode of the destination operand (2 bits)
S — Workspace Register of the source operand (4 bits)
T_s — Addressing mode of the source operand (2 bits)
C — Shift or Bit count (4 bits)
W — Indicates Workspace Register to be used (4 bits)

Figure 5-4. 9900 Instruction Formats

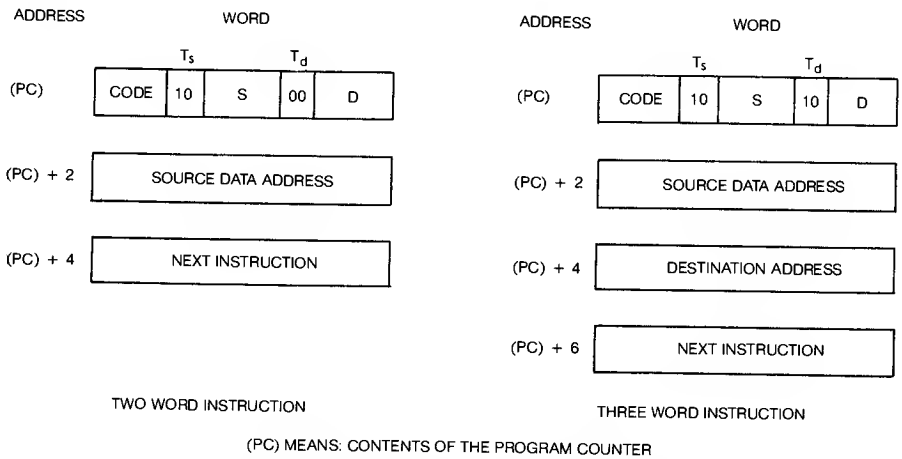


Figure 5-5. Example Memory Requirements for Format 1 Instructions.

Program Counter (PC)

The program counter, abbreviated PC, contains the address of the instruction to be executed as illustrated in *Figure 5-6.* Normally, after executing an instruction, the contents of the program counter are incremented by two to locate the next instruction word in sequence in memory. The programmer can control the contents of the program counter (and thus control where the next instruction is to be found) by using branch or jump instructions. These instructions offer the alternatives of taking the next instruction in sequence or jumping to another part of program memory for the next instruction.

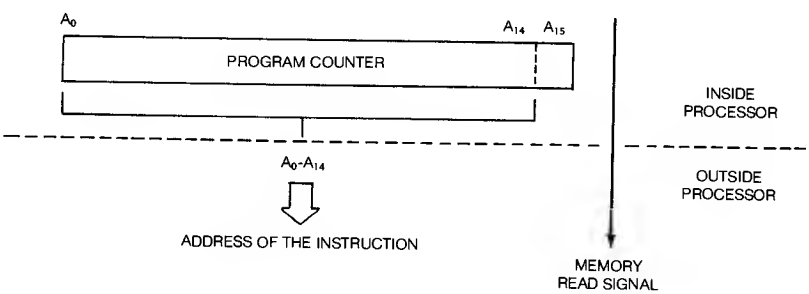
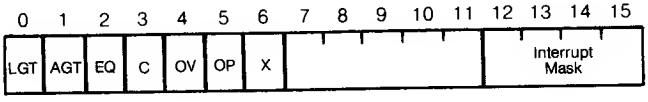


Figure 5-6. Purpose of the Program Counter

Status Register (ST)

The purpose of the status register is to store the general arithmetic and logic conditions that result from the execution of each instruction. This information lets the programmer know if the last operation caused a result equal to or greater than some reference number (often zero). It includes the information regarding the sign of the result (was it a negative or a positive number), the parity of the result (an odd or even number of one bits), and if a carry or overflow occurred (indicating that the 16 bit word length was insufficient to hold the result). The status register also contains a 4 bit code known as the interrupt mask which defines which of 16 hardware subsystem interrupt signals will be recognized and responded to by the processor. The information contained in the status register is defined in *Figure 5-7.*



Status Register Bit		
0	**LGT**	— *Logical Greater Than* — set in a comparison of an unsigned number with a smaller unsigned number.
1	**AGT**	— *Arthmetic Greater Than* — set when one signed number is compared with another that is less positive (nearer to −32,768).
2	**EQ**	— *Equal* — set when the two words or two bytes being compared are equal.
3	**C**	— *Carry* — set by carry out of most significant bit of a word or byte in a shift or arithmetic operation.
4	**OV**	— *Overflow* — set when the result of an arthmetic operation is too large or too small to be correctly represented in 2's complement form. OV is set in addition if the most significant bit of the two operands are equal and the most significant bit of the sum is different from the destination operand most significant bit. OV is set in subtraction if the most significant bits of the operands are not equal and the most significant bit of the result is different from the most significant bit of the destination operand. In single operand instructions affecting OV, the OV is set if the most significant bit of the operand is changed by the instruction.
5	**OP**	— *Odd Parity* — set when there is an odd number of bits set to one in the result.
6	**X**	— *Extended Operation* — set when the PC and WP registers have been to set to values of the transfer vector words during the execution of an extended operation.
7-11		— Reserved for special Model 990/10 computer applications.
12-15		— *Interrupt Mask* — All interrupts of level equal to or less than mask value are enabled.

Figure 5-7. *TMS9900 Status Register Contents*

Workspace Pointer (WP)

This register addresses the first word in a group of 16 consecutive memory words called a workspace as illustrated in *Figure 5-8.* These workspace words are called workspace registers and are treated by the processor as if they were registers on the processor chip. These workspace registers can be used as accumulators for arithmetic operations or for storage of often used data. When the workspace register contains the data used by the instruction, the T_s or T_d fields in the instruction format (see *Figure 5-4*) are 00. This way of locating instruction operands is an addressing mode called workspace register addressing. The workspace register can also be used to store the address of the data to be used instead of storing the data itself. In this case the T_s or T_d fields of the instruction code or format will be 01. This type of addressing (method of data location) is known as register indirect addressing. Workspace registers 1 through 15 can also be used to store the base address to which an offset will be added to determine a data address. This type of addressing is called indexed addressing and the T_s or T_d fields for this type of addressing will be a 10.

Some of the workspace registers are reserved for specific tasks as shown in *Figure 5-8.* If a certain type of subroutine branch called a branch and link (BL) is performed, register 11 is used to save the contents of the program counter at the time of the branch. In
▸5 another type of subroutine branch, the branch and link workspace (BLWP) instruction, registers 13, 14, and 15, are used to save the values of WP, PC, and ST registers, respectively, that were in the processor at the time the branch instruction occurred. These registers then allow the programmer to return to the situation or program context that existed prior to the branch. Register 12 is used to form the address of certain input and output bits that make up part of the communications register unit (CRU) subsystem.

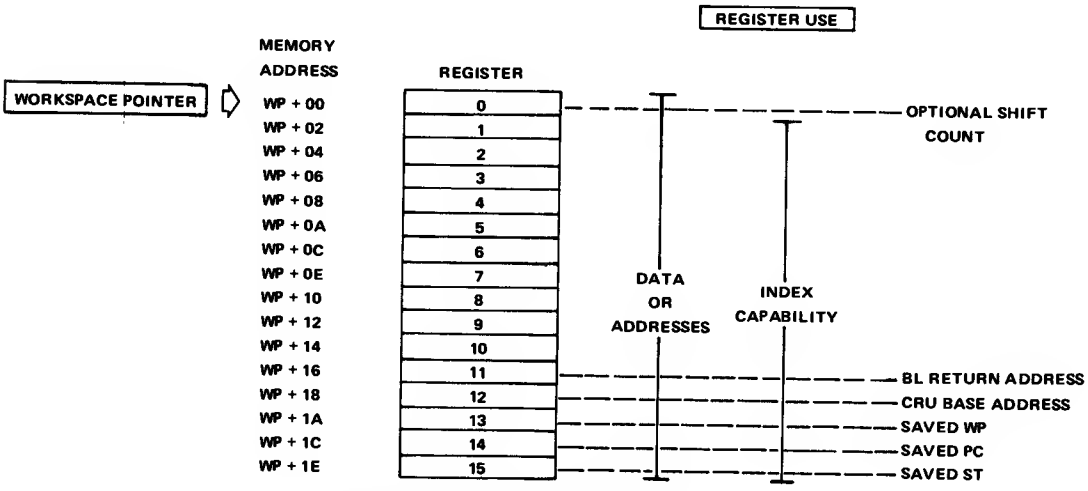


Figure 5-8. 9900 Workspace Structure

The relationships between the workspace registers and the instruction operations must be understood by the programmer to effectively utilize the 9900. Much of the addressing of data involves the use of workspace registers and branch and input/output instructions must use the dedicated registers 11 through 15. The use of the workspace in performing the basic program functions offered by the 9900 will be covered in detail throughout this chapter.

Program Environment or Context

The contents of the three processor registers (PC, WP, and ST) completely define the status of the system program at any given time. As illustrated in *Figure 5-9,* the program counter keeps track of that part of the system program currently being executed by specifying the current instruction location. The status register keeps track of the logical and arithmetic conditions that result from the execution of each instruction. The workspace pointer keeps track of the location in memory of the sixteen general purpose workspace registers currently being used by the program. The contents of the processor and workspace registers define the current program environment or *context* of the system. A change in the contents of these registers will change the environment to a new part of program memory and a new workspace area. Thus, the system will be switched to a new environment or program context by such a change. Similarly, by restoring the contents of PC, WP, and ST to original values, the program environment will be switched back to the original context and continue executing in the original program environment.

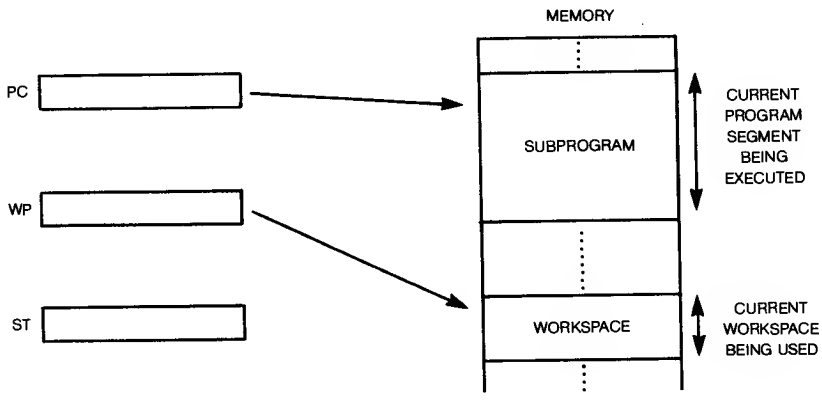


Figure 5-9. Program Context

MEMORY ORGANIZATION

The 9900 system memory must provide storage locations for the system program and subprograms and storage for system data. Since the physical devices used for storing instructions are often a different type of memory device from those used to store data, the program is usually stored in consecutive blocks of memory separate from the blocks of data. This is illustrated in *Figure 5-10.* Also shown in *Figure 5-10* are groups of memory locations that must be reserved for program and workspace addresses used by certain subprograms. Thus, the memory is subdivided into three types of storage of storage locations: program memory, data memory, and reserved or dedicated memory.

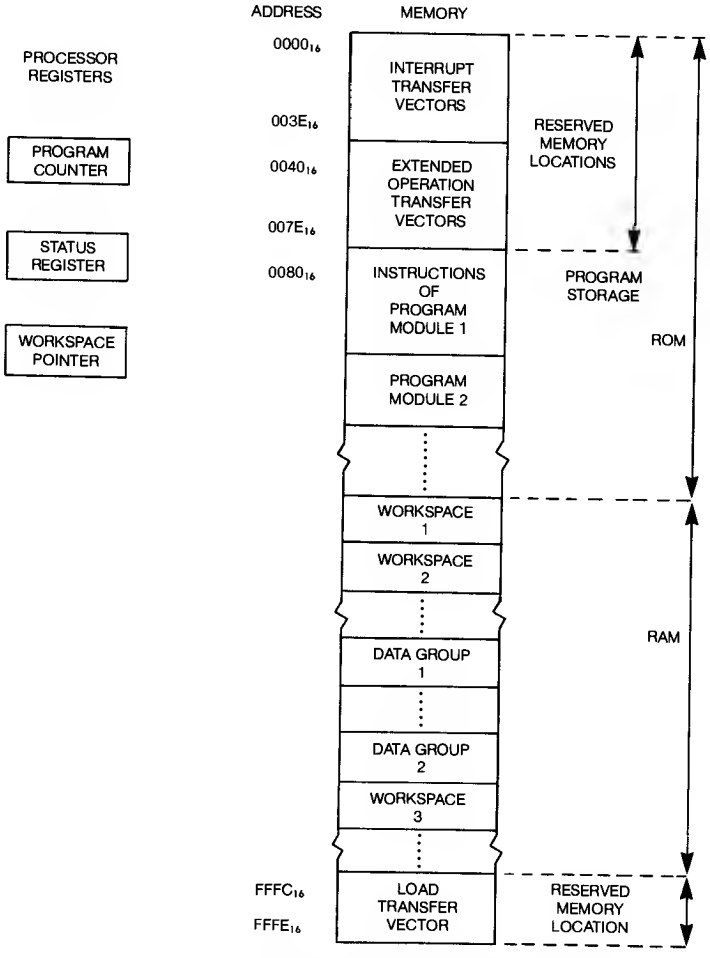


Figure 5-10. 9900 Memory Organization

RAM/ROM Partitioning

The program storage should be non-volatile so that the system program is not lost when the system power is turned off. Further, it is often desirable for the program memory to be a read-only memory or ROM. High volume read-only memory devices are mask programmable by the manufacturer. Alternatively, the program storage can be placed in a programmable read only memory (PROM). These devices may be economically programmed in smaller quantities. Programming may be performed by the user or by the distributor of the devices. Since both PROM and ROM devices provide word storage in consecutive addresses and the processor executes programs by going through instructions in sequence, the instructions that comprise a given subprogram should be placed in consecutive addresses in a block of memory words called a program module. It is not necessary that all program modules be adjacent to each other in memory, though certainly it is reasonable to do so.

System data storage, excluding input/output registers, provide storage for data being processed by the system program. These storage locations are usually located in consecutive blocks of memory. Since the data memory must provide both read and write capability, it is often called read-write memory. A more common terminology is random access memory or RAM, though this is somewhat misleading, since the program memory in ROM may also be randomly accessed.

The range of addresses that are assigned to the RAM storage locations and those that are assigned to the ROM locations are somewhat arbitrary. The reserved locations are the first locations in program memory, so that part of the ROM addresses are these reserved location areas. Often hardware considerations such as the simplification of the address decoding circuitry may decide the range of addresses that are used for each type of memory.

Reserved Memory

The program modules, workspaces, and general data storage can generally be placed anywhere in memory, as long as the following reserved locations are preserved:

1) The first 32 words of memory (addresses 0 through $3E_{16}$) are reserved for interrupt transfer vectors.
2) The next 32 words of memory (addresses 40_{16} through $7E_{16}$) are reserved for extended operation transfer vectors.
3) The last two words of memory (addresses $FFFC_{16}$ and $FFFF_{16}$) are reserved for a load or reset transfer vector.

These transfer vectors provide storage for a value to be placed in the workspace pointer and a value to be placed in the program counter in order to switch the program context from its current environment to a subprogram and new workspace. This new subprogram and workspace context is used to respond to a hardware interrupt signal, a hardware reset signal, or an instruction called an extended operation (XOP).

WORKSPACE UTILIZATION

THE WORKSPACE CONCEPT AND USES

The advanced memory-to-memory architecture of the 9900 affords multiple register files in main memory for efficient data manipulation and flexible subroutine linkage. The usage of the workspace must follow certain constraints for optimum performance. Each workspace is a contiguous block of 16 words in main memory. All 16 general purpose registers are available to the programmer for use in any of four ways:

1) *Operand Registers* — to contain data for arithmetic and logical operations.
2) *Accumulators* — to store intermediate results of arithmetic operations.
3) *Address Registers* — to specify memory location of operands.
4) *Index Registers* — to provide an offset from a base address to define an operand location.

The workspace pointer in the processor contains the address of workspace register 0. The address of any workspace register R is:

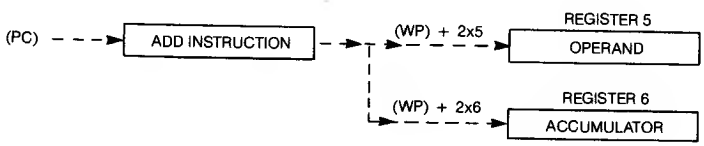
$$\text{Memory Address of Register R} = (WP) + 2R$$

where (WP) means the contents of the workspace pointer.

▶5 When a workspace register is specified as an operand in an instruction, (workspace register addressing mode) the workspace register contains binary data for use by the instruction. As an example, consider the addition of the data in register 5 to the data in register 6. The instruction format is:

A 5,6

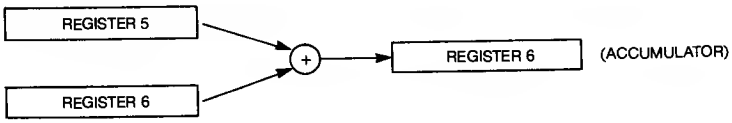
with address calculations of:



which is interpreted as follows:

1) The contents of the program counter addresses the instruction in ROM.
2) The instruction indicates workspace register addressing causing the calculation of the workspace addresses to locate the data to be used by the instruction (contained in registers 5 and 6) in RAM.

The resulting hardware operation with the data thus located is:

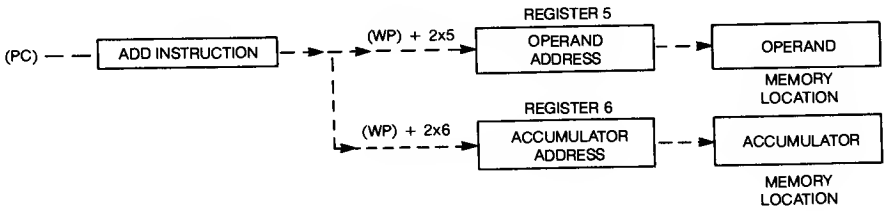


In this example, register 5 is functioning as an operand and register 6 is functioning as an accumulator. The difference between an operand and an accumulator register is that operands remain unchanged by an operation, while accumulators assume new values, the result of the operations.

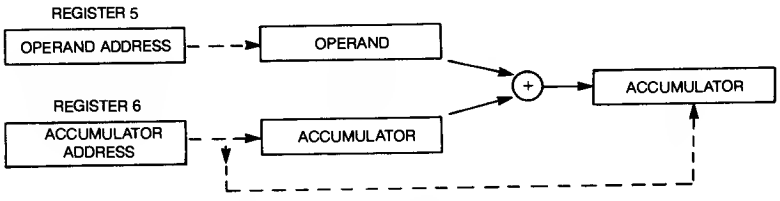
The contents of a workspace register may be the address of an operand or an accumulator in main memory. Address registers are accessed through workspace register indirect addressing, with or without autoincrementing. If autoincrementing is not used, the content of the workspace register (the address of the data) is not changed by the operation. If autoincrementing is used, the address contained in the workspace register is incremented by one for byte operations and by two for word operations. An example of an addition instruction in which both the operand and the accumulator are specified by register indirect addressing is: 5◀

```
A          *5,*6
```

with the address computations:



with the resulting hardware operation:

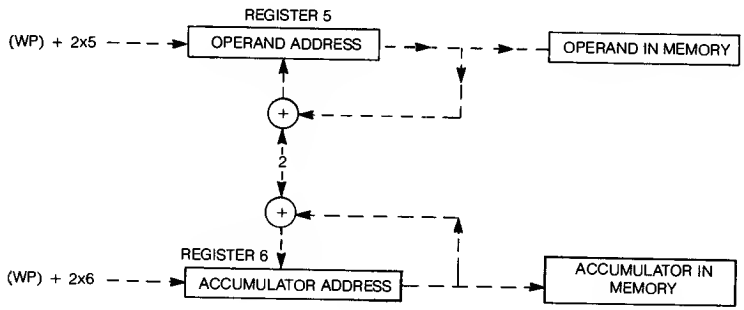


The contents of the address registers are not changed in execution since autoincrementing is not used.

Autoincrementing is often used when accessing structured data and data arrays. To add this feature to this example, the following format would be used:

A *5+, *6+

which would result in the same events as described for standard workspace register indirect addressing with the addition of an incrementing by two of the contents of the address register 5 and 6:

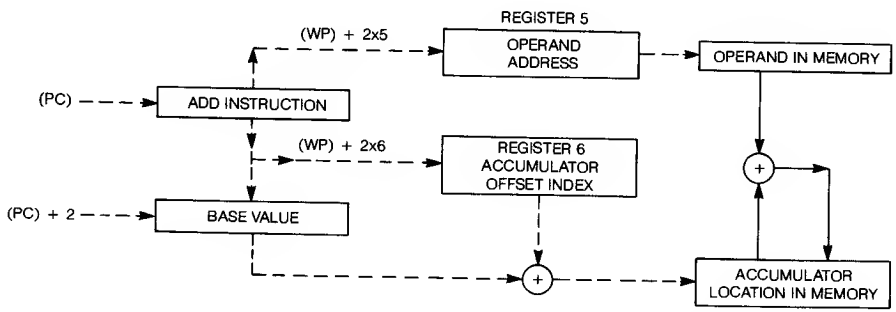


▶5 The addresses are modified (incremented by two) after the operand and accumulator addressing operations are completed.

When the workspace register is used as an index register, its contents specify an offset from a base address. The sum of this offset and the base address contained in the instruction defines the memory location of program data. Workspace registers act as index registers when the indexed addressing mode is used. The only restriction on the use of workspace registers as index registers is that register 0 cannot be used as an index register. An example of using register 5 as an indirect address register for the operand and register 6 as an index register for addressing the accumulator would be:

A *5, @BASE (6)

The binary address BASE is the second word of the two word add instruction, with address calculations as follows:



The operand data is added to the accumulator data and the sum is stored in the accumulator location.

Dedicated Areas of Workspaces

Any register of a workspace may be used as a general purpose register (with the exception of register 0 not being available as an index register). A few of the registers are used by 9900 hardware in certain ways, and the software designer must observe these constraints to assure the integrity of stored data and program and hardware linkages. *Figure 5-11* shows the way the workspace is viewed by the hardware.

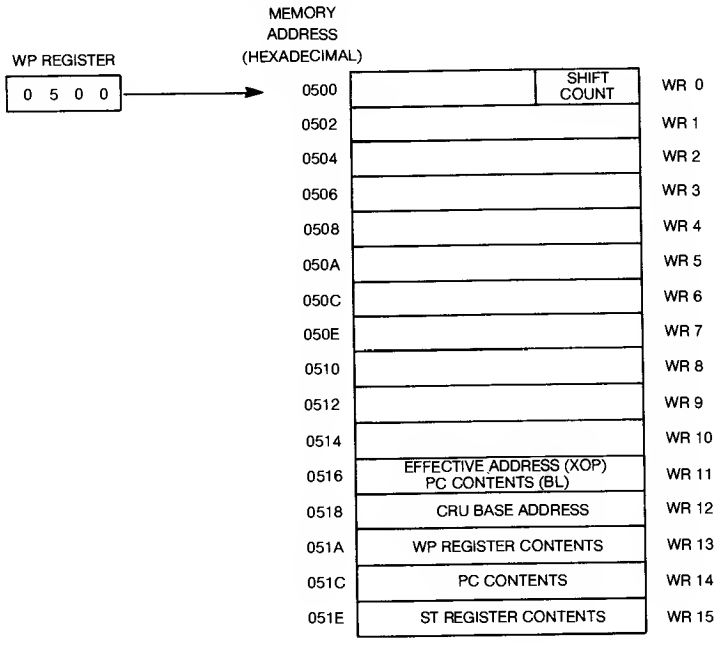


Figure 5-11. Reserved Areas of 9900 Workspaces

An examination of *Figure 5-11* reveals the following areas that may have to be reserved in a workspace:

Registers 13, 14, and 15 — Context Switches

These three workspace registers are loaded with current values of the workspace pointer, program counter, and status register with each context switch. A context switch occurs in response to an interrupt or in executing a BLWP or XOP instruction. When an RTWP return instruction is executed, the processor restores these values to the processor registers from the last three workspace registers. To insure that this return linkage is not destroyed, the programmer must insure that subprogram operations or subsequent context switches do not alter the contents of registers 13, 14, or 15.

Register 0 — Shift Instruction

Bits 12 through 15 of register 0 may specify a bit count for shift instructions. The 9900 shift instructions have the format:

```
OPCODE      R, SCNT
```

where the OPCODE is one of the shift instruction mnemonics SLA, SRC, SRL, or SRA, R is the operand register, and SCNT specifies the number of bit position to be shifted. When SCNT is zero, bits 12 through 15 of register 0 specifies the shift count. If both SCNT and bits 12 through 15 of register 0 are zero, a 16 bit shift will occur.

Register 11 — XOP and BL Instructions

Register 11 is used to save address information in extended operation instructions (XOP) and Branch and Link subroutine jump (BL) instructions. The BL instruction provides a means of subroutine linkage without the overhead of a context switch. Previous contents of register 11 are replaced with the program counter contents when a BL occurs. Return to the calling procedure is accomplished with the RT pseudo-instruction or by an indirect branch B *11. No critical data should be stored in register 11 if a BL instruction is to be executed.

▶5 In the case of the extended operation instruction, an address is passed to register 11 during the XOP context switch. For example:

```
XOP      VAR, OPNUM
```

OPNUM is the XOP number and locates the XOP transfer vector in main memory through the formula:

$$\text{Transfer Vector Address} = 40_{16} + 4 \times \text{OPNUM}$$

The effective address of the source operand VAR is placed into register 11 of the XOP workspace. Even if VAR is not provided, register 11 contents will be altered by executing an XOP instruction.

Register 12 — CRU Bit Addressing

The 9900 communications register unit (CRU) is a direct command-driven I/O interface. The five CRU instructions (SBO, SBZ, TB, LDCR, and STCR) all depend on the presence of a CRU hardware base address in bits 3 through 14 of workspace register 12. None of these instructions alter the content of register 12.

Workspace Location

Workspaces may be located anywhere in main memory. In practice, 66 words of memory are reserved to implement necessary hardware functions (transfer vectors). Workspaces and data may be stored in any other memory area, known as general memory. The memory locations reserved for 9900 transfer vectors for interrupts and extended operation instructions are memory addresses 0000_{16} through $007E_{16}$. The last two words of memory (addresses $FFFC_{16}$ and $FFFE_{16}$) are reserved for a load function transfer vector, so the last data or instruction word can occur at address $FFFA_{16}$.

Within general addresses 0080_{16} through $FFFA_{16}$, workspaces can be independent, or used in common by different program segments or subprograms. To reduce memory requirements of a software system, routines can share workspaces. The effect of a BL call to a subroutine is illustrated in Figure 5-12. The program counter is changed to fetch the instructions from the subroutine, but the *workspace pointer is not changed,* which results in a workspace shared by the called and the calling procedures.

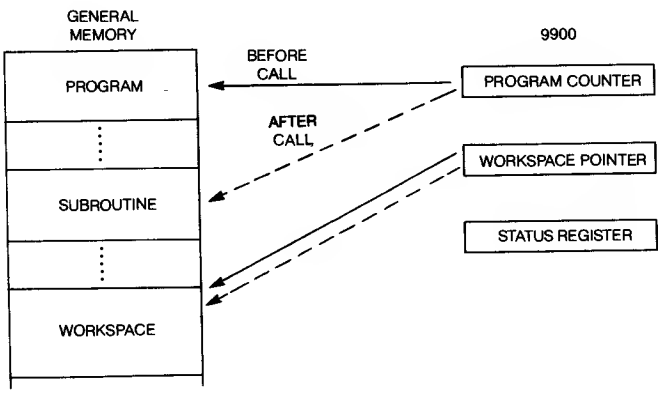


Figure 5-12. Shared Workspace Subroutine Call

When a routine requires the use of a large number of workspace registers, an independent workspace will be needed for that routine. In some cases, independent workspaces are used for routines when little common data is needed. When workspaces have no common memory words, parameter or data passing can be done by using the old program counter and workspace pointer. For example, in a context switch, which saves the old workspace pointer in the new workspace register 13, any of the old workspace registers can be accessed by referring to the contents of the new register 13. The contents of register 13 addresses the old workspace register 0. The use of register 13 as an index register allows the programmer access to any other of the old workspace registers. Thus, to access old register 0 as an operand in an add instruction, the following instruction would be used:

```
A          *13,7
```

This instruction specifies the contents of old register 0 (addressed by the contents of new register 13) as an operand and new register 7 as an accumulator. To address old register 10, the following indexed addressing approach could be used:

```
A          @20(13),7
```

This instruction adds 20 to the contents of new register 13 to generate the address of old workspace register 10, which is then used as an operand in the add operation. The effect
▶5 of a context switch in providing an independent workspace is illustrated in *Figure 5-13.*

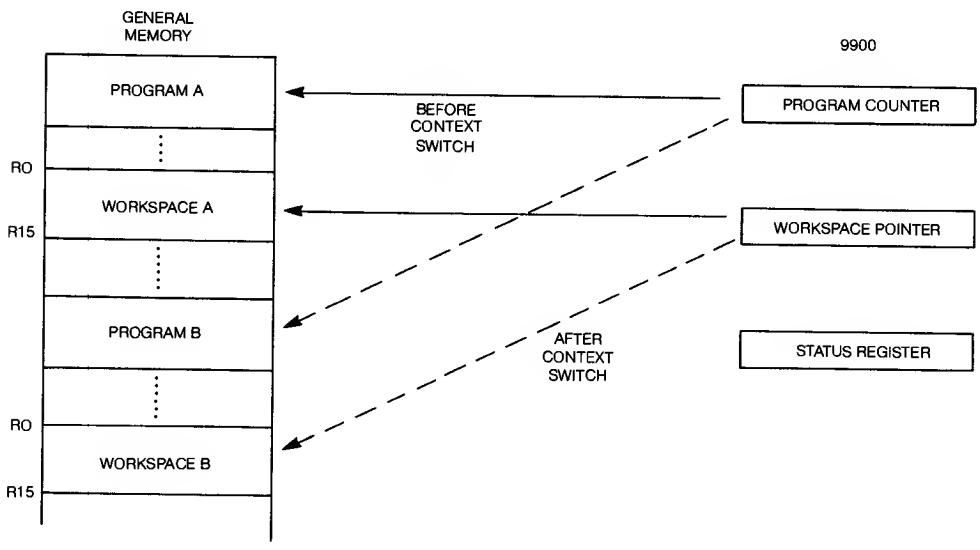


Figure 5-13. Independent Workspaces

SUBROUTINE TECHNIQUES

Software systems are implemented with a set of subprograms, usually subroutines. Subroutines offer several advantages over incorporation of all code into a large main program:

1) Repetition of code is reduced. Modular coding of repeated processes saves memory requirements of software.
2) Documentation is simplified. The clarity of complex programs is enhanced by breaking the overall task into manageable subsystems.
3) Debugging time is reduced. A complicated system can be made functional one module at a time.

These advantages point out the importance of understanding the characteristics of 9900 subroutine calls. The most important characteristics are the way the subroutine linkages back to the calling program are handled and the way parameters are passed between the calling program and the subroutine. The linkage procedures for the types of subroutine calls are discussed first.

Types of Subroutines

Three types of subroutine calls are used with the 9900. The following table summarizes the calls and returns for each type: 5◀

Call to Subroutine		*Return to Calling Procedure*	
Mnemonic	*Meaning*	*Mnemonic*	*Meaning*
BL	Branch & Link	RT or B *11	Return
BLWP	Branch & Link Workspace Pointer	RTWP	Return with Workspace Pointer
XOP	Extended Operation	RTWP	Return with Workspace Pointer

The branch and link instruction is a fast transfer to a routine that shares the workspace with the calling procedure. Execution of a BL causes the contents of the program counter to be stored in workspace register 11. The new program counter value is the single argument of the BL instruction. An example of a typical BL instruction is:

```
PT      BL      @SUB1
```

SUB1 is the label of the first instruction of the subroutine being called. After execution of the BL instruction, program flow will continue at the symbolic address SUB1. Upon execution of the BL instruction, the update value of the program counter (address PT+4) is stored in workspace 11 (PT is the symbolic address of the BL instruction). This process of a shared workspace subroutine call is illustrated in *Figure 5-14.* Return to the calling procedure is through the RT pseudo-instruction which is equivalent to the indirect branch;

```
B       *11
```

Since the BL instruction always reloads Workspace register 11, special steps must be taken to insure that the critical return address is not overstored. Generally, register 11 should not be used to save a variable whose value will be needed after a BL instruction occurs. Similarly, after a BL instruction has been executed (and before a RT instruction
▶5 has been executed), register 11 cannot be used by any instruction that would change the contents of register 11, such as using register 11 as an accumulator or executing another BL instruction. If multiple levels of BL calls are to be used, a push-down stack must be established to save intermediate return linkage. Techniques for setting up a stack are discussed under the topics of multiple level subroutine calls and reentrancy.

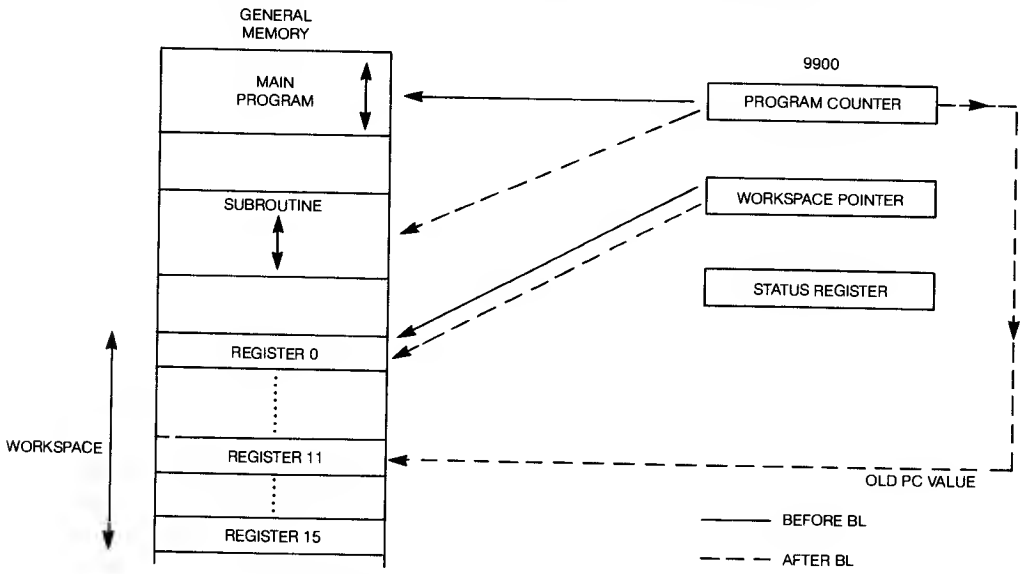


Figure 5-14. Effects of BL Instruction

The branch and load workspace pointer (BLWP) is a subroutine call that initiates a context switch. When a context switch occurs, the programming environment is changed to allow the subroutine to use a new register file (workspace). BLWP has the following effect as illustrated in *Figure 5-15:*

1) A transfer vector located by the argument of the BLWP instruction supplies a new workspace pointer value and program counter value.
2) The old values of WP, PC, and ST are saved in registers 13, 14, and 15, respectively, of the new workspace.
3) Execution proceeds in the subroutine using the new PC value.

The 9900 format for a typical BLWP using Symbolic addressing is:

```
PCL     BLWP   @TVAL
```

where PCL is an arbitrary label and the symbolic address of the location of the BLWP instruction in general memory. TVAL is the symbolic address of the transfer vector, which in turn provides new values for the workspace pointer and the program counter. The contents of workspace register 13 through 15 of the new workspace are reserved for storage of the return linkage. Since the BLWP can store return linkage in an independent workspace, multiple subroutine levels may be implemented without a return stack as long as no two subroutines use the same workspace (transfer vector). Although the example in *Figure 5-15* uses symbolic addressing mode, other addressing modes can be used.

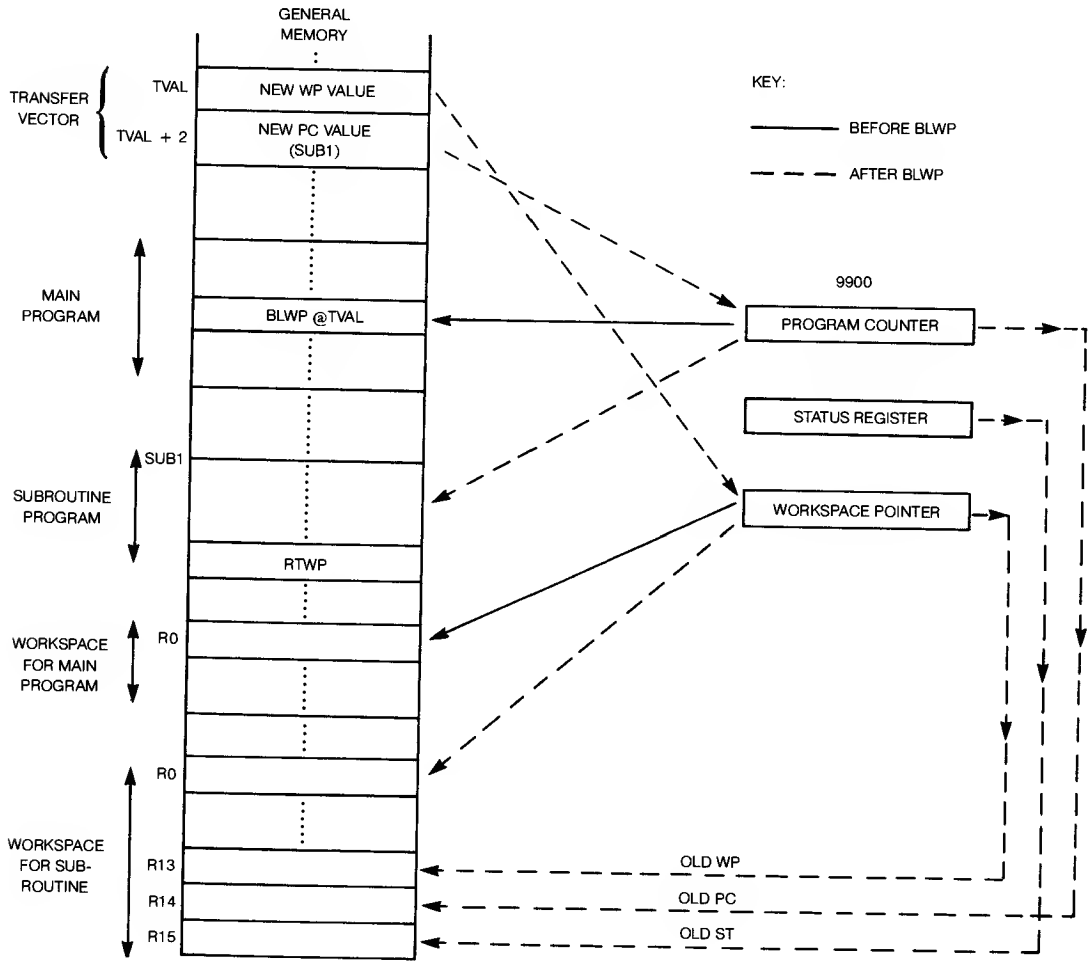


Figure 5-15. Execution of BLWP Instruction (BLWP @TVAL)

Extended operation instructions (XOP) offer a means of expanding the 9900 instruction set. The implementation of an XOP is similar to the execution of a BLWP; *the instructions differ only in the location of the transfer vector and in the parameter passing feature offered by the XOP.* The execution of an XOP is illustrated in *Figure 5-16* and consists of the following events:

1) Identify the XOP number (N) and locate a transfer vector in memory at the address $0040_{16} + 4 \times N$.
2) Use the transfer vector word one as the new workspace pointer value and the second word of the vector as the new program counter value.
3) Save the old contents of WP, PC, and ST in new workspace registers 13, 14, and 15, respectively.
4) Store the effective address of the source operand in new workspace register 11.

Thus, XOP initiates a context switch with the added benefit of direct passing of a parameter address to the new workspace (register 11). By using an assembler directive DXOP, the user can define a mnemonic string to present one of the 16 XOP transfer vectors. This mnemonic can then be used in the program as a user defined instruction, improving the clarity of the program coding. For example, to define XOP 15 as the mnemonic SAMPL, the following directive can be used:

```
DXOP   SAMPL, 15
```

5◀

Then, instead of using the standard XOP entry in the program:

```
XOP    @PARAM, 15
```

The programmer can insert the newly defined mnemonic:

```
SAMPL  @PARAM
```

The XOP call is a software trap to a user-defined routine. It functions as though the routine were a single instruction added to the 9900 set of operation codes, hence the name "extended operation."

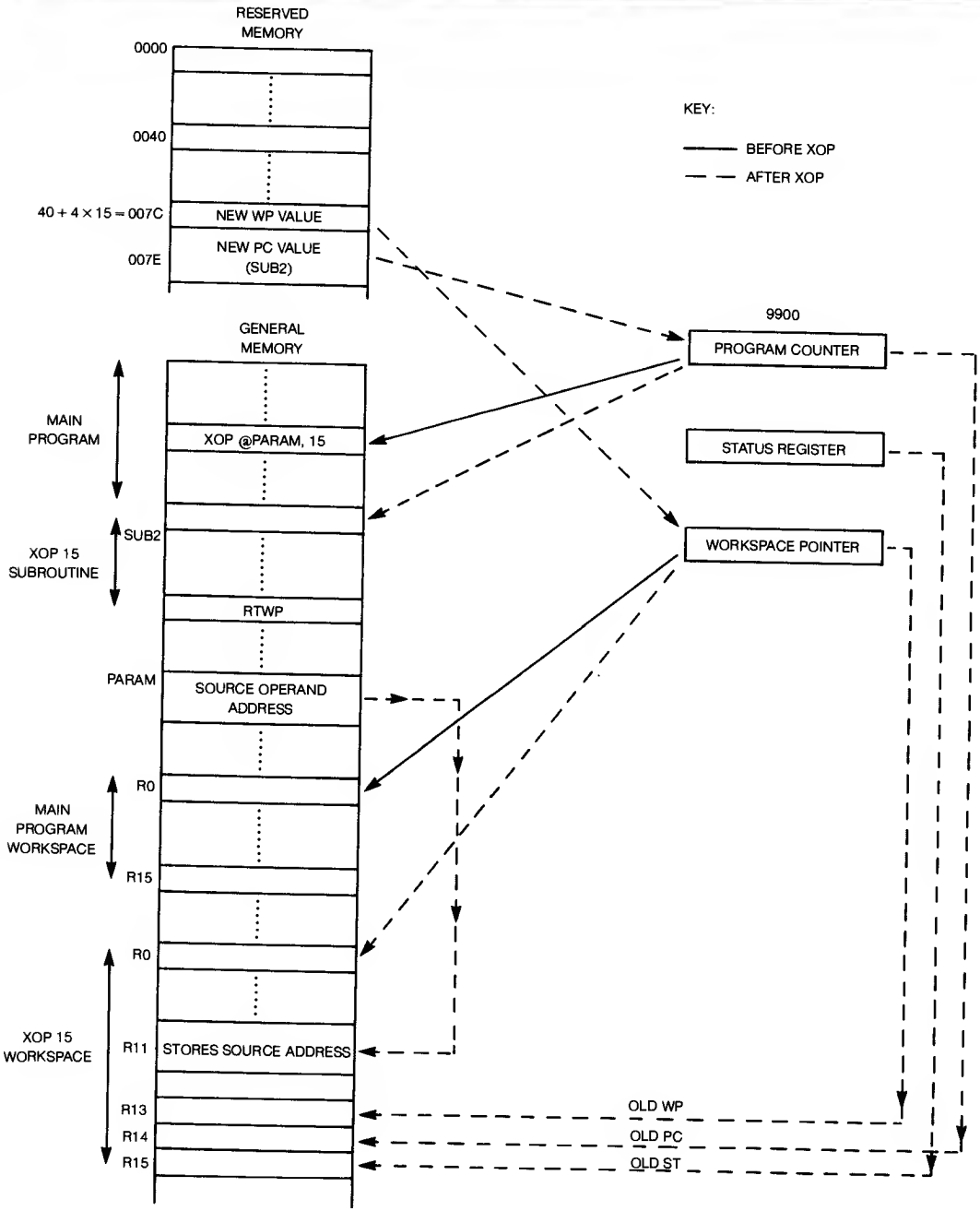


Figure 5-16. Execution of XOP Instruction (XOP @PARAM, 15)

Parameter Passing

Most subroutines require access to data generated by the calling procedure. Different subroutine call types mandate different parameter passing techniques. All three types of subroutine calls, BL, BLWP, and XOP, *transfer the old contents of the program counter to the called procedure.* This return linkage provides a powerful tool for parameter passing as illustrated in *Figure 5-17.* A parameter list can be assembled in a block of words following the call and accessed through the old program counter by workspace register indirect autoincrement addressing. Regardless of the number of parameters used in any given call, the program counter must be incremented past the whole list, so that the return will be to the next instruction in the calling program. A subroutine call using only one of two passed parameters is shown in the following example:

```
0200          BL     @ANG      Call Subroutine ANG
0202  FLAG1   DATA   >0        Parameter 1 is 0000₁₆
0204  FLAG2   DATA   >1        Parameter 2 is 0001₁₆

0600  ANG     MOV    *11+,3    Move Parameter 1 into R3
0602          C      3,2       Compare Parameter 1 to contents of R2
0604          JEQ    FIRSTEQ   Try next test if equal
0606          INCT   R11       Move Return PC past parameter 2
0608          B      *11       Return
060A  FIRSTEQ
```

The subroutine ANG checks the first parameter against the contents of R2. If an inequality is found, the branch to continue the routine at FIRSTEQ is not taken. The move instruction which loaded parameter 1 into the workspace increments the program counter in R11 by 2 so that register 11 now points to parameter 2. The INCT instruction is required to increment the program counter value in R11 past parameter 2 to point to the next instruction in the calling program.

When parameters are passed in this way using the program counter, good programming practice dictates that they be constants or addresses only and not variables. Variable quantities should be stored in memory external to program code. To 'nest' variable data in program code causes in-line code modification, which would produce code that would be inoperative if stored in ROM.

The example above dealt with the BL subroutine call, though the same technique can be applied to BLWP or XOP calls. These calls store the program counter in workspace register 14, so the indirect address register must be 14 instead of 11.

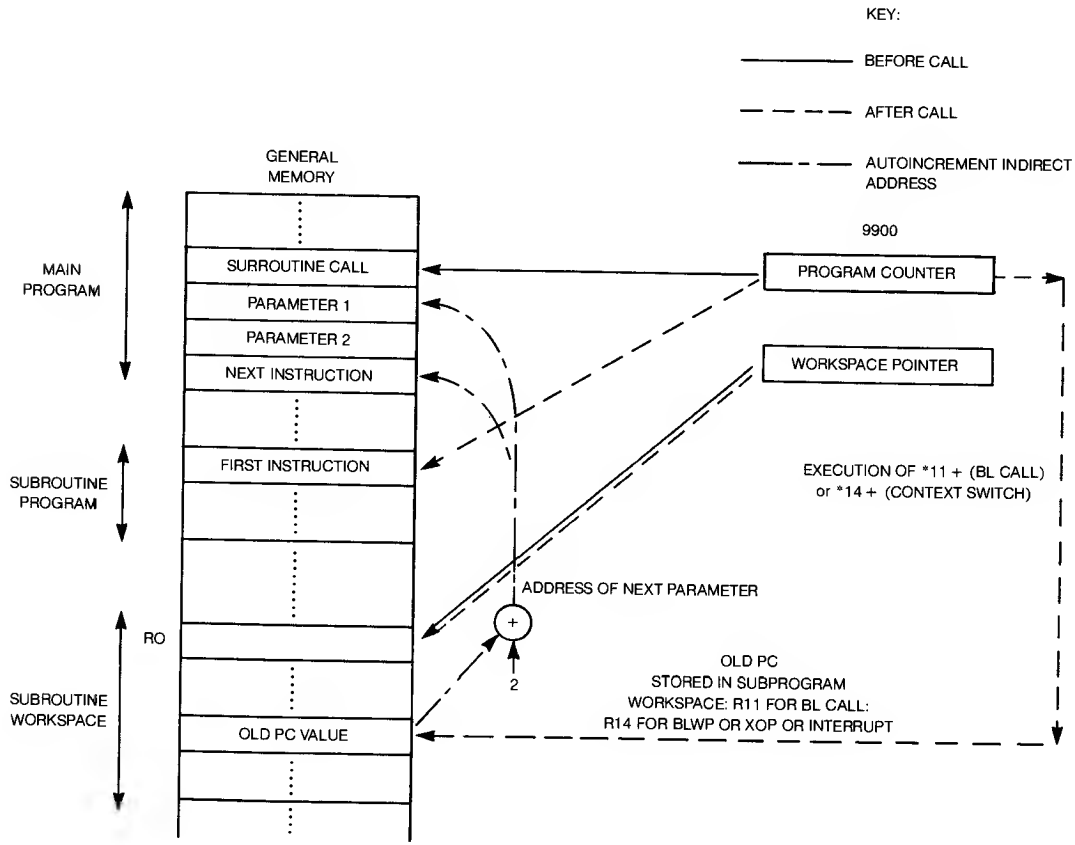


Figure 5-17. Parameter Passing Using Old Program Counter Value.

Another method of parameter passing is used when a context switch occurs. Both BLWP and XOP cause the old contents of the workspace pointer to be stored in the new workspace register 13. By using register 13 in the called procedure, access is gained to parameters in the old workspace as illustrated in *Figure 5-18.* Direct access to old register 0 is provided, but to use other parts of the old workspace, indexed addressing provides the most convenient access to old registers 1 through 15 without changing the old workspace pointer value. For example, to move the contents of old workspace register 2 to new workspace register 5, the following instruction can be used:

```
MOV     @R2*2(13),R5
```

which causes the address of the operand to be the contents of register 13 plus 4, which is the address of old workspace register 2. Similarly, to move the contents of old workspace register 7 to old workspace register 6:

```
MOV     @R7*2(13),@R6*2(13)
```

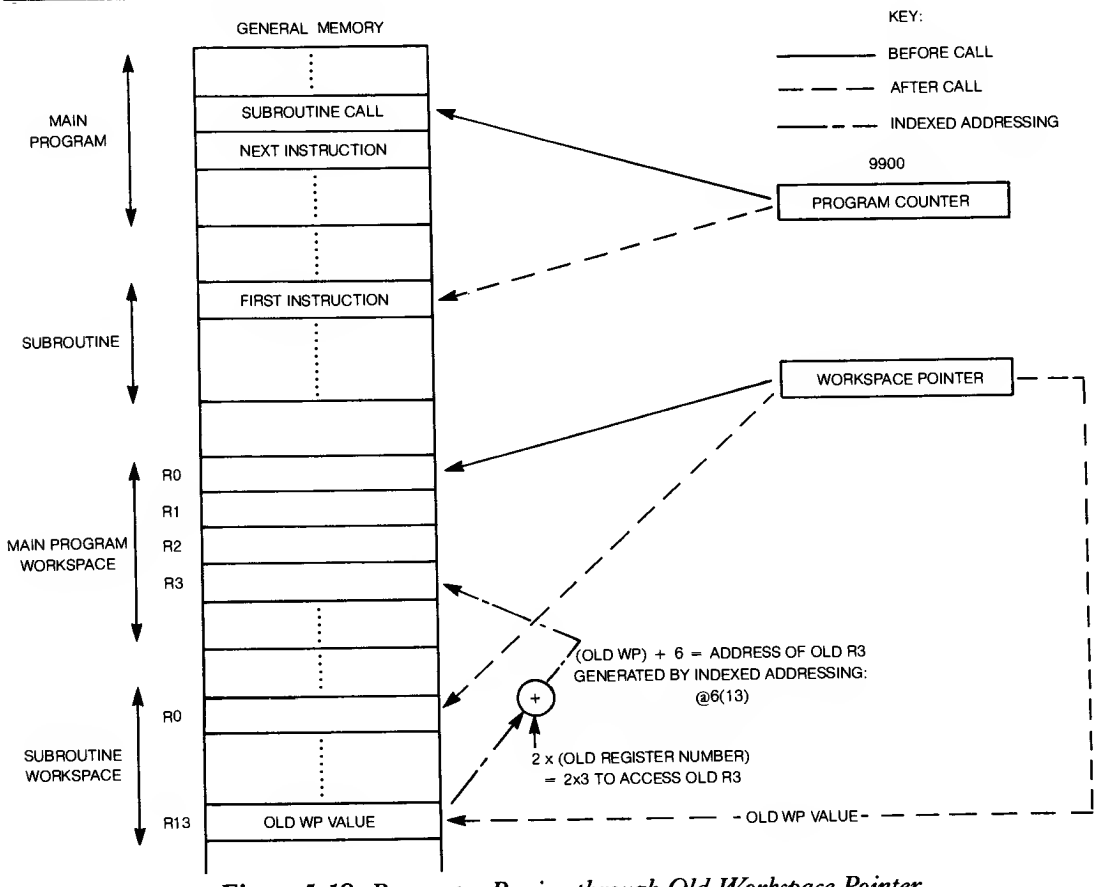


Figure 5-18. Parameter Passing through Old Workspace Pointer

A final type of parameter passing applies only to XOP context switches. The single argument of an XOP call specifies the effective address of a source operand. This form of parameter passing avoids the risk of changing the old PC and WP. The overhead of changing the WP and PC pointers is also avoided to increase execution speed. As an example, if XOP 9 has been defined as FADD by a DXOP directive, the call:

```
FADD   @LIST
```

causes the address stored at location LIST to be placed in register 11 of the subroutine workspace. Then, workspace register indirect addressing can access the parameter. For example, if in the FADD subroutine it is desired that the parameter be incremented by two, the following instruction would be used.

```
INCT   *11
```

The use of the parameter through its address in register 11 is straightforward and doesn't interfere with the return linkage. This type of parameter passing has already been illustrated in *Figure 5-16.*

Multiple Level Shared Workspace Subroutines

Since the BL instruction always reloads the workspace register 11, special steps must be taken to insure that the information in register 11 is not overstored. In the case of multiple levels of BL called subroutines, routines which call other routines before returning to the main program, a pushdown stack should be established to save intermediate return linkage. To create a return linkage stack for multiple levels of subroutines which share a workspace, the following procedure is employed:

1) Allocate one workspace register to the stack pointer function.
2) For each subroutine, "push" the contents of workspace register 11 before the next call, and "pop" the stack to restore the register 11 contents after each call is complete.

An example of a stack manipulation code following this procedure to push and pop return linkage is as follows, with register 5 acting as a stack pointer:

```
        ⋮    Subroutine code before next level call

        DECT   5          Decrement Stack Pointer          }
▶5      MOV    11, *5     Load return PC onto Stack        }  Push Operation
        BL     @SUBNXT    Call next level of subroutine
        MOV    *5+, 11    After return, restore current    }
        ⋮                 return address and restore       }
                          stack pointer                    }  Pop Operation
```

This code allows the current subroutine to call subroutine SUBNXT without destroying the current subroutine's return linkage. The main program employs a standard BL call, and the lowest level routine would not use the stack, since its register 11 would not be replaced with a subsequent BL call. An example of this stack operation procedure with 3 nested subroutines is illustrated in *Figure 5-19.*

Shared Workspace Mapping

Software systems for small computers must efficiently utilize available memory. This section presents an organized technique for sharing workspaces between subroutines to reduce system memory requirements.

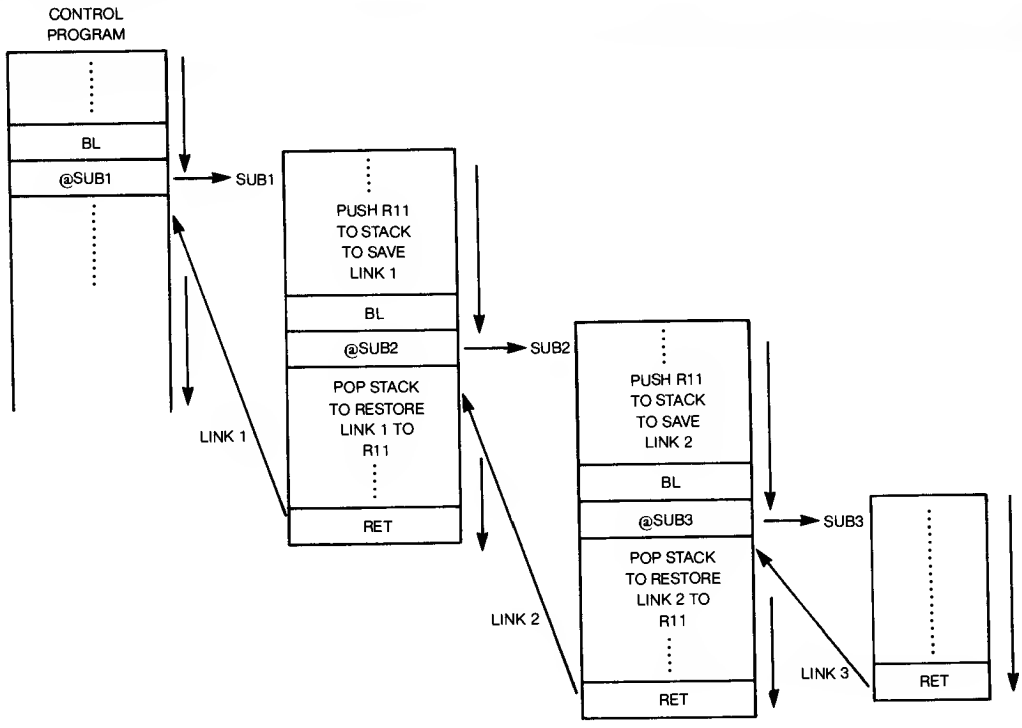


Figure 5-19. Stack Operations in Nested Subroutines.

The first step in system development is to write a main program and its associated subroutines with totally independent workspaces. Avoidance of shared workspaces at the start can prevent the undesirable aspect of destruction of critical data, including return linkage.

After independent software is written, the programmer begins the process of identifying potential shared workspaces. First, the relationship between called and calling procedures is summarized graphically as shown in *Figure 5-20.* This graph represents the fact that procedure A can call either procedure B or C. Procedure B may call D or E, while E can call D or G, and so on throughout the graph. Having identified routine relationships, *Figure 5-20* can be changed to a form that reflects subroutine levels. All procedures at the same level are called from a higher level and may call routines at a lower level.

Thus, *Figure 5-20* would be changed to the form illustrated in *Figure 5-21.* This information is equivalent to the information contained in *Figure 5-20*, but it clarifies the relationship between procedures. The routines on a particular level can never call another routine at the same or at a higher level. Thus, all routines at the same level can share a common workspace since return linkage will not be overstored by a subsequent call. Therefore, for the example described in *Figures 5-20 and 5-21,* five independent workspaces will suffice for a software system of eight procedures, saving 3 workspaces or 48 words of memory. By employing this simple technique, the software designer can write efficient code with an assurance of the integrety of return linkage.

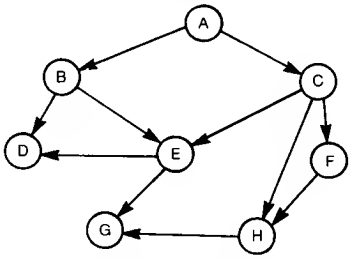


Figure 5-20. Graphical Representation of Interrelation of Calls

▶5

Re-entrant Programming

Re-entrant programming is a technique that allows one set of program code to be executed on multiple data sets concurrently. To be re-entrant, program code must have the following characteristics:

1) All data contained in a re-entrant routine must be common to all procedures which call it, and must be read-only to all using procedures.
2) All data unique to calling routines must be stored and used in a workspace unique to the calling procedure.
3) Re-entrant code must not alter data or instructions within its code during execution.

Re-entrant coding is a general programming technique that has many applications. Device service routines which control the operations of several similar units should be re-entrant. By passing a CRU base address with other unique data to a re-entrant service subprogram, any one of a group of calling procedures can access such a multi-purpose I/O routine, thereby saving system memory requirements. This is a case in which one routine is used for several applications at random time intervals. A re-entrant subroutine is so loosely coupled to its calling procedure that a re-entrant routine can be interrupted during execution, used on different data, and return to complete the original process without losing data integrity. Since re-entrant code is immune to problems with data resulting from interrupts, it finds application in interrupt service routines, commonly called procedures, in a multiprogramming environment such as assemblers or in real-time control applications.

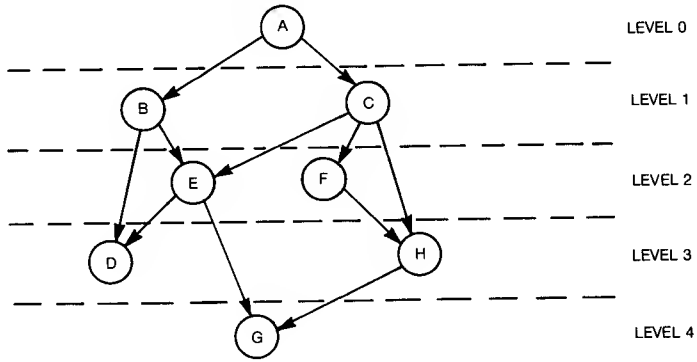


Figure 5-21. Levels of Subroutine Calls

Figure 5-22 illustrates a program flow in which subroutine A must be re-entrant. The alternative to writing subroutine A in re-entrant code is to make two copies of A, one for each time A can be executed concurrently. The re-entrant approach is more efficient in memory usage than is the multiple copy approach. In this program flow, the main program calls its first level subroutine which in turn calls subroutine A as a second level subroutine. During execution of routine A, an interrupt occurs, which in this example the interrupt handler program sequence calls several routines, including routine A. If A employs re-entrant programming, the same words of code can implement routine A for both parts of the program flow.

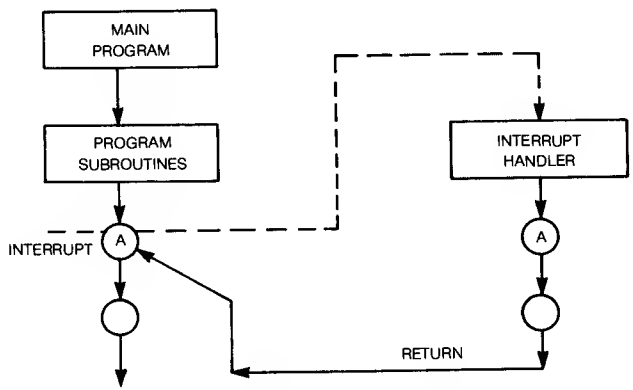


Figure 5-22. Interrupt Requiring Re-entrant Programming

As an example of re-entrant coding, consider the problem of forming a starting and an ending address for a block of data to be operated on by a subroutine. Register 1 is to hold the starting address, register 2 is to hold the ending address, and register 3 is to hold the current data address within the block.

Usually the first part of any program is the initialization of the system. The LWPI instruction is used to initialize the workspace pointer (WP register) to define the location of the 16 workspace registers. If the workspace of a program sequence is to be located at a starting address of 400_{16}, the following instruction will initialize the workspace pointer to that value:

```
LWPI   >0400
```

Under the interrupts discussion the use of the LIMI (load interrupt mask immediate) instruction to establish which interrupts would be responded to was covered. For example, if the programmer wants to disable all interrupts above level 7 for a program segment, the following instruction must be used at the first of the segment:

```
LIMI   7
```

Similarly, the load immediate (LI) instruction is used to initialize values in workspace registers. The LI can be followed by MOV instructions to further initialize other memory locations. As an example, to initialize register 3 and memory location TEST to the value $00FF_{16}$, the following instructions can be used:

```
LI     3, >00FF
MOV    3, @TEST
```

▶5 In some initialization sequences several registers have to be initialized to the same value, such as zero. For example, if 10 consecutive memory words starting at location 1000_{16} are to be cleared (zeroed) then a program loop is suggested. One possible implementation of this initialization task would be:

```
        LI     2,>1000     Set R2 to the starting address 1000₁₆
        LI     3,>100A     Set R3 to the address past the last data location to
                           be cleared
LOOP    CLR    *2+         Clear data, increment the address by two
        C      2,3         Is address past the 10th data location

        JNE    LOOP        If not jump to LOOP to continue clears
                           else go the next sequence of instructions
          .
          .
          .
          .
```

In this data initialization program segment, like most program segments, registers must be initialized to establish program limits, addresses, and other conditions. In this sequence register 2 was initialized to the starting data address and register 3 was initialized to indicate the first word address after the 10th data word to be cleared. Had this loop been implemented with a loop counter, the register acting as a loop counter would have been initialized to 10.

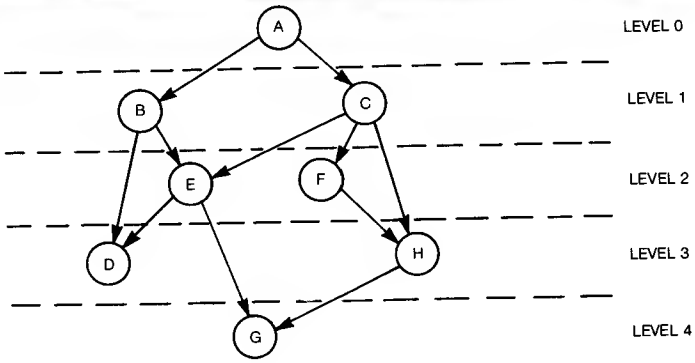


Figure 5-21. Levels of Subroutine Calls

Figure 5-22 illustrates a program flow in which subroutine A must be re-entrant. The alternative to writing subroutine A in re-entrant code is to make two copies of A, one for each time A can be executed concurrently. The re-entrant approach is more efficient in memory usage than is the multiple copy approach. In this program flow, the main program calls its first level subroutine which in turn calls subroutine A as a second level subroutine. During execution of routine A, an interrupt occurs, which in this example the interrupt handler program sequence calls several routines, including routine A. If A employs re-entrant programming, the same words of code can implement routine A for both parts of the program flow.

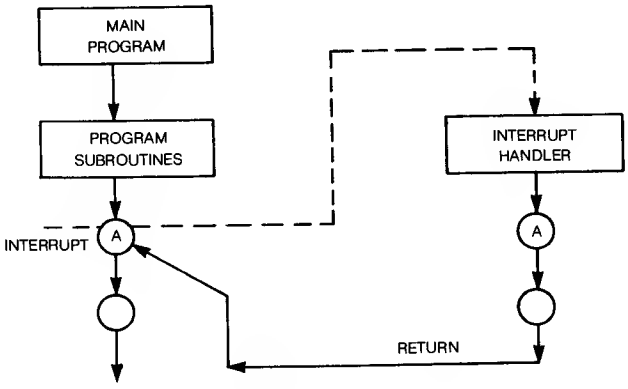


Figure 5-22. Interrupt Requiring Re-entrant Programming

As an example of re-entrant coding, consider the problem of forming a starting and an ending address for a block of data to be operated on by a subroutine. Register 1 is to hold the starting address, register 2 is to hold the ending address, and register 3 is to hold the current data address within the block.

This structure is in a subroutine that must be re-entrant, i.e., one that can be called concurrently by two different program segments as illustrated in *Figure 5-22.* Different program segments that use this subroutine will probably be dealing with blocks at different starting addresses and of different sizes. For example, the main program stream may be operating on a block of 10 words starting at address 1000_{16} while the interrupt handler may have to operate on a 32 word block starting at address 2000_{16}. Forming the starting and ending addresses as follows:

```
BLKLN  EQU  32
       LI   1,>1000      Load R1 with starting address
       LI   2,2*BLKLN    Load R2 with 2x (words in block)
       A    1,2          Form end address for block (1014₁₆)
```

would not result in re-entrant code. This sequence would be correct for the main program stream but is not correct for the interrupt handler stream. The code can be made to work for both streams by not placing the load immediates in the subroutine itself but by placing them in the program stream that calls the subroutine. Then, when either the main program or the interrupt handler gets ready to call the subroutine, the starting address and ending address can be established within the workspace for that environment. The subroutine can then concentrate on performing its manipulations, without being concerned with the address initialization process. Suppose that this addressing scheme is used in a subroutine that clears a block of memory words. *Figure 5-23* shows the re-entrant and non-re-entrant forms of this subroutine. The re-entrant form can be used in the situation of *Figure 5-22* since execution depends on the workspace being used. The subroutine can be executing with the registers 1 through 3 of the main program when an interrupt occurs. The interrupt handler uses a different workspace so when it calls the CLEAR subroutine, new starting and ending addresses are used, without affecting where the subroutine was in the main program execution. Then, when the subroutine and interrupt handler have been executed and the context is switched back to the main program, the subroutine will continue executing with the values in registers 2 and 3 of the main program environment.

This is not true of the non-re-entrant code. By moving the contents of the interrupt register 1 to the FIN location, the number of blocks to be cleared by the main program execution CLEAR subroutine could be changed, if the interrupt occurs after the MOV 1, @FIN instruction. Because the value FIN is unique for each calling program or computed in the subroutine, the code may not properly be re-entered. That is, it should not be used when an interrupting procedure may execute the same code. The re-entrant version could be used by any number of interrupting procedures without affecting execution results in either the main program or the interrupting program environments. Entrance to the routine would be performed by executing a BL @ CLRLUP.

Re-Entrant Coding

```
        MOV   1,3      Set R3 at first address of block

        A     1,2      Compute end address and place in R2

COM     C     3,2      Address past end?

        JH    PRET     If so, return
        CLR   *3+      Else, clear word and go to next address
        JMP   COM      Jump to continue clearing
PRET    B     *11      Return
```

Non-Re-entrant Coding

```
CLRLUP  MOV   1,3      Set R3 at first address of block

        MOV   1,@FIN   Set up first address in FIN

        A     2,@FIN   Compute final address and store at FIN

COM     C     3,@FIN   Address past end?

        JH    PRET     If so, return
        CLR   *3+      Else, clear word and go to next address
        JMP   COM      Jump to continue clearing
PRET    B     *11      Return
```

5

Figure 5-23. Subroutine Example of Re-entrant Coding.

PROGRAMMING TASKS

The programming techniques of workspace and subroutine usage, program loops, macros, and data representation must be applied to the development of programs and subprograms to perform the basic system functions of state initialization, pattern recognition, arithmetic, and input/output. Each of these system functions represents a programming task that involves programming structures peculiar to each function. This section discusses the basic requirements of the software for each function and presents some of the programming approaches that are used to meet those requirements.

INITIALIZATION

When the system is first turned on, the first few instructions encountered must initialize the state of the system to a desired predetermined starting state. The system initialization procedure is usually started by the RESET or LOAD functions. Similarly, as the system enters a subprogram or a new program sequence, the state of certain memory locations must be initialized to a desired starting state. Further, in developing the software, the transfer vectors and other program constants must be initialized by the assembly language software. The assembly language directives available for this purpose include the equate (EQU) and the data (DATA) directives. The application of these directives to the problem of initializing the reserved memory locations and program constants are covered in detail under the assembler directive discussion in Chapter 7.

Usually the first part of any program is the initialization of the system. The LWPI instruction is used to initialize the workspace pointer (WP register) to define the location of the 16 workspace registers. If the workspace of a program sequence is to be located at a starting address of 400_{16}, the following instruction will initialize the workspace pointer to that value:

```
LWPI   >0400
```

Under the interrupts discussion the use of the LIMI (load interrupt mask immediate) instruction to establish which interrupts would be responded to was covered. For example, if the programmer wants to disable all interrupts above level 7 for a program segment, the following instruction must be used at the first of the segment:

```
LIMI   7
```

Similarly, the load immediate (LI) instruction is used to initialize values in workspace registers. The LI can be followed by MOV instructions to further initialize other memory locations. As an example, to initialize register 3 and memory location TEST to the value $00FF_{16}$, the following instructions can be used:

```
LI     3, >00FF
MOV    3, @TEST
```

▶5 In some initialization sequences several registers have to be initialized to the same value, such as zero. For example, if 10 consecutive memory words starting at location 1000_{16} are to be cleared (zeroed) then a program loop is suggested. One possible implementation of this initialization task would be:

```
        LI     2,>1000     Set R2 to the starting address 1000₁₆
        LI     3,>100A     Set R3 to the address past the last data location to
                           be cleared
LOOP    CLR    *2+         Clear data, increment the address by two
        C      2,3         Is address past the 10th data location

        JNE    LOOP        If not jump to LOOP to continue clears
                           else go the next sequence of instructions
          .
          .
          .
          .
```

In this data initialization program segment, like most program segments, registers must be initialized to establish program limits, addresses, and other conditions. In this sequence register 2 was initialized to the starting data address and register 3 was initialized to indicate the first word address after the 10th data word to be cleared. Had this loop been implemented with a loop counter, the register acting as a loop counter would have been initialized to 10.

Generally, most program initialization tasks can be handled by using a combination of the techniques presented in this section. The immediate load instructions are the most commonly used operations in performing the initialization operation, followed by the use of assembler initialization directives to establish vectors and other data constant initialization.

Masking and Testing

In many cases only certain portions of a word are of interest. The program segment may be examining or modifying a single bit or a group of bits. The bits that are not involved in the operation must be masked off so that they will not affect status bits and thus affect program decisions. There are several ways of approaching this masking and single bit testing problem.

If a single I/O bit is to be examined or modified, the simplest approach is to use the CRU single bit instruction SBO, SBZ, or TB to perform the desired operation. This is possible only if the hardware has been set up to address the desired bit as a CRU bit. If the bit is not accessible through CRU addressing, one of the selective masking instructions must be used. The set to ones or zeroes instructions (SOC, SOCB, SZC, and SZCB) can be used to selectively set or clear bits. The compare ones or zeroes corresponding instructions (COC and CZC) can be used to test selected bits. Of course these instructions can be used to test or change single or multiple bits. An alternative single bit approach is to use the circulate instruction (SRC) to get the desired bit into the carry status bit for examination or changing.

To see how these non-CRU masking instructions are used, consider the task of examining the value of bit 12 of the data of workspace register 1. The mask is contained in location MASK which will contain all zeroes in all bits except for bit 12 which will contain a one. Thus, location MASK will contain 0008_{16}. Then, the instruction:

```
CZC     @MASK, 1
```

will set the equal status bit if bit 12 of R1 contains a zero. The instruction:

```
COC     @MASK, 1
```

will set the equal status bit if bit 12 of R1 contains a one. In these cases, the JEQ or JNE instructions can be used to test the equal status bit after the comparison.

Alternatively, the instruction:

```
SRC     1,4
```

will cause bit 12 to be in the carry flip flop. However, this instruction will change the contents of R1 by moving all bits to the right 4 positions. The JC or JNC instructions are used to test the bit value in the carry status bit.

To selectively set bit 12 of R1, any of the following instructions could be used:

```
ORI     1,>0008
SOC     @MASK,1
```

To selectively clear bit 12 of R1, either or the following instructions could be used:

```
ANDI    1,>FFF7
```

or:

```
SZC     @MASK,1
```

If groups of bits are to be changed or examined, the above techniques can be used if all bits are to be ones or zeroes. For example, if bit 13 of register 2 is to be tested, the following instructions would jump to point P1 in the program if bit 13 of R2 is one:

```
ANDI  2,>0004      Zero all bits but bit 13 of R2; compare to 0

JNE   P1           If EQ=0, bit 13 was one and jump to point P1
```

A more complicated test would be to check bits 13 and 15 of R3. A jump to P2 is to be made if both of these bits are one. The following instructions would accomplish this test and program decision:

```
▶5
H5      DATA    5
        .
        .
        .
        COC     @H5,3     Compare to 5 to see if both bits are one.
        JEQ     P2        If they are, jump to point P2.
```

Thus, a combination of masks (ANDI) compares, and conditional jumps can be used to examine all features of system words and react appropriately.

If a group of bits is to be examined or modified arithmetically, a slightly different approach may be used. If for example the least four bits of R1 are to be compared to 8, one approach would be to provide a copy of the R1 contents in R2. Then the first 12 bits of R2 are zeroed with:

```
ANDI  2,>000F
```

or with: SZC @MASK1,2 where the contents of location MASK1 are $FFF0_{16}$. Then, the contents of R2 can be compared to 8. The entire sequence would then be:

```
        MOV  1,2
        ANDI 2,>F
        CI   2,8
        JLT  P1
          •
          •     Sequence of instructions
          •        to handle case where least
          •        four bits of R1 (and R2) are
          •        greater than or equal to 8
          •
P1        •     Sequence of instructions
          •        to handle case where least
          •        four bits of R1 (and R2) are
                   less than 8
```

This technique is useful in Decimal to Binary or Binary to Decimal number conversion and in implementing BCD (Binary Coded Decimal) arithmetic.

5

Of the techniques that can be used in masking and testing, the ANDI, ORI, SOC, and SZC instructions change the word they operate on. The Compare techniques, CZC and COC, do not affect the words being operated on but they do affect the status bits. Often, when part of a word is modified (such as portion of the word is zeroed by an ANDI instruction) the word must later be reassembled after all bit group operations have been completed. The programmer should see that such operations are performed on copies of the word so that further masking operations use the original word. As an example, if a word is to be broken down into four bit groups (to implement BCD arithmetic), at least four copies of the word are required or four accumulators must be used. If register 1 contains the master copy, and registers 2 through 5 contain the four bit groups, the following sequence of instructions would generate the desired four bit groupings in the four accumulators from the master copy in register 1:

```
MOV  @MASTER,1   Move word to be separated into R1
MOV  1,2         Move a copy of the word into
MOV  1,3         accumulators R2 through R5
MOV  1,4
MOV  1,5
ANDI 2,>000F     Mask all but least four bits in R2
ANDI 3,>00F0     Mask all but next four bits in R3
ANDI 4,>0F00     Mask all but next four bits in R4
ANDI 5,> F000    Mask all but most significant four bits in R5
```

With this program sequence, the original word can be broken into bit groups for further testing and modification. R1 still contains the original word for reference and further manipulation. However, by using ANDI mask instructions, several memory words are required to hold intermediate results. This would not have been necessary if compare (selective bit) instructions had been used. The specific application usually dictates which approach is to be used.

Arithmetic Operations

Basic arithmetic can be performed with addition and subtraction, though certain operations such as multi-word arithmetic require the use of shift instructions and conditional branch instructions such as the jump on carry or jump on greater than.

Multi-Precision Arithmetic

The 9900 arithmetic instructions perform mathematical functions on 16 bit words. For applications that require a greater numerical accuracy or a larger number (the 16 bit word can hold a magnitude number from 0 to 65,535), multiple word numbers must be used. The basic arithmetic instructions must then be used in such a way as to implement the desired mathematical functions on these multiple word numbers. This section deals with techniques for treating several words as a single binary value, that is, extended precision arithmetic.

A 16 bit two's complement word can represent a signed value in the range −32,768 to +32,767. The negate (NEG) and absolute value (ABS) instructions provide fast conversion between positive and negative 16 bit words. For sign conversion on binary values represented by multiple words, special conversion techniques are required. The process for converting a three word positive value to its negative or two's complement value is shown in *Figure 5-24.* The three word number is stored in registers 0, 1, and 2 of the workspace. The complementing procedure is to form the one's complement of the three word number using the invert (INV) instruction and then to add 1 to the result. The JNC instructions in the program check to see if a carry is to be propagated from a less significant word to a more significant word in the process of adding one to the three word number. If carries occur, the addition is handled by the increment (INC) instruction. Conversion of a number to its absolute value is accomplished by checking the sign bit (most significant bit) and executing the negate routine (COMP) on negative values.

Memory Structure

Register 0	Register 1	Register 2	
A_0	A_1	A_2	= A (48 bit number)

Procedure:

1) Form 1's complement of A, using the invert (INV) instruction.
2) Convert the 1's complement of A to the two's complement of A by adding 1 to the 1's complement of A.

2's complement of A = 1's complement of A + 1

Program:

```
COMP  INV  0     Complement contents of R0
      INV  1     Complement contents of R1
      NEG  2     Negate contents of R2
      JNC  EXIT  If no carry operation is complete, return
      INC  1     If carry, add one to contents of R1
      JNC  EXIT  If no carry operation is complete, return
      INC  0     If carry, add one to contents of R0
EXIT  RT         Return
```

Figure 5-24. *Process to form the Negative of A(−A).*

The process of adding or subtracting two multi-word numbers is to perform the operation on the least significant words, then on the next most significant words with the previous carry or borrow, and so on until the complete result is formed. Subtraction could be performed by first using the negate procedure of *Figure 5-24* on the value to be subtracted and then adding this two's complement result to the other number. 5◀

Multiple word multiplication can be handled by using the 9900 multiply (MPY) instruction to provide 32 bit partial products and then adding all partial products to achieve the final desired product. The procedure is illustrated in *Figure 5-25* for multiplication of one 32 bit number by a second 32 bit number. The multiplication of a 16 bit number by a second 16 bit number is performed by the MPY instruction. Thus, four applications of the MPY would form the required four partial products. Then, by adding these products in the correct positions, the 64 bit product is formed. The basic memory structure used by the example in *Figure 5-25* can be understood by looking at the operation of the MPY instruction. The accumulator or destination operand must be a workspace register. Then, the product is stored in two successive workspace registers, the most significant 16 bits in the destination workspace register and the least significant 16 bits in the next workspace register. The source operand which specifies the multiplier may be specified with any addressing mode, though the example of *Figure 5-25* uses register addressing for this operand. Thus, the instruction:

```
MPY    1,8
```

multiples the contents of register 1 by the contents of register 8 and places the 32 bit product in registers 8 and 9. While the multiplier register 1 contents are unchanged, the multiplicand register 8 contents are changed to the most significant part of the product.

Thus, there must be several copies of each multiplicand to be able to form several partial products. In 32 bit by 32 bit multiplication, there are two multipliers (B_0 in register 0 and B_1 in register 1) and two multiplicands. Since each multiplicand is involved in two partial products, there must be two copies of each multiplicand. In *Figure 5-25* the copies of the A_0 multiplicand are saved in registers 4 and 6 and the copies of the A_1 multiplicand are saved in registers 2 and 8. Then, the following four MPY instructions form the four required partial products:

```
MULT MPY    1,8    Form the A1xB1 product in R8 and R9
     MPY    1,4    Form the B1xA0 product in R4 and R5
     MPY    0,2    Form the B0xA1 product in R2 and R3
     MPY    0,6    Form the B0xA0 product in R6 and R7
```

Which can be followed by the additions to form the complete 64 bit product in registers 6 through 9:

```
     A      3,5    Add two of three 16 bit groups in positions 2^16 to 2^31
     JNC    P0     If no carry, add in R8 contents
     INC    7      If carry, add one to R7 accumulator
P0   A      5,8    Finish adding 2^16 to 2^31 bits
     JNC    P1     If no carry, procede to next position adds
     INC    7      If carry, add one to R7 accumulator
P1   A      2,4    Add part of 2^32 to 2^47 bits in R2 and R4
     JNC    P2     If no carry, procede to rest of addition
     INC    6      If carry, add 1 to R6 accumulator
P2   A      4,7    Finish adding 2^32 to 2^47 bits
     JNC    FIN    If no carry, operation is complete
     INC    6      If carry, add one to R6 accumulator
FIN  RT     return
```

The process illustrated by *Figure 5-25* is for multiplication of two 32 bit magnitude numbers. Multiplication of negative numbers can be handled with the same program by converting all numbers to their absolute value, saving the sign. Then, after the magnitude multiplication is complete, the sign of the product is the exclusive OR of the multiplier and multiplicand sign bits. If desired, the product can be complemented or negated and stored in two's complement form.

Basis of Procedure:

$A \times B = (A_0 x 2^{16} + A_1) \times (B_0 x 2^{16} + B_1) =$

$A \times B = A_0 x B_0 x 2^{32} + (A_1 x B_0 + A_0 x B_1) \times 2^{16} + A_1 x B_1$

Where multiplying by 2^n implies shifting the number n positions to the left with respect to a number multiplied by 2^0 or 1.

Memory Structure:

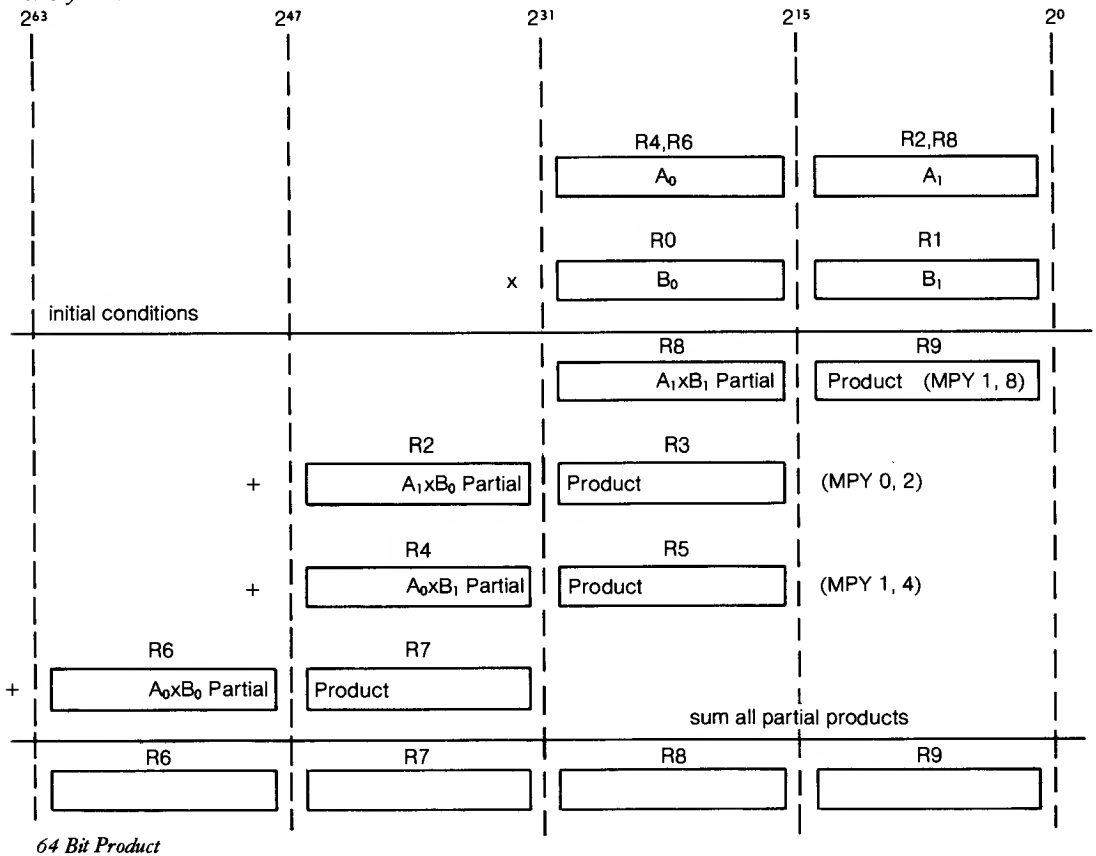


Figure 5-25. *Multiple Precision Multiplication*

Floating Point Arithmetic

If the system requires the ability to represent fractional numerical quantities instead of integer numbers, a method must be defined that will provide for the location of the radix point of such numbers. Just as the decimal point of 75.39 defines a quantity:

$$7x10^1 + 5x10^0 + 3x10^{-1} + 9x10^{-2}$$

the binary point in 101.01 defines a quantity:

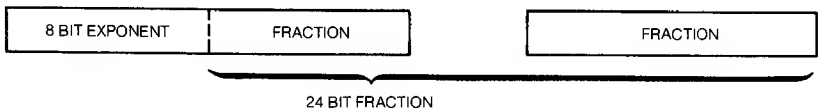
$$1x2^2 + 0x2^1 + 1x2^0 + 0x2^{-1} + 1x2^{-2}$$

Although a group of bits can be configured in many ways to define a floating point number, most floating point representations share the following characteristics:

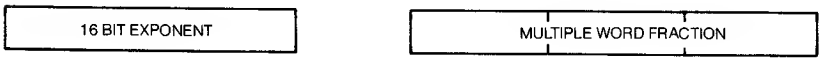
1) Floating point numbers are represented as a fraction and an exponent (mantissa and characteristic).
2) The fraction is by convention normalized to lie in the range $\frac{1}{2} \leq F < 1$, e.g., the binary point lies to the left of the first one bit.
3) The exponent defines the power of 2 by which the fraction is multiplied to evaluate the floating point number.

Possible floating point representatives include:

▶5



and:



When addition or subtraction of two floating point numbers is performed, the following operations must be performed:

1) Equalize exponents; increment the exponent of the smaller quantity until it is the same as the larger exponent. With each exponent increment, shift the corresponding fraction to the right with zero fill from the left.

2) Add or subtract the fractions as required. If a carry results from an addition, the sum fraction is shifted right, shifting the carry into the fraction, and the exponent is incremented. If the difference resulting from the subtraction is not normalized (zero in first bit position) the fraction must be shifted left until a one is in the leftmost position. With each left shift, the resultant exponent is decremented.

Floating point multiplication can be performed by multiplying the fractions and adding the exponents. Similarly, floating point division can be performed by dividing the fractions and subtracting the exponents. After such an operation the fraction must be normalized and the exponents must be checked for overflow or underflow. Signed numbers can be handled in the same way that the multiplication of signed integers is handled.

INPUT/OUTPUT

The most fundamental and necessary instructions of a processor are its input and output instructions and techniques. Without input and output, the system would not be able to control or communicate with the external world, and as a result would be of no use. There are two general ways of implementing input and output operations. One obvious approach is to use special input and output instructions that are interpreted by the hardware to apply to the input and output devices. The 9900 instructions that provide this capability are the CRU or communications register unit instructions (SBO, SBZ, TB, LDCR, and STCR) and the input and output hardware that respond to these instructions make up the communications register unit. Another approach to inputting and outputting information is to simply treat the input and output devices as one of the system memory locations, in which case any of the 9900 instructions can be used in accessing these locations. This approach is called memory mapped input/output, since the devices are assigned a portion of the available memory addresses, and the hardware must decode the appropriate address to activate a given device. Each of these approaches has its advantages and disadvantages, and the programmer must be aware of these trade-offs in order to provide an optimum approach to system input/output.

Memory Mapped Input/Output

The principle advantages of using memory mapped input/output are:

1) The full instruction set is available for manipulating the data in the input/output device.
2) The hardware is straightforward, since address decoding and device timing signals are required for RAM and ROM memory anyway and these can be simply extended to handle the I/O subsystem as well.
3) Transfers of information are made 8 or 16 bits at a time, offering a high bit rate transfer.

The disadvantages of memory mapped I/O are:

1) Since some of the 'memory' locations are being used by input or output devices, less memory is available for instructions and general data storage.
2) Bit transfers must be made 8 or 16 bits at a time. This is wasteful if a given device can handle a single or a few bits at a time.
3) It is a more expensive technique in terms of pinouts, board space, and layout time.
4) The hardware interface must accomodate the full width memory bus.

The most commonly used instruction in memory mapped I/O operation is the MOV instruction to effect data transfers. However, it is quite possible to set up an input output subsystem as general purpose storage or as a workspace and perform shifts, additions, multiplications, logical operations, and so on, on the data contained in the I/O subsystem.

Generally, if I/O transfers are to be made 8 or 16 bits at a time and if the system is not memory bound (memory is needed for program and system data), memory mapped I/O is often used. Certainly, if performing arithmetic, logic, or other instructions directly on input/output data is required or advantageous, memory mapped I/O must be used. If single or multiple bit transfers are all that is required, and transfer rate is not critical, then memory mapped I/O has no advantage over CRU I/O. CRU I/O hardware is normally simpler and less expensive.

CRU Input/Output

The CRU instructions provide for single bit transfers with the SBO (set bit to one), SBZ (set bit to zero), and TB (test bit) instructions. Multiple bit transfers with the bits transferred one at a time are possible using the LDCR (load communications register) and STCR (store communications register) instructions. The advantages of the CRU instruction approach to I/O are:

1) Any number of bits (up to 16) can be transferred with the appropriate CRU instruction. Thus, the designer can set up the data transfer to exactly meet the requirements of the subsystem being serviced. This is especially useful in control situations where single sense bits are to be examined and single on-off output control signals are needed.
2) No memory locations are used by the subsystem. The CRU instructions can access 4096 input and 4096 output bits (which is equivalent to 256 data words) in addition to the 65,536 memory bytes.

The disadvantages of the CRU I/O are:

1) Only data transfers are provided. Arithmetic, logical or other operations must be performed on the data after it has been moved to one of the general data storage locations in RAM.
2) The hardware must include the capability to decode and implement the CRU transfers; however, the added IC complexity is more than offset by reduced package size.
3) Single bit transfer speed may be too slow for some applications.

Input/Output Methods

There are three ways that an input/output transfer can be handled or initiated. The processor can be interrupted, causing the program to jump to a subroutine that handles the input/output task. The program can encounter an instruction to transfer data from an input location or to an output location, for the purpose of displaying results, actuating control elements, or inputting system status. The processor can be bypassed entirely and the data transferred directly to or from system memory, using direct memory access.

Interrupt Driver Input/Output

If the timing of input/output transfer is to be controlled by an external system, then the interrupt driven I/O method must be used. This approach is used in inputting data when the time of input is random. The external system inputs the data and signals the processor with an interrupt to indicate that data is in one of the 9900 system input registers. The 9900 responds by performing a context switch to a subprogram that will process the data in that register. The interrupt driven approach may also be used in outputting data when the processor needs to know when the data has been taken by an external system. Once the external system takes the data, it can signal the processor with an interrupt signal. The processor responds by performing a context switch to a subprogram that will then send more data to that output location.

The interrupt driven I/O procedure provides a mechanism of implementing a communications sequence known as handshaking. This communications protocol is illustrated in *Figure 5-26.* In the input mode, the data-present signal latches the 9900 system input register and serves as the interrupt signal. If desired the processor's reading of the contents of that register can be used to generate the data-taken signal. Upon receiving the data taken signal, the external system can then send more data to the 9900 system. In the output mode, the register write operation that sends data to the output register in the 9900 system can be used as a data-present signal to the external system. When the external system takes this data, it can use an interrupt signal to notify the 9900 that the register is ready to receive more data.

The interrupt driven approach has the main advantage of providing a means of setting up a handshaking communications with another system. It can handle data communications that occur at random or unpredictable times. The main disadvantage of this type of I/O is that it does involve the processor, slowing down its work on main system programs and subprograms. Further, since a context switch is involved in responding to an interrupt, such an approach may require more memory than one of the other two approaches.

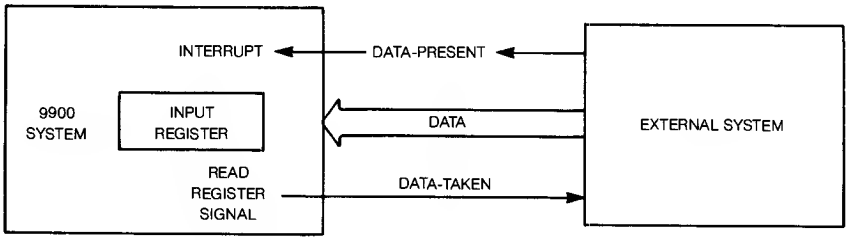
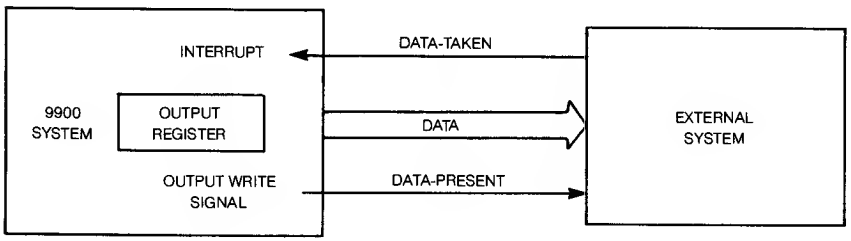


Figure 5-26a. Handshaking Input Transfer.



▶5 *Figure 5-26b. Handshaking Output Transfer.*

Programmed Input/Output

The simplest method of I/O is the programmed I/O. The times and conditions under which inputs and outputs are to occur are controlled by the system program. For example, the processor may update a display memory whenever the display is to be changed. The program determines the new information to be displayed and then outputs this information to the display storage locations. Similarly, the program may require information about the status of a subsystem in order to determine subsequent operations. Such status may be the condition of threshold detectors, the status of serial data transmission or reception, or some other system condition. When the program encounters a point at which it needs to check such status words, it simply inputs the desired word. Of course, as in all I/O methods, the input or output operations can be handled by either CRU instructions or through standard MOV or other instructions in memory mapped I/O devices.

The programmed input/output method is a high speed, low hardware cost approach to input/output, since no subroutine overhead is involved. However, it does not handle the random input situation very well, unless the program is devoted solely to waiting for the next signal input to occur. Even then, the program may not check for input occurrence and perform the desired input instruction before the external system sends new data, destroying the old data.

CHAPTER 6

Instruction Set

6

SOFTWARE FEATURES OF THE 9900

In order to understand the operation of the 9900 instructions, the basic software features of the 9900 must be understood. These features include the processor-memory interrelationships, the available addressing modes, the terminology and formats used in the 9900 assembly language, and the interrupt and subroutine procedures used by the 9900.

Processor Registers and System Memory

There are three registers in the 9900 that are of interest to the programmer; their functions are illustrated in *Figure 6-1:*

Program Counter—This register contains the address of the instruction to be executed by the 9900. This instruction address can point to or locate an instruction anywhere in system memory, though instructions normally are not placed in the first 64 words of memory. These locations are reserved for interrupt and extended operation transfer vectors.

Workspace Pointer—This register contains the address of the first word of a group of 16 consecutive words of memory called a workspace. The workspace can be located anywhere in memory that is not already dedicated to transfer vector or program storage. These 16 workspace words are called workspace registers 0 through 15, and are treated by the 9900 processor as data registers much as other processors treat on-chip data registers for high access storage requirements.

▶6 *Status Register*—The status register stores the summary of the results of processor operations, including such information as the arithmetic or logical relation of the result to some reference data, whether or not the result can be completely contained in a 16-bit data word, and the parity of the result. The last bits of the status register contain the system interrupt mask which determines which interrupts will be responded to.

These three 16-bit registers completely define the current state of the processor: what part of the overall program is being executed, where the general purpose workspace is located in memory, and what the current status of operations and the interrupt system is. This information completely defines the current program environment or context of the system. A change in the program counter contents and workspace register contents switches the program environment or context to a new part of program memory with a new workspace area. Performing such a context switch or change in program environment is a very efficient method of handling subroutine jumps to subprograms that require the use of a majority of the workspace registers.

Program Counter

Figure 6-1 illustrates the use of the three processor registers. The program counter is the pointer which locates the instruction to be executed. All instructions require one or more 16-bit words and are always located at *even* addresses. Multiple word instructions include one 16-bit operation word and one or two 16-bit operand addresses. Two of the processors in the 9900 family (TMS9900, SBP9900) employ a 16-bit data bus and receive the instructions 16 bits at a time. The other processors (TMS9980A/81, TMS9985, TMS9940) use an 8-bit data bus and require extra memory cycles to fetch instructions. In both cases the even and odd bytes are located at even and odd addresses respectively as illustrated in *Figure 6-2.* In addition, data may be stored as 16-bit words located at even addresses or as 8-bit bytes at either even or odd addresses.

Workspace

The workspace is a set of 16 contiguous words of memory, the first of which is located by the workspace pointer. The individual 16-bit words, called workspace registers, are located at even addresses (see *Figure 6-1*). All of the registers are available for use as general registers; however, some instructions make use of certain registers as illustrated in *Figure 6-3.* Care should be exercised when using these registers for data or addresses not related to their special functions.

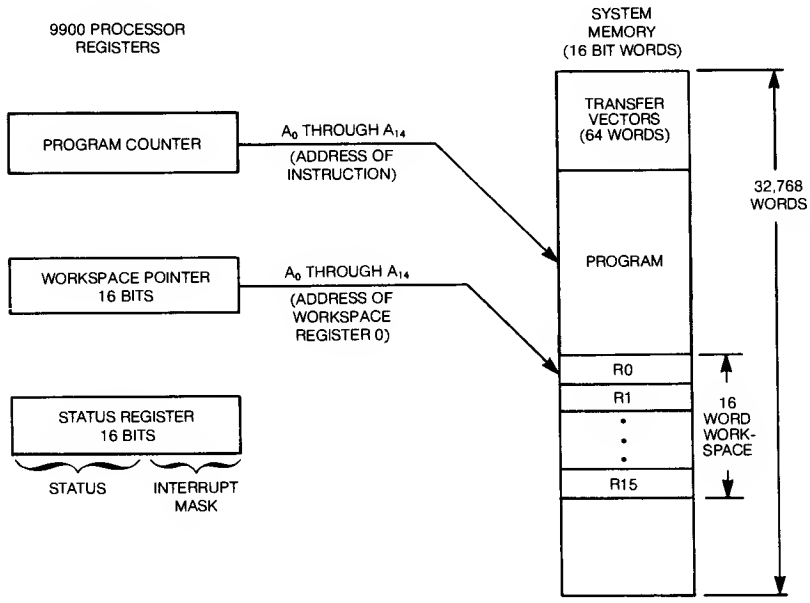


Figure 6-1. 9900 System Memory and Processor Registers.

Status Register

The status register contents for the 9900 are defined in *Figure 6-4.* The 9900 interrupt mask is a 4-bit code, allowing the specification of 16 levels of interrupt. Interrupt levels equal to or less than the mask value will be acknowledged and responded to by the 9900. The 9940 status register is similar, except the interrupt mask occupies bits 14 and 15 of the status register, providing for four interrupt levels in the 9940.

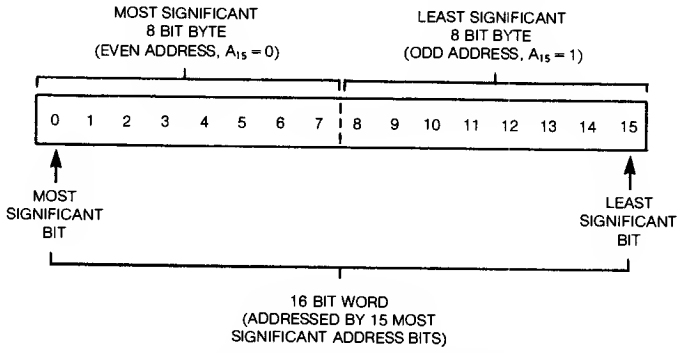


Figure 6-2. Word and Byte Definition.

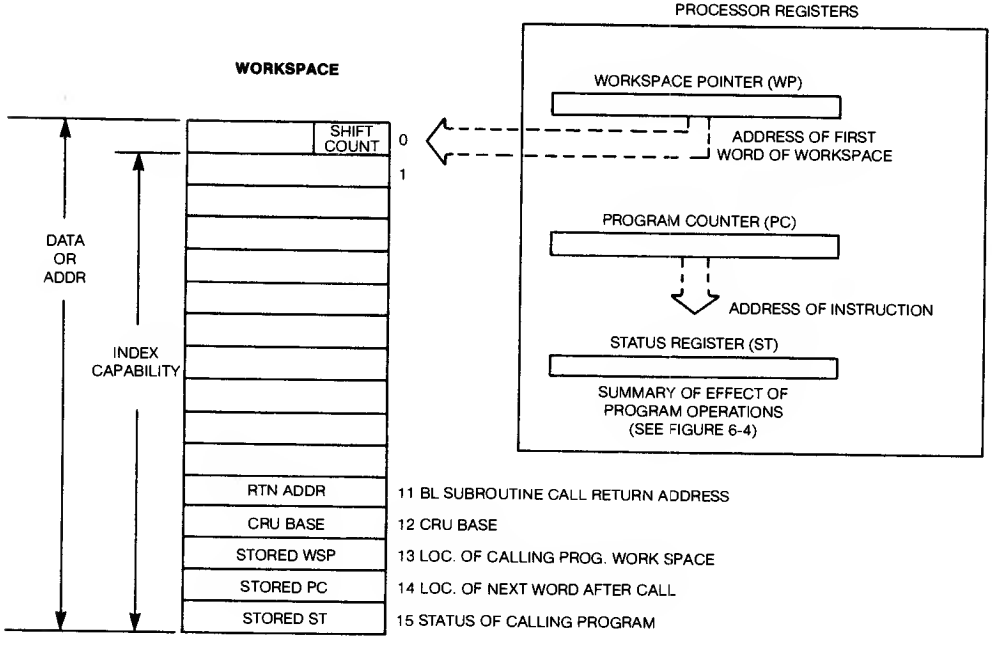
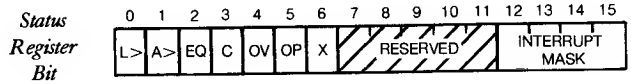


Figure 6-3. Workspace Register Utilization.



0 LGT — *Logical Greater Than*—set in a comparison of an unsigned number with a smaller unsigned number.

1 AGT — *Arithmetic Greater Than*—set when one signed number is compared with another that is less positive (nearer to −32,768).

2 EQ — *Equal*—set when the two words or two bytes being compared are equal.

3 C — *Carry*—set by carry out of most significant bit of a word or byte in a shift or arithmetic operation.

4 OV — *Overflow*—set when the result of an arithmetic; operation is too large or too small to be correctly represented in 2's complement form. OV is set in addition if the most significant bit of the two operands are equal and the most significant bit of the sum is different from the destination operand most significant bit. OV is set in subtraction if the most significant bits of the operands are not equal and the most significant bit of the result is different from the most significant bit of the destination operand. In single operand instructions affecting OV, the OV is set if the most significant bit of the operand is changed by the instruction.

5 OP — *Odd Parity*—set when there is an odd number of bits set to one in the result.

6 X — *Extended Operation*—set when the PC and WP registers have been to set to values of the transfer vector words during the execution of an extended operation.

7-11 Reserved for special Model 990/10 computer applications.

12-15 *Interrupt Mask*—All interrupts of level equal to or less than mask value are enabled.

Figure 6-4. *9900 Status Register Contents*

Addressing Modes

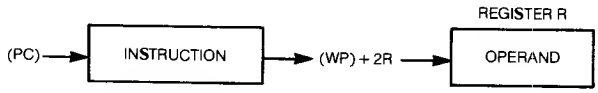
The 9900 supports five general purpose addressing modes or methods of specifying the location of a memory word:

Workspace Register Addressing

The data or address to be used by the instruction is contained in the workspace register number specified in the operand field of the instruction. For example, if the programmer wishes to decrement the contents of workspace register 2, the format of the decrement instruction would be:

```
DEC     2
```

The memory address of the word to be used by the instruction is computed as follows:



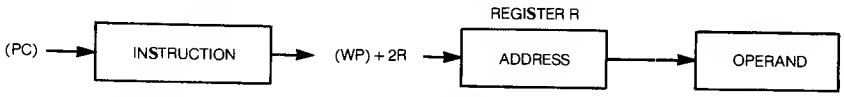
This type of addressing is used to access the often used data contained in the workspace.

Workspace Register Indirect Addressing

The address of the data to be used by the instruction is contained in the workspace register specified in the operand field (the workspace register number is preceded by an asterisk). This type of addressing is used to establish data counters so the programmer can sequence through data stored in successive locations in memory. If register 3 contains the address of the data word to be used, the following instruction would be used to clear (CLR) that data word:

```
CLR     *3
```

In this instruction the contents of register 3 would not be changed, but the data word addressed by the contents of register 3 would be cleared (set to all zeroes — 000_{16}). The word address is computed as follows for this type of addressing:

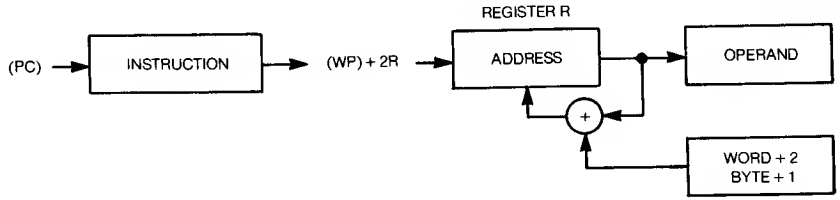


Workspace Register Indirect Addressing With Autoincrement—

This addressing mode locates the data word in the same way that workspace register indirect addressing does, with the added feature of incrementing the contents of the address register after the instruction has been completed. The address in the register is incremented by one if a byte operation is performed and by two if a word operation is performed. Thus, to set up a true data counter to clear a group of successive words in memory whose address will be contained in register 3, the following instruction would be used:

```
CLR     *3+
```

where the asterisk (*) indicates the workspace register indirect addressing feature and the plus (+) indicates the autoincrementing feature. With this type of addressing, the following computations occur:



Symbolic or Direct Addressing

The address of the memory word is contained in the operand field of the instruction and is contained in program memory (ROM) in the word immediately following the operation code word for the instruction. For example, to clear the memory word at address 1000_{16}, the following format would be used:

CLR @>1000

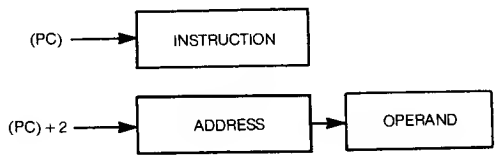
where the at sign (@) indicates direct addressing and the greater than (>) sign indicates a base 16 (hexadecimal) constant. Alternatively, the data word to be cleared could be named with a symbolic name such as COUNT and then the instruction would be:

CLR @COUNT

and if COUNT is later equated to 1000_{16}, this instruction would clear the data word at address 1000_{16}. The instruction would occupy two words of program memory:

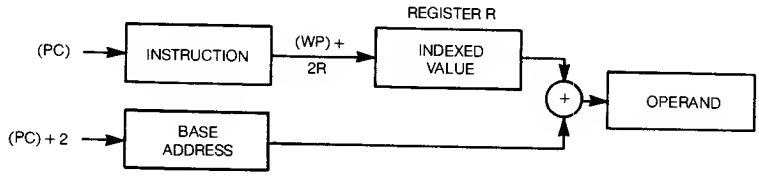
(PC)	$04C0_{16}$	Operation Code for Clear
(PC) + 2	1000_{16}	Address of Data

The address of the memory word is thus contained in the instruction itself and is located by the program counter. Since this address is part of the instruction, it cannot be modified by the program. As a result, this type of addressing is used for program variables that occupy a single memory word such as program counters, data masks, and so on. The address computations for direct addressing are as follows:



Indexed Addressing

Indexed addressing is a combination of symbolic and register indirect addressing. It provides for address modification since part of the address is contained in the workspace register used as an index register. Registers 1 through 15 can be used as index registers. The memory word address is obtained by adding the contents of the index register specified to the constant contained in the instruction:



Thus, to locate the data word whose address is two words down from the address contained in register 5, and to clear this memory word, the following instruction is used:

```
CLR     @4(5)
```

This instruction will cause the processor to add 4 to the contents of register 5 to generate the desired address. Alternatively, a symbolic name could be used for the instruction constant:

```
CLR     @DISP(5)
```

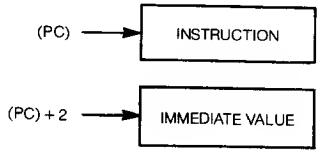
with the value for the symbol DISP defined elsewhere in the assembly language program.

Special Addressing Modes

Three additional types of special purpose addressing are used by the 9900.

Immediate Addressing

Immediate addressing instructions contain the data to be used as a part of the instruction. In these instructions the first word is the instruction operation code and the second word of the instruction is the data to be used:



Program Counter Relative Addressing

Conditional branch or jump instructions use a form of program counter relative addressing. In such instructions the address of the instruction to be branched to is relative to the location of the branch instruction. The instruction includes a signed displacement with a value between -128 and $+127$. The branch address is the value of the program counter plus two plus twice the displacement. For example, if LOOP is the label at location 10_{16} and the instruction:

```
JMP     LOOP
```

is at location 18_{16}, the displacement in the instruction machine code generated by the assembler will be -5 or FB_{16}. This value is obtained by adding two to the current program counter:

$$18_{16} + 2 = 1A_{16}$$

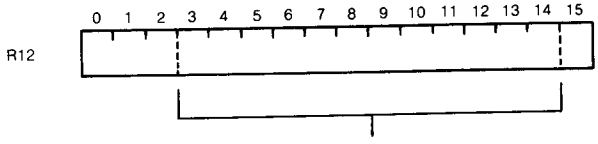
and subtracting from this result the location of LOOP:

$$1A_{16} - 10_{16} = A_{16} = 10 \text{ decimal.}$$

The displacement of 5 is one-half this value of 10 and it is negative since LOOP is 5 words prior to the $18_{16} + 2$ location.

CRU Addressing

CRU addressing uses the number contained in bits 3 through 14 of register 12 to form a hardware base address:



CRU Hardware Base address = Contents of R12 divided by 2

Thus if R12 contains 0400_{16} (the software base address), bits 3 through 14 will be 0200_{16}. This hardware base address is used to indicate the starting CRU bit address for multiple bit CRU transfer instructions (STCR and LDCR). It is added to the displacement contained in single bit CRU instructions (TB, SBO, SBZ) to form the CRU bit address for these instructions. For example, to set CRU bit 208 to a one, with register 12 containing 400_{16}, the following CRU instruction would be used:

```
SBO     8
```

so that the CRU bit address is $200_{16} + 8. = 208_{16}$.

ASSEMBLY LANGUAGE PROGRAMMING INFORMATION †

In order to understand the instruction descriptions and applications the assembly language nomenclature must be understood. Assembly language is a readily understood language in which the 9900 instructions can be written. The machine code that results from the assembly of programs written in this language is called object code. Such object code may be absolute or relocatable, depending on the assembly language coding. Relocatable code is that which can be loaded into any block of memory desired, without reassembling or without changing program operation. Such code has its address information relative to the first instruction of the assembly language program so that once a loader program specifies the location of this first instruction, the address of all instructions are adjusted to be consistent with this location. Absolute code contains absolute addresses which cannot be changed by the loader or any operation other than reassembling the program. Generally, relocatable code is preferable since it allows the program modules to be located anywhere in memory of the final system.

Assembly Language Formats

The general assembly language source statements consist of four fields as follows:

LABEL MNEMONIC OPERANDS COMMENT

The first three fields must occur within the first 60 character positions of the source record. At least one blank must be inserted between fields.

Label Field

6 The label consists of from one to six characters, beginning with an alphabetic character in character position one of the source record. The label field is terminated by at least one blank. When the assembler encounters a label in an instruction it assigns the current value of the location counter to the label symbol. This is the value associated with the label symbol and is the address of the instruction in memory. If a label is not used, character position 1 may be a blank, or an asterisk.

Mnemonic or Opcode Field

This field contains the mnemonic code of one of the instructions, one of the assembly language directives, or a symbol representing one of the program defined operations. This field begins after the last blank following the label field. Examples of instruction mnemonics include A for addition and MOV for data movement. The mnemonic field is required since it identifies which operation is to be performed.

Operands Field

The operands specify the memory locations of the data to be used by the instruction. This field begins following the last blank that follows the mnemonic field. The memory locations can be specified by using constants, symbols, or expressions, to describe one of several addressing modes available. These are summarized in *Figure 6-5.*

†Excerpts from Model 990 computer TMS 9900 Microprocessor Assembly Language Programmer's Guide.

Type of Addressing	*Operand Format*	*Memory Location Specified*	*MOV Instruction Example Coding*		*Result*	*T_d or T_s Field Code*
Workspace Register	n	Workspace Register n Rn	MOV	3,5	R3 ⟶ R5	00
Workspace Register Indirect	*n	Address given by the contents of workspace register n M(Rn)	MOV	*3,*5	M(R3) ⟶ M(R5)	01
Workspace Register Indirect, Autoincrement	*n +	As in register Indirect; address register Rn is incremented after the operation (by one for byte operations, by two for word operations)	MOV	*3+,*5+	M(R3) ⟶ M(R5) R3 + 2 ⟶ R3 R5 + 2 ⟶ R5	11
Symbolic Memory	@exp	Address is given by value of exp. M(exp)	MOV	@ONE, @10	M(ONE) ⟶ M(10)	10
Indexed Memory	@exp(n)	Address is the sum of the contents of Rn and the value of exp M(Rn + exp)	MOV	@2(3), @DP(5)	M(R3 + 2) ⟶ M(R5 + DP)	10

Notes:

n is the number of the workspace register: $0 \leq n \leq 15$; n may not be 0 for indexed addressing.

exp is a symbol, number, or expression

The T_d and T_s fields are two bit portions of the instruction machine code. There are also S and D four bit fields, which are filled in with the four bit code for n. n is 0 for symbolic or direct addressing.

Figure 6-5. Addressing Modes

Comments Field

Comments can be entered after the last blank that follows the operands field. If the first character position of the source statement contains an asterisk (*), the entire source statement is a comment. Comments are listed in the source portion of the assembler listing, but have no affect on the object code.

Terms and Symbols

Symbols are used in the label field, the operator field, and the operand field. A symbol is a string of alphanumeric characters, beginning with an alphabetic character.

Terms are used in the operand fields of instructions and assembler directives. A term is a decimal or hexadecimal constant, an absolute assembly-time constant, or a label having an absolute value. Expressions can also be used in the operand fields of instructions and assembler directives.

Constants

Constants can be decimal integers (written as a string of numerals) in the range of −32,768 to +65,535. For example:

257

Constants can also be hexadecimal integers (a string of hexadecimal digits preceded by >). For example:

>09AF

6

ASCII character constants can be used by enclosing the desired character string in single quotes. For example:

'DX' = 4458_{16} 'R' + 0052_{16}

Throughout this book the subscript 16 is used to denote base 16 numbers. For example, the hexadecimal number 09AF will be written $09AF_{16}$.

Symbols

Symbols must begin with an alphabetic character and contain no blanks. Only the first six characters of a symbol are processed by the assembler.

The assembler predefines the dollar sign ($) to represent the current location in the program.

A given symbol can be used as a label only once, since it is the symbolic name of the address of the instruction. Symbols defined with the DXOP directive are used in the OPCODE field. Any symbol in the OPERANDS field must have been used as a label or defined by a REF directive.

Expressions

Expressions are used in the OPERANDS fields of assembly language statements. An expression is a term or a series of terms separated by the following arithmetic operations:

+ addition

− subtraction

* multiplication

/ division

The operator precedence is +, −, *, / (left to right).

The expression must not contain any imbedded blanks or extended operation defined (DXOP directive defined) symbols. Unary minus (a minus sign in front of a number or symbol) is performed first and then the expression is evaluated from left to right. An example of the use of the unary minus in an expression is:

LABEL+TABLE+(−INC)

which has the effect of the expression:

LABEL+TABLE−INC

The relocatability of an expression is a function of the relocatability of the symbols and constants that make up the expression. An expression is relocatable when the number of relocatable symbols or constants added to the expression is one greater than the number of relocatable symbols or constants subtracted from the expressions. All other expressions are absolute. The expression given earlier would be relocatable if the three symbols in the expression are all relocatable.

The following are examples of valid expressions.

BLUE+1

2*16+RED

440/2−RED

Survey of the 9900 Instruction Set

The 9900 instructions can be grouped into the following general categories: data transfer, arithmetic, comparison, logical, shift, branch, and CRU input/output operations. The list of all instructions and their effect on status bits is given in *Figure 6-6.*

Mnemonic	L>	A>	EQ	C	OV	OP	X	Mnemonic	L>	A>	EQ	C	OV	OP	X
A	X	X	X	X	X	-	-	DIV	-	-	-	-	X	-	-
AB	X	X	X	X	X	X	-	IDLE	-	-	-	-	-	-	-
ABS	X	X	X	X	X	-	-	INC	X	X	X	X	X	-	-
AI	X	X	X	X	X	-	-	INCT	X	X	X	X	X	-	-
ANDI	X	X	X	-	-	-	-	INV	X	X	X	-	-	-	-
B	-	-	-	-	-	-	-	JEQ	-	-	-	-	-	-	-
BL	-	-	-	-	-	-	-	JGT	-	-	-	-	-	-	-
BLWP	-	-	-	-	-	-	-	JH	-	-	-	-	-	-	-
C	X	X	X	-	-	-	-	JHE	-	-	-	-	-	-	-
CB	X	X	X	-	-	X	-	JL	-	-	-	-	-	-	-
CI	X	X	X	-	-	-	-	JLE	-	-	-	-	-	-	-
CKOF	-	-	-	-	-	-	-	JLT	-	-	-	-	-	-	-
CKON	-	-	-	-	-	-	-	JMP	-	-	-	-	-	-	-
CLR	-	-	-	-	-	-	-	JNC	-	-	-	-	-	-	-
COC	-	-	X	-	-	-	-	JNE	-	-	-	-	-	-	-
CZC	-	-	X	-	-	-	-	JNO	-	-	-	-	-	-	-
DEC	X	X	X	X	X	-	-	JOC	-	-	-	-	-	-	-
DECT	X	X	X	X	X	-	-	JOP	-	-	-	-	-	-	-
LDCR	X	X	X	-	-	1	-	SBZ	-	-	-	-	-	-	-
LI	X	X	X	-	-	-	-	SETO	-	-	-	-	-	-	-
LIMI	-	-	-	-	-	-	-	SLA	X	X	X	X	X	-	-
LREX	-	-	-	-	-	-	-	SOC	X	X	X	-	-	-	-
LWPI	-	-	-	-	-	-	-	SOCB	X	X	X	-	-	X	-
MOV	X	X	X	-	-	-	-	SRA	X	X	X	X	-	-	-
MOVB	X	X	X	-	-	X	-	SRC	X	X	X	X	-	-	-
MPY	-	-	-	-	-	-	-	SRL	X	X	X	X	-	-	-
NEG	X	X	X	X	X	-	-	STCR	X	X	X	-	-	1	-
ORI	X	X	X	-	-	-	-	STST	-	-	-	-	-	-	-
RSET	-	-	-	-	-	-	-	STWP	-	-	-	-	-	-	-
RTWP	X	X	X	X	X	X	X	SWPB	-	-	-	-	-	-	-
S	X	X	X	X	X	-	-	SZC	X	X	X	-	-	-	-
SB	X	X	X	X	X	X	-	SZCB	X	X	X	-	-	X	-
SBO	-	-	-	-	-	-	-	TB	-	-	X	-	-	-	-
								X	2	2	2	2	2	2	2
								XOP	2	2	2	2	2	2	2
								XOR	X	X	X	-	-	-	-

Notes: 1. When an LDCR or STCR instruction transfers eight bits or less, the OP bit is set or reset as in byte instructions. Otherwise these instructions do not affect the OP bit.

2. The X instruction does not affect any status bit; the instruction executed by the X instruction sets status bits normally for that instruction. When an XOP instruction is implemented by software, the XOP bit is set, and the subroutine sets status bits normally.

Figure 6-6. *Status Bits Affected by Instructions*

Data Transfer Instructions

Load— used to initialize processor or workspace registers to a desired value.

Move— used to move words or bytes from one memory location to another.

Store— used to store the status or workspace pointer registers in a workspace register.

Arithmetic Instructions

Addition and Subtraction—perform addition or subtraction of signed or unsigned binary words or bytes stored in memory.

Negate and Absolute Value—changes the sign or takes the absolute value of data words in memory.

Increment and Decrement—Adds or subtracts 1 or 2 from the specified data words in memory.

Multiply—Performs unsigned integer multiplication of a word in memory with a workspace register word to form a 32 bit product stored in two successive workspace register locations.

Divide—Divides a 32 bit unsigned integer dividend (contained in two successive workspace registers) by a memory word with the 16 bit quotient and 16 bit remainder stored in place of the dividend.

Compare Instructions

These instructions provide for masked or unmasked comparison of one memory word or byte to another or a workspace register word to a 16 bit constant.

Logical Instructions

OR and AND—masked or unmasked OR and AND operations on corresponding bits of two memory words. A workspace register word can be ORed or ANDed with a 16 bit constant.

Complement and Clear — The bits of a selected memory word can be complemented, or cleared or set to ones.

Exclusive OR—A workspace register word can be exclusive ORed with another memory word on a bit by bit basis.

Set Bits Corresponding—Set bits to one (SOC) or to zero (SZC) whose positions correspond to one positions in a reference word.

Shift Instructions

A workspace register can be shifted arithmetically or logically to the right. The registers can be shifted to the left (filling in vacated positions with zeroes) or circulated to the right. The shifts and circulates can be from 1 to 16 bit positions.

Branch Instructions

The branch instructions and the JMP (jump) instruction unconditionally branch to different parts of the program memory. If a branch occurs, the PC register will be changed to the value specified by the operand of the branch instruction. In subroutine branching the old value of the PC is saved when the branch occurs and then is restored when the return instruction is executed. The conditional jump instructions test certain status bits to determine if jump is to occur. When a jump is made the PC is loaded with the sum of its previous value and a displacement value specified in the operand portion of the instruction.

Control/CRU Instructions

These instructions provide for transferring data to and from the communications register input/output unit (CRU) using the CRUIN, CRUOUT and CRUCLK pins of the 9900.

Instruction Descriptions

The information provided for each instruction in the next section of this chapter is as follows:

6

Name of the instruction.

Mnemonic for the instruction.

Assembly language and machine code formats.

Description of the operation of the instruction.

Effect of the instruction on the Status Bits.

Examples.

Applications.

The format descriptions and examples are written without the label or comment fields for simplicity. Labels and comments fields can be used in any instruction if desired.

Each instruction involves one or two operand fields which are written with the following symbols:

G—Any addressing mode is permitted except I (Immediate).

R—Workspace register addressing.

exp—A symbol or expression used to indicate a location.

value—a value to be used in immediate addressing.

cnt—A count value for shifts and CRU instructions.

CRU—CRU (Communications Register Unit) bit addressing.

The instruction operation is described in written and equation form. In the equation form, an arrow(⟶) is used to indicate a transfer of data and a colon (:) is used to indicate a comparison. In comparisons, the operands are not changed. In transfers, the source operand (indicated with the subscript s) is not changed while the destination operand (indicated with the subscript d) is changed. For operands specified by the symbol G, the M(G) nomenclature is used to denote the memory word specified by G. MB(G) is used to denote the memory byte specified by G. Thus, transferring the memory word contents addressed by G_s to the memory word location specified by G_d and comparing the source (G_s) data to zero during the transfer, can be described as:

$M(G_s) \longrightarrow M(G_d)$

$M(G_s):0$

which is the operation performed by the MOV instruction:

MOV G_s,G_d

A specific example of this instruction could be:

MOV @ONE,3

which moves the contents of the memory word addressed by the value of the symbol ONE to the contents of workspace register 3:

M(ONE)⟶R3

M(ONE) : 0

DATA TRANSFER INSTRUCTIONS

The MOV instructions are used to transfer data from one part of the system to another part. The LOAD instructions are used to initialize registers to desired values. The STORE instructions provide for saving the status register (ST) or the workspace pointer (WP) in a specified workspace register.

LOAD IMMEDIATE

LI

Format: **LI R,value**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	0	0	0	0	0	R				(0200 + R) 0≤R≤15

Operation: The 16 bit data value in the word immediately following the instruction is loaded into the specified workspace register R.

value ⟶ R
immediate operand: 0

Affect on Status: **LGT,AGT,EQ**

Examples: **LI 7,5** 5 ⟶ R7

LI 8,>FF $00FF_{16}$ ⟶ R8

Applications: The LI instruction is used to initialize a workspace register with a program constant such as a counter value or data mask.

▸6

LOAD INTERRUPT MASK IMMEDIATE

LIMI

Format: **LIMI value**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	(0300)

Operation The low order 4 bit value (bits 12-15) in the word immediately following the instruction is loaded into the interrupt mask portion of the status register:

BITS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ST																

4 BIT VALUE

Affect on Status: Interrupt mask code only

Example: **LIMI 5**

Enables interrupt levels 0 through 5

Application: The LIMI instruction is used to initialize the interrupt mask to control which system interrupts will be recognized.

LOAD Workspace Pointer Immediate — LWPI

Format: **LWPI value**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	(02E0)

Operation: The 16 bit value contained in the word immediately following the instruction is loaded into the workspace pointer (WP):

value ⟶ WP

Affect on Status: None

Example: **LWPI >0500**

Causes 0500_{16} to be loaded into the WP.

Application: LWPI is used to establish the workspace memory area for a section of the program.

MOVE Word — MOV

Format: **MOV G_s,G_d**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	0	T_d		D				T_s		S				(C---)

Operation: The word in the location specified by G_s is transferred to the location specified by G_d, without affecting the data stored in the G_s location. During the transfer, the word (G_s data) is compared to 0 with the result of the comparison stored in the status register:

$M(G_s) \longrightarrow M(G_d)$
$M(G_s):0$

Status Bits Affected: **LGT, AGT, and EQ**

Examples:

MOV	**R1,R3**	R1 ⟶ R3, R1:0
MOV	***R1,R3**	M(R1) ⟶ R3, M(R1):0
MOV	**@ONES,*1**	M(ONES) ⟶ M(R1), M(ONES):0
MOV	**@2(5),3**	M(R5 + 2) ⟶ R3, M(R5 + 2):0
MOV	***R1 +,*R2 +**	M(R1) ⟶ M(R2), M(R1):0, (R1) + 2 ⟶ R1, (R2) + 2 ⟶ R2

Application: MOV is used to transfer data from one part of the system to another part.

MOVE Byte

MOVB

Format: **MOVB G_s,G_d**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	T_d		D				T_s		S				(D---)

Operation: The Byte addressed by G_s is transferred to the byte location specified by G_d. If G is workspace register addressing, the most significant byte is selected. Otherwise, even addresses select the most significant byte; odd addresses select the least significant byte. During the transfer, the source byte is compared to zero and the results of the comparison are stored in the status register.

$MB(G_s) \longrightarrow MB(G_d)$
$MB(G_s):0$

Status Bits Affected: **LGT, AGT, EQ, OP**

Examples: **MOVB @>1C14,3**
MOVB *8,4

These instructions would have the following example affects:

Memory Location	*Contents Initially*	*Contents After Transfer*
1C14	**2016**	**2016**
R3	**542B**	**202B**
R8	**2123**	**2123**
2123	**1040**	**1040**
R4	**0A0C**	**400C**

▶6

The underlined data are the bytes selected.

Application: MOVB is used to transfer 8 bit bytes from one byte location to another.

SWPB

Swap Bytes

Format: **SWPB G**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	1	0	1	1	T_s		S				(06C0 + T_s, S)

Operation: The most significant byte and the least significant bytes of the word at the memory location specified by G are exchanged.

Affect on Status: None

		Before	After
Example: **SWPB 3**	R3 Contents:	F302	02F3

Application: Used to interchange bytes if needed for subsequent byte operations.

STST

Store Status

Format: **STST R**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	0	1	1	0	0	R				(02C0 + R) $0 \le R \le 15$

Operation: The contents of the status register are stored in the workspace register specified:

ST ⟶ R

Affect on Status: None

Example: **STST 3** ST is transferred to R3

Application: STST is used to save the status for later reference.

STWP

Store Workspace Pointer

STWP

Format: **STWP R**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	0	1	0	1	0	R				(02A0 + R) 0≤R≤15

Operation: The contents of the workspace pointer are stored in the workspace register specified:

WP ⟶ R

Affect on Status: None

Example: **STWP 3** WP is transferred into R3

Appliation: STWP is used to save the workspace pointer for later reference.

▸6

ARITHMETIC INSTRUCTIONS

These instructions perform the following basic arithmetic operations: addition (byte or word), subtraction (byte or word), multiplication, division, negation, and absolute value. More complicated mathematical functions must be developed using these basic operations. The basic instruction set will be adequate for many system requirements.

ADD WORDS

A

Format: **A** G_s,G_d

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	0	1	0	T_d		D				T_s		S				(A---)

Operation: The data located at the address specified by G_s is added to the data located at the address specified by G_d. The resulting sum is placed in the G_d location and is compared to zero:

$$M(G_s) + M(G_d) \longrightarrow M(G_d)$$
$$M(G_s) + M(G_d):0$$

Status Bits Affected: **LGT, AGT, EQ, C, OV**

Examples:

A	**5,@TABLE**	R5 + M(TABLE) ⟶ M(TABLE)
A	**3,*2**	R3 + M(R2) ⟶ M(R2)

with the sums compared to 0 in each case. Binary addition affects on status bits can be understood by studying the following examples:

M(G$_s$)	*M(G$_d$)*	*Sum*	*LGT*	*AGT***	*EQ*	*C*	*OV**
1000	**0001**	**1001**	**1**	**1**	**0**	**0**	**0**
F000	**1000**	**0000**	**0**	**0**	**1**	**1**	**0**
F000	**8000**	**7000**	**1**	**1**	**0**	**1**	**1**
4000	**4000**	**8000**	**1**	**0**	**0**	**0**	**1**

*OV (overflow) is set if the most significant bit of the sum is different from the most significant bit of M(G_d) and the most significant bit of both operands are equal.

**AGT (arithmetic greater than) is set if the most significant bit of the sum is zero and if EQ (equal) is 0.

Application: Binary addition is the basic arithmetic operation required to generate many mathematical functions. This instruction can be used to develop programs to do multiword addition, decimal addition, code conversion, and so on.

AB

ADD BYTES

AB

Format: **AB** **G_s,G_d**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	0	1	1	T_d		D				T_s		S				(B---)

Operation: The source byte addressed by G_s is added to the destination byte addressed by G_d and the sum byte is placed in the G_d byte location. Recall that even addresses select the most significant byte and odd addresses select the least significant byte. The sum byte is compared to 0.

$MB(G_s) + MB(G_d) \longrightarrow MB(G_d)$
$MB(G_s) + MB(G_d):0$

Status Bits Affected: **LGT, AGT, EQ, C, OV, OP**

Example: **AB** **3,*4+** R3 + MB(R4) ⟶ MB(R4), R4 + 2 ⟶ R4
AB **@TAB,5** MB(TAB) + R5 ⟶ R5

To see how the AB works, the following example should be studied:
AB **@>2120,@>2123**

Memory Location	*Data Before Addition*	*Data After Addition*
2120	**F320**	**F320**
2123	**2106**	**21F9**

▶6 The underlined entries are the addressed and changed bytes.

Application: AB is one of the byte operations available on the 9900. These can be useful when dealing with subsystems or data that use 8 bit units, such as ASCII codes.

ADD IMMEDIATE

AI

Format: **AI R,Value**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	0	0	0	1	0	R				(0220 + R) 0≤R≤15

Operation: The 16 bit value contained in the word immediately following the instruction is added to the contents of the workspace register specified.

R + Value ⟶ R, R + Value:0

Status Bits Affected: **LGT, AGT, EQ, C, OV**

Example: **AI 6,>C**

Adds C_{16} to the contents of workspace register 6. If R6 contains 1000_{16}, then the instruction will change its contents to $100C_{16}$, and the LGT and AGT status bits will be set.

Application: This instruction is used to add a constant to a workspace register. Such an operation is useful for adding a constant displacement to an address contained in the workspace register.

SUBTRACT WORDS

S

Format: **S G_s,G_d**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	1	1	0	T_d		D				T_s		S				(6---)

Operation: The source 16 bit data (location specified by G_s) is subtracted from the destination data (location specified by G_d) with the result placed in the destination location G_d. The result is compared to 0.

$M(G_d) - M(G_s) \longrightarrow M(G_d)$
$M(G_d) - M(G_s):0$

Status Bits Affected: **LGT, AGT, EQ, C, OV**

Examples: **S @OLDVAL,@NEWVAL**

would yield the following example results:

Memory Location	*Before Subtraction Contents*	*After Subtraction Contents*
OLDVAL	**1225**	**1225**
NEWVAL	**8223**	**6FFE (8223-1225)**

All status bits affected would be set to 1 except equal which would be reset to 0.

Application: Provides 16 bit binary subtraction.

Subtract Bytes

SB

Format: **SB** G_sG_d

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	1	1	1	T_d		D				T_s		S				(7---)

Operation: The source byte addressed by G_s is subtracted from the destination byte addressed by G_d with the result placed in byte location G_d. The result is compared to 0. Even addresses select the most significant byte and odd addresses select the least significant byte. If workspace register addressing is used, the most significant byte of the register is used.

$MB(G_d) - MB(G_s) \longrightarrow MB(G_d)$
$MB(G_d) - MB(G_s):0$

Status Bits Affected: **LGT, AGT, C, EQ, OV, OP**

Format: **SB** ***6 + ,1**

R1 – MB(R6) ⟶ R1
R1 – MB(R6):0
R6 + 1 ⟶ R6

This operation would have the following example result:

Memory Location	*Contents Before Instruction*	*Contents After Instruction*
R6	**121D**	**121E**
121D	**3123**	**4123**
R1	**1344**	**F044**

▶6 The underlined entries indicated the addressed and changed bytes. The LGT (logical greater than) status bit would be set to 1 while the other status bits affected would be 0.

Application: SB provides byte subtraction when 8 bit operations are required by the system.

INC

Increment

Format: **INC G**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	0	1	1	0	T_s		S				(05--)

Operation: The data located at the address indicated by G is incremented and the result is placed in the G location and compared to 0.

M(G) + 1 ⟶ M(G)
M(G) + 1 : 0

Status Bits Affected: **LGT, AGT, EQ, C, OV**

Examples: **INC @TABL** M(TABL) + 1 ⟶ M(TABL)
INC 1 (R1) + 1 ⟶ R1

Application: INC is used to increment byte addresses and to increment byte counters. Autoincrementing addressing on byte instructions automatically includes this operation.

INCT

Increment by Two

Format: **INCT G**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	0	1	1	1	T_s		S				(05--)

Operation: Two is added to the data at the location specified by G and the result is stored at the G location and is compared to 0:

M(G) + 2 ⟶ M(G)
M(G) + 2 : 0

Status Bits Affected: **LGT, AGT, EQ, C, OV**

Example: **INCT 5** (R5) + 2 ⟶ R5

Application: This can be used to increment word addresses, though autoincrementing on word instructions does this automatically.

DEC/DECT

DECREMENT

DEC

Format: **DEC G**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	1	0	0	0	T_S		S				(06--)

Operation: One is subtracted from the data at the location specified by G, the result is stored at that location and is compared to 0:

M(G) – 1 ⟶ M(G)
M(G) – 1 : 0

Status Bits Affected: **LGT, AGT, EQ, C, OV**

Example: **DEC @TABL** M(TABL) – 1 ⟶ M(TABL)

Application: This instruction is most often used to decrement byte counters or to work through byte addresses in descending order.

DECREMENT BY TWO

DECT

Format: **DECT G**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	1	0	0	1	T_S		S				(06--)

▶6 *Operation:* Two is subtracted from the data at the location specified by G and the result is stored at that location and is compared to 0:

M(G) – 2 ⟶ M(G)
M(G) – 2 : 0

Status Bits Affected: **LGT, AGT, EQ, C, OV**

Example: **DECT 3** (R3) – 2 ⟶ R3

Application: This instruction is used to decrement word counters and to work through word addresses in descending order.

NEGATE

NEG

Format: **NEG G**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	0	1	0	0	T_s		S				(05--)

Operation: The data at the address specified by G is replaced by its two's complement. The result is compared to 0:

$-M(G) \longrightarrow M(G)$
$-M(G) : 0$

Status Bits Affected: **LGT, AGT, EQ, OV** (OV set only when operand = 8000_{16})

Example: **NEG 5** $-(R5) \longrightarrow R5$

If R5 contained $A342_{16}$, this instruction would cause the R5 contents to changed to $5CBE_{16}$ and will cause the LGT and AGT status bits to be set to 1.

Application: NEG is used to form the 2's complement of 16 bit numbers.

ABSOLUTE VALUE

ABS

Format: **ABS G**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	1	1	0	1	T_s		S				(07---)

6◀

Operation: The data at the address specified by G is compared to 0. Then the absolute value of this data is placed in the G location:

$M(G) : 0$
$|M(G)| \longrightarrow M(G)$

Status Bits Affected: **LGT, AGT, EQ, OV** (OV set only when operand = 8000_{16})

Example: **ABS @LIST(7)** $|M(R7 + LIST)| \longrightarrow M(R7 + LIST)$

If the data at R7 + LIST is $FF3C_{16}$, it will be changed to $00C4_{16}$ and LGT will be set to 1.

Application: This instruction is used to test the data in location G and then replace the data by its absolute value. This could be used for unsigned arithmetic algorithms such as multiplication.

MPY

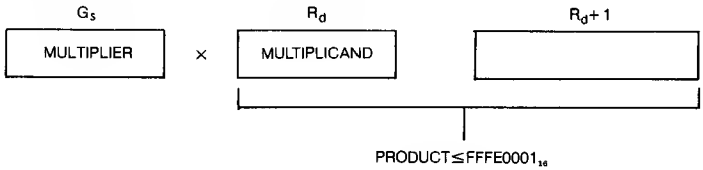
MULTIPLY

MPY

Format: **MPY** G_s,R_d

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	1	1	0	D				T_s		S				(3----)

Operation: The 16 bit data at the address designated by G_s is multiplied by the 16 bit data contained in the specified workspace register R. The unsigned binary product (32 bits) is placed in workspace registers R and R + 1:



Affect on Status: None

Example: **MPY @NEW,5**

If the data at location NEW is 0005_{16} and R5 contains 0012_{16}, this instruction will cause R5 to contain 0000_{16} and R6 to contain $005A_{16}$.

Application: MPY can be used to perform 16 bit by 16 bit binary multiplication. Several such 32 bit subproducts can be combined in such a way to perform multiplication involving larger multipliers and multiplicands such as a 32 bit by 32 bit multiplication.

▶6

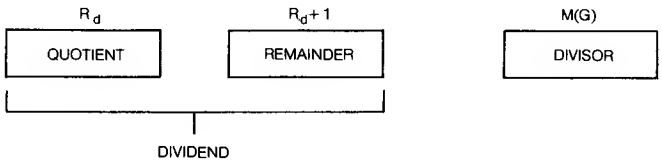
DIVIDE

DIV

Format: **DIV** G_s,R_d

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	1	1	1	D				T_s		S				(3---)

Operation: The 32 bit number contained in workspace registers R_d and $R_d + 1$ is divided by the 16 bit data contained at the address specified by G_s. The workspace register R_d then contains the quotient and workspace $R_d + 1$ contains the 16 bit remainder. The division will occur only if the divisor at G is greater than the data contained in R_d:



Affect on Status: Overflow (OV) is set if the divisor is less than the data contained in R_d. If OV is set, R_d and $R_d + 1$ are not changed.

Example: **DIV @LOC,2**

If R2 contains 0 and R3 contains $000D_{16}$ and the data at address LOC is 0005_{16}, this instruction will cause R2 to contain 0002_{16} and R3 to contain 0003_{16}. OV would be 0.

Application: DIV provides basic binary division of a 32 bit number by a 16 bit number.

COMPARISON INSTRUCTIONS

These instructions are used to test words or bytes by comparing them with a reference constant or with another word or byte. Such operations are used in certain types of division algorithms, number conversion, and in recognition of input command or limit conditions.

COMPARE WORDS

C

Format: **C** **G_s,G_d**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	0	0	0	T_d		D				T_s		S				(8----)

Operation: The 2's complement 16 bit data addressed by G_s is compared to the 2's complement 16 bit data addressed by G_d. The contents of both locations remain unchanged.

$M(G_s) : M(G_d)$

Status Bits Affected: **LGT, AGT, EQ**

Example: **C** **@T1,2**

This instruction has the following example results:

Data at Location T1	*Data in R2*	*Results of Comparison* LGT	AGT	EQ
FFFF	**0000**	**1**	**0**	**0**
7FFF	**0000**	**1**	**1**	**0**
8000	**0000**	**1**	**0**	**0**
8000	**7FFF**	**1**	**0**	**0**
7FFF	**7FFF**	**0**	**0**	**1**
7FFF	**8000**	**0**	**1**	**0**

Application: The need to compare two words occurs in such system functions as division, number conversion, and pattern recognition.

Compare Bytes

CB

Format: **CB** **G_s,G_d**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	0	0	1	T_d		D				T_s		S				(9---)

Operation: The 2's complement 8 bit byte addressed by G_s is compared to the 2's complement 8 bit byte addressed by G_d:

$MB(G_s) : MB(G_d)$

Status Bits Affected: **LGT, AGT,EQ,OP**

OP (odd parity) is based on the number of bits in the source byte.

Example: **CB** **1,*2**

with the typical results of (assuming R2 addresses an odd byte):

			Results of Comparison		
R1 data	*M(R2) Data*	*LGT*	*AGT*	*EQ*	*OP*
FFFF	**FF00**	**1**	**0**	**0**	**0**
7F00	**FF00**	**1**	**1**	**0**	**1**
8000	**FF00**	**1**	**0**	**0**	**1**
8000	**FF7F**	**1**	**0**	**0**	**1**
7F00	**007F**	**0**	**0**	**1**	**1**

The underlined entries indicate the byte addressed.

Application: In cases where 8 bit operations are required, CB provides a means of performing byte comparisons for special conversion and recognition problems.

Compare Immediate

CI

Format: **CI R,Value**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	0	1	0	0	0	R				(0280 + R) $0 \le R \le 15$

Operation: CI compares the specified workspace register contents to the value contained word immediately following the instruction:

R : Value

Status Bits Affected: **LGT, AGT, EQ**

Example: **CI 9,>F330**

If R9 contains 2183_{16}, the equal (EQ) and logical greater than (LGT) bits will be 0 and arithmetic greater than (AGT) will be set to 1.

Application: CI is used to test data to see if system or program limits have been met or exceeded or to recognize command words.

Compare Ones Corresponding

COC

Format: **COC G_s,R**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	0	0	0	D				T_s		S				(2---)

▶6

Operation: The data in the location addressed by G_s act as a mask for the bits to be tested in workspace register R. That is, only the bit position that contain ones in the G_s data will be checked in R. Then, if R contains ones in all the bit positions selected by the G_s data, the equal (EQ) status bit will be set to 1.

Status Bits Affected: **EQ**

Example: **COC @TESTBIT, 8**

If R8 contains $E306_{16}$ and location TESTBIT contains $C102_{16}$,

TESTBIT Mask = 1100 0001 0000 0010
R8 = 1110 0011 0000 0110

equal (EQ) would be set to 1 since everywhere the test mask data contains a 1 (underlined positions), R8 also contains a 1.

Application: COC is used to selectively test groups of bits to check the status of certain sub-systems or to examine certain aspects of data words.

Compare Zeroes Corresponding

CZC

Format: **CZC** **G_s,R**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	0	0	1	D				T_s		S				(2---)

Operation: The data located in the address specified by G_s act as a mask for the bits to be tested in the specified workspace register R. That is, only the bit positions that contain ones in the G_s data are the bit positions to be checked in R. Then if R contains zeroes in all the selected bit positions, the equal (EQ) status bit will be set to 1.

Status Bits Affected: **EQ**

Examples: **CZC** **@TESTBIT,8**

If the TESTBIT location contains the value $C102_{16}$ and the R8 location contains 2301_{16},

TESTBIT Data = 1100 0001 0000 0010
R8 = 0010 0011 0000 0001
X

the equal status bit would be reset to zero since not all the bits of R8 (note the X position) are zero in the positions that the TESTBIT data contains ones.

Application: Similar to the COC instruction.

6

LOGIC INSTRUCTIONS

The logic instructions allow the processor to perform boolean logic for the system. Since AND, OR, INVERT, and Exclusive OR (XOR) are available, any boolean function can be performed on system data.

AND IMMEDIATE

ANDI

Format: **ANDI R,Value**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	0	0	1	0	0	R				(0240 + R) 0≤R≤15

Operation: The bits of the specified workspace register R are logically ANDed with the corresponding bits of the 16 bit binary constant value contained in the word immediately following the instruction. The 16 bit result is compared to zero and is placed in the register R:

R AND Value ⟶ R
R AND Value : 0

Recall that the AND operation results in 1 only if *both* inputs are 1.

Status Bits Affected: **LGT, AGT, EQ**

Example: **ANDI 0,>6D03**

If workspace register 0 contains $D2AB_{16}$, then (D2AB) AND (6D03) is 4003_{16}:

▶6

Value =	0110	1101	0000	0011
R0 =	1101	0010	1010	1011
R0 AND Value =	0100	0000	0000	0011 = 4003_{16}

This value is placed in R0. The LGT and AGT status bits are set to 1.

Application: ANDI is used to zero all bits that are not of interest and leave the selected bits (those with ones in Value) unchanged. This can be used to test single bits or isolate portions of the word, such as a four bit group.

OR Immediate

ORI

Format: **ORI R,Value**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	0	0	1	1	0	R				(0260 + R) 0≤R≤15

Operation: The bits of the specified workspace register R are ORed with the corresponding bits of the 16 bit binary constant contained in the word immediately following instruction. The 16 bit result is placed in R and is compared to zero:

R OR Value ⟶ R
R OR Value : 0

Recall that the OR operation results in a 1 if *either* of the inputs is a 1.

Status Bits Affected: **LGT, AGT, EQ**

Example: **ORI 5,>6D03**

If R5 contained $D2AB_{16}$, then R5 will be changed to $FFAB_{16}$:

R5 =	1101	0010	1010	1011
Value =	0110	1101	0000	0011
	1111	1111	1010	1011 = $FFAB_{16}$ = R5 OR Value

with LGT being set to 1.

Application: Used to implement the OR logic in the system.

Exclusive OR

XOR

Format: **XOR** **G_sR_d**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	0	1	0	D				T_s		S				(2——)

Operation: The exclusive OR is performed between corresponding bits of the data addressed by G_s and the contents of workspace register R_d. The result is placed in workspace register R_d and is compared to 0:

$M(G_s)$ XOR $R_d \longrightarrow R_d$
$M(G_s)$ XOR R_d : 0

Status Bits Affected: **LGT, AGT, EQ**

Example: **XOR** **@CHANGE,2**

If location CHANGE contains $6D03_{16}$ and R2 contains $D2AA_{16}$, R2 will be changed to $BFA9_{16}$:

CHANGE Data =	0110	1101	0000	0011
R2 =	1101	0010	1010	1010
M(CHANGE) XOR R2 =	1011	1111	1010	1001 = $BFA9_{16}$

and the LGT status bit will be set to 1. Note that the exclusive OR operation will result in a 1 if *only one* of the inputs is a 1.

Application: XOR is used to implement the exclusive OR logic for the system.

▶6

Invert

INV

Format: **INV** **G**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	0	1	0	1	T_s		S				(05——)

Operation: The bits of the data addressed by G are replaced by their complement. The result is compared to 0 and is stored at the G location:

$\overline{M(G)} \longrightarrow M(G)$
$\overline{M(G)}$: 0

Status Bits Affected: **LGT, AGT, EQ**

Example: **INV** **11**

If R11 contains $00FF_{16}$, the instruction would change the contents to $FF00_{16}$, causing the LGT status bit to set to 1.

Application: INV is used to form the 1's complement of 16 bit binary numbers, or to invert system data.

CLEAR

CLR

Format: **CLR G**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	0	0	1	1	T_s		S				(04——)

Operation: 0000_{16} is placed in the memory location specified by G.

$0000_{16} \longrightarrow M(G)$

Affect on Status: None

Example: **CLR *11**

would clear the contents of the location addressed by the contents of R11, that is:

$0000_{16} \longrightarrow M(R11)$

Application: CLR is used to set problem arguments to 0 and to initialize memory locations to zero during system start-up operations.

SET TO ONE

SETO

Format: **SETO G**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	1	1	0	0	T_s		S				(07——)

Operation: $FFFF_{16}$ is placed in the memory location specified by G: $FFFF_{16} \longrightarrow M(G)$

Affect on Status: None

Example: **SETO 11**

would cause all bits of R11 to be 1.

Application: Similar to CLR

6◀

SET ONES CORRESPONDING

SOC

Format: **SOC** G_s,G_d

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	1	0	T_d		D				T_s		S				(E——)

Operation: This instruction performs the OR operation between corresponding bits of the data addressed by G_s and the data addressed by G_d. The result is compared to 0 and is placed in the G_d location:

$$M(G_s) \text{ OR } M(G_d) \longrightarrow M(G_d)$$
$$M(G_s) \text{ OR } M(G_d) : 0$$

Status Bits Affected: **LGT, AGT, EQ**

Example: **SOC 3,@NEW**

If location NEW contains $AAAA_{16}$ and R_3 contains $FF00_{16}$, the contents at location NEW will be changed to $FFAA_{16}$ and the LGT status bit will be set to 1.

Application: Provides the OR function between any two words in memory.

SET ONES CORRESPONDING, BYTE

SOCB

Format: **SOCB** G_s,G_d

▶6

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	1	1	T_d		D				T_s		S				(F——)

Operation: The logical OR is performed between corresponding bits of the byte addressed by G_s and the byte addressed by G_d with the result compared to 0 and placed in location G_d:

$$MB(G_s) \text{ OR } MB(G_d) \longrightarrow MB(G_d)$$
$$MB(G_s) \text{ OR } MB(G_d) : 0$$

Status Bits Affected: **LGT, AGT, EQ, OP**

Example: **SOCB 5,8**

If R5 contains $F013_{16}$ and R8 contains $AA24_{16}$, the most significant byte of R8 will be changed to FA_{16} so that R8 will contain $FA24_{16}$ and the LGT status bit will be set to 1.

Application: The SOCB provides the logical OR function on system bytes.

Set to Zeroes Corresponding

SZC

Format: **SZC** G_s,G_d

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	1	0	0	T_d		D				T_s		S				(4——)

Operation: The data addressed by G_s forms a mask for this operation. The bits in the destination data (addressed by G_d) that correspond to the one bits of the source data (addressed by G_s) are cleared. The result is compared to zero and is stored in the G_d location.

$$\overline{M(G_s)} \text{ AND } M(G_d) \longrightarrow M(G_d)$$
$$\overline{M(G_s)} \text{ AND } M(G_d) : 0$$

Status Bits Affected: **LGT, AGT, EQ**

Example: **SZC 5,3**

If R5 contains $6D03_{16}$ and R3 contains $D2AA_{16}$, this instruction will cause the R3 contents to change to $92A8_{16}$:

R5 (Mask) = 0110 1101 0000 0011
R3 = 1101 0010 1010 1010
Result = 1001 0010 1010 1000 = $92A8_{16}$

with the LGT status bit set. The underlined entries indicate which bits are to be cleared.

Application: SZC allows the programmer to selectively clear bits of data words. For example, when an interrupt has been serviced, the interrupt request bit can be cleared by using the SZC instruction.

SZCB

SET TO ZEROES CORRESPONDING, BYTES

SZCB

Format: **SZCB** G_sG_d

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	1	0	1	T_d		D				T_s		S				(5---)

Operation: The byte addressed by G_s will provide a mask for clearing certain bits of the byte addressed by G_d. The bits in the G_d byte that will be cleared are the bits that are one in the G_s byte. The result is compared to zero and is placed in the G_d byte:

$$\overline{MB(G_s)} \text{ AND } MB(G_d) \longrightarrow MB(G_d)$$
$$\overline{MB(G_s)} \text{ AND } MB(G_d) : 0$$

Status Bits Affected: **LGT, AGT, EQ, OP**

Example: **SZCB @BITS,@TEST**

If location BITS is an *odd* address which locates the data $18\underline{F0}_{16}$, and location TEST contains an *even* address which locates the data $\underline{AA}24_{16}$, the instruction will clear the first four bits of TEST data changing it to $0A24_{16}$.

Application: Provides selective clearing of bits of system bytes.

▶6

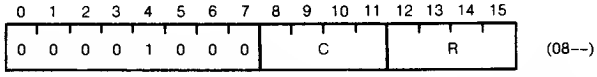
SHIFT INSTRUCTIONS

These instructions are used to perform simple binary multiplication and division on words in memory and to rearrange the location of bits in the word in order to examine a given bit with the carry (C) status bit.

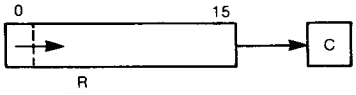
SHIFT RIGHT ARITHMETIC

SRA

Format: **SRA R,Cnt**



Operation: The contents of the specified workspace register R are shifted right Cnt times, filling the vacated bit position with the sign (most significant bit) bit: The shifted number is compared to zero:



Status Bits Affected: **LGT, AGT, EQ, C**

Number of Shifts: Cnt (number contained in the instruction from 0 to 15) specifies the number of bits shifted unless Cnt is zero in which case the shift count is taken from the four least significant bits of workspace register 0. If both Cnt and these four bits are 0, a 16 bit position shift is performed.

6◀

Example: **SRA 5,2** *Shift R5 2 bit positions right*
SRA 7,0

If R0 least four bits contain 6_{16}, then the second instruction will cause register 7 to be shifted 6 bit positions (Cnt in that instruction is 0):

If R7 Before Shift = 1011 1010 1010 1010 = $BAAA_{16}$
R7 After Shift = 1111 1110 1110 1010 = $FEEA_{16}$
If R5 Before Shift = 0101 0101 0101 0101 = 5555_{16}
R5 After Shift = 0001 0101 0101 0101 = 1555_{16}
After the R7 shift the LGT would be set, and Carry = 1
After the R5 shift LGT and AGT would be set and Carry = 0

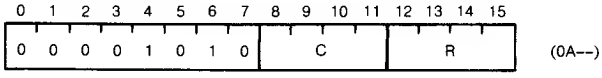
Application: SRA provides binary division by 2^{Cnt}.

SLA

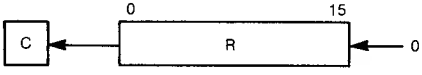
Shift Left Arithmetic

SLA

Format: **SLA R,Cnt**



Operation: The contents of workspace register R are shifted left Cnt times (or if Cnt = 0, the number of times specified by the least four bits of R0) filling the vacated positions with zeroes. The carry contains the value of the last bit shifted out to the left and the shifted number is compared to zero:



Status Bits Affected: **LGT, AGT, EQ, C, OV**

Example: **SLA 10,5**

If workspace register 10 contains 1357_{16} the instruction would change its contents to $6AE0_{16}$, causing the arithmetic greater than (AGT), logical greater than (LGT), and overflow (OV) bits to set. Carry would be zero, the value of the last bit shifted.

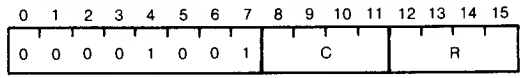
Application: SLA performs binary multiplication by 2^{Cnt}

▶6

Shift Right Logical

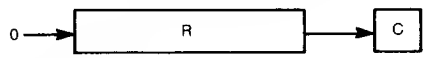
SRL

Format: **SRL R,Cnt**



(09--)

Operation: The contents of the workspace register specified by R are shifted right Cnt times (or if Cnt = 0, the number of times specified by the least four bits or R0) filling in the vacated positions with zeroes. The carry contains the value of the last bit shifted out to the right and the shifted number is compared to zero:



Status Bits Affected: **LGT, AGT, EQ, C**

Example: **SRL 0,3**

If R0 contained $FFEF_{16}$, the contents would become $1FFD_{16}$ with the AGT, LGT, and C bits set to 1:

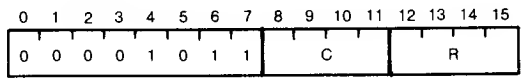
R0 Before Shift = 1111 1111 1110 1111 = $FFEF_{16}$
R0 After Shift = 0001 1111 1111 1101 = $1FFD_{16}$

Application: Performs binary division by 2^{Cnt}

Shift Right Circular

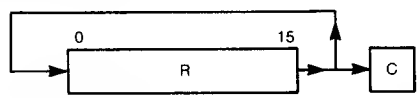
SRC

Format: **SRC R,Cnt**



(0B--)

Operation: On each shift the bit shifted out of bit 15 is shifted back into bit 0. Carry contains the value of the last bit shifted and the shifted number is compared to 0. The number of shifts to be performed is the number Cnt, or if Cnt = 0, the number contained in the least significant four bits of R0:



Status Bits Affected: **LGT, AGT, EQ, C**

Example: **SRC 2,7**

If R2 initially contains $FFEF_{16}$, then after the shift it will contain $DFFF_{16}$ with LGT and C set to 1.

R2 Before Shift = 1111 1111 1110 1111 = $FFEF_{16}$
R2 After Shift = 1101 1111 1111 1111 = $DFFF_{16}$

Application: SRC can be used to examine a certain bit in the data word, change the location of 4 bit groups, or swap bytes.

UNCONDITIONAL BRANCH INSTRUCTIONS

These instructions give the programmer the capability of choosing to perform the next instruction in sequence or to go to some other part of the memory to get the next instruction to be executed. The branch can be a subroutine type of branch, in which case the programmer can return to the point from which the branch occurred.

BRANCH

B

Format: **B** G_s

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	0	0	0	1	T_s		S				(04--)

Operation: The G_s address is placed in the program counter, causing the next instruction to be obtained from the location specified by G_s.

Affect on Status: None

Example: **B** ***3**

If R3 contains $21CC_{16}$, then the next instruction will be obtained from location $21CC_{16}$.

Application: This instruction is used to jump to another part of the program when the current task has been completed.

▶6

Branch and Link

BL

Format: **BL** G_s

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	1	0	1	0	T_s		S				(06--)

Operation: The source address G_s is placed in the program counter and the address of the instruction following the BL instruction is saved in workspace register 11.

$G_s \longrightarrow PC$
(Old PC) $\longrightarrow$ R11

Affect on Status: None

Example: **BL @TRAN**

Assume the BL instruction is located at 3200_{16} and the value assigned to TRAN is 2000_{16}. PC will be loaded with the value 2000_{16} (TRAN) and R11 will be loaded with the value 3202_{16} (old PC value).

Application: This is a shared workspace subroutine jump. Both the main program and the subroutine use the same workspace registers. To get back to the main program at the branch point, the following branch instruction can be used at the end of the subroutine:

```
B       *11
```

which causes the R11 contents (old PC value) to be loaded into the program counter.

Branch and Load Workspace Pointer

BLWP

Format: **BLWP** G_s

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	0	0	0	0	T_s		S				(04--)

Operation: The word specified by the source G_s is loaded into the workspace pointer (WP) and the next word in memory ($G_s + 2$) is loaded into the program counter (PC) to cause the branch. The old workspace pointer is stored in the new workspace register 13, the old PC value is stored in the new workspace register 14, and the status register is stored in new workspace register 15:

$M(G_s) \longrightarrow WP$
$M(G_s + 2) \longrightarrow PC$
(Old WP) $\longrightarrow$ New R13
(Old PC) $\longrightarrow$ New R14
(Old ST) $\longrightarrow$ New R15

Affect on Status: None

Example: **BLWP *3**

Assuming that R3 contains 2100_{16} and location 2100_{16} contains 0500_{16} and location 2102_{16} contains 0100_{16}, this instruction causes WP to be loaded with 0500_{16} and PC to be loaded with 0100_{16}. Then, location $051A_{16}$ will be loaded with the old WP value, the old PC value will be saved in location $051C_{16}$, and the status (ST) will be saved in location
▶6 $051E_{16}$. The next instruction will be taken from address 0100_{16} and the subroutine workspace will begin at 0500_{16} (R0). BLWP and XOP do not test IREQ at the end of instruction execution.

Application: This is a context switch subroutine jump with the transfer vector location specified by G_s. It uses a new workspace to save the old values of WP, PC, and ST (in the last three registers). The advantage of this subroutine jump over the BL jump is that the subroutine gets its own workspace and the main program workspace contents are not disturbed by subroutine operations.

XOP

Extended Operation

Format: **XOP** **G_s,n**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	0	1	1	D				T_s		S				(2---)

Operation: n specifies which extended operation transfer vector is to be used in the context switch branch from XOP to the corresponding subprogram. The effective address G_s is placed in R11 of the subprogram workspace in order to pass an argument or data location to the subprogram:

$M(n \times 4 + 0040_{16}) \longrightarrow WP$
$M(n \times 4 + 0042_{16}) \longrightarrow PC$
(Old WP) $\longrightarrow$ New R13
(Old PC) $\longrightarrow$ New R14
(Old ST) $\longrightarrow$ New R15
$G_s \longrightarrow$ New R11

Affect on Status: Extended Operation (X) bit is set.

Example: **XOP** ***1,2**

Assume R1 contains 0750_{16}. WP is loaded with the word at address 48_{16} (first part of transfer vector for extended operation 2) and PC is loaded with the word at address $4A_{16}$. If location 48_{16} contains 0200_{16}, this will be the address of R0 of the subprogram workspace. Thus, location 0236_{16} (new R11) will be loaded with 0750_{16} (contents of R1 in main program), location $023A_{16}$ (new R13) will be loaded with the old WP value, location $023C_{16}$ will be loaded with the old PC value, and location $023E_{16}$ (new R15) will be loaded with the old status value:

$M(48_{16}) \longrightarrow WP$
$M(4A_{16}) \longrightarrow PC$
(Old WP) $\longrightarrow M(023A_{16})$ New R13
(Old PC) $\longrightarrow M(023C_{16})$ New R14
(Old ST) $\longrightarrow M(023E_{16})$ New R15
$0750_{16} \longrightarrow M(0236_{16})$ New R11

Application: This can be used to define a subprogram that can be called by a single instruction. As a result, the programmer can define special purpose instructions to augment the standard 9900 instruction set.

Return with Workspace Pointer

RTWP

Format: **RTWP**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	(0380)

Operation: This is a return from a context switch subroutine. It occurs by restoring the WP, PC, and ST register contents by transferring the contents of subroutine workspace registers R13, R14, and R15, into the WP, PC, and ST registers, respectively.

R13 ⟶ WP
R14 ⟶ PC
R15 ⟶ ST

Status Bits Affected: **All (ST receives the contents of R15)**

Application: This is used to return from subprograms that were reached by a transfer vector operation such as an interrupt, extended operation, or BLWP instruction.

Unconditional Jump

JMP

Format: **JMP EXP**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	1	0	0	0	0	DISP								(10--)

▸6 *Operation:* The signed displacement defined by EXP is added to the current contents of the program counter to generate the new value of the program counter. The location jumped to must be within −128 to +127 words of the present location.

Affect on Status: None

Example: **JMP THERE**

If this instruction is located at 0018_{16} and THERE is the label of the instruction located at 0010_{16}, then the Exp value placed in the object code would be FB (for −5). Since the Assembler makes this computation, the programmer only needs to place the appropriate label or expression in the operand field of the instruction.

Application: If the subprogram to be jumped to is within 128 words of the JMP instruction location, the unconditional JMP is preferred over the unconditional branch since only one memory word (and one memory reference) is required for the JMP while two memory words and two memory cycles are required for the B instruction. Thus, the JMP instruction can be implemented faster and with less memory cost than can the B instruction.

X

EXECUTE

Format: **X** **G_s**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	0	0	1	0	T_s		S				(04--)

Operation: The instruction located at the address specified by G_s is executed.

Status Bits Affected: **Depends on the instruction executed**

Example: **X** ***11**

If R11 contains 2000_{16} and location 2000_{16} contains the instruction for CLR 2 then this execute instruction would clear the contents of register 2 to zero.

Application: X is useful when the instruction to be executed is dependent on a variable factor.

6◀

CONDITIONAL JUMP INSTRUCTIONS

JH
JL
JHE
JLE
JGT
JLT
JEQ
JNE
JOC
JNC
JNO
JOP

These instructions perform a branching operation only if certain status bits meet the conditions required by the jump. These instructions allow decision making to be incorporated into the program. The conditional jump instruction mnemonics are summarized in *Table 6-1* along with the status bit conditions that are tested by these instructions.

Format: **Mnemonic Exp**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	1	CODE				DISP								(1---)

Operation: If the condition indicated by the branch mnemonic is true, the jump will occur using relative addressing as was used in the unconditional JMP instruction. That is, the Exp defines a displacement that is added to the current value of the program counter to determine the location of the next instruction, which must be within 128 words of the jump instruction.

Effect on Status Bits: None

Example:

```
C      R1, R2
JNE    LOOP
```

The first instruction compares the contents of registers one and two. If they are not equal, EQ = 0 and the JNE instruction causes the branch to LOOP to be taken. If R1 and R2 are equal, EQ = 1 and the branch is not taken.

▶6

Table 6-1. Status Bits Tested by Instructions

Mnemonic	*L>*	*A>*	*EQ*	*C*	*OV*	*OP*	*Jump if:*	*CODE**
JH	X	–	X	–	–	–	$L> \bullet \overline{EQ}=1$	B
JL	X	–	X	–	–	–	$L>+EQ=0$	A
JHE	X	–	X	–	–	–	$L>+EQ=1$	4
JLE	X	–	X	–	–	–	$\overline{L>}+EQ=1$	2
JGT	–	X	–	–	–	–	$A>=1$	5
JLT	–	X	X	–	–	–	$A>+EQ=0$	1
JEQ	–	–	X	–	–	–	$EQ=1$	3
JNE	–	–	X	–	–	–	$EQ=0$	6
JOC	–	–	–	X	–	–	$C=1$	8
JNC	–	–	–	X	–	–	$C=0$	7
JNO	–	–	–	–	X	–	$OV=0$	9
JOP	–	–	–	–	–	X	$OP=1$	C

Note: In the Jump if column, a logical equation is shown in which • means the AND operation, + means the OR operation, and a line over a term means negation or inversion.

*CODE is entered in the CODE field of the OPCODE to generate the machine code for the instruction.

Application: Most algorithms and programs with loop counters require these instructions to decide which sequence of instructions to do next.

CRU INSTRUCTIONS

The communications register unit (CRU) performs single and multiple bit programmed input/output for the microcomputer. All input consists of reading CRU line logic levels into memory, and all output consists of setting CRU output lines to bit values from a word or byte of memory. The CRU provides a maximum of 4096 input and 4096 output lines that may be individually selected by a 12 bit address which is located in bits 3 through 14 of workspace register 12. This address is the hardware base address for all CRU communications.

SET BIT TO LOGIC ONE

SBO

Format: **SBO disp**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	1	1	1	0	1	DISP								(1D--)

Operation: The CRU bit at disp plus the hardware base address is set to one. The hardware base address is bits 3 through 14 of workspace register 12. The value disp is a signed displacement.

1 ⟶ Bit (disp + base address)

Affect on Status: None

Example: **SBO 15**

If R12 contains a software base address of 0200_{16} so that the hardware base address is 0010_{16} (the hardware base address is one-half the value of the contents of R12 excluding bits 0, 1 and 2), the above instruction would set CRU line $010F_{16}$ to a 1.

Application: Output a one on a single bit CRU line.

Set Bit to Logic Zero

SBZ

Format: **SBZ** **disp**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	1	1	1	1	0	DISP								(1E--)

Operation: The CRU bit at disp plus the base address is reset to zero. The hardware base address is bits 3 through 14 of workspace register 12. The value disp is a signed displacement.

0 ⟶ Bit (disp + hardware base address)

Affect on Status: None

Example: **SBZ** **2**

If R12 contains 0000_{16}, the hardware base address is 0 so that the instruction would reset CRU line 0002_{16} to zero.

Application: Output a zero on a single bit CRU line.

Test Bit

TB

Format: **TB** **disp**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	1	1	1	1	1	DISP								(1F--)

▸6

Operation: The CRU bit at disp plus the base address is read by setting the value of the equal (EQ) status bit to the value of the bit on the CRU line. The hardware base address is bits 3 through 14 of workspace register 12. The value disp is a signed displacement.

Bit (disp + hardware base address) ⟶ EQ

Status Bits Affected: **EQ**

Example: **TB** **4**

If R12 contains 0140_{16}, the hardware base address is $A0_{16}$ (which is one-half of 0140_{16}):

R12 Contents = 0000 0001 0100 0000

Note that the underlined hardware base address is $0A0_{16}$. Equal (EQ) would be made equal to the logic level on CRU line $0A0_{16} + 4 =$ CRU line $0A4_{16}$.

Application: Input the CRU bit selected.

LOAD CRU

LDCR

Format: **LDCR G_s,Cnt**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	1	0	0	C				T_s		S				(3---)

Operation: Cnt specifies the number of bits to be transferred from the data located at the address specified by G_s, with the first bit transferred from the least significant bit of this data, the next bit from the next least significant bit and so on. If Cnt = 0, the number of bits transferred is 16. If the number of bits to be transferred is one to eight, the source address is a byte address. If the number of bits to be transferred is 9 to 16, the source address is a word address. The source data is compared to zero before the transfer. The destination of the first bit is the CRU line specified by the hardware base address, the second bit is transferred to the CRU line specified by the hardware base address + 1, and so on.

Status Bits Affected: **LGT, AGT, EQ**
OP (odd parity) with transfer of 8 or less bits.

Example: **LDCR @TOM,8**

Since 8 bits are transferred, TOM is a byte address. If TOM is an even number, the most significant byte is addressed. If R12 contains 0080_{16}, the hardware base address is 0040_{16} which is the CRU line that will receive the first bit transferred. 0041_{16} will be the address of the next bit transferred, and so on to the last (8th) bit transferred to CRU line 0047_{16}. This transfer is shown in *Figure 6-7.*

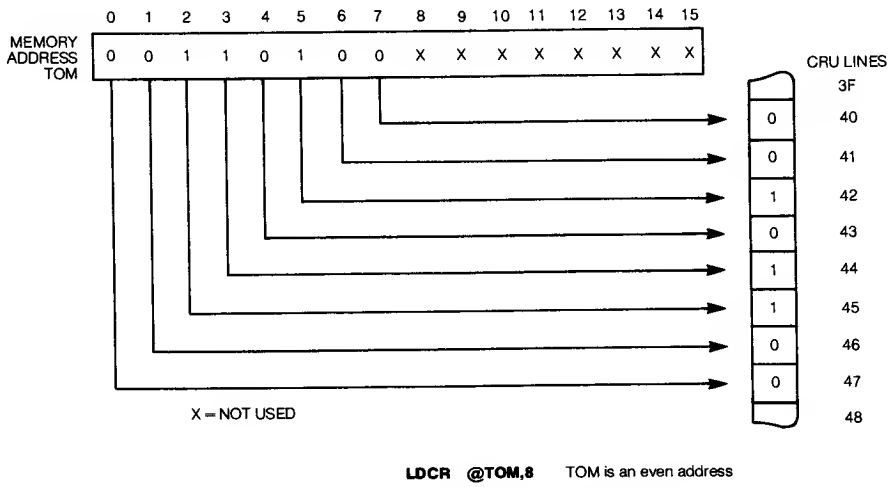
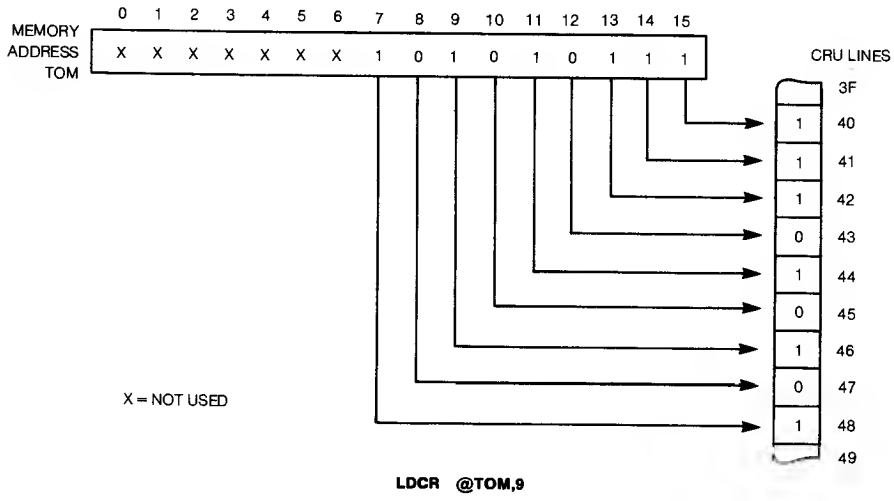


Figure 6-7. LDCR byte transfer

Application: The LDCR provides a number of bits (from 1 to 16) to be transferred from a memory word or byte to successive CRU lines, starting at the hardware base address line; the transfer begins with the least significant bit of the source field and continues to successively more significant bits. A further example of word versus byte transfers is given in *Figure 6-8,* in which a 9 bit (word addressed source) transfer is shown.



▶6

Figure 6-8. LDCR Word transfer

STCR

Store CRU

Format: **STCR G_s,Cnt**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	1	0	1	C				T_s		S				(3---)

Operation: Cnt specifies the number of bits to be transferred from successive CRU lines (starting at the hardware base address) to the location specified by G_s, beginning with the least significant bit position and transferring successive bits to successively more significant bits. If the number of bits transferred is 8 or less, G_s is a byte address. Otherwise, G_s is a word address. If Cnt = 0, 16 bits are transferred. The bits transferred are compared to zero. If the transfer does not fill the entire memory word, the unfilled bits are reset to zero.

Status Bits Affected: **LGT, AGT, EQ**
OP for transfers of 8 bits or less

Example: **STCR 2,7**

Since 7 bits are to be transferred this is a byte transfer so that the bits will be transferred to the most significant byte of R2. *Figure 6-9* illustrates this transfer assuming that R12 contains 90_{16} so that the hardware base address is 48_{16} for the first bit to be transferred.

Note: Bits 8-15 are unchanged if transfer is less than 8 bits.

6◂

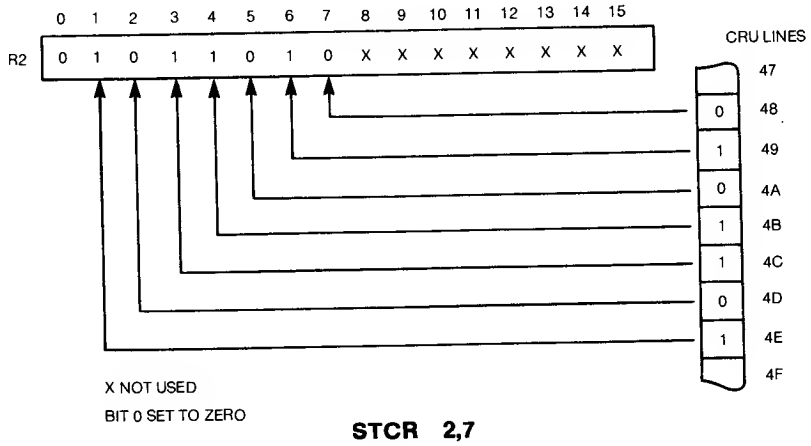


Figure 6-9. STCR Example

CONTROL INSTRUCTIONS

The control instructions are primarily applicable to the Model 990 Computer. These instructions are RSET (Reset), IDLE, CKOF (Clock off), CKON (Clock on), LREX (restart). The Model 990/10 also supports the long distance addressing instructions: LDS (Load long distance source) and LDD (Long distance destination). The use of these instructions are covered in the appropriate Model 990 computer programmer's manuals.

The control instructions have an affect on the 9900 signals on the address lines during the CRU Clock as shown below:

instruction	A_0	A_1	A_2	OP CODE (0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)	
LREX	H	H	H	0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0	(03E0)
CKOF	H	H	L	0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0	(03C0)
CKON	H	L	H	0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0	(03A0)
RSET	L	H	H	0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0	(0360)
IDLE	L	H	L	0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0	(0340)
CRU	L	L	L		

The IDLE instruction puts the 9900 in the idle condition and causes a CRUCLK output every six clock cycles to indicate this state. The processor can be removed from the idle state by 1) a $\overline{\text{RESET}}$ signal, 2) any interrupt that is enabled, or 3) a $\overline{\text{LOAD}}$ signal.

For the 9900 the above instructions are referred to as external instructions, since external hardware can be designed to respond to these signals. The address signals A_0, A_1, and A_2 can be decoded and the instructions used to control external hardware.

SPECIAL FEATURES OF THE 9940

The 9940 instruction set includes the instructions already presented. Two of these instructions are slightly different for the 9940. These are the extended operation and the load interrupt mask immediate instructions. There are two new arithmetic instructions that provide for binary coded decimal (BCD) addition and subtraction. The 9940 uses extended operations 0 through 3 to generate the load interrupt mask and the decimal arithmetic instructions. Thus, the 9940 extended operations 4 through 15 are available to the programmer.

LOAD IMMEDIATE INTERRUPT MASK

LIIM

Format: **LIIM n**

$0 \leq n \leq 3$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	0	1	1	0	0	1	0	0	0	0	0	n		(2C8–)

Operation: The interrupt mask bits 14 and 15 of the status register are loaded with n. Subsequent to this instruction, interrupt levels greater than n will be ignored by the processor, and interrupts of level n or less will be responded to by the processor.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ST																
															n	

Status Bits Affected: **Interrupt Mask (Bits 14 and 15)**

Example: **LIIM 2**

This operation will load the interrupt mask with 2, that is bit 14 would be set to a 1 and bit 15 would be reset to zero. This would disable interrupts of level 3, but would enable other interrupt levels.

Application: This instruction is used to control the 9940 interrupt system.

Extended Operation

XOP

Format: **XOP** **G_s,n**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	0	1	1	D				T_s		S				(2——)

Operation: n specifies the extended operation transfer vector to be used in the context switch to the extended operation subprogram. The TMS9940 restricts the range of n ($4 \leq n \leq 15$) so that there are only 12 XOP's available. This is because the first four are used by the processor to implement the LIIM, DCA, and DCS instructions. The transfer vector procedure for the programmer-defined extended operations is:

$M(40_{16} + 4xn) \longrightarrow$ (WP)
$M(42_{16} + 4xn) \longrightarrow$ (PC)
$G_s \longrightarrow$ (New WR11)
(Old WP) $\longrightarrow$ (New WR13)
(Old PC) $\longrightarrow$ (New WR14)
(Old ST) $\longrightarrow$ (New WR15)

Status Bits Affected: **None**

Example and Applications: **XOP** ***1,4**

This instruction will cause an extended operation 4 to occur with the new workspace register 11 containing the address found in workspace register 1. The new WP value will be obtained from $40_{16} + 4 \times 4 = 50_{16}$ and the new PC value will be obtained from 52_{16}.

▶6

DCA

Decimal Correct Addition

Format: **DCA** G_s

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	0	1	1	0	0	0	0	T_s		S				(2C—)

Operation: The byte addressed by G_s is corrected according to the table given in *Figure 6-10.* This operation is a processor defined extended operation with n = 0 so that the sequence of events described under the XOP discussion will occur in executing this instruction.

Status Bits Affected: **LGT, AGT, EQ, C, P, and DC (Digit Carry).**

Example: **DCA** ***10**

This instruction would cause the byte addressed by the contents of the current workspace register 10 to be decimal adjusted in accordance with the truth table of *Figure 6-10.*

Application: This instruction is used immediately after the binary addition of two bytes (AB instruction) to correct any decimal digits outside the BCD code range of 0000_2 through 1001_2. It also keeps decimal addition accurate by responding to digit carries. For example, if 8_{16} is added to 8_{16} in BCD addition, 16_{16} should be generated. However, if this operation is performed with binary addition, 10_{16} results:

```
    0 0 0 0     1 0 0 0
+   0 0 0 0     1 0 0 0
-----------------------
    0 0 0 1     0 0 0 0      Digit Carry = 1
```

The DCA detects the digit carry and adds 0110_2 to the least significant digit to get the correct 16_{16}.

Decimal Correct Subtraction

DCS

Format: **DCS** G_s

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	0	1	1	0	0	0	1	T_s		S				(2C——)

Operation: The byte addressed by G_s is corrected according to the table given in *Figure 6-10.* This instruction is a processor defined extended operation with n = 1, so that the sequence of events described under extended operation will occur in executing this instruction.

Status Bits Affected: **LGT, AGT, EQ, C, P, and DC**

Example: **DCS 3**

This instruction would cause the most significant byte of register 3 to be corrected in accordance with the truth table of *Figure 6-10.*

Application: As in the DCA instruction, this instruction extends the 9940 capability to include decimal subtraction. The programmer first performs binary subtraction on bytes (the SB instruction) and then immediately performs the DCS operation on the result byte to correct the result so that it is within the BCD code range 0000_2 through 1001_2.

▸6

0	7	
X	Y	8-BIT BYTE CONTAINING RESULT OF BINARY ADD OR SUBTRACT OF 2 BCD DIGITS
MSB	LSB	

BYTE BEFORE EXECUTION				BYTE AFTER DCA				BYTE AFTER DCS			
C	X	DC	Y	C	X	DC	Y	C	X	DC	Y
0	X<10	0	Y<10	0	X	0	Y	—	—	—	—
0	X<10	1	Y<10	0	X	0	Y+6	—	—	—	—
0	X<9	0	Y≥10	0	X+1	1	Y+6	—	—	—	—
1	X<10	0	Y<10	1	X+6	0	Y	—	—	—	—
1	X<10	1	Y<10	1	X+6	0	Y16	—	—	—	—
1	X<10	0	Y≥10	1	X+7	1	Y+6	—	—	—	—
0	X≥10	0	Y<10	1	X+6	0	Y	—	—	—	—
0	Z≥10	1	Y<10	1	X+6	0	Y+6	—	—	—	—
0	X≥9	0	Y≥10	1	X+7	1	Y+6	—	—	—	—
0	X	0	Y	—	—	—	—	0	X+10	1	Y+10
0	X	1	Y	—	—	—	—	0	X+10	0	Y
1	X	0	Y	—	—	—	—	1	X	1	Y+10
1	X	1	Y	—	—	—	—	1	X	0	Y

Figure 6-10. Result of DCA and DCS Instructions of the 9940.

CHAPTER 7

Program Development: Software Commands— Descriptions and Formats

7

INTRODUCTION

The purpose of this chapter is to provide reference data for the various software development systems available for the 9900 family of microprocessors and microcomputers. Most of the information is reproduced in reference card form on heavy stock and inserted at the back of the book. Perforations allow easy removal. So pull out the ones you will use and fold for pocket reference. The specific choice of programming system will dictate which cards to pull.

Table 7-1 lists the sections in the chapter. One or more cards are made for those sections marked with a bullet. The section on Assembly Language programming describes the basic format for coding instructions and assembler directives. It is a general topic, applicable to all of the programming systems.

The 9900 reference card will come in handy for product design and programming activities for any of the processors. Explanation of the terms, mnemonics instruction execution rules, etc. can be found in Chapters 4, 5, and 6.

The complete TM 990/402 Line-by-Line Assembler User's Guide is included because this EPROM resident software is used in Chapter 9. It should serve as an illustration of the need for some form of an assembler in writing even the simplest programs. Contrast the programming effort of Chapter 3 will be the extended application of Chapter 9, and you will appreciate the power of this LBL assembler.

Reference material for the other programming systems is in the form of lists of commands and their syntax. These pages are not stand-alone documents. Software documentation is supplied with each of the programming systems and is required for full explanations of the commands and their use. Experienced designers always need assistance in recalling exact command mnemonics and their formats. Thus, this chapter supports you in any
▸7 programming environment by appropriate reminders.

Table 7-1

Assembly language programming and assembler directives
- 9900 Reference Data
  TM 990/402 Line-by-Line Assembler
- TIBUG Monitor
- TM 990/302 Software Development board
- TXDS Commands for the FS 990 PDS
- AMPL Reference data
- POWER BASIC Commands
- Cross Support reference data
  - Assembler
  - Simulator
  - Utilities

Assembly Language Programming: Formats and Directives

7

ASSEMBLY LANGUAGE PROGRAMMING

An assembly language is a computer oriented language for writing programs. The TMS9900 recognizes instructions in the form of 16 bit (or longer) binary numbers, called instruction or operation codes (Opcodes). Programs could be written directly in these binary codes, but it is a tedious effort, requiring frequent reference to code tables. It is simpler to use names for the instructions, and write the programs as a sequence of these easily recognizable names (called mnemonics). Then, once the program is written in mnemonic or assembly language form, it can be converted to the corresponding binary coded form (machine language form). The assembler programs described here indicate parts of PX9ASM, TXMIRA and SDSMAC, which operate on cassette, floppy disc, and moving head disc systems respectively. Several other assemblers are available from TI which provide fewer features, but operate with much smaller memory requirements.

Assembly Language Application

The assembly language programming and program verification through simulation or execution are the main elements involved in developing microprocessor programs. The overall program development effort consists of the following steps:

- Define the problem.
- Flowchart the solution to the problem.
- Write the assembly language program for the flowchart.
- Execute the Assembler to generate the machine code.
- Correct any format errors indicated by the Assembler.
- Execute the corrected machine code program on a TMS9900 computer or on a Simulator to verify program operation.

This program development sequence is defined in flowchart form in *Figure 7-1.*

▶7

Assembly Language Formats

The general assembly language source statements consists of four fields as follows:

LABEL MNEMONIC OPERANDS COMMENT

The first three fields must occur within the first 60 character positions of the source record. At least one blank must be inserted between fields.

Label Field

The label consists of from one to six characters, beginning with an alphabetic character in character position one of the source record. The label field is terminated by at least one blank. When the assembler encounters a label in an instruction it assigns the current value of the location counter to the label symbol. This is the value associated with the label symbol and is the address of the instruction in memory. If a label is not used, character position 1 must be a blank.

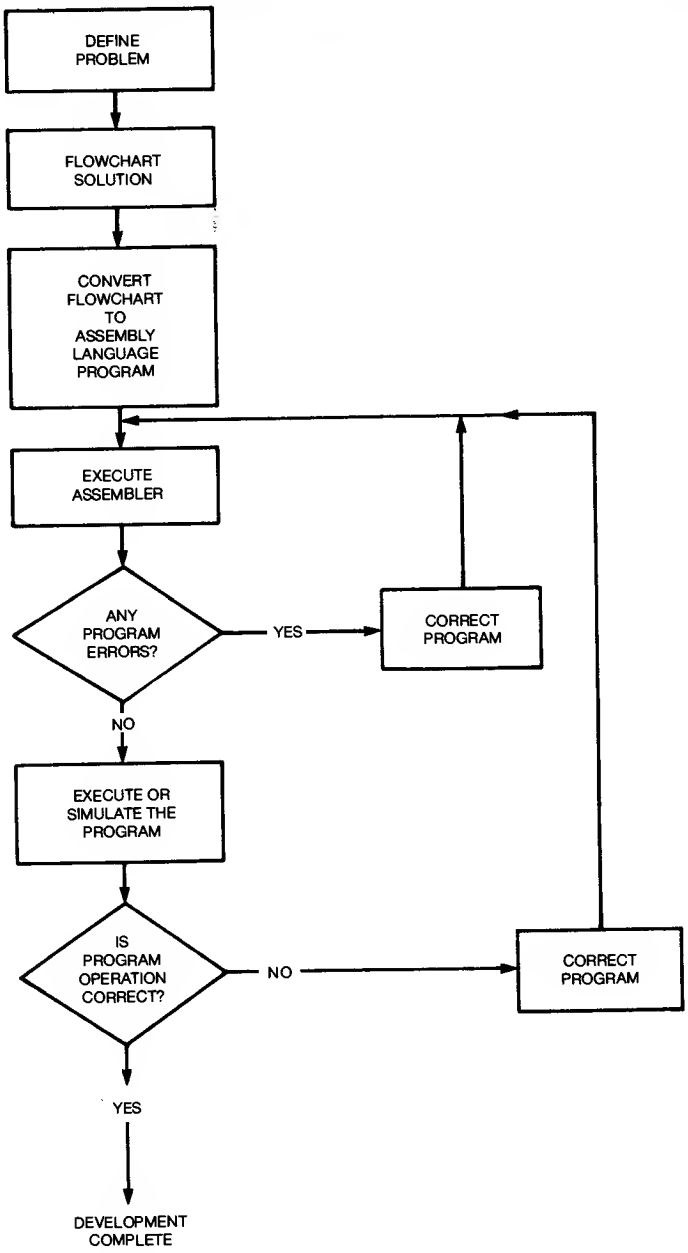


Figure 7-1. Program Development Flowchart

Mnemonic or Opcode Field

This field contains the mnemonic code of one of the instructions, one of the assembly language directives, or a symbol representing one of the program defined operations. This field begins after the last blank following the label field. Examples of instruction mnemonics include A for addition and MOV for data movement. The mnemonic field is required since it identifies which operation is to be performed.

Operands Field

The operands specify the memory locations of the data to be used by the instruction. This field begins following the last blank that follows the mnemonic field. The memory locations can be specified by using constants, symbols, or expressions, to describe one of several addressing modes available.

Comment Field

Comments can be entered after the last blank that follows the operands field. If the first character position of the source statement contains an asterisk (*), the entire source statement is a comment. Comments are listed in the source portion of the Assembler listing, but have no affect on the object code.

Terms and Symbols

Symbols are used in the label field, the operator field, and the operand field. A symbol is a string of alphanumeric characters, beginning with an alphabetic character.
Terms are used in the operand fields of instructions and assembler directives. A term is a decimal or hexadecimal constant, an absolute assembly-time constant, or a label having an absolute value. Expressions can also be used in the operand fields of instructions and assembler directives.

▶7

Constants

Constants can be decimal integers (written as a string of numerals) in the range of −32,768 to +65,535. For example:

257

Constants can also be hexadecimal integers (a string of hexadecimal digits preceded by >). For example:

>09AF

ASCII character constants can be used by enclosing the desired character string in single quotes. For example:

'DX'

Throughout this book the subscript 16 is used to denote base 16 numbers. For example, the hexadecimal number 09AF is written $09AF_{16}$.

Symbols

Symbols must begin with an alphabetic character and contain no blanks. Only the first six characters of a symbol are processed by the Assembler.

The Assembler predefines the dollar sign ($) to represent the current location in the program. The symbols R0 through R15 are used to represent workspace registers 0 through 15, respectively.

A given symbol can be used as a label only once, since it is the symbolic name of the address of the instruction. Symbols defined with the DXOP directive are used in the OPCODE field. Any symbol in the OPERANDS field must have been used as a label or defined by a REF directive.

Expressions

Expressions are used in the OPERANDS fields of assembly language statements. An expression is a constant, a symbol, or a series of constants and symbols separated by the following arithmetic operators:

+ addition
− subtraction
* multiplication
/ division

Unary minus is performed first and then the expression is evaluated from left to right. A unary minus is a minus sign (negation) in front of a number or a symbol.

The expression must not contain any imbedded blanks or extended operation defined (DXOP directive) symbols.

The multiplication and division operations must be used on absolute code symbols. The result of evaluating the expression up to the multiplication or division operator must be an absolute value. There must not be more than one more relocatable symbol added to an expression than are subtracted from it. 7◀

The following are examples of valid expressions:

BLUE+1	The sum of the value of symbol BLUE plus 1.
GREEN−4	The result of subtracting 4 from the value of symbol GREEN.
2*16+RED	The sum of 32 and the value of symbol RED.
440/2−RED	220 minus the value of symbol RED.

ASSEMBLER DIRECTIVES

General Information

The assembler directives are used to assign values to program symbolic names, address locations, and data. There are directives to set up linkage between program modules and to control output format, titles, and listings.
The assembler directives take the general form of:

LABEL DIRECTIVE EXPRESSION COMMENT

The LABEL field begins in column one and extends to the first blank. It is optional on all directives except the EQU directive which requires a label. There is no label in the OPTION directive. When no label is present, the first character position in the field must be a blank. When a label is used (except in an EQU directive) the label is assigned the current value of the location counter.

The two required directives are:

IDT	Assign a name to the program
END	Terminate assembly

The most commonly used optional directives are:

EQU	Assign a value to a label or a data name.
RORG	Relocatable Origin
BYTE	Assign values to successive bytes of memory
DATA	Assign 16 bit values to successive memory words
TEXT	Assign ASCII values to successive bytes of memory

Other directives include:

AORG	Absolute (non-relocatable) Origin
DORG	Dummy Origin
BSS	Define bytes of storage beginning with symbol
BES	Define bytes of storage space ending with symbol
DXOP	Define an extended operation
NOP	No operation Pseudo-instruction
RT	Return from subroutine Pseudo-instruction
PAGE	Skip to new page before continuing listing
TITL	Define title for page headings
LIST	Allows listing of source statements
UNL	Prevents listing of source statements
OPTION	Selects output option to be used
DEF	Define symbol for external reference
REF	Reference to an external source

Required Directives

Two directives must be supplied to identify the beginning and end of the assembly language program. The IDT directive must be the first statement and the END directive must be the last statement in the assembly language program.

Program Identifier — IDT

This directive assigns a name to the program and must precede any directive that generates object code. The basic format is:

```
IDT   'Name'
```

The name is the program name consisting of up to 8 characters. As an example, if a program is to be named Convert, the basic directive would be:

```
IDT   'CONVERT'
```

The name is printed only when the directive is printed in the source listing.

Program End — END

This directive terminates the assembly. Any source statement following this directive is ignored. The basic format is:

```
END
```

Initialization Directives

These directives are used to establish values for program symbols and constants.

Define Assembly-Time Constant — EQU

Equate is used to assign values to program symbols. The symbol to be defined is placed in the label field and the value or expression is placed in the Expression field:

```
Symbol   EQU   Expression
```

The symbol can represent an address or a program parameter. This directive allows the program to be written in general symbolic form. The equate directive is used to set up the symbol values for a specific program application.

The following are examples of the use of the Equate directive:

```
TIME    EQU   HOURS+5
N       EQU   8
VAR     EQU   >8000
```

BYTE
DATA
TEXT

Initialize Memory

These directives provide for initialization of successive 8 bit bytes of memory with numerical data (BYTE directive) or with ASCII character codes (TEXT directive). The DATA directive provides for the initialization of successive 16 bit words with numerical data.

The formats are the same for all three directives:

Directive Expression-list

The Label and Comment are optional. The expression or value list contains the data entries for the 8 bit bytes (BYTE directive), or the 16 bit words (DATA directive), or a character string enclosed in quotes (TEXT directive).

Examples of the use and effects of these directives are shown in *Figure 7-2.*

Program Location Directives

These directives affect the location counter by causing the instructions to be located in specified areas of memory.

AORG
RORG
DORG

▸7

Origin Directives

These directives set the address of the next instruction to the value listed in the expression field of the directive:

Directive Expression

The expression field is required on all except the RORG directive. It is a value or an expression (containing only previously defined symbols). This value is the address of the next instruction and is the value that is assigned to the label (if any) and to the location counter. The AORG and DORG expressions must result in an absolute value and contain no character constants.

Example Directives:

```
KONS    BYTE  >10, -1, 'A', 'B', N + 3
WD1     DATA  >01FF, 3200, -'AF', 8, N+>1000
MSG1    TEXT  'EXAMPLE'
```

AFFECTS ON MEMORY LOCATION	MEMORY DATA: DIRECTIVE ENTRY	RESULTING DATA (BINARY FORM)				RESULTING DATA (HEXADECIMAL)
KONS	>10,−1	0001	0000	1111	1111	1 OFF
KONS+2	'A', 'B'	0100	0001	0100	0010	4142
KNOS+4	N+3	0000	1011	X	X	0B--
•	•	•	•	•	•	•
•	•	•	•	•	•	•
WD1	>01FF	0000	0001	1111	1111	01FF
WD1+2	3200	0000	1100	1000	0000	0C80
WD1+4	−'AF'	1011	1110	1011	1010	BEBA
WD1+6	8	0000	0000	0000	1000	0008
WD1+8	N+>1000	0001	0000	0000	1000	1008
•	•	•	•	•	•	•
•	•	•	•	•	•	•
•	•	•	•	•	•	•
MSG1	'EX'	0100	0101	0101	1000	4558
MSG1+2	'AM'	0100	0001	0101	1101	414D
MSG1+4	'PL'	0101	0000	0100	1100	504C
MSG1+6	'E'	0100	0101	X	X	4E--

XX (--) is original unaltered data in this location. N is assumed to be previously defined as 8.

Figure 7-2. Initialization Directive Examples

The AORG directive causes this value to be absolute and fixed. For example: 7◀

```
AORG >1000+X
```

If X has been previously defined to have an absolute value of 6, the next instruction would be unalterably located at the address 1006_{16}. If a label had been included, it would have been assigned this same value.

The RORG directive causes this value to be relative or relocatable so that subsequent operations by the assembler or simulator can relocate the block of instructions to any desired area of memory. Thus, a relocatable block of instructions occupying memory locations 1000_{16} to 1020_{16} could be moved by subsequent simulator (or other software) operations to locations 2000_{16} to 2020_{16}. An example RORG statement is:

```
SEG1    RORG >1000
```

This directive would cause SEG1 and the value of the location counter (address of the next instruction) to be set to 1000_{16}. This and all subsequent locations are relocatable.

SEG2 RORG

This directive would cause subsequent instructions to be at relocatable addresses. SEG2 and the address of the next instruction would be set to the value of the location counter.

The DORG directive causes the instructions to be listed but the assembler does not generate object code that can be passed on to simulators or other subsystems. However, symbols defined in the dummy section would then be legitimate symbols for use in the AORG or RORG program sections. For example:

DORG 0

The labels with the subsequent dummy section of instructions will be assigned values relative to the start of the section (the instruction immediately following this directive). No object code would be generated for this section.

An RORG directive is used after a DORG or AORG section to cause the subsequent instructions to be relocatable object code. If no origin directives are included in the assembly language program, all object code is relocatable starting at (referenced to) an address of 0.

Storage Allocation Directives

BES
BSS

These directives reserve a block of memory (range of addresses) for data storage by advancing the location counter by the amount specified in the expression field. Thus, the instruction after the directive will be at an address equal to the expression value plus the address of the instruction just before the directive.

▶7

Basic Formats:

BES Expression

BSS Expression

If a label is included in the BSS directive it is assigned the value of the location counter at the *first byte* if the storage block. If the label is included in the BES directive it is assigned the value of the location counter for the instruction *after* the block.

The Expression designates the number of bytes to be reserved for storage. It is a value or an expression containing no character constants. Expressions must contain only previously defined symbols and result in an absolute value.

Examples:

BUFF1 BES >10

A 16 byte buffer is provided. Had the location counter contained the value 100_{16} (FF_{16} was the address of the previous instruction), the new value of the location counter would be 110_{16}, and this would be the value assigned to the symbol BUFF1. The next instruction after the buffer would be at address 110_{16}.

BUFF2 BSS 20

If the previous instruction is located at FF_{16}, BUFF2 will be assigned the value 100_{16}, and the next instruction will be located at 114_{16}. A 20 byte area of storage with addresses 100_{16} through 113_{16} has been reserved.

Word Boundary — EVEN

This directive causes the location counter to be set to the next even address (beginning of the next word) if it currently contains an odd address. The basic format is:

EVEN

The label is assigned the value of the location counter prior to the EVEN directive.

Program Listing Control Directives

These directives control the printer, titling, and listing provided by the assembler.

Output Options — OPTION

The basic format of this directive is:

OPTION Keyword-list

No label is permitted. The keywords control the listing as follows: 7◀

Keyword	*Listing*
XREF	Print a cross reference listing.
OBJ	Print a hexadecimal listing of the object code.
SYMT	Print a symbol table with the object code.

Example:

OPTION XREF,SYMT
Print a cross reference listing and the symbol table with the object code.

Advance Page

PAGE

This directive causes the assembly listing to continue at the top of the next page. The basic format is:

```
PAGE
```

Page Title

TITL

This directive specifies the title to be printed at the top of each page of the assembler listing. The basic format is:

```
TITL   'String'
```

The String is the title enclosed in single quotes. For example:

```
TITL   'REPORT GENERATOR'
```

Source Listing Control

LIST
UNL

These directives control the printing of the source listing. UNL inhibits the printing of the source listing: LIST restores the listing. The basic formats are:

```
UNL
LIST
```

Extended Operation Definition

DXOP

This directive names an extended operation. Its format is:

▶7

```
DXOP   SYMBOL, Term
```

The symbol is the desired name of the extended operation. Term is the corresponding number of the extended operation. For example:

```
DXOP   DADD,13
```

defines DADD as extended operation 13. Once DADD has been so defined, it can be used as the name of a new operation, just as if it were one of the standard instruction mnemonics.

Program Linkage Directives

These directives enable program modules to be assembled separately and then integrated into an executable program.

External Definition — DEF

This directive makes one or more symbols available to other programs for reference. Its basic format is:

```
DEF    Symbol-list
```

Symbol-list contains the symbols to be defined by the program being assembled. For example:

```
DEF    ENTER,    ANS
```

causes the assembler to include the Symbols ENTER and ANS in the object code so that they are available to other programs. When DEF does not precede the source statements that contain the symbols, the assembler identifies the symbols as multi-defined symbols.

External Reference — REF

This directive provides access to symbols defined in other programs. The basic format is:

```
REF    Symbol-list
```

The Symbol-list contains the symbols to be included in the object code and used in the operand fields of subsequent source statements. For example:

```
REF    ARG1,ARG2
```

causes the symbols ARG1 and ARG2 to be included in the object code so that the corresponding address can be obtained from other programs.

Note: If a REF symbol is the first operand of a DATA directive causing the value of the symbol to be in 0 absolute location, the symbol will not be linked correctly in location 0. 7◀

ASSEMBLER OUTPUT

Introduction

The types of information provided by Assemblers include:

Source Listing	— Shows the source statements and the resulting object code.
Error Messages	— Errors in the assembly language program are indicated.
Cross Reference	— Summarizes the label definitions and program references.
Object Code	— Shows the object code in a tagged record format to be passed on to a computer or simulator for execution.

Source Listing

Assemblers produce a source listing showing the source statements and the resulting object code. A typical listing is shown in *Figure 7-3.*

```
0229                        *
0230                        *        DEMONSTRATE EXTERNAL REFERENCE LINKING
0231                        *
0232                                 REF     EXTR
0233   028C                          RORG
0234   028C    C820                  MOV     @EXTR, @EXTR
       028E    0000
       0290    028E'
0235   0292    28E0                  XOR     @EXTR, 3
       0294    0290'
0236   6000                          AORG    >B000
0237   B000    3220                  LOCR    @EXTR, B
       B002    0294'
023B   B004    0420                  BLWP    @EXTR
       B006    B002
0239   B008    0223                  AI      3, EXTR
       B00A    B006
0240   B00C    38A0                  MPY     @EXTR, 2
       B00E    B00A
0241   0296                          RORG
0242   0296    C820                  MOV     @EXTR, @EXTR
       029B    B00E
       029A    029B'
0243   029C    28E0                  XOR     @EXTR, 3
       029E    029A'
0244   C000                          AORG    >C000
0245   C000    3220                  LOCR    @EXTR, B
       C002    029E'
0246   C004    0420                  BLWP    @EXTR
       C006    C002
0247   C008    0223                  AI      3, EXTR
       C00A    C006
0248   C00C    38A0                  MPY     @EXTR, 2
       C00E    C00A
```

Figure 7-3. Typical Source Listing.

The first line available in a listing is the title line which will be blank unless a TITL directive has been used. After this line, a line for each source statement is printed. For example:

```
0018    0156    C820    MOV    @INIT+3,@3
        0158    012B'
        015A    0003
```

In this case the source statement:

```
MOV    @INIT+3,@3
```

produces 3 lines of object code. The source statement number 18 applies to the entire 3 line entry. Each line has its own location counter value (0156, 0158, and 015A). C820 is the OPCODE for MOV with symbolic memory addressing.

012B' is the value for INIT + 3. 0003 is for the direct address 3. The apostrophe (') after 012B indicates this address is program-relocatable. Source statements are numbered sequentially, whether they are listed or not (listing could be prevented by using the UNLIST directive).

9900
Reference Data

7

INSTRUCTION FORMAT

FORMAT (USE)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1 (ARITH)	OP CODE			B	T_D		D				T_S		S			
2 (JUMP)	SIGNED DISPLACEMENT*															
3 (LOGICAL)	OP CODE						D				T_S		S			
4 (CRU)	OP CODE						C				T_S		S			
5 (SHIFT)	OP CODE								C				W			
6 (PROGRAM)	OP CODE										T_S		S			
7 (CONTROL)	OP CODE												NOT USED			
	OP CODE											NU	W			
8 (IMMEDIATE)	IMMEDIATE VALUE															
9 (MPY,DIV,XOP)	OP CODE						D				T_S		S			

KEY

B = BYTE INDICATOR
(1 = BYTE, 0 = WORD)
T_D = D ADDR, MODIFICATION
D = DESTINATION ADDR.
T_S = ADDR. MODIFICATION

S = SOURCE ADDR.
C = XFR OR SHIFT LENGTH (COUNT)
W = WORKSPACE REGISTER NO.
* = SIGNED DISPLACEMENT OF − 128 TO + 127 WORDS
NU = NOT USED

T_D/T_S FIELD

	CODE	EFFECTIVE ADDRESS	MNEMONIC
00:	REGISTER	WP + 2 · [S OR D]	Rn
01:	INDIRECT	(WP + 2 · [S OR D])	*Rn
10:	INDEXED (S OR D ≠ 0)	(WP + 2 · [S OR D]) + (PC); PC ← PC + 2	NUM (Rn)
10:	SYMBOLIC (DIRECT, S OR D = 0)	(PC); PC ← PC + 2	NUM
11:	INDIRECT WITH AUTO INCREMENT	(WP + 2 · [S OR D]); INCREMENT EFF. ADDR.	*Rn+

STATUS REGISTER

▶7

0	1	2	3	4	5	6	7 — 11	12 — 15
L>	A>	=	C	O	P	X	RESERVED	INTERRUPT MASK

0 — LOGICAL GREATER THAN
1 — ARITHMETIC GREATER THAN
2 — EQUAL/TB INDICATOR
3 — CARRY FROM MSB
4 — OVERFLOW
5 — PARITY (ODD NO. OF BITS SET)
6 — XOP IN PROGRESS

INTERRUPT MASK
F = ALL INTERRUPTS ENABLED
0 = ONLY LEVEL 0 ENABLED

INTERRUPTS

TRAP ADDR	WP
TRAP ADDR + 2	PC

LEVEL	ID	TRAP ADDR	LEVEL	ID	TRAP ADDR
0	RESET	0000	8	EXTERNAL	0020
1	EXTERNAL	0004	9	EXTERNAL	0024
2	EXTERNAL	0008	10	EXTERNAL	0028
3	EXTERNAL	000C	11	EXTERNAL	002C
4	EXTERNAL	0010	12	EXTERNAL	0030
5	EXTERNAL	0014	13	EXTERNAL	0034
6	EXTERNAL	0018	14	EXTERNAL	0038
7	EXTERNAL	001C	15	EXTERNAL	003C

NOTES: 1) XOP VECTORS 0—15 OCCUPY MEMORY LOCATIONS 0040-007C
2) LOAD VECTOR OCCUPIES MEMORY LOCATIONS FFFC–FFFF

BLWP TRANSFERS	RTWP TRANSFERS	BL TRANSFER	XOP TRANSFER
WP → NEW W13	CURRENT W13 → WP	PC → W11	EFF. ADDR. → NEW W11
PC → NEW W14	CURRENT W14 → PC		WP → NEW W13
ST → NEW W15	CURRENT W15 → ST		PC → NEW W14
			ST → NEW W15
			1 → ST6

INSTRUCTIONS BY MNEMONIC

MNEMONIC	OP CODE	FORMAT	RESULT COMPARED TO ZERO	STATUS AFFECTED	INSTRUCTIONS
A	A000	1	Y	0-4	ADD(WORD)
AB	B000	1	Y	0-5	ADD(BYTE)
ABS	0740	6	Y	0-4	ABSOLUTE VALUE
AI	0220	8	Y	0-4	ADD IMMEDIATE
ANDI	0240	8	Y	0-2	AND IMMEDIATE
B	0440	6	N	—	BRANCH
BL	0680	6	N	—	BRANCH AND LINK (W11)
BLWP	0400	6	N	—	BRANCH LOAD WORKSPACE POINTER
C	8000	1	N	0-2	COMPARE (WORD)
CB	9000	1	N	0-2,5	COMPARE (BYTE)
CI	0280	8	N	0-2	COMPARE IMMEDIATE
CKOF	03C0	7	N	—	EXTERNAL CONTROL
CKON	03A0	7	N	—	EXTERNAL CONTROL
CLR	04C0	6	N	—	CLEAR OPERAND
COC	2000	3	N	2	COMPARE ONES CORRESPONDING
CZC	2400	3	N	2	COMPARE ZEROES CORRESPONDING
DEC	0600	6	Y	0-4	DECREMENT (BY ONE)
DECT	0640	6	Y	0-4	DECREMENT (BY TWO)
DIV	3C00	9	N	4	DIVIDE
IDLE	0340	7	N	—	COMPUTER IDLE
INC	0580	6	Y	0-4	INCREMENT (BY ONE)
INCT	05C0	6	Y	0-4	INCREMENT (BY TWO)
INV	0540	6	Y	0-2	INVERT (ONES COMPLEMENT)
JEQ	1300	2	N	—	JUMP EQUAL (ST2 = 1)

INSTRUCTIONS BY MNEMONIC

JGT	1500	2	N	—	JUMP GREATER THAN (ST1 = 1)
JH	1B00	2	N	—	JUMP HIGH (STO = 1 AND ST2 = 0)
JHE	1400	2	N	—	JUMP HIGH OR EQUAL (STO OR ST2 = 1)
JL	1A00	2	N	—	JUMP LOW (STO AND ST2 = 0)
JLE	1200	2	N	—	JUMP LOW OR EQUAL (STO = 0 OR ST2 = 1)
JLT	1100	2	N	—	JUMP LESS THAN (ST1 AND ST2 = 0)
JMP	1000	2	N	—	JUMP UNCONDITIONAL
JNC	1700	2	N	—	JUMP NO CARRY (ST3 = 0)
JNE	1600	2	N	—	JUMP NOT EQUAL (ST2 = 0)
JNO	1900	2	N	—	JUMP NO OVERFLOW (ST4 = 0)
JOC	1800	2	N	—	JUMP ON CARRY (ST3 = 1)
JOP	1C00	2	N	—	JUMP ODD PARITY (ST5 = 1)
LDCR	3000	4	Y	0-2,5	LOAD CRU
LI	0200	8	N	0-2	LOAD IMMEDIATE
LIMI	0300	8	N	12-15	LOAD IMMEDIATE TO INTERRUPT MASK
LREX	03E0	7	N	12-15	EXTERNAL CONTROL
LWPI	02E0	8	N	—	LOAD IMMEDIATE TO WORKSPACE POINTER
MOV	C000	1	Y	0-2	MOVE (WORD)
MOVB	D000	1	Y	0-2,5	MOVE (BYTE)
MPY	3800	9	N	—	MULTIPLY
NEG	0500	6	Y	0-4	NEGATE (TWO'S COMPLEMENT)
ORI	0260	8	Y	0-2	OR IMMEDIATE
RSET	0360	7	N	12-15	EXTERNAL CONTROL
RTWP	0380	7	N	0-6,12-15	RETURN WORKSPACE POINTER
S	6000	1	Y	0-4	SUBTRACT (WORD)
SB	7000	1	Y	0-5	SUBTRACT (BYTE)
SBO	1D00	2	N	—	SET CRU BIT TO ONE
SBZ	1E00	2	N	—	SET CRU BIT TO ZERO
SETO	0700	6	N	—	SET ONES
SLA	0A00	5	Y	0-4	SHIFT LEFT (ZERO FILL)
SOC	E000	1	Y	0-2	SET ONES CORRESPONDING (WORD)
SOCB	F000	1	Y	0-2,5	SET ONES CORRESPONDING (BYTE)
SRA	0800	5	Y	0-3	SHIFT RIGHT (MSB EXTENDED)
SRC	0800	5	Y	0-3	SHIFT RIGHT CIRCULAR
SRL	0900	5	Y	0-3	SHIFT RIGHT (LEADING ZERO FILL)
STCR	3400	4	Y	0-2,5	STORE FROM CRU
STST	02C0	8	N	—	STORE STATUS REGISTER
STWP	02A0	8	N	—	STORE WORKSPACE POINTER
SWPB	06C0	6	N	—	SWAP BYTES
SZC	4000	1	Y	0-2	SET ZEROES CORRESPONDING (WORD)
SZCB	5000	1	Y	0-2,5	SET ZEREOS CORRESPONDING (BYTE)
TB	1F00	2	N	2	TEST CRU BIT
X	0480	6	N	—	EXECUTE
XOP	2C00	9	N	6	EXTENDED OPERATION
XOR	2800	3	Y	0-2	EXCLUSIVE OR
DCA	2C00	9	N	0-3,5,7	DECIMAL CORRECT ADD
DCS	2C00	9	N	0-3,5,7	DECIMAL CORRECT SUB
LIIM	2C00	9	N	14,15	LOAD INTERRUPT MASK

ILLEGAL OP CODES 0000-01FF;0320-033F;0780-07FF;0C00-OFFF

INSTRUCTIONS BY OP CODE

OP CODE	MNEMONIC	OP CODE	MNEMONIC
0000-01FF	ILLEGAL	1000	JMP
0200	LI	1100	JLT
0220	AI	1200	JLE
0240	ANDI	1300	JEQ
0260	ORI	1400	JHE
0280	LI	1500	JGT
0240	STWP	1600	JNE
02C0	STST	1700	JNC
02E0	LWPI	1800	JOC
0300	LIMI	1900	JND
0320-033F	ILLEGAL	1A00	JL
0340	IDLE	1B00	JH
0360	RSET	1C00	JOP
0380	RTWP	1D00	SBO
03A0	CKON	1E00	SBZ
03C0	CKOF	1F00	TB
03E0	LREX	2000	COC
0400	BWLP	2400	CZC
0440	B	2800	XOR
0480	X	2C00	XOP
04C0	CLR	3000	LDCR
0500	NEG	3400	STCR
0540	INV	3800	MPY
0580	INC	3C00	DIV
05C0	INCT	4000	SZC
0600	DEC	5000	SZCB
0640	DECT	6000	S
0680	BL	7000	SB
06C0	SWPB	8000	C
0700	SETO	9000	CB
0740	ABS	A000	A
0780-07FF	ILLEGAL	B000	AB
0800	SRA	C000	MOV
0900	SRL	D000	MOVB
0A00	SLA	E000	SOC
0B00	SRC	F000	SOCB
0C00	ILLEGAL		

7

PSEUDO-INSTRUCTIONS

MNEMONIC	PSEUDO-INSTRUCTIONS	CODE GENERATED
NOP	NO OPERATION	1000
RT	RETURN	0458

PIN DESCRIPTIONS

PIN #	FUNCTION	PIN #	FUNCTION	PIN #	FUNCTION
1	V_{BB}	23	A1	44	D3
2	V_{CC}	24	A0	45	D4
3	WAIT	25	$\phi 4$	46	D5
4	$\overline{\text{LOAD}}$	26	V_{SS}	47	D6
5	HOLDA	27	V_{DD}	48	D7
6	$\overline{\text{RESET}}$	28	$\phi 3$	49	D8
7	IAQ	29	DBIN	50	D9
8	$\phi 1$	30	CRUOUT	51	D10
9	$\phi 2$	31	CRUIN	52	D11
10	A14	32	$\overline{\text{INTREQ}}$	53	D12
11	A13	33	IC3	54	D13
12	A12	34	IC2	55	D14
13	A11	35	IC1	56	D15
14	A10	36	IC0	57	NC
15	A9	37	NC	58	NC
16	A8	38	NC	59	NC
17	A7	39	NC	60	CRUCLK
18	A6	40	NC	61	$\overline{\text{WE}}$
19	A5	41	D0	62	READY
20	A4	42	D1	63	$\overline{\text{MEMEN}}$
21	A3	43	D2	64	$\overline{\text{HOLD}}$
22	A2				

ASSEMBLER DIRECTIVES

MNEMONIC	DIRECTIVE
AORG	ABSOLUTE ORIGIN
BES	BLOCK ENDING WITH SYMBOL
BSS	BLOCK STARTING WITH SYMBOL
BYTE	INITIALIZE BYTE
DATA	INITIALIZE WORD
▶7 DEF	EXTERNAL DEFINITION
DORG	DUMMY ORIGIN
DXOP	DEFINE EXTENDED OPERATION
END	PROGRAM END
EQU	DEFINITE ASSEMBLY — TIME CONSTANT
EVEN	WORD BOUNDARY
IDT	PROGRAM IDENTIFIER
LIST	LIST SOURCE
PAGE	PAGE EJECT
REF	EXTERNAL REFERENCE
RORG	RELOCATABLE ORIGIN
TEXT	INITIALIZE TEXT
TITL	PAGE TITLE
UNL	NO SOURCE LIST

USASCII/HOLLERITH CHARACTER CODE

CHAR.	USASCII (HEXADECIMAL)	HOLLERITH*	CHAR.	USASCII (HEXADECIMAL)	HOLLERITH*
NUL	00		3	33	3
SOH	01		4	34	4
STX	02		5	35	5
ETX	03		6	36	6
EOT	04		7	37	7
ENQ	05		8	38	8
ACK	06		9	39	9
BEL	07		:	3A	2-8
BS	08		;	3B	11-6-8
HT	09		<	3C	12-4-8
LF	0A		=	3D	6-8
VT	0B		>	3E	0-6-8
FF	0C		?	3F	0-7-8
CR	0D		@	40	4-8
S0	0E		A/a	41/61	12-1
SI	0F		B/b	42/62	12-2
DLE	10		C/c	43/63	12-3
DC1	11		D/d	44/64	12-4
DC2	12		E/e	45/64	12-5
DC3	13		F/f	46/66	12-6
DC4	14		G/g	47/67	12-7
NAK	15		H/h	48/68	12-8
SYN	16		I/i	49/69	12-9
ETB	17		J/j	4A/6A	11-1
CAN	18		K/k	4B/6B	11-2
EM	19		L/l	4C/6C	11-3
SUB	1A		M/m	4D/6D	11-4
ESC	1B		N/n	4E/6E	11-5
FS	1C		O/o	4F/6F	11-6
GS	1D		P/p	50/70	11-7
RS	1E		Q/q	51/71	11-8
US	1F		R/r	52/72	11-9
SPACE	20	BLANK	S/s	53/73	0-2
!	21	11-2-8	T/t	54/74	0-3
"	22	7-8	U/u	55/75	0-4
#	23	3-8	V/v	56/76	0-5
$	24	11-3-8	W/w	57/77	0-6
%	25	0-4-8	X/x	58/78	0-7
&	26	12	Y/y	59/79	0-8
'	27	5-8	Z/z	5A/7A	0-9
(	28	12-5-8	[	5B	12-2-8
)	29	11-5-8	\	5C	
*	2A	11-4-8	]	5D	12-7-8
+	2B	12-6-8	^	5E	11-7-8
,	2C	0-3-8	—	SF	0-5-8
-	2D	11	`	60	
.	2E	12-3-8	{	7B	
/	2F	0-1	>	7C	
0	30	0	}	7D	
1	31	1	~	7E	
2	32	2	DEL	7F	

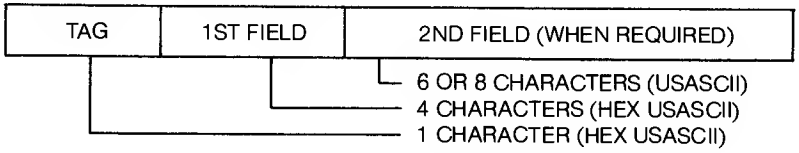
*PUNCH IN CARD ROWS

7

HEX-DECIMAL TABLE

EVEN BYTE				ODD BYTE			
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,766	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,066	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15

OBJECT RECORD FORMAT AND CODE



TAG	FIRST FIELD	SECOND FIELD	MEANING
0	LENGTH OF ALL RELOCATABLE CODE	PROGRAM ID (8-CHARACTER)	PROGRAM START
1	ADDRESS	(NOT USED)	ABSOLUTE ENTRY ADDRESS
2	ADDRESS	(NOT USED)	RELOCATABLE ENTRY ADDRESS
3	LOCATION OF LAST APPEARANCE OF SYMBOL	6 CHARACTER SYMBOL	EXTERNAL REFERENCE LAST USED IN RELOCATABLE CODE
4	LOCATION OF LAST APPEARANCE OF SYMBOL	6 CHARACTER SYMBOL	EXTERNAL REFERENCE LAST USED IN ABSOLUTE CODE
5	LOCATION	6 CHARACTER SYMBOL	RELOCATABLE EXTERNAL DEFINITION
6	LOCATION	6 CHARACTER SYMBOL	ABSOLUTE EXTERNAL DEFINITION
7	CHECKSUM FOR CURRENT RECORD	(NOT USED)	CHECKSUM
8	ANY VALUE	(NOT USED)	IGNORE CHECKSUM VALUE
9	LOAD ADDRESS	(NOT USED)	ABSOLUTE LOAD ADDRESS
A	LOAD SDDRESS	(NOT USED)	RELOCATABLE LOAD ADDRESS
B	DATA	(NOT USED)	ABSOLUTE DATA
C	DATA	(NOT USED)	RELOCATABLE DATA
D	LOAD BIAS	(NOT USED)	LOAD BIAS OR OFFSET (NOT A PART OF ASSEMBLER OUTPUT)
E			ILLEGAL
F	(NOT USED)	(NOT USED)	END OF RECORD

TM990/402 Line-by-Line Assembler User's Guide

7

GENERAL

The TM 990/402 Line-By-Line Assembler (LBLA) is a standalone program that assembles into object code the 69 instructions used by the TM 990/100M/101M/180M microcomputers. Comments can be a part of the source statement; however, assembler directives are not recognized. Assembler TM 990/402-1 consists of two EPROM's and supports the TM 990/100M microcomputer. TM 990/402-2 consists of one EPROM and supports the TM 990/180M microcomputer.

INSTALLATION

Remove the TMS 2708 chip(s) from the package and install as follows (see *Figure 1*):

(1) Turn off power to the TM 990/1XXM microcomputer.

(2) Place the chip(s) into the proper socket(s) as shown in *Figure 1.* The shaded components in *Figure 1* denote the LBLA EPROM's correctly placed in their sockets. The corresponding socket number (UXX number) is marked on the EPROM.

NOTES

1. Place the TMS 2708(s) into the socket(s) with pin 1 in the lower left corner as denoted by a 1 on the board and on the EPROM. Be careful to prevent bending of the pins.
2. Do not remove EPROM's containing the monitor as shown in *Figure 1.* The monitor is used by the assembler.

(3) Verify proper positioning in the sockets. Apply power to the microcomputer board.

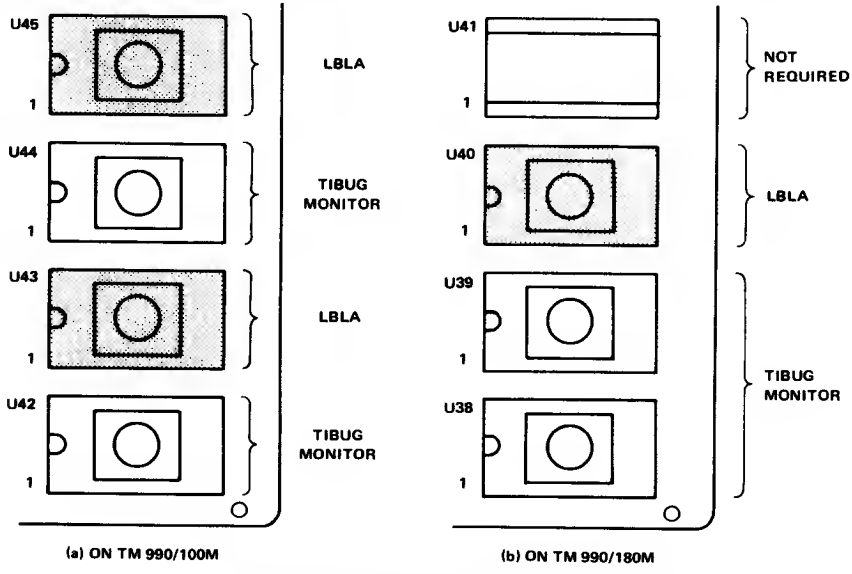


Figure 1. Placement of TMS 2708 Eprom's

OPERATION

Setup

NOTE
The examples in this guide use memory addresses obtainable in RAM on the TM 990/100M microcomputer. To exemplify the TM 990/180M addressing scheme, the reader should substitute a 3 for the F in the most significant digit (left most) of a four-digit memory address in the following examples (e.g., $3EE0_{16}$ for $FEE0_{16}$).

- With the Line-By-Line Assembler EPROMs installed, call up the monitor by pressing the RESET switch in the upper left corner of the board and then pressing the A key at the terminal.
- Invoke the R keyboard command and set the Program Counter (PC) to $09E6_{16}$. This is the memory address entry point for the Line-By-Line Assembler.
- Invoke the E (execute) command. The assembler will execute and print the memory address (M.A.) $FE00_{16}$ for the TM 990/100 or $3E00_{16}$ for the TM 990/180M. The printhead will space to the assembly language opcode input column and wait for input from the keyboard.

```
?R
W=0BA4
P=000F        9E6 ←———— LBLA ENTRY ADDRESS
?E
FE00
```

Inputs To Assembler

The Line-By-Line Assembler accepts assembly language inputs from a terminal. As each instruction is input, the assembler interprets it, places the resulting machine code in an absolute address, and prints the machine code (in hexadecimal) next to its absolute address:

```
MEMORY ADDRESS OF ASSEMBLED MACHINE CODE
MACHINE CODE ASSEMBLED BY ASSEMBLER
INSTRUCTION MNEMONIC
ONE SPACE (MAXIMUM)
OPERANDS
AT LEAST ONE SPACE (MINIMUM)
COMMENTS

FE00  02E0  LWPI >FE80          SET UP WORKSPACE ADDRESS
FE02  FE80
FE04  0200  LI R0,10            SET UP COUNTER VALUE
FE06  000A
FE08  0201  LI R1,>FEA0         ADDRESS OF VALUES IN R1
FE0A  FEA0
FE0C  0202  LI R2,>FEB0         ADDRESS OF STORAGE AREA IN R2
FE0E  FEB0
FE10  CCB1  MOV ◆R1+,◆R2+       MOVE VALUES TO STORAGE AREA
FE12  0600  DEC R0              DECREMENT COUNTER
FE14  1301  JEQ >FE18           EXIT IF COUNTER = ZERO
FE16  10FC  JMP >FE10           LOOP BACK UNTIL 10 VALUES MOVED
FE18
```

Use only one space between the mnemonic and the operand. If you use the comment field, use at least one space between the operand and comment. If no comment is used, complete the instruction with a space and carriage return. If a comment is used, only a carriage return is required.

No loader tags are created; code is loaded in contiguous memory addresses by the assembler. The location can be changed as desired (explained in paragraph 3.2.2).

Labels cannot be used. Addressing is by byte displacement (jump instructions) or by absolute memory address.

NOTE

Be aware that the workspace for the TIBUG monitor begins in RAM at address $FFB0_{16}$ for the TM 990/100M and begins at address $3FB0_{16}$ for the TM 990/180M. Understand that assembled object code should not be entered at or above these addresses.

Program Preparation

Set up your program using flow charts with code written on a coding pad. Do not use assembler directives.

Changing Absolute Load Address

Code is located at the address written on the assembler output. When initialized, the assembler loads code contiguously starting at M.A. $FE00_{16}$ ($3E00_{16}$ for TM 990/180M). This address can be changed at any time during assembly by typing a slash (/) followed by the desired M.A.:

```
FE80   8081   C R1,R2              COMPARE VALUES
FE82   1301   JEQ >FE86            IF EQUAL, SKIP ERROR ROUTINE
FE84   06A0   8L @>FF20            OTHERWISE DO ERROR ROUTINE
FE86   FF20
FE88          /FF20        <------ CHANGE ADDRESS
FF20   2FA0   XOP @>FF26,14        SEND ERROR MESSAGE (See TIBUG Monitor)
FF22   FF26
FF24   0458   B ◆R11               RETURN TO CALLING PROGRAM
FF26   0A0D   +>0A0D
FF28   4552   $ERROR FOUND
FF2A   524F
FF2C   5220
FF2E   464F
FF30   554E
FF32   4420
FF34   0000   +0000
FF36          /FE86        <------ CHANGE ADDRESS
FE86
```

Note that this is similar to using an AORG (absolute origin) 990 assembler directive.

Entering Instructions

Any of the 69 instructions applicable to the TM 990/1XXM microcomputers can be interpreted by the Line-By-Line Assembler. The following apply:

(1) Place one space between instruction mnemonic and operand.

(2) Terminate entire instruction with a space and a carriage return. Lines with comments need only a carriage return. Character strings require two carriage returns.

(3) Do not use labels; addressing is through byte displacement (jump instructions) or absolute addresses:

```
FE8C   1607   JNE   $+16
FC8E   10E8   JMP   >FE60
FE90   C8A2   MDV   @>FD20(R2), @>FE10(R2)
FE92   FD20
FE94   FE10
FE96
```

(4) Register numbers are in decimal and can be predefined (preceded by an R):

```
FE96   020C   LI 12,>D00
FE98   0D00
FE9A   020D   LI R13,>FFFF
FE9C   FFFF
FE9E
```

(5) Jump instruction operand can be $+n, $−n, or >M where n is a decimal value of bytes (+256≥n≥−254) and M is a memory address in hexadecimal. The dollar sign must be followed by a sign and number (JMP $ is not allowed).

```
FE20   1304   JEQ   $+10          EXIT
FE22   1304   JEQ   $+>A          EXIT
FE24   1304   JEQ   $+%1010       EXIT
FE26   1304   JEQ   >FE30         EXIT
FE28   10FF   JMP   $+0           LDDP AT THIS ADDRESS [>FE28]
FE2A   10FF   JMP   $−0           LODP AT THIS ADDRESS
```

7◀

(6) Absolute numerical values can be in binary, decimal, or hexadecimal.

- Binary values are preceded by a percent sign (%). One to 16 ones and zeroes can follow; unspecified bits on the left will be zero filled:

```
FE58   0204   LI R4,%10101010     >AA IN R4
FE5A   00AA
FE5C   000A   +%1010              DATA STATEMENT
FE5E   FFF6   −%1010              DATA STATEMENT
FE60
```

- Decimal values have no prefix in an operand:

```
FE6C   0205   LI R5,100        LOAD COUNTER
FE6E   0064
FE70   0206   LI R6,32768      SET LIMIT
FE72   8000
FE74   8000   +32768
FE76   8000   -32768
FE78   7FFF   +32767
FE7A   8001   -32767
FE7C   FFFF   -1
FE7E
```

- Hexadecimal values are preceded by the greater-than sign (>):

```
FE7E   02E0   LWPI>FF00        SET WP ADDRESS
FE80   FF00
FE82   FFFF   +>FFFF           DATA STATEMENT
FE84   0001   +>FFFF           DATA STATEMENT
FE86
```

NOTE

In operands, absolute value must be unsigned values only. However, there is a method for using the assembler to compute and assemble a negative value; this method is especially useful with the immediate instructions (e.g., AI, CI, LI). Enter the instruction using the negative value. The assembled value will be all zeroes in the last assembled word. Use the slash command (paragraph 3.2.2) to assemble at the previous address, then enter the negative value as a data statement as shown in the following example:

```
FE1A   0201   LI R1,->100      <-- USE SIGNED OPERAND
FE1C   0000                    <-- SIGNED NUMBER ASSEMBLIES AS 0000 (IN M.A.>FE1C)
FE1E          /FE1C            <-- SET OBJECT LOAD ADDRESS TO PREVIOUS ADDRESS
FE1C   FF00   ->100            <-- ->100(>FF00) NOW IN M.A.>FE1C
FE1E
```

▶7

(7) Absolute addresses are used instead of labels:

```
FEA0   C820   MOV   @>FE10,@>FED0      MOVE TO STORAGE
FEA2   FE10
FEA4   FE00
FEA6   16FC   JNE   >FEA0              LOOP BACK TO MOVE INSTRUCTION
FEA8
```

(8) Character strings are preceded by a dollar sign and are terminated with two carriage returns.

```
FF10   4142   $ABCD        1233
FF12   4344
FF14   2020
FF16   2031
FF1B   3233
FF1A   3320                      ← UNUSED RIGHT BYTE FILLED WITH>20 (SPACE)
```

(9) Character strings of one or two characters can be designated by encoding the string in quotes. If not part of an operand, a plus or minus sign must precede the value. If the string is larger than two characters, the last two characters are interpreted.

```
FEAA   3132   +'12'        CHARACTERS DNE AND TWD
FEAC   000C   +12          VALUE OF POSITIVE TWELVE
FEAE   FFF4   -12          VALUE OF NEGATIVE TWELVE
FEB0   0000   +            + FDLLOWED BY CTRL KEY AND NULL KEY PRESSED
FEB2   0202   LIR2, 'ABCD' ASSEMBLED LAST TWO CHARACTERS (C AND D)
FEB4   4344
FEB6   0202   LI R2, 'E'   CHARACTER E IN RIGHT BYTE
FEBB   0045
FEBA   0202   LI R2,>E     VALUE >E IN RIGHT BYTE
FEBC   000E
FEBE
```

(10) Signed numerical values of up to 16 bits can be designated by preceding the value with a plus or minus sign. If more than 16 bits are entered in binary or hexadecimal, the last 16 bits entered are used. If more than 16 bits are entered in decimal, the assembled value is the same as the remainder had the number between divided by 2^{15} ($65{,}536_{10}$).

```
FE1B   00FF   +%1111111100000001 1111111
FE1A   FF01   -%1111111100000001 1111111
FE1C   AAEE   +>AAAAAAEE
FE1E   8000   +32768
FE20   8001   +32769
FE22   0000   +65536
FE24   FFFF   +131071
FE26   0000   +131072
FE28   8000   -32768
FE2A   8001   -32767
FE2C   7FFF   -32769
FE2E
```

7◀

Errors

When the assembler detects an error, it types an error symbol and readies the terminal for re-entering data at the same memory address. The following error symbols are used:

- D (Displacement error). The jump instruction destination is more than +256 or −254 bytes away.

```
FF38          JNC    $+300◆D
FF38          JNC    >F000◆D
FF38    1708  JNC    >FF50
FF3A
```

- R (Range error). The operand is out of range for its field:

```
FF30          LI     R44,◆R
FE30    0204  LI     R4,200
FF32    00C8
```

- S (Syntax error). The instruction syntax was incorrect:

```
FF34          MOZ◆S  } INCORRECT MNEMONICS
FF34          MOS◆S  }
FF34    C802  MOV R2, @>FE90
FF36    FE90
```

EXITING TO THE MONITOR

Return control to monitor by pressing the escape (ESC) key.

PSEUDO-INSTRUCTIONS

▶7 The TM 990/402 also interprets two pseudo-instructions. These pseudo-instructions are not additional instructions but actually are additional mnemonics that conveniently represent two members of the instruction set:

- The NOP mnemonic can be used in place of a JMP $+2 instruction which is essentially a no-op (no operation). This can be used to replace an existing instruction in memory, or it can be included in code to force additional execution time in a routine. Both NOP and JMP $+2 assemble to the machine code 1000_{16}.
- The RT mnemonic can be used in place of a B *R11 instruction which is a common return from a branch and link (BL) subroutine. Both RT and B *R11 assemble to the machine code $045B_{16}$.

Note the following examples:

```
FE00 1000 JMP $+2        JUMP TO NEXT INSTRUCTION
FE02 1000 NOP            ALSO ASSEMBLES TO >1000
FE04 045B B ◆R11         RETURN COMMAND
FE06 0458 RT             ALSO A RETURN COMMAND
```

TIBUG Monitor

7

TIBUG COMMANDS

INPUT	RESULTS
B	Execute under Breakpoint
C	CRU Inspect/Change
D	Dump Memory to Cassette/Paper Tape
E	Execute
F	Find Word/Byte in Memory
H	Hex Arithmetic
L	Load Memory from Cassette/Paper Tape
M	Memory Inspect/Change
R	Inspect/Change User WP, PC, and ST Registers
S	Execute in Step Mode
T	1200 Baud Terminal
W	Inspect/Change Current User Workspace

COMMAND SYNTAX CONVENTIONS

CONVENTION SYMBOL	EXPLANATION
<>	Items to be supplied by the user. The term within the angle brackets is a generic term.
[ ]	Optional Item — May be included or omitted at the user's discretion. Items not included in brackets are required.
{ }	One of several optional items must be chosen.
(CR)	Carriage Return
Λ	Space Bar
LF	Line Feed
R or Rn	Register (n = 0 to 15)
WP	Current User Workspace Pointer contents
PC	Current User Program Counter contents
ST	Current User Status Register contents

USER ACCESSIBLE UTILITIES

XOP	FUNCTION
8	Write 1 Hexadecimal Charter to Terminal
9	Read Hexadecimal Word from Terminal
10	Write 4 Hexadecimal Characters to Terminal
11	Echo Character
12	Write 1 Character to Terminal
13	Read 1 Character from Terminal
14	Write Message to Terminal
	NOTE All characters are in ASCII code.

TIBUG ERROR MESSAGES

ERROR	CONDITION
0	Invalid tag detected by the loader.
1	Checksum error detected by the loader.
2	Invalid termination character detected.
3	Null input field detected by the dump routine.
4	Invalid command entered.

7◀

COMMAND	SYNTAX
Execute under Breakpoint (B)	B<address><(CR)>
CRU Inspect/Change (C)	C<base address>{^,}<count><(CR)>
Dump Memory to Cassette/Paper Tape (D)	D<start address>{^,}<stop address>{^,}<entry address>{^,}IDT = <name><^> (MONITOR PROMPT)
Execute Command (E)	E
Find Command (F)	F<start address>{^,}<stop address>{^,}<value>{(CR)}
Hexadecimal Arithmetic (H)	H<number 1>{^,}<number 2><(CR)>
Load Memory from Cassette or Paper Tape (L)	L<bias><(CR)>
Memory Inspect/Change, Memory Dump (M)	Memory Inspect/Change Syntax M<address><(CR)> Memory Dump Syntax M<start address>{^,}<stop address><(CR)>
Inspect/Change User WP,PC, and ST Registers (R)	R<(CR)>
Execute In Single Step Mode (S)	S
TI 733 ASR Baud Rate (T)	T
Inspect/Change User Workspace (W)	W [Register Number] <(CR)>

▸7

TM 990/302
Software Development Board

7

EPROM's which may be programmed by the '302

2708
2716
2516
2532
9940

SOFTWARE COMPONENTS

	Access Command
Executive	(CR)
Text Editor	TE
Symbolic Assembler	SA
Debug Package	DP
EPROM Programmer	EP
Relocating Loader	RL
EIA Interface	EI
I/O Scheduler/Handler	SR

LUNO ASSIGNMENTS

Device	Logical Unit No.
Dummy	0
Terminal (LOG)	1
Audio Cassette 1	2
Audio Cassette 2	3
Second EIA Connector	4
Memory	5

SOFTWARE COMPONENT CALLS

Text Editor	TEØ(input device),(output device)
Symbolic Assembler	SAØ(source device), (object device), (listing device)
Debug Package	DPØ(output device)
EPROM Programmer	EP
Relocating Loader	RLØ(input device)
Set Baud Rate	SRØ(nnnn)
Escape	ESC (return to executive)

TEXT EDITOR COMMANDS

D	Delete lines n thru m
I	Insert at line n with optional auto increment by m
K	Keep buffer and print new top line in the buffer
G	Get buffer and print new bottom line in the buffer
P	Print lines n thru m
Q	Flush the input file until end of input file and return to executive
R	Resequence input to output, n is initialized line # and m is the increment

COMMAND	SYNTAX
Delete Lines n thru m (Rn,m)	D (starting line #)[,(ending line #)]
Insert After Line n with optional auto increment by m (In,m)	I (line number after which new data is entered) [,(auto increment value)]
Get Buffer (G)	G
Keep Buffer (K)	K
Print lines n thru m (Pn,m)	P (first line # to be printed) [,(last line # to be printed)]
Quit Text Editor (Q)	Q
Resequence Output (Rn,m)	R (initial line number) [,(increment value)]

7◀

ASSEMBLER DIRECTIVES

AORG	[label]øAORGø(value)ø[comment]
BSS	[label]øBSSø(value)ø[comment]
BYTE	[label]øBYTEø(value),(value),(value),....ø[comment]
DXOP	[label]øDXOPø(symbol),(value)ø[comment]
END	[label]øENDø(symbol)ø[comment]
EQU	[label]øEQUø(expression)ø[comment]
DATA	[label]øDATAø(exp),(exp),...ø[comment]
EVEN	[label]øEVENø[comment]
IDT	[label]øIDTb(string)ø[comment]
TEXT	[label]øTEXTø(–),'string'ø[comment)

DEBUG Package

Verb	Command
SB	Set Software Breakpoint and Execute
IM	Inspect/Change Memory
IC	Inspect/Change CRU
IR	Inspect/Change MPU Registers
ST	Set Software Trace
RU	Single Step for 1 or more instructions with or without trace
DM	Dump Memory

▸7

DEBUG COMMANDS

Set Breakpoint and Execute	SBø(address)
Inspect/Change Memory	IMø(address)
Inspect/Change CRU	ICø(CRU base addr.)(no. of bits)
Inspect/Change MPU registers	IR
Set Software Trace	STø(0 or 1)
Run 1 or more Instructions	RUø(no. of instructions in decimal)
Dump Memory	DMø(starting addr.),(ending addr.)

EPROM PROGRAMMING CRU ASSIGNMENTS

CRU BASE ADDRESS$_{16}$	INPUT/OUTPUT	FUNCTION
1710	I/O	EPROM DATA BIT 0
1712	I/O	:
1714	I/O	:
1716	I/O	:
1718	I/O	:
171A	I/O	:
171C	I/O	:
171E	I/O	EPROM DATA BIT 7
1720	O	EPROM ADDRESS LSB
1722	O	:
1724	O	:
1726	O	:
1728	O	:
172A	O	:
172C	O	:
172E	O	:
1730	O	:
1732	O	:
1734	O	:
1736	O	:
1738	O	EPROM ADDRESS MSB
173A	O	EPROM PROGRAM ENABLE
173E	O	EPROM PROGRAMMING PULSE

7◂

EPROM PROGRAMMING RESPONSES

PP = Program EPROM
RE = Read EPROM to Memory
CE = Compare EPROM to Memory
Memory Bounds: MEM BDS? (start addr.),(stop addr.)
EPROM Start addr: EPROM START? (start addr.)
Programming Mode: MODE? P(parallel) or I(in line)
Starting Byte: ST byte ? (0 or 1 if P above)

PREDEFINED CRU ADDRESSES FOR I/O DEVICES

Device	CRU Address
Users Terminal (9902)	80_{16}
Timer (9901)	100_{16}
EIA Interface (9902)	180_{16}
Recorder 1 Forward	1700_{16}
Recorder 2 Forward/9940 Flag 1	1702_{16}
Recorder 2 Write Data/9940 Flag 2	1704_{16}
Recorder 1 Read Data/9940 Flag 3	1706_{16}
Personality Card Code Bit 0	1708_{16}
Personality Card Code Bit 1	$170A_{16}$
Personality Card Code Bit 2	$170C_{16}$
Switch Code Bit	$170E_{16}$
EPROM Data	1710_{16} — $171E_{16}$
EPROM Address	1720_{16} — 1738_{16}
EPROM Program Enable	$173A_{16}$
EPROM Programming Pulse	$173C_{16}$

TXDS Commands for FS 990 Software Development System

7

Examples of manuals available in support of the TXDS System:

TXDS PROGRAMMER'S GUIDE (#946258-9701)

This manual enables the user to employ the Terminal Executive Development System (TXDS) in conjunction with the TX990 Operating System and the Model 990/4 and 990/10 Computer System hardware configuration to develop, improve, change, or maintain (1) the user's customized Operating System and the user's applications programs or (2) any other type of user-produced programs (e.g., the user's own supervisor call processors or the user's own utility programs). It is assumed the reader is familiar with the Model 990 Computer System assembly language and the concepts of the TX990 Operating System.

The sections and appendixes of this manual are organized as follows:

I Introduction — Provides a general description of the TXDS utility programs and their capabilities. Also includes a description of the control functions of the TXDS Control Program.

II Loading and Executing a Program — Provides a step-by-step procedure for loading and executing (1) each of the TXDS and TX990 Operating System utility programs and (2) a user program. Also describes the TXDS Control Program and how to correctly respond to its prompts.

III Verification of Operation — Provides several short step-by-step procedures to checkout proper operation of the TXDS software.

IV Creating and Editing Program Source Code — Describes the capabilities of the TXEDIT utility program and how the user can employ those capabilities to edit or generate the text of source programs and object programs.

V Assembling Source Programs — Describes how the user can employ the TXMIRA utility program to assemble source files (i.e., source code programs).

7 VI TX990 Cross Reference (TXXREF) Utility Program — Describes how the user can employ the TXXREF utility program to produce a listing of each user-defined symbol in a 990 assembly source program along with the line numbers on which the symbol is defined and all of the line numbers on which the symbol is referenced.

VII Linking Object Modules — Describes how the user can employ the TXDS Linker utility program to form a single object module from a set of independently assembled object modules (in the form of object code or compressed object code.)

VIII TXDS Copy Concatenate (TXCCAT) Utility Program — Describes how the user can employ the TXCCAT utility program to copy one to three files to a single output file.

IX TXDS Standalone Debug Monitor (TXDBUG) Utility Program — Describes how the user can employ the TXDBUG utility program to debug programs which have been designed to operate in a "standalone" situation without support of an operating system.

X TXDS PROM (TXPROM) Programmer Utility Program — Describes how the user can employ the TXPROM programming utility program to control the Programming Module (PROM) hardware to make customized ROMs containing user-created data or programs.

XI TXDS BNPF/High Low (BNPFHL) Dump Utility Program — Describes how the user can employ the BNPFHL utility program to produce a BNPF or high/low file format.

XII TXDS IBM Diskette Conversion Utility (IBMUTL) Program — Describes how the user can employ the IBMUTL utility program to transfer standard IBM-formatted diskette datasets to TX990 Operating System files and to transfer TX990 Operating System files to standard IBM-formatted diskette datasets.

XIII TXDS Assign and Release LUNO Utility Program — Describes how the operator can assign and release LUNOs in systems which do not include OCP.

A Glossary — Clarifies selected words used in this TX990 Operating System Programmer's Guide.

B Compressed Object Code Format — Describes the compressed object code format.

C Task State Codes — Lists and describes the task state codes.

D I/O Error Codes — List and describes the I/O error codes available to the user, when coding a program, for printout or display on a terminal device.

The following documents contain additional information related to the TX990 Operating System and are referenced herein this manual:

TITLE	PART NUMBER
Model 990 Computer TX990 Operating System Programmer's Guide	946259-9701
Model 990 Computer TMS9900 Microprocessor Assembly Language Programmer's Guide	943441-9701
Model 990 Computer Model FD800 Floppy Disc System Installation and Operation	945253-9701
Model 990 Computer Model 913 CRT Display Terminal Installation and Operation	943457-9701
Model 990 Computer Model 911 Video Display Terminal Installation and Operation	943423-9701
Model 990 Computer Model 733 ASR/KSR Data Terminal Installation and Operation	945259-9701
Model 990 Computer Model 804 Card Reader Installation and Operation	945262-9701
Model 990 Computer Models 306 and 588 Line Printers Installation and Operation	945261-9701
Model 990 Computer PROM Programming Module Installation and Operation	945258-9701
990 Computer Family Systems Handbook	945250-9701
Model 990 Computer Communications Systems Installation and Operation	945409-9701

List of Commands and Special Keys/Characters

COMMAND SYNTAX	DESCRIPTION
	SETUP COMMANDS
SL	Start Line Numbers (SL) command causes line numbers to be printed with each line of text.
SN	Stop Line Numbers (SN) comman causes line numbers not to be printed.
SP	Set Print Margin (SP) command sets the right boundary for print display.
SM	Set Margin (SM) for Find command sets the left and right boundaries for the Find command.
ST	Set Tabs (ST) command sets up to five tab stops.
	PRINTER-MOVEMENT COMMANDS
D	Down (D) command moves the pointer down toward the bottom of the buffer.
U	Up (U) command moves the pointer up towards the first line in the buffer.
T	Top (T) command moves the pointer to the first line in the buffer.
B	Bottom (B) command moves the pointer to the last line in the buffer.
	EDIT COMMANDS
▸7 C	Change (C) command removes lines from the buffer and inserts new ones in their place. The new lines are input from the terminal.
I	Insert (I) command takes input from the terminal and places the new lines into the buffer.
M	Move (M) command moves lines from one place in the buffer to another.
R	Remove (R) command deletes lines from the buffer.
F	Find string (F) command searches for the first occurrence of a character string in a line and replaces it with another string of characters.
	PRINT COMMANDS
L	Limits (L) command causes the first line and the last line to be displayed.
P	Print (P) command displays lines of text.

List of Commands and Special Keys/Characters (Continued)

COMMAND SYNTAX	DESCRIPTION
	OUTPUT COMMANDS
K	Keep (K) command takes lines of text out of the buffer and puts them in the output file.
Q	Quit (Q) command takes lines of text out of the buffer or the input files and puts them in the output file.
E	An (E) command terminates without writing an EOF to the output file.
	TERMINATE-SEQUENCE COMMANDS
T or C	Allows the user to make multiple single directional editing passes on a source or object program.
	SPECIAL KEYS/CHARACTERS
CTRL-H	Pressing the control key and the H key simultaneously on the hard copy terminal causes the terminal to backspace a character to enable rewriting over an entered character-error.
RUB OUT	The RUB OUT key causes the line just entered to be deleted so that a new line can replace it.
CTRL-I	Pressing the control (CTRL) key and the I key simultaneously on a hard-copy terminal causes a tab stop to be entered in the input string, although only one space will be echoed on the terminal.
ESC/RESET	Pressing the ESCape or RESET key on the system console causes a display to be aborted.
position keys	When using a VDT, only the left position key (←) and the right (→) position key are recognized. The up and down position keys cause garbage to be entered into the input string. The left position key causes characters to be deleted from the character string; a right position key causes whatever was under the cursor to be entered.
DELETE LINE	DELETE LINE on a VDT acts the same as a RUB OUT on a hardcopy terminal.
TAB	A SPACE character is echoed. The TAB is interpreted by the text editor and spaces are inserted to fill the text line to the next TAB setting.

7◂

TXMiRA Options

OPTION	DESCRIPTION
Mnnnnn	Overrides memory size default; default is 2400 bytes
X	Produce cross-reference
L	Produce assembly listing
T	Expand TEXT code on listing
S	Produce sorted symbol list
C	Produce compressed object output where n is a decimal digit

TXLINK Options

OPTION	DESCRIPTION
Mnnnnn	Override default memory size, default is 11800 bytes.
C	Compressed object output.
Iaaaaaaaa	IDT for linked object.
P	Partial link desired.
L	Print load map and symbol list.

Note: n is a decimal digit and a is an alphanumeric character.

TXCCAT Options

7

OPTION	DESCRIPTION
TRnnnn	Truncate record to length nnnn.
FLnnnn	Fix records to size nnnn by padding with blanks or by truncation.
SKnnnn	Skip nnnn input records, prior to output.
LFnn	List file, page length = nn, default = 55.
SLnn	Space lines on listing, nn = space count, default = 0.
NL	Number lines on listing.
RI	Do not rewind input on open.
RO	Do not rewind output on open.

Note: n is a decimal digit and the maximum field size is given by the number of n's.

TXDBUG Keyboard Commands

DEBUG Commands

IC	Inspect Communications Register Unit (CRU)
IM	Inspect Memory
IR	Inspect AU Register (WP, PC, ST)
IS	Inspect Snapshot
IW	Inspect Workspace Registers
MC	Modify Communications Register Unit (CRU)
MM	Modify Memory
MR	Modify Registers
MW	Modify Workspace Registers
SB	Set Breakpoint
SP	Set H/W Write Protect Option
SR	Set Trace Region
SS	Set Snapshot
ST	Set Trace
CB	Clear Breakpoint
CP	Clear H/W Write Protect Option
CR	Clear Trace Region
CS	Clear Snapshot

AMPL
Reference Data

▸7

EXPLANATION OF THE NOTATION USED IN THIS CARD

	Notation	Explanation
Optional Items	[item]	Bracketed item may be omitted.
	{item 1 / item 2}	Exactly one item must be selected. from the items in braces.
Substitution	expr 'file'	Any expression may be used. File or device name required.
Repetition	item . . .	A list of items may be used.
Required	<item>	Replace with item.

CHARACTER SET

Type	Characters	Use
Special	RETURN SPACE ! " $ / ( ) * + , — . / : ; < = > ? @	Any printable character may be used in a quoted string. RETURN terminates line and statement. ";" may separate statements. SPACE separates adjacent numbers and identifiers.
Numerals	0 — 9	
Letters	A — Z,a — z	

NOTE: All AMPL reserved words use only upper case (UPPER CASE LOCK).

SYMBOL NAMES

Type	Example	Definition
System	RO ETRC	Up to four alphanumeric characters; all system symbols are predefined.
User-defined	USRVAR X3 BRKADR GO	Up to six alphanumeric characters; assignment defines a variable. ARRAY statement defines an array. PROC/FUNC statement defines a procedure/function.
Program label	IDT. .DEF	Up to six alphanumeric characters. Period after IDT and before DEF labels, defined by LOAD command.

CONSTANTS

Type	Example	Range
Decimal	10833	1 . . . 32767
Hexadecimal	02A51, >2A51	>0 . . . >FFFF
Octal	125121	!0 . . . !177777
Binary	<10101001010001	<0 . . . <1111111111111111
ASCII	"*Q"	
Instruction	#XOR *R1,R9#	
Keyword	IAQ	See keyword constant table.

EXPRESSIONS

Type	Example	Definition
Subexpression	(expr)	
Identity	+ expr	Value of <expr>.
Negation	− expr	Two's complement of <expr>.
Target memory	@addr	<addr> used as word address into emulator or target memory.
Proc/Func Argument	ARG expr	Argument in position <expr> of call list; ARG 0 is number of arguments in list.
Proc/Func local variable	LOC expr	Word <expr> of local variable array; LOC 0 is length of local variable array.
Multiplication	expr1*expr2	Signed product (warning on overflow).
Division	expr1/expr2	Signed quotient (warning on divide by zero).
Remainder	expr1 MOD EXPR2	Signed remainder of division (warning on divide by zero).
Addition	expr1 + expr2	Signed sum.
Subtraction	expr1 - expr2	Signed difference.
NOTE: Result of relational operator is either FALSE (0) or TRUE (-1).		
Equality	expr1 EQ expr2 expr1 NE expr2	16-bit comparison.
Arithmetic inequality	expr1 LT expr2 expr1 LE expr2 expr1 GT expr2 expr1 GE expr2	Signed, 16-bit comparison.
Logical inequality	expr1 LO expr2 expr1 LOE expr2 expr1 HI expr2 expr1 HIE expr2	Unsigned, 16-bit comparison.
Complement	NOT expr	16-bit one's complement.
Conjunction	expr1 AND expr2 expr 1 NAND expr2	16-bit boolean AND. 16-bit boolean not AND.
Disjunction	expr1 OR expr2 expr1 XOR expr2	16-bit boolean OR. 16-bit boolean exclusive OR.

NOTE: Operators are given in order of precedence, highest to lowest. Solid lines separate precedence groups; within each group, precedence is equal and evaluation is left to right. Evaluation results in a 16-bit integer value.

UNSIGNED ARITHMETIC

Syntax	Definition
MPY (expr1, expr2)	Low-order 16 bits of unsigned product. <expr1>* <expr2>; high order 16 in MDR.
DIV (divisor, dividend)	Unsigned quotient of 32-bit number (MDR, <dividend>) over <divisor>; remainder in MDR.
MDR	High-order 16-bits of MPY product and of DIV dividend; remainder of DIV; unsigned carry of + and-.

ARRAY DEFINITION

ARRAY name(expr1[,expr2]), . . .	User <name> (previously undefined or name of deleted array) is defined as one- or two-dimension array.

DISPLAY STATEMENTS

expr[:f . . . f]	Value of expression
'LITERAL STRING'	Literal string
add1 [TO addr2] [:f . . . f] ? [:f . . . f]	Target memory

Format specification/[:f . . . f]

ASCII	A	set default	G	octal	O[i]
binary	B[i]	hexadecimal	H[i]	symbolic	S
decimal	D[i]	instruction	I	unsigned	U[i]
name =	E	newline	N[j]	space	X[j]

Note:	1< =i< =9	field width 'i' digits, then two blanks
	i=0	default field width, no trailing blanks
	1< =j< =9	repeat 'j' times
	j=0	repeat 10 times

Response to display/modify mode(?):

forward step	RETURN, +	replace contents	<expr>
back step	—	open new address	@<addr>
exit	;	change display	:f . . . f

DISASSEMBLER

Instruction	DST	Destination address.
operands	SRC	Source address.

NOTE: Additional instructions of the TMS9940 (DCA, DCS, LIIM, SM) will assemble correctly (#DCA *RC1#) but will disassemble as XOP instructions. See TMS9940 specifications for details.

7◀

ASSIGNMENT STATEMENTS

Type	Example	Definition
Variable	sym = expr	User-defined or writable system symbol or REF program label.
Target memory	@addr = expr	Put value of <expr> at target <addr>
Proc/Func argument	ARG n = expr	Local copy of argument in position <n> of call list.
Command local	LOC n = expr	Word <n> of local storage array.
Array	A[(i1[,i2])] = e	User defined array name; zero, one, or two index expressions.

NOTE: Precedence of @, ARG, and LOC may require parenthesis around following expression.

COMPOUND STATEMENTS

Syntax	Definition
BEGIN statements END	Statements are executed sequentially. Use in place of any single statement syntax.

CONTROL STATEMENTS

IF expr THEN s1 [ELSE s2]	<s1> is executed if <expr> is TRUE (nonzero). Otherwise, <s2> is executed, if included.
CASE expr OF expr 1::s1; . . . . exprn::sn [ELSE s] END	Statement <si> at first label expression <expr> equal to <expr> is executed. If none, statement <s> is executed, if included.
WHILE expr DO statement	While <expr> is TRUE (nonzero), <statement> is executed.
REPEAT statement UNTIL expr	<statement> is executed. If <expr> FALSE (zero), <statement> is executed until <expr> is TRUE.
FOR var = expr1 TO expr 2[BY expr3] DO statement	Value of <expr1> is assigned to <var>. <statement> is executed until <var> is equal to <expr2>; <expr3> is added to <var>, and <statement> repeated. Default value of <exp3> is 1.
ESCAPE	Exit from innermost enclosing WHILE, REPEAT, or FOR statement.

PROCEDURE/FUNCTION/FORM DEFINITION

PROC name [(args[,locs])] statements END FUNC name [(args[,locs])] statements END	User-defined <name> (previously undefined or deleted procedure/function) is bound to <statements>. <args> is the required number of arguments. <locs> is the size of local storage array.
RETURN [expr]	Pass control back to calling statement. In a procedure, <expr> is ignored. In a function, value of <expr> replaces the function call in the calling expression.
FORM name 'prompt' [= [{ constant / 'string' }]]; . . .	END <name> must be a previously defined procedure or function, semicolon required between prompts.

PROCEDURE/FUNCTION CALLS

proc name [(expr, . . .)]	User-defined or system procedure/function with list of argument expressions.
func name [(expr, . . .)]	Command definition determines number of arguments required. Some system commands require quoted strings as arguments.

NOTE: Procedure/functions with defined FORM when called with no arguments will prompt for arguments using the FORM.

example FORM:

COMMENTARY ENTRY	comment, not a prompt required argument, with default value required argument, must enter value default given if value not entered
PROMPT 1 = default value PROMPT 2 = PROMPT 3* =	

FORM control function keys:

Next prompt:	TAB,↓,→FIELD, SKIP, RETURN
Previous prompt:	↓,←FIELD
First prompt:	HOME
Erase value:	ERASE FIELD, ERASE INPUT
Redisplay default:	INSERT LINE
Duplicate previous value:	F4
Complete form:	ENTER
Abort form:	CMD

7◀

INPUT/OUTPUT COMMANDS

Syntax	**Definition**
HCRB	Host computer CRU base address.
HCRR (offset,width)	Read host computer CRU field.
HCRW (offset,width,value)	Write <value> into host CRU field.
COPY ('file' | edit id)	AMPL input from 'file' AMPL input from edit buffer
LIST ('file' | OFF | ON | EOF)	Initialize listing device or file. Disable listing output. Enable listing output. Close listing device or file with EOF.
NL	Print newline.

```
unit = OPEN [ ( ('file' | edit id) [ , (0 | IN | OUT | IO) [ , (0 | REWIND | EXTEND) [ , (SEQ | REL) ] ] ] ) ]
```

no arguments — list all open units and edit buffers.
initialize 'file'/<edit id> I/O unit
- 0 — device IO, file IN only
- IN — for input only
- OUT — for output only
- IO — for input/output
- REWIND — position to beginning of file
- EXTEND — position to end of file
- SEQ — auto-create sequential file
- REL — auto-create rel-rec file

▶7

```
event-READ [ (unit [ , (0 | DIRECT) [ , (0 | GRAPH)
    [ , (VDT , (0 | f row) [ , (0 | f col) [ , s col] ] | SEQ | REL [, rec # ]) ] ] ] ) ]
```

no arguments — read console
Read record from (unit)
- 0 — issue read ASCII
- DIRECT — issue read direct
- GRAPH — read graphics on 922 VDT
- VDT — read in cursor positioning mode
  - f row — field start row
  - f col — field start column
  - s col — cursor start column

INPUT/OUTPUT COMMANDS (continued)

SEQ — read sequentially
REL — read sepecified record
rec # — record number to read
<event>/256 = cursor column after read if VDT
<event> AND 255 = event key value if VDT,
else >OD for end of record,
>13 for end of file.

value = EVAL [(unit)] — Evaluate expression in <unit>'s buffer;
if no <unit>, READ/EVAL the console.

DPLY [(unit)] — AMPL display unit for output to <unit>;
if no <unit>, to console.

okay = MOVE (from unit, to unit) — Move contents of <from unit>'s buffer to <to unit>'s buffer
<okay> = 0 if moved
= >FFFF if too big and not moved.

REW[(unit)] — Rewind (unit) — repositions, file clears console
no argument — clears console

$$\text{Cursor} = \text{WRIT}\left(\text{unit}\left[,\left\{\begin{matrix}0\\ \text{DIRECT}\end{matrix}\right\}\left[,\left\{\begin{matrix}0\\ \text{GRAPH}\end{matrix}\right\}\right]\right.\right.$$
$$\left.\left.\left[,\left\{\begin{matrix}\text{VDT} & \left\{\begin{matrix}0\\ \text{f row}\end{matrix}\right\} & [,\ [\text{f col}]]\\ \text{SEQ} & & \\ \text{REL} & [,\ \text{rec \#}\ ] & \end{matrix}\right\}\right]\right]\right]\right)$$

no arguments — write console
Write record to (unit),
0 — issue write ASCII
DIRECT — issue write direct
GRAPH — write graphics on 911 VDT
VDT — write in cursor positioning mode
f row — field start row
f col — field start column
SEQ — write sequentially
REL — read specified record
rec # — record number to read
<cursor>/256 = cursor column after write if VDT

$$\text{CLSE (unit}\left[,\left\{\begin{matrix}\text{EOF}\\ \text{UNLOAD}\end{matrix}\right\}\right]$$

Release I/O <unit>,
EOF — write end-of-file mark
UNLOAD — unload unit

SYSTEM SYMBOLS

V — variable F — function P — procedure

CLR	P — clear	MDEL	P — symbols
CLSE	P — I/O close	MDR	V — arithmetic
COPY	P — copy	MIN	V — minutes
CRUB	V — CRU base	MOVE	F — I/O buffer
CRUR	F — CRU read	MPY	F — multiply
CRUW	P — CRU write	MSYM	P — symbols
DAY	V — day	NL	P — newline
DBUF	P — delete buffer	OPEN	F — I/O open
DELE	P — delete symbol	PC	V — registers
DIV	F — divide	R0-R15	V — registers
DPLY	P — display	READ	F — I/O read
DR	P — registers	REW	P — I/O rewind
DST	V — destination	RSTR	P — restore
DUMP	P — dump	SAVE	P — save
EBRK	P — emulator	SEC	V — seconds
ECLK	V — emulator	SRC	V — source
EDIT	F — edit	ST	V — register
EHLT	F — emulator	TBRK	P — trace module
EINT	P — emulator	TEVT	P — trace module
EMEM	V — emulator	THLT	F — trace module
ERUN	P — emulator	TINT	P — trace module
EST	F — emulator	TNCE	V — trace module
ETB	F — emulator	TNE	V — trace module
ETBH	F — emulator	TRUN	P — trace module
ETBO	V — emulator	TST	F — trace module
▸7 ETRC	P — emulator	TTB	F — trace module
ETYP	V — emulator	TTBH	F — trace module
EVAL	F — evaluate	TTBN	V — trace module
EXIT	P — exit AMPL	TTBO	V — trace module
HCRB	V — host CRU	TTRC	P — trace module
HCRR	F — CRU read	USYM	P — user symbols
HCRW	P — CRU write	VRFY	P — verify
HR	V — hour	WAIT	F — delay AMPL
IOR1	V — I/O	WP	V + register
KEEP	P — keep edit	WRIT	P — I/O write
LIST	P — list	YR	V — year
LOAD	P — load object		

EDIT

Syntax	Definition
edit id = EDIT[({'file' / edit id} [,record])]	Create edit buffer with 'file'. Edit existing buffer. No argument creates an empty buffer.
KEEP (edit id, 'file')	Save edit buffer onto 'file' and delete edit buffer.
DBUF (edit id)	Delete edit buffer.

EDIT CONTROL FUNCTION KEYS

Function	911 KEY	913 KEY	CONTROL CHARACTER
edit/compose mode	F7	F7	V
quit edit mode	CMD	HELP	X
roll up	F1	F1	A
roll down	F2	F2	B
set tab	F3	F3	C
clear tab	F4	F4	D
tab	TAB (shift SKIP)	TAB	I
back tab	FIELD	BACK TAB	T
newline	RETURN	NEWLINE	RETURN
insert line	unlabeled gray	INSERT LINE	O
delete line	ERASE INPUT	DELETE LINE	N
erase line	ERASE FIELD	CLEAR	W
truncate line	SKIP	SET	K
insert character	INS CHAR	INSERT CHAR	
delete character	DEL CHAR	DELETE CHAR	
cursor up	↓	↓	U
cursor down	↓	↓	J
cursor right	→	→	R
cursor left	←	←	H
top of screen	HOME	HOME	

7

GENERAL COMMANDS

Syntax	Definition
USYM	List all user symbols, procedures, functions, and arrays.
DELE ('name'....)	Delete user procedure, function, or array.
SAVE ('file')	Save all user defined symbols, functions, and arrays on 'file'.
RSTR ('file')	Restore user defined symbols, procedures, functions, and arrays from 'file'.
CLR	Delete all user symbols, procedures, functions and arrays.
MSYM	List object program labels.
MDEL	Delete all object program labels.
EXIT	Exit from AMPL back to operating system.

TIMING

YR	Year (1976 to 1999)
DAY	Julian day (1 to 366)
HR	Hour (0 to 23)
MIN	Minute (0 to 59)
SEC	Second (0 to 59)
WAIT (expr)	Suspend AMPL for <expr>*50 milliseconds (<expr> = 20 is one second).

TARGET MEMORY COMMANDS

EMEM	Emulator memory mapping: 9900/9980 map 8K bytes (0->1FFF) 9940 define RAM and ROM sizes.
LOAD ('file'[,bias[,IDT] [+DEF] [+REF]]]):	Load object program by bias and enter program labels into table.
VRFY ('file' [,bias])	Verify object program, listing differences between object and target memory.
DUMP ('file',low,high[,start])	Dump program from target <low> to <high> in nonrelocatable format.

▶7

EMULATOR CONTROL COMMANDS

Syntax	Definition
EINT ('EM0n' [,{1/0}[,'TM0n']])	Initialize Emulator device, clock 0 = prototype/ 1 = emulator.
ECLK	Processor clock.
ETYP	Processor type: -1 = TMS9940, 0 = SBP9900, 1 = TMS9900, 2 = TMS9980.
ETRC ({MA/IAQX/IAQ} [,count[,low,high]])	Trace qualifier, completion break count (OFF-255), address range.
EBRK ({MA/IAQ/MR/MW} [+ ILLA] [,address]...)	Address breakpoint(s) (ILLA only valid for TMS9940).
ERUN	Run emulation at PC, WP, ST.
EST	Emulation status (3 LSBits): HOLD, IDLE, Running
EHLT	Halt emulation, return status.
ETBH (index[,{MR/MW/IAQ}])	Indexed bus signal from buffer. (TRUE if expression matches).
ETB (index)	Indexed address from trace buffer.
ETBO, ETBN	Emulator Trace buffer limits: Oldest, Newest sample indices.

7◀

TRACE MODULE CONTROL

Syntax	Definition
TINT ('TM0n')	Initialize trace module.
TTRC ([INT] { OFF / [±Q0] [±Q1][±Q2][±Q3] / [±IAQ][±DBIN] } [,count[, { ON / OFF }]])	Qualify data samples, trace completion counter (OFF-255), latch option on D0-D3.
TEVT ({ OFF / [±D0] [±D1] [±D2] [±D3] / [±IAQ] [±DBIN) / EXT } [,value[,mask]])	Qualify D0-D3 event (or EXTernal), <value> and <mask> for D4-D19.
TBRK (count [,<delay>[,INV] [+EDGE]]])	Set event counter (OFF-FFFF), set delay counter (OFF-244), count INVerted/EDGE events.
TRUN	Start Trace module tracing.
TST	Trace module status (3 LSB's), event occurred, trace full, tracing.
THLT	Halt trace module, return status.
TNE	Number of events since last TRUN.
TNCE	Number of event count overflows.
TTBH (index[, { [±D0] [±D1][±D2] [±D3] / [±IAQ][±DBIN] }])	D0-D3 of indexed samples, (TRUE if expression matches).
TTB (<index>)	D4-D19 indexed samples (data bus)
▶7 TTBO, TTBN	Trace module trace buffer limits: Oldest, Newest sample indices.

TRACE MODULE INTERCONNECT TO EMULATOR

Q0	Memory address bit 15 (TMS9940 only).
D0	Byte memory cycle (TMS9940 only).
Q1,D1, IAQ	Instruction Acquisition.
Q2,D2,DBIN	DataBusIN = MR(read), MW = -DBIN(write).
Q3	Emulator trace qualifier and range (ETRC).
D3, External Event	Emulator address breakpoint (EBRK).
D4-D19	Emulator data bus (bits 0-15).
External Clock	Emulator memory cycle clock.
Control Cable	Synchronizes emulation and tracing. Trace module will halt emulator for EINT ('EM0n', clock 'TM0n').

TARGET REGISTERS

PC,WP,ST	Processor registers.
R0-R15	Workspace registers.
DR	Display all registers.

CRU READ/WRITE

CRUB	CRU interface base address.
CRUR (offset,width)	Read target CRU field.
CRUW (offset,width,value);	Write <value> into target CRU field

KEYWORDS

ARG	FORM	THEN	GE
ARRAY	FUNC	TO	GT
BEGIN	IF	UNTIL	HI
BY	LOC	WHILE	HIE
CASE	MOD	AND	LE
DO	NULL	NAND	LO
ELSE	OF	OR	LOE
END	PROC	XOR	LT
ESCAPE	REPEAT	NOT	NE
FOR	RETURN	EQ	

KEYWORD CONSTANTS

D0	EXT	IO	Q2
D1	EXTEND	MA	Q3
D2	GRAPH	MR	REF
D3	IAQ	MW	REL
DBIN	IAQX	N	REWIND
DEF	IDT	OFF	SEQ
DIRECT	ILLA	ON	UNLOAD
EDGE	IN	OUT	VDT
EOF	INT	Q0	Y
ETBN	INV	Q1	

ERROR MESSAGES

0 — ! UNDEFINED ERROR CODE !
1 — I/O ERROR, OS ERROR CODE RETURNED
2 — INSUFFICIENT MEMORY TO CONTINUE
3 — ! SEGMENT VIOLATION !
4 — I/O ERROR: INVALID UNIT ID
5 — I/O ERROR: READ/WRITE VIOLATION
6 — I/O ERROR: INSUFFICIENT MEMORY FOR OPEN
7 — ! DELETE UNIT CONTROL BLOCKS ERROR !
8 — TOO MANY IDT DEF/REF SYMBOLS IN LOAD
9 — EXCEEDED 15 LOAD OPERATIONS SINCE LAST CLR
10 — CANNOT ALLOCATE MEMORY FOR USER SYMBOL TABLE
11 — ! ERROR IN I/O UNIT CHAIN POINTERS !
12 — OVERLAY ERROR
101 — VARIABLE CANNOT BE READ
102 — VARIABLE CANNOT BE WRITTEN
103 — SYMBOL IS UNDEFINED
104 — ! INVALID CODEGEN BRANCH TABLE INDEX !
105 — INSUFFICIENT MEMORY TO COMPILE STATEMENT
106 — SYMBOL IS DEFINED; CANNOT BE REDEFINED
107 — INSUFFICIENT MEMORY TO COMPILE PROC/FUNC
108 — INPUT RECORD CANNOT BE CLASSIFIED
109 — INPUT STRING EXCEEDS MAXIMUM ALLOWED LENGTH
110 — ! INVALID SCANNER BRANCH TABLE INDEX !
▶7 111 — UNRECOGNIZABLE INPUT ITEM
112 — ! UNDEFINED OPERATOR !
114 — SYMBOL NOT AN IDT/DEF/REF LOAD SYMBOL
115 — USER SYMBOL TABLE FULL
116 — CONSTANT EXCEEDS 16 BITS
117 — SYNTAX ERROR
118 — ! INVALID KEYWORD STRING LENGTH !
119 — SYNTAX ERROR IN ONE-LINE-ASSEMBLY STATEMENT
120 — INCORRECT NUMBER OF ARRAY SUBSCRIPTS
121 — ESCAPE SPECIFIED OUTSIDE A LOOP CONSTRUCT
122 — ARRAY REDEFINED WITH INCORRECT SUBSCRIPTS

NOTE: A hexadecimal number is also printed with some error messages. Refer to the AMPL System Operation Guide for complete explanation.

ERROR MESSAGES

201 — SYMBOL NOT FOUND TO DELETE
202 — SYMBOL CANNOT BE DELETED
203 — INVALID DISPLAY FORMAT CHARACTER FOLLOWING:
204 — NO LIST DEVICE ASSIGNED
205 — EMULATOR I/O ERROR CODE RETURNED
209 — INVALID INDEX INTO EMULATOR TRACE BUFFER
210 — !CANNOT ALLOCATE FORM CURRENT VALUE SEGMENT!
211 — INSUFFICIENT MEMORY TO SAVE FORM PARAMETERS
214 — INVALID RESTORE FILE
215 — INSUFFICIENT MEMORY TO COMPLETE THE RESTORE
216 — BAD TRACE OR COMPARISON MODE SELECTED
219 — TRACE MODULE I/O ERROR CODE RETURNED
220 — CANNOT EDIT ON THIS DEVICE TYPE
221 — TRACE INTERFACE CHANGE ILLEGAL WHILE TRACING
222 — INVALID INDEX INTO TRACE MODULE BUFFER
223 — INSUFFICIENT ARGUMENTS IN PROC/FUNC CALL
224 — STACK OVERFLOW; DELETE PROC/FUNC/ARRAY
225 — DELETED PROC/FUNC/ARRAY REFERENCED
226 — INSUFFICIENT ARGUMENTS IN FORM FOR PROC/FUNC
227 — ! INVALID FORM SEGMENT ID !
228 — ! INVALID FORM CURRENT VALUE SEGMENT ID !
229 — INVALID CHARACTER IN LOAD FILE
230 — CHECKSUM ERROR IN LOAD FILE
231 — ARITHMETIC OVERFLOW
233 — PROC/FUNC CALL ARGUMENT OUT OF RANGE
234 — INVALID "ARG" OR "LOC" INDEX FOR WRITING
235 — INVALID "ARG" OR "LOC" INDEX FOR READING
237 — ARRAY ALREADY DEFINED
238 — INVALID ARRAY DIMENSION
240 — REFERENCE TO UNDECLARED ARRAY
241 — INVALID ARRAY SUBSCRIPT
242 — ! ERROR ARRAY SEGMENT LENGTH !
243 — DELETED IDT/DEF/REF LOAD SYMBOL REFERENCED
244 — ALL IDT/DEF/REF LOAD SYMBOLS DELETED
245 — INVALID DEVICE TYPE TO "EINT" OR "TINT"

NOTE: Error messages withing exclamation marks (!) are AMPL internal system errors. Contact Texas Instruments if problem persists.

POWER BASIC
MP 307

▶7

REFERENCE CARD FOR DEVELOPMENT AND EVALUATION BASIC

This card contains a summary of all POWER BASIC† statements and commands for Development and Evaluation BASIC. An explanation preceded by an asterisk (*) indicates the statement or command is not supported by Evaluation BASIC. A * indicates the statement is supported only by the Development BASIC software enhancement package.

COMMANDS

CONtinue

*Execution continues from last break.

LIST

LIST the user's POWER BASIC program. In LIST will list from specified line number through end of program or until ESC key is struck.

LOAD

Reads a previously recorded POWER BASIC program from an auxiliary device or configures POWER BASIC to execute a BASIC program in EPROM.
LOAD reads program from 733ASR digital cassette.
LOAD 1 or LOAD 2 * reads program from audio cassette drive No. 1 or No. 2.
LOAD <address>* configures POWER BASIC to execute BASIC program in EPROM at specified address.

NEW

Prepare for entry of NEW POWER BASIC program or set the lower RAM memory bound after auto-sizing.
NEW clears pointers of POWER BASIC and prepares for entry of new program.
NEW <address>* sets the lower RAM memory bound used by POWER BASIC after auto-sizing or power-up.

PROGRAM

Program current POWER BASIC application program into EPROM.*

RUN

Begin program execution at the lowest line number.

SAVEn (n is interpreted as in LOADn command)

Record current user program on auxiliary device.

SIZE

Display current program size, variable space allocated, and available memory in bytes.

†Trademark of Texas Instruments

EDITING

The phrase "(ctrl)" indicates that the user holds down the control key while depressing the key corresponding to the character immediately following.

(CR)	Enter edited line.
(ctrl)In	*Insert n blanks.
(ctrl)Dn	*Delete n characters.
(ctrl)H	Backspace one character.
(ctrl)F	Forward space one character.
In(ctrl)E	Display for editing source line indicated by line number (In).
(ctrl)T	Toggle from one partition to the other partition (only in Evaluation BASIC).
(esc)	Cancel input line or break program execution.
(Rubout) or (DEL)	Backspace and delete character.

STATEMENTS

InBAUD <exp 1,> <exp 2>

*sets baud rate of serial I/O port(s).

InBASE <(exp)>

Sets CRU base address for subsequent CRU operations

InCALL Name <subroutine address>[, <var 1>, <var 2>, <var 3>, <var 4>]

*Transfers to external subroutines. If variable is contained in parentheses, the address will be passed; otherwise, the value will be passed.

InDATA {<exp> | <string const>} [, {<exp> | <string const>}] . . .

defines internal data block.

▶7

In DEF FN<x>[(<arg 1> [, arg 2] [, arg 3])] = <exp>

*Defines user arithmetic function.

InDIM <var (dim[, dim] . . .)> [, ...]

Allocates user variable space for dimensioned or array variables.

InEND

Terminates program execution and returns to edit mode.

In ERROR<In>

*Specifies a subroutine that will be called via a GOSUB statement when an error occurs.

In ESCAPE
InNOESC

*Enables or disables the excape key to interrupt program execution (always enabled in Evaluation BASIC).

lnFOR <sim-var> = <exp> TO <exp> [STEP <exp>]
lnNEXT <sim-var>

Open and close program loop. Both identify the same control variable. FOR assigns starting, ending, and optionally stepping values.

lnGOSUB<ln>

Transfer of control to an internal subroutine beginning at the specified line.

lnPOP

*Removal of most previous return address from GOSUB stack without an execution transfer.

lnRETURN

Return from internal subroutine.

lnGOTO<ln>

Transfers program execution to specified line number.

lnIF<exp>THEN<statement>
lnELSE<statement>

Causes conditional execution of the statement following THEN. *ELSE statements execute when IF condition is false.

lnIMASK<LEVEL>

*Set interrupt mask of TMS 9900 processor to specified level.

lnTRAP<level>TO<ln>

*Assign interrupt level to interrupt subroutine.

lnIRTN

*Return from BASIC interrupt service routine.

lnINPUT<var> $\left[\left\{\begin{matrix}\cdot\\;\end{matrix}\right\}\text{<var>}\right]\cdots\left[\left\{\begin{matrix}\cdot\\;\end{matrix}\right\}\right]$

Accesses numeric constants and strings from the keyboard into variables in the INPUT list.

ln [LET] <var> = <exp>

Evaluates and assigns values to variables or array elements.

lnON $\left\{\begin{matrix}\text{<var>}\\\text{<exp>}\end{matrix}\right\}$ THEN GOTO ln [,ln] . . .

lnON $\left\{\begin{matrix}\text{<var>}\\\text{<exp>}\end{matrix}\right\}$ THEN GOSUB ln [,ln] . . .

*Transfers execution to the line number specified by the expression or variable.

lnPRINT <exp> [,exp] . . .

Print (format free) the evaluated expressions.

lnRANDOM [exp]

*Set the seed to the specified expression value.

lnREAD $\left\{\begin{matrix}\text{<numeric var>}\\\text{<string var>}\end{matrix}\right\}\left[,\left\{\begin{matrix}\text{<numeric var>}\\\text{<string var>}\end{matrix}\right\}\right]\cdots$

Assigns values from the internal data list to variables or array elements.

7

InREM [text]

Inserts comments.

InRESTOR [exp]

Without an argument, resets pointer to beginning of data sequence; with an argument, resets pointer to line number specified.

InSTOP

Terminates program execution and returns to Edit mode.

InTIME

Sets, displays, or stores the 24 hour time of day clock.

InTIME <exp>, <exp>, <exp>
Sets and starts clock.
InTIME <string-var>
Enables storing clock time into a string variable.
InTIME
Prints clock time as HR:MN:SD.

InUNIT <exp>

*Designates device(s) to receive all printed output.

FUNCTIONS

ABS <(exp)>	*Absolute value of expression.
ASC <(string-var)>	*Returns decimal ASCII code for first character of string variable.
ATN <(exp)>	Arctangent of expression in radians.
BIT <(var, exp)> BIT <(var, exp 1)> = <exp 2>	*Reads or modifies any bit within a variable. Returns a 1 if bit is set and 0 if not set. Selected bit is set to 1 if assigned value is non-zero and to zero if the assigned value is zero.
▶7 COS >(exp)>	Cosine of the expression in radians.
CRB <(exp)>	Reads CRU bit as selected by CRU base + exp. Exp is valid for −127 thru 128.
CRB <(exp 1)> = <(exp 2)>	Sets or resets CRU bit as selected by CRU base + exp 1. If exp 2 is non-zero, the bit will be set, else reset. Exp 1 is valid for −127 thru 128.
CRF <(exp)>	Reads n CRU bits as selected by CRU base where exp evaluates to n. Exp is valid for 0 thru 15. If exp = 0, 16 bits will be read.
CRF <(exp 1)> = <(exp 2)>	Stores exp 1 bits of exp 2 to CRU lines as selected by CRU BASE. Exp 1 if valid for 0 thru 15. If exp 1 = 0, 16 bits will be stored.
EXP <(exp)>	*Raise the constant e to the power of the evaluated expression.
INP <(exp)>	Returns the signed integer portion of the expression.

LOG <(exp)>	*Returns natural logarithm of the expression.
MEM <(exp)>	Reads byte from user memory at address specified by exp. Exp must be in the integer range, (0 to 65535).
MEM <(exp 1)> = <(exp 2)>	Stores byte exp 2 into user memory specified by exp 1. Exp 1 and exp 2 must be in the integer range.
MCH <(string 1), (string 2)>	*Returns the number of characters to which the two strings agree.
NYK <(exp)>	Conditionally samples the keyboard in run time mode. If exp <>0, return decimal value of last key struck and clear key register. (0 if no key struck.) If exp = 0, return a 1 if the last key struck has the same decimal value as the expression. Clear key register if TRUE, else return 0 if FALSE.
RND	Returns a random number between 0 and 1.
SIN <(exp)>	Sine of the expression in radians.
SQR <(exp)>	Square root of expression.
SRH <(string 1), (string 2)>	*Return the position of string 1 in string 2, 0 if not found.
SYS <(exp)>	*Obtains system parameters generated during program execution. Example: SYS(0) = INPUT control character, SYS(1) = Error code number, SYS(2) = error line number.
TIC <(exp)>	Returns the number of time tics less the expression value. One TIC equals 40 milliseconds (1/25 second).

STRINGS

ASCII Character Conversion Function	ASC (string-var) *Convert first character of string to ASCII numeric representation.
Assignment	<string-var> = {<string-var> / <string-constant>} Store string into string-var ending with a null.
Character Match Function	MCH (<string 1>, <string 2>) *Return the number of characters to which the 2 strings agree.
Character Search Function	SRH (<string 1>, <string 2>) *Return the position of string 1 in string 2. Zero is returned if not found.
Concatenate	<string-var> = {<string-var> / <string-constant>} + {<string-var> / <string-constant>} [+ {...}]

7◀

Convert to ASCII	<string-var> = <exp> <string-var> = # <string>, <exp> *Convert exp to ASCII characters ending with a null. # string specifies a formatted conversion.
Convert to Binary	<var 1> = <string>, <var 2> *Convert string into binary equivalent. Var 2 receives the delimiting non-numeric character in first byte.
Deletion	<String-var> = /<exp> *Delete exp characters from string-var.
Insertion	<string-var> = /<string> *Pick byte into string-var.
Pick	<string-var> = {<string-var> / <string-constant>}, <exp> Pick number of characters specified by exp into string-var ending with a null.
Replace	<string-var> = {<string-var> / <string-constant>}; <exp> Replace number of characters specified by exp of string-var with string.
String Length Function	<var> = LEN <(string-var)> <var> = LEN "string" *Return the length of string.

INPUT OPTIONS

string-var	Prompt with colon and input character data. Example: INPUT $A
▶7 ,	Delimit expressions. Example A, B
;	Suppress prompting or CR LF if at end of line. Examples: INPUT ;A INPUT A;
# exp	Allow a maximum of exp characters to be entered. Example: INPUT # 1"Y or N"$1
%exp	*Must enter exactly exp number of characters. Example: INPUT %4"CODE"C
?<ln>	*Upon an invalid input or entry of a control character, a GOSUB is performed to the line # . SYS(0) will be equal to − 1 if there was an invalid input. Otherwise, SYS(0) will equal the decimal equivalent of the control character. Example: INPUT ?100;A

OUTPUT OPTIONS

;	Delimit expressions or suppress CR LF if at end of line. Examples: PRINT A;B PRINT A;
,	Tab to next print field. Example: PRINT A, B
TAB <(exp)>	Tab to exp column. Example: PRINT TAB (50);A
string	Print string or string-var. Example: PRINT "HI";$A(0)
# exp	*Print exp as hexadecimal in free format. Example: PRINT # 123
# ,exp	*Print exp as hexadecimal in byte format. Example: PRINT # ,50
# ;exp	*Print exp as hexadecimal in word format. Example: PRINT # ,A
<hex value>	*Direct output of ASCII codes. Example: PRINT "<OD> <OA>"
# string	*Print under specified format where: PRINT # "9999"I 9 = digit holder PRINT # "000-00-0000"SS 0 = digit holder or force 0 PRINT # "$$$,$$$.00"DLR $ = digit holder and floats $ PRINT # "SSS.0000"4*ATN1 S = digit holder and floats sign PRINT # "<<<.00>"I < = digit holder and float on negative >number PRINT # "990.99E"N E = sign holder after decimal PRINT # "990.99"N . = decimal point specifier PRINT # "999,990.99"N , = suppressed if before significant digit PRINT # "999,990∧00"I ∧ = translates to decimal point PRINT # "HI = 99"I any other character is printed.

7

GENERAL INFORMATION

ARITHMETIC OPERATIONS

A = B	Assignment
A − B	Negation or subtraction
A + B, $A + $B	Addition or string concatenation
A*B	Multiplication
A/B	Division
A∧B	Exponentiation
− A	Unary Minus
+ A	Unary Plus

LOGICAL OPERATORS

LNOT A	*1's complement of integer.
A LAND B	*Bit wise AND.
A LOR B	*Bit wise OR.
A LXOR B	*Bit wise exclusive OR.

RELATIONAL OPERATORS

1 if TRUE and 0 if FALSE

A = B	TRUE if equal, else FALSE.
A = = B	*TRUE if approximately equal (1E-7), else FALSE
A<B	TRUE if less than, else FALSE.
A< = B	TRUE if less than or equal, else FALSE.
A>B	TRUE if greater than, else FALSE.
A> = B	TRUE if greater than or equal, else FALSE.
A<>B	TRUE if not equal, else FALSE.
NOT A	*TRUE if zero, else FALSE.
A AND B	*TRUE if both non-zero, else FALSE.
A OR B	*TRUE if either non-zero, else FALSE.

OPERATOR PRECEDENCE

1. Expressions in parentheses
2. Exponentiation and negation
3. *,/
4. +,−
5. <=,<>
6. >=,<
7. =,>
8. = =, LXOR
9. NOT, LNOT
10. AND, LAND
11. OR, LOR
12. (=)ASSIGNMENT

SPECIAL CHARACTERS

:: Separates statements typed on same line.
! Tail remark used for comments after program statement
; Equivalent to PRINT.

ERROR CODES

1 = SYNTAX ERROR
2 = UNMATCHED PARENTHESIS
3 = INVALID LINE NUMBER
4 = ILLEGAL VARIABLE NAME
5 = TOO MANY VARIABLES
6 = ILLEGAL CHARACTER
7 = EXPECTING OPERATOR
8 = ILLEGAL FUNCTION NAME
9 = ILLEGAL FUNCTION ARGUMENT
10 = STORAGE OVERFLOW
11 = STACK OVERFLOW
12 = STACK UNDERFLOW
13 = NO SUCH LINE NUMBER
14 = EXPECTING STRING VARIABLE
15 = INVALID SCREEN COMMAND
16 = EXPECTING DIMENSIONED VARIABLE
17 = SUBSCRIPT OUT OF RANGE
18 = TWO FEW SUBSCRIPTS
19 = TOO MANY SUBSCRIPTS
20 = EXPECTING SIMPLE VARIABLE
21 = DIGITS OUT OF RANGE (0< # of digits <12)
22 = EXPECTING VARIABLE
23 = READ OUT OF DATA
24 = READ TYPE DIFFERS FROM DATA TYPE
25 = SQUARE ROOT OF NEGATIVE NUMBER
26 = LOG OF NON-POSITIVE NUMBER
27 = EXPRESSION TOO COMPLEX
28 = DIVISION BY ZERO
29 = FLOATING POINT OVERFLOW
30 = FIX ERROR
31 = FOR WITHOUT NEXT
32 = NEXT WITHOUT FOR
33 = EXP FUNCTION HAS INVALID ARGUMENT
34 = UNNORMALIZED NUMBER
35 = PARAMETER ERROR
36 = MISSING ASSIGNMENT OPERATOR
37 = ILLEGAL DELIMITER
38 = UNDEFINED FUNCTION
39 = UNDIMENSIONED VARIABLE
40 = UNDERFINED VARIABLE
41 = EXPANSION EPROM NOT INSTALLED
42 = INTERRUPT W/O TRAP
43 = INVALID BAUD RATE
44 = TAPE READ ERROR
45 = EPROM VERIFY ERROR
46 = INVALID DEVICE NUMBER

7◀

Cross Support

▶7

The Cross Assembler data base which is assigned to PUNIT, is read by the FORTRAN program as the first file at execution time. It is the actual Cross Assembler program written in internal code, and it is suggested that it be assigned to a permanent disk file.

INTERNAL NAME	DEFAULT UNIT	DEVICE TYPE	RECORD LENGTH	FUNCTION
IUNIT	5	CR,CS MT,DF	80	TMS 9900 Source Input
LUNIT	6	CS,MT	80	Listing Output
OUNIT	7	CS,MT	80	TMS9900 Object Output
SUNIT	10	MT,DF	80	Assembly Scratch
PUNIT	11	CR,CS	80	Data Base INPUT

CR—CARD READER; CS—CASSETTE TAPE; MT—MAGNETIC TAPE; DF—DISKFILE; CP—CARD PUNCH; LP—LINE PRINTER

CROSS ASSEMBLER SYSTEM FILES

AORG places the expression value in the location counter, and defines the succeeding locations as absolute.

ABSOLUTE ORIGIN **AORG**

Syntax Definition:

[<label>]ø . . . AORGø . . . <wd-exp>ø . . .[<comment>]

RORG places the expression value in the location counter, and defines the succeeding locations as relocatable.

RELOCATABLE ORIGIN **RORG**

Syntax Definition:

[<label>ø . . . RORGø . . . [<exp>]ø . . .[<comment>]

DORG places the expression value in the location counter, and defines the succeeding locations as a dummy section. No object code is generated in a dummy section.

DUMMY ORIGIN **DORG**

Syntax Definition:

<label>ø . . . DORGø . . . <exp>ø . . .[<comment>]

BSS first assigns the label, if present, and increments the location counter by the value of the expression.

BLOCK STARTING WITH SYMBOL **BSS**

Syntax Definition:

[<label>]ø . . . BSSø . . . <wd-exp>ø . . .[<comment>]

BSS first increments the location counter by the value of the expression, and then assigns the label, if present.

BLOCK ENDING WITH SYMBOL **BES**

Syntax Definition:

[<label>]ø . . . BESø . . . <wd-exp>ø . . .[<comment>]

EQU assigns an assembly-time constant to the label.

▶7 DEFINE ASSEMBLY-TIME CONSTANT **EQU**

Syntax Definition:

<label>ø . . . EQUø . . . <exp>ø . . .[<comment>]

EVEN first assigns the label, if present, and then aligns the location counter on a word boundary (even address).

WORD BOUNDARY **EVEN**

Syntax Definition:

[<label>]ø . . . EVENø . . .[<comment>]

OPTIONS allows cross referencing when XREF is specified, and allows printing of the symbol table when SYMT is present.

OUTPUT OPTIONS **OPTION**

Syntax Definition:

ø . . . OPTIONø . . . <keyword>[,<keyword>] . . . ø . . . [<comment>]

IDT assigns a name to the program, and must precede any code-generating directive or instruction.

PROGRAM IDENTIFIER **IDT**

Syntax Definition:

[<label>]ø . . . IDTø . . . <string>ø . . .[<comment>]

TITL supplies a string to be printed at the top of each subsequent source listing page.

PAGE TITLE **TITL**

Syntax Definition:

[<label>]ø . . . TITLø . . . <string>ø . . . [<comment>]

LIST restores printing of the source listing.

LIST SOURCE **LIST**

Syntax Definition:

[<label>]ø . . . LISTø . . . [<comment>]

UNL inhibits printing of the source listing.

NO SOURCE LIST **UNL**

Syntax Definition:

[<label>]ø . . . UNLø . . . [<comment>]

PAGE directs the assembler to continue the source listing on the next page.

PAGE EJECT **PAGE**

Syntax Definition:

[<label>]ø . . . PAGEø . . . [<comment>]

BYTE places expressions in successive bytes, optionally assigning the label the address of the first byte.

INITIALIZE BYTE **.BYTE**

Syntax Definition:

[<label>]ø . . . BYTEø . . . <exp>[,<exp>] . . . ø . . . [<comment>]

DATA places expressions in successive words, optionally assigning the label the address of the first word. 7◀

INITIALIZE WORD **DATA**

Syntax Definition:

[<label>]ø . . . DATAø . . . <exp>[,<exp>] . . . ø . . . [<comment>]

TEXT places characters in successive bytes, arithmetically negating the last character, and optionally assigns the label the address of the first character.

INITIALIZE TEXT **TEXT**

Syntax Definition:

[<label>]ø . . . TEXTø . . . [−]<string>ø . . . [<comment>]

DEF makes symbols available to other programs as external references.

EXTERNAL DEFINITION **DEF**

Syntax Definition:

[<label>]ƀ . . . DEFƀ . . . <symbol>[,<symbol>] . . . ƀ . . . [<comment>]

REF directs the assembler to look externally for symbols.

EXTERNAL REFERENCE **REF**

Syntax Definition:

[<label>]ƀ . . . REFƀ . . . <symbol>[,<symbol>] . . . ƀ . . . [<comment>]

DXOP assigns an extended operation to a symbol.

DEFINE EXTENDED OPERATIONS **DXOP**

Syntax Definition:

[<label>]ƀ . . . DXOPƀ . . . <symbol>,<term>ƀ . . . [<comment>]

END terminates the assembly

PROGRAM END **END**

Syntax Definition:

[<label>]ƀ . . . ENDƀ . . . [<symbol>]ƀ . . . [<comment>]

NOP places a no-operation code in the object file.

NO OPERATION **NOP**

Syntax Definition:

[<label>]ƀ . . . NOPƀ . . . [<comment>]

RT assembles as a return from subroutine by substituting a branch through register 11.

RETURN **RT**

Syntax Definition:

[<label>]ƀ . . . RTƀ . . . [<comment>]

▶7

SIMULATOR FILES

INTERNAL NAME	DEFAULT UNIT	DEVICE TYPE	RECORD LENGTH	FUNCTION	WHERE USED
INCOPY	4	MT,DF	80	Batch copy file	C
INCOM	5	TE,CR MT,DF	80	Simulation command	C
OUTPRT OUTTRC	6	MT,DF TE,CR	80 or 136	Listing output	L,C,R
INLOD	10	TE,CR MT,DF	80	Linker commands	L
OUTCOM	11	TE,LP	80 or 136	Prompts and error msg. for linker output	L
OUTSAV	17	MT,CP DF	80	Absolute object	L,S
INSCR	20	MT,DF	136	Input scratch file	C,R,S
OUTSCR	21	MT,DF	136	Output scratch file	L,C,R

Device type legend
TE—terminal; CR—card reader; MT—magnetic tape; DF—disk file; CP—card punch

Where used legend
L—link processor; C—command processor; R—run processor; S-save processor

In addition to the above unit number assignments, the user must also assign unique FORTRAN logical unit numbers to each TMS9900 object code module to be included in the LINK processor.

SIMULATOR DIRECTIVES

ORIGIN COMMAND. The "ORIGIN" command can be used to specify where relocatable code is to be loaded.

ORIGIN hex-number

INCLUDE COMMAND. The "INCLUDE" command directs the loader to load an object module from a data set (e.g., card reader, disc, tape). The data set must be a sequential data set and may contain one or more object modules. At least one "INCLUDE" command should be used in the LINK processor command stream. The format for the command is as follows:

INCLUDE n

ENTRY COMMAND. The "ENTRY" command specifies the program entry point to the loader. The format for the command is as follows:

ENTRY name

SUMMARY OF CONTROL LANGUAGE STATEMENTS

The formats of the control statements for the "COMMAND" processor are shown below, with a brief description following:

[label] {R, RUN} [*] {F, FOR} n [{FR, FROM} i1] [{T, TO}] i2 [,label]	Specifies where to start and stop simulation. Control passes to statement at label operand when a breakpoint occurs.
[label] {T, TRACE} [list]	Specifies locations to be traced.
[label] {NOT, NOTRACE} [list]	Disables trace for specified locations.
[label] {RE, REFER} [list]	Specifies locations for reference breakpoint.
▶7 [label] {NOR, NOREFER} [list]	Disables reference breakpoint at specified locations.
[label] {A, ALTER} [list]	Specifies locations for alteration breakpoint.
[label] {NOA, NOALTER} [list]	Disables alteration breakpoint at specified locations.
[label] {P, PROTECT} [list]	Specifies areas for memory protection.
[label] IF (logical expression) label	Conditional transfer of control program.
[label] {J, JUMP} label	Unconditional transfer of control program.
[label] {TI, TIME} [n]	Prints the value of 9900 time and optionally sets a new value.
[label] {D, DISPLAY} [D] {CP, CPU} [register list]	Prints contents of registers.
[label] {D, DISPLAY} [D, C] {M, MEMORY} list	Prints contents of memory.

[label] {D, DISPLAY} {S, SYMBOL} [$, symbol, number]	Prints values from symbol table.
[label] {D, DISPLAY} {CR, CRU} {I, INPUT, O, OUTPUT} list	Prints CRU values.
[label] {S, SET} {C, CPU} register-value list	Places values into registers.
[label] {S, SET} {M, MEMORY} location-value list	Places values into memory.
[label] {S, SET} {I, INT} level, n_1 [,n_2,n_3]	Sets up one or more interrupts.
[label] {E, END}	Disables breakpoints and traces, and initializes simulation. Passes control to next control statement.
[label {I, INPUT} {n_1 n TO n_2} [F, FIRST, L, LAST, A, ALL] [data]	Defines input lines and fields, and supplies data for program being simulated.
[label] {O, OUTPUT} {n_1 n TO n_2}	Defines output lines and fields, or prints output of program being simulated.
[label] {CONN, CONNECT} list	Connects input CRU lines to output CRU lines.
[label] {C, CONVERT} expression list	Evaluates and prints values of expressions in decimal and hexadecimal form.
{B, BATCH}	Specifies batch mode.
[label] {L, LOAD}	Loads Wp and PC from locations $FFFC_{16}$ and $FFFE_{16}$.
[label] {CL, CLOCK} t	Specify clock period.
[label] {M, MEMORY} {RA, RAM, RO, ROM} {R, READ} $= n_1$ [{W, WRITE} $= n_2$] list	Define available memory. Default is 32K RAM.
[label] {SA, SAVE}	Create absolute object module.
[label] {W, WIDTH} n	Specifies number of columns available for printing.

7◀

MONITOR COMPLETION CODES

The simulator signals completion by executing and writing an appropriate STOP I statement, where I takes on one of the following values:

CODE	MEANING
0	Normal completion
1	Abnormal completion from LNKPRC
2	Premature EOF —If this error occurs it indicates that a premature EOF was encountered while attempting to reposition the BATCH command file.
3	Internal error; invalid label value
4	Roll memory overflow
5	Loader error —If this error occurs it means an attempt was made to load an object file into simulated memory and it failed causing termination of the link processor.
8	Abnormal completion from LOADER
9	Abnormal completion from CMDPRC
99	Internal error —Illegal completion from CMDPRC Internal error
999	Internal error —Illegal parameter passed to WRITER

If an error of 99 or 999 results, an internal error has occurred and the error should be reported to TEXAS INSTRUMENTS INC.

▶7

LINK PROCESSOR ERRORS

CODE	MESSAGE
L01	Load not completed
L02	Multiply defined external symbol (name)
L03	Empty object file on unit
L04	Attempt to load undefined memory
L05	Tag D follows tag 0
L06	Invalid tag character
L09	Undefined external memory
L13	Empty memory on save
L14	(name) not in external symbol table
L18	Maximum memory size exceeded
L19	Missing end
L21	Checksum error (computed value)
L22	Odd origin value specified—even value used
L24	Ref chain loop
L25	Object module does not start with tag 0
L26	Odd value (value) specified for tag (tag) even value used
L27	Missing F tag in record (number)
L28	Bad REF chain for (name)
L29	Bad object format in object module
L30	Illegal hex digit in field (digit)

COMMAND PROCESSOR ERRORS

CODE NUMBER	CODE NAME	MESSAGE	CODE NUMBER	CODE NAME	MESSAGE
1	BADCHR	Bad character	18	RANGE	Range error
2	BADCMD	Unrecognizable command	19	SYNTAX	Syntax error
3	BADIGT	Bad digit	20	TOOMNY	Too many values
4	BADMOD	Bad module name	21	UNDEF	Undefined symbol
5	BADREG	Bad register mnemonic			
6	BADVAL	Bad value			
7	CRUSPC	CRU specification error			
8	FLDCNT	Too few/many fields			
9	HITEOF	Hit EOF			
10	HITEOL	Hit end-of-line			
11	MEMDEF	Undefined			
12	MISSEQ	Missing equal sign			
13	NODATA	No data found			
14	NOROL	No data rolls available			
15	NOSET	Set not performed			
16	NOTIMP	Command not implemented			
17	ORDER	Command out of order			

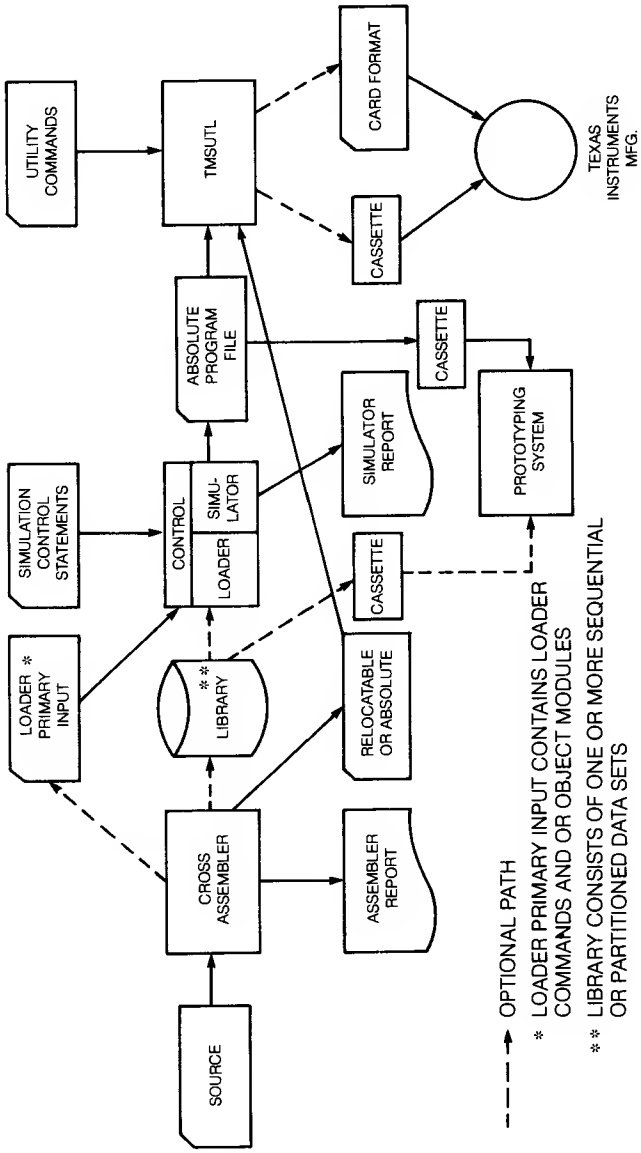
RUN PROCESSOR ERRORS

CODE	MESSAGE
1	PC interrupt vector entry in undefined memory
2	WP interrupt vector entry in undefined memory
3	Register out of address space (WP 65502)
4	Registers in undefined memory
5	Registers in ROM
6	PC interrupt vector refer breakpoint
7	WP interrupt vector refer breakpoint
8	Register alter breakpoint
9	Register protect breakpoint
10	Register refer breakpoint
11	Undefined opcode
12	Undefined memory reference
13,14	Unused
15	PC refer breakpoint
16	Unimplemented opcode
17,18,19	Unused
20	Destination address in undefined memory
21	Destination refer breakpoint
22	Destination alter breakpoint
23	Destination ROM breakpoint
24	Unused
25	Source address in undefined memory
26	Source refer breakpoint
27	Source alter breakpoint
28	Source ROM breakpoint

TMSUTL

CONCEPT

TMSUTL is a general purpose ultility program that accepts as input TI microprocessor object format, PROM manufacturing formats, or ROM manufacturing formats. This data is syntax checked, output options are gathered, the input data converted and an output file is produced.



7

INPUT, OUTPUT CONTROL CARD FORMATS

GENERAL DESCRIPTION

INPUT frmt [addr1 addr2] [WIDTH = x] [PARTITION = y]

- frmt — is the format number (integer 1-12).
- addr1 — is the starting address where input data is to be stored.
- addr2 — is the maximum address where data is to be stored.
- x — is the bit width of the input words.
- y — is the number of input data set partitions 1 Y 4

OUTPUT num addr1 addr2 WIDTH = x PARTITION = y

- num — is the format number (integer 1-12).
- addr1 — is the minimum address to be output.
- addr2 — is the maximum address to be output.
- x — is the bit width of an output word.
- y —

EOF—End of COMMAND FILE indicator

AVAILABLE FORMATS

FORMAT #	FORMAT	INPUT	OUTPUT
1	Hexadecimal #1 (PROM)	X	X
2	Hexadecimal #2 (ROM)	X	X
3	BNPF	X	X
4	271 & 371 ROM/HILO of prototyping System	X	X
5	TMS8080/TMS1000 Absolute Object from SIM8080/SIM1000 Loader/Simulator	X	X
▶7 6	TMS1000 Absolute ROM Object from Assembler	X	X
7	TMS1000 Listed Absolute Object	X	X
8	TMS1000 OPLA Data	X	
9	TMS9900 Standard Absolute Object of Cross Support System (Assembler or Loader/Simulator) & Prototyping System	X	X
10	TMS9900 Compressed Absolute Object of Prototyping System	X	X
11	TI4700 ROM	X	X
12	TI4800 ROM	X	X

TMSUTL FORMAT PATHS

Output Format →	1	2	3	4	5	6	7	8	9	10	11	12
1) Hexadecimal #2 (PROM)	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	YES
2) Hexadecimal #2 (ROM)	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	YES
3) BNPF	YES	YES	YES	YES	YES	YES	YES	NO	YES	YES	YES	YES
4) 271 & 371 ROM/ HILO of Prototyping System	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	YES
5) TMS1000 / TMS8080 Absolute Object from Loader/Simulator	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES
6) TMS1000 Absolute ROM Objects from Assembler for masking	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES
7) TMS1000 Listed Absolute Object	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES
8) TMS1000 OPLA Data	YES	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO
9) TMS9900 Standard Absolute Object of Cross Support System (Assembler or Loader/Simulator) & Prototyping System	YES	YES	YES	YES	NO	NO	NO	NO	YES	YES	YES	YES
10) TMS9900 Compressed Absolute Object of Protoyping System	YES	YES	YES	YES	NO	NO	NO	NO	YES	YES	YES	YES
11) TI4700 ROM	YES	YES	YES	YES	YES	NO	YES	NO	NO	NO	YES	YES
12) TI4800 ROM	YES	YES	YES	YES	YES	NO	YES	NO	NO	NO	YES	YES

DATA DELIMITERS

The following is a table of data delimiters or end-of-module records for Input Data.

FORMAT #	TYPES
1. Hex format 1	End of file record (:00)
2. Hex format 2	Trailer record — "END OF TEXT" (hollerith code 12-9-3) character followed by 79 non-blank characters (without asterisks)
3. BNPF	End of file record ($ in column 1)
4. 271/371 ROM and HILO of Prototyping System	End of file record ($END)
5. TMS8080/TMS1000 Absolute Object from Loader/Simulator	End record (+ END)
6. TMS1000 Absolute ROM Object	End of file record ($END)
7. TMS1000 Listed Absolute Object	End of file record ($END)
8. TMS1000 OPLA Data	End of file record ($END)
9. TMS9900 Standard Absolute Object	End of module record (:)
10. TMS9900 Binary Compressed Absolute Object	End of file record ($END)
11. TI4700 ROM	End of file record ($END)
12. TI4800 ROM	End of file record ($END)

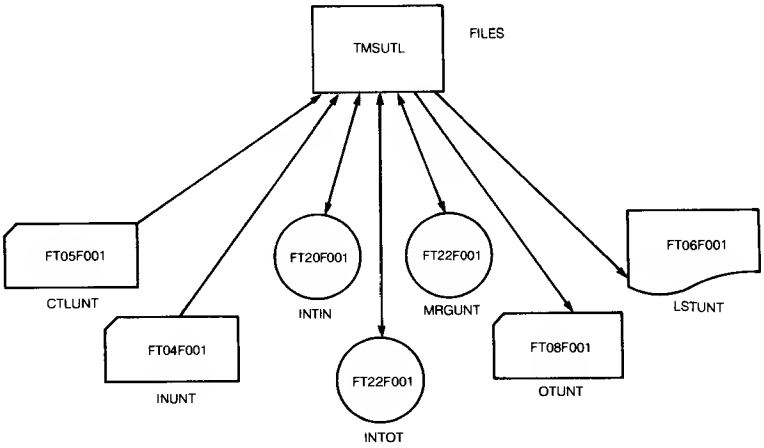
ADDRESS RANGES FOR FORMATS

FORMAT#	FORMAT	ADDRESS RANGE
1	Hexadecimal #1 (PROM)	$(0\text{-}FFFF)_H$
2	Hexadecimal #2 (ROM)	None
3	BNPF	None
4	271 & 371 ROM/HILO of Prototyping System	None
5	TMS8080/TMS1000 Absolute Object from Loader/Simulator	(0-255)
6	TMS1000 Absolute ROM Object	$(0\text{-}800)_H$
7	TMS1000 Listed Absolute Object	(0-1 Chapter 0-15 page 0-3F location)$_H$
8	TMS1000 OPLA Data	$(0\text{-}1F)_H$
9	TMS9900 Standard Absolute Object	$(0\text{-}FFFF)_H$
10	TMS9900 Compressed Absolute Object	$(0\text{-}FFFF)_H$
11	TI4700 ROM	$(0\text{-}400)_H$
12	TI4800 ROM	$(0\text{-}400)_H$

INPUT AND OUTPUT WIDTHS FOR FORMATS

FORMAT#	FORMAT	WIDTH (BITS)
1	Hexadecimal #1 (PROM)	8
2	Hexadecimal #2 (ROM)	8
3	BNPF	2 or 4 or 8 or 16
4	271 & 371 ROM/HILO of Prototyping System	4 or 8
5	TMS8080/TMS1000 Absolute Object from Loader/ Simulator	8
6	TMS1000 Object from Assembler	8
7	TMS1000 Listed Absolute Object	8
8	TMS1000 OPLA Data	8 or 16
9	TMS9900 Standard Absolute Object	16
10	TMS9900 Compressed Absolute Object	16
11	TI4700 ROM	8
12	TI4800 ROM	4 or 8

FILES DEFINITIONS & DESCRIPTIONS



CTLUNT — Input file for control cards.

INUNT — Input file for data.

INTIN — Intermediate file for storage of input data. It must be a rewindable file with a logical record length of 80 bytes.

INTOT — Intermediate file for storage of internal data. It must be a rewindable file with a logical record length of 80 bytes.

OTUNT — Output file for translated data.

LSTUNT — Print file for listing of data and error messages.

MRGUNT — Intermediate file for storage of internal data. It must be a rewindable file with a logical record length of 80 bytes.

TMSUTL ERROR MESSAGES

- ··· INPUT CONTROL CARD MISSING. Input control card missing or misplaced; it should be the first control card.
- ··· INVALID CONTROL CARD FIELD. Control card has an invalid field. Dollar signs point to the beginning and the end of the field.
- ··· OUTPUT FORMAT INCOMPATIBLE WITH INPUT FORMAT. The output format specified can not be converted from the input format specified.
- ··· OUTPUT FORMAT MISSING. Output control card missing or misplaced; it should follow the Input card.
- ··· ADDR2 ADDR1 OR BOTH NOT SPECIFIED. Either minimum or maximum address is invalid. Addr1 must be less than or equal to Addr2.
- ··· WIDTH INVALID FOR I/O FORMAT SPECIFIED. For the format specified the bit width is invalid.
- ··· PARTITION ERR. The Input bit width times the number of input partitions is not equal to the width times the number of output partitions.
- ··· ERROR DETECTED ON INPUT CARD. The format of a data card is invalid, check the field pointed to by the dollar signs.
- ··· INPUT OUT OF SEQUENCE. The addresses of the input data are not in sequential order.
- ··· # OF WORDS INPUT FOR CURRENT PARTITION NOT EQUAL TO THAT IN PREVIOUS PARTITION. The number of words input for each partition is not equal. Check the input data.
- ··· ADDRESS OUT OF RANGE. Either Addr1 or Addr2 is out of range or the address read on the input data is out of range of the format specified.

STOP CODES	ERROR
1	Input data error. (A message describing the error is output before this is issued.)
2	Format not implemented yet in EOF.
3	Format not implemented yet in TRANS.

STOP CODES	ERROR
90	DECHEX unable to find H or blank.
91	Data will not fit in card field passed to AFORMT.
92	Invalid format number in EOF.
93	Invalid width passed to INWORD.
94	SHFTR called with invalid arguments.
95	TRANS called with an invalid format number.

CHAPTER 8

Product Data Book

8-

Feature \ Device Type	TMS 9900	TMS 9900-40	SBP 9900A	TMS 9980A/ TMS 9981	TMS 9985	TMS 9940
Number of bytes addressable	65k		65k	16k	65k 256 RAM	2k EPROM 128 RAM
Number of Interrupts	16		16	5	5	4
Number of Pins	64		64	40	40	40
Power Supply Requirements	+5, −5, +12		500 Ma *(Note 1)*	+5,	+5	+5
Technology	N-MOS		I²L	N-MOS	N-MOS	N-MOS
Environmental (Temperature)	0-70°C		−55-125°C	0-70°C	0-70°C	0-70°C
Clock Rate	3.3 MHz	4 MHz	3 MHz	10MHz	5 MHz	5 MHz
Relative Thruput	1.0	1.3	0.9	0.6	0.65-0.8 *(Note 2)*	1.2
Number of Address Bus Lines	15		15	14	16	*(Note 3)*
Number of Data Bus Lines	16		16	8	8	*(Note 3)*
Clock	TIM 9904		SN54LS124	On chip	On chip	On chip

Note 1: Voltage for the SBP 9900A is 1.5 to 30 volts with a series resistor.
Note 2: The relative thruput is 0.65 with an off-chip RAM and 0.8 with an on-chip RAM.
Note 3: There are 32 general purpose pins which can be programmed for I/O. While the memory and data buses are not available, 8 address bits are accessible for CRU I/O expansion.

FAMILY DESCRIPTION

The TMS 9900 micrprocessor is a single-chip 16-bit central processing unit (CPU) produced using N-channel silicon-gate MOS technology. The instruction set of the TMS 9900 includes the capabilities offered by full minicomputers. The unique memory-to-memory architecture features multiple register files, resident in memory, which allow faster response to interrupts and increased programming flexibility. The separate bus structure simplifies the system design effort. Texas Instruments provides a compatible set of MOS and TTL memory and logic function circuits to be used with a TMS 9900 system. The system is fully supported by software and a complete prototyping system.

8

There is a TMS 9900-40 part designed for 4-MHz operation. Refer to the separate "−40" (dash forty) electrical specification tables for exact characteristics.

The SBP 9900A is basically the same as the TMS 9900 but it employs I^2L technology to enhance environmental specifications. It is a static, bipolar microprocessor operating from a single phase clock over the frequency range from 0 to 3 MHz. The I/O is fully TTL compatible so that no special peripheral circuits are required. The power supply is specified as a single input injector current which may be varied over the range from 10 mA to 1 A with a corresponding change in speed (described in the specific SBP 9900A section). The architecture is the same for the SBP 9900A as the TMS 9900 with minor differences in clock and control lines.

Software compatibility with other 9900 microprocessor family members provides a common body of hardware/software within Texas Instruments 990 minicomputer family.
The TMS 9980A/TMS 9981 is another software-compatible member of TI's 9900 family of microprocessors. Designed to minimize the system cost for smaller systems, the TMS 9980A/TMS 9981 is a single-chip 16-bit central processing unit (CPU) which has an 8-bit data bus, on-chip clock, and is packaged in a 40-pin package. The instruction set of the TMS 9980A/TMS 9981 includes the capabilities offered by full minicomputers and is exactly the same as the 9900's.

The TMS 9940 is a single-chip, 16-bit microprocessor containing a CPU, memory (RAM and EPROM/ROM), and extensive I/O. Except for four instructions which do not apply to the TMS 9940 microcomputer configuration, the TMS 9940 instruction set matches that of the TMS 9900 and includes capabilities offered by *minicomputers*. In addition, the TMS 9940 instruction set includes two instructions which facilitate manipulation of binary coded decimal (BCD) data, and a single-word load-interrupt-mask (LIIM) instruction.
The unique memory-to-memory architecture features multiple register files, resident in the RAM, which allow faster response to interrupts and increased programming flexibility. The memory consists of 128 bytes of RAM and 2048 bytes of EPROM. The TMS 9940 implements four levels of interrupts including an internal decrementer which can be programmed as a timer or an event counter. All members of the TMS 9900 family of peripheral circuits are compatible with the TMS 9940. The TMS 9940 is fully supported by software and hardware development systems and by factory applications engineers and technical answering services.

The TMS 9985 is a software compatible member of TI's 9900 family of microprocessors and microcomputers and contains a 16-bit CPU, 256 bytes of RAM, on chip timer/event counter, external 16-bit address bus and 8-bit data bus, and is in a 40-pin package. The instruction set of the TMS 9985 includes the capabilities offered by full minicomputers and is exactly the same as the TMS 9940 microcomputer's. The unique memory-to-memory architecture features multiple register files, resident in memory, which allows faster response to interrupts and increased programming flexibility. In addition, the TMS 9985 has excellent I/O flexibility with CRU, memory mapped I/O and direct memory access.

COMMON KEY FEATURES

- 16-Bit Architecture
- 69 Powerful Instructions Include:
  - Multiply and Divide
  - 5 Addressing Modes
  - Bit, Byte, and Word Addressing
  - One, Two and Three Word Instructions
- Rapid Hardware Context Switching
- Multiple 16-Word Register Files (Workspaces) Reside in Memory
- Separate I/O, Memory and Interrupt Bus Structures
- Programmed and DMA I/O Capability
- Communications Register Unit (CRU) for Low and Medium Speed Devices
- Efficient Memory-to-Memory Architecture
- Extended Operations (XOP) Feature Allows Users to Augment the Instruction Set
- Maskable Vectored Priority Interrupts for Multiprogramming Requirements
- Software Compatible with 990 Minicomputer Family

Key Features of Specific Devices

TMS 9900

- 3.3-MHz Speed (4.0 MHz Speed for the TMS 9900-40)
- Up to 65,536 Bytes of Memory
- 16 Prioritized Interrupts
- 64-Pin Package
- N-Channel Silicon-Gate Technology
- 0-70°C Ambient Temperature Range
- Directly TTL Compatible I/O

SBP 9900A

- 3-MHz Speed
- Up to 65,536 Bytes of Memory
- 16 Prioritized Interrupts
- 64-Pin Package
- I^2L Technology
- −55 to 125°C Ambient Temperature Range
- Single dc Power Supply
- Directly TTL Compatible I/O

TMS 9980A/TMS9981

- 10-MHz Speed
- Up to 16,384 Bytes of Memory
- 8-Bit Memory Data Bus
- 4 Prioritized Interrupts
- On-Chip 4-Phase Clock Generator
- 40-Pin Package
- N-Channel Silicon-Gate Technology
- 0-70°C Ambient Temperature Range

TMS 9980A/TMS 9981 Differences

The TMS 9980A and the TMS 9981, although very similar, have several differences of which the user should be aware.

1. The TMS 9980A requires a V_{BB} supply (pin 21) while the TMS 9981 has an internal charge pump to generate V_{BB} from V_{CC} and V_{DD}.
2. The TMS 9981 has an optional on-chip crystal oscillator in addition to the external clock mode of the TMS 9980A.
3. The pin-outs are not the same for D0-D7, INT0-INT2, and $\overline{\phi 3}$.

TMS 9985

- 5-MHz Speed
- Up to 65,536 Bytes of Memory
- 8-Bit Memory Data Bus
- 5 Prioritized Interrupts
- 40-Pin Package
- N-Channel Silicon-Gate Technology
- 0-70°C Ambient Temperature Range
- On Chip Timer/Event Counter
- 256 Bits of RAM on Chip
- Separate Memory, I/O and Interrupt Bus Structures
- On Chip Programmable Flags (16)
- Multiprocessor System Interface
- Single 5-Volt Supply
- Speed Selected Versions

TMS 9940

- 5-MHz Speed
- 2048 Bytes of EPROM or ROM
- 128 Bytes of RAM
- 4 Prioritized Interrupts
- 40-Pin Package
- N-Channel Silicon-Gate Technology
- 0-70°C Ambient Temperature Range
- On-Chip Timer/Event Counter
- 32 Bits General Purpose I/O
- 256 Bits I/O Expansion
- Multiprocessor System Interface
- Single 5-Volt Power Supply
- Power Down Capability for Low Standby Power
- Easy Test Function
- Offered as either an EPROM device as a mask ROM device
- Speed Selected Versions

Organization of CPU Data Manuals and Instruction Set

Data manuals for the five CPU's in the 9900 family are reproduced in this section with the TMS 9900 first, followed by the SBP 9900A, TMS 9980A/81, and TMS 9940 data manuals. Following this there is an abbreviated version of the TMS 9985 manual. Since the information regarding the instruction set is common to all of the CPU's, it has been removed from the individual manuals and is printed at the end of this section.

TMS 9900

1. INTRODUCTION

1.1 DESCRIPTION

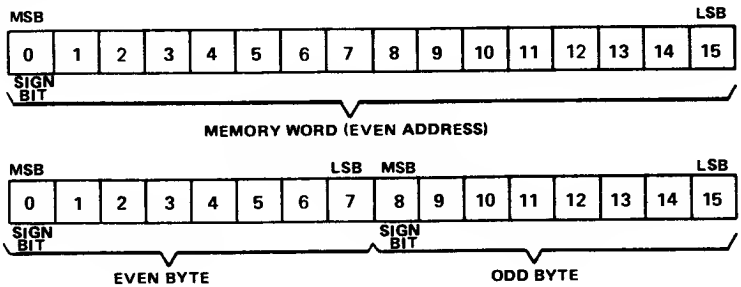
The TMS 9900 microprocessor is a single-chip 16-bit central processing unit (CPU) produced using N-channel silicon-gate MOS technology (see Figure 1). The instruction set of the TMS 9900 includes the capabilities offered by full minicomputers. The unique memory-to-memory architecture features multiple register files, resident in memory, which allow faster response to interrupts and increased programming flexibility. The separate bus structure simplifies the system design effort. Texas Instruments provides a compatible set of MOS and TTL memory and logic function circuits to be used with a TMS 9900 system. The system is fully supported by software and a complete prototyping system.

1.2 KEY FEATURES

- 16-Bit Instruction Word
- Full Minicomputer Instruction Set Capability Including Multiply and Divide
- Up to 65,536 Bytes of Memory
- 3.3 – MHz Speed
- Advanced Memory-to-Memory Architecture
- Separate Memory, I/O, and Interrupt-Bus Structures
- 16 General Registers
- 16 Prioritized Interrupts
- Programmed and DMA I/O Capability
- N-Channel Silicon-Gate Technology

2. ARCHITECTURE

The memory word of the TMS 9900 is 16 bits long. Each word is also defined as 2 bytes of 8 bits. The instruction set of the TMS 9900 allows both word and byte operands. Thus, all memory locations are on even address boundaries and byte instructions can address either the even or odd byte. The memory space is 65,536 bytes or 32,768 words. The word and byte formats are shown below.



2.1 REGISTERS AND MEMORY

The TMS 9900 employs an advanced memory-to-memory architecture. Blocks of memory designated as workspace replace internal-hardware registers with program-data registers. The TMS 9900 memory map is shown in Figure 2. The first 32 words are used for interrupt trap vectors. The next contiguous block of 32 memory words is used by the extended operation (XOP) instruction for trap vectors. The last two memory words, $FFFC_{16}$ and $FFFE_{16}$, are used for the trap vector of the LOAD signal. The remaining memory is then available for programs, data, and workspace registers. If desired, any of the special areas may also be used as general memory.

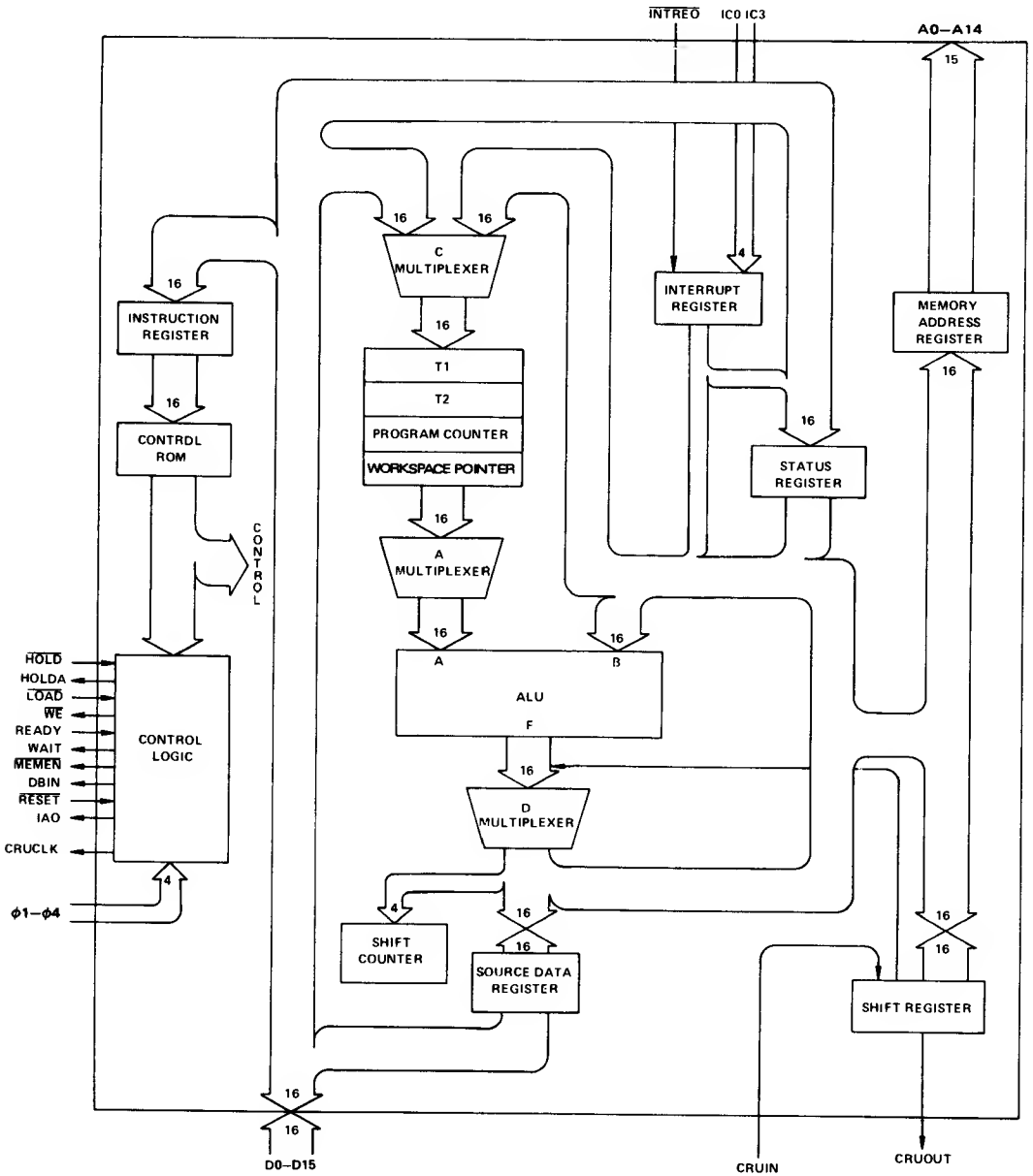


FIGURE 1 – ARCHITECTURE

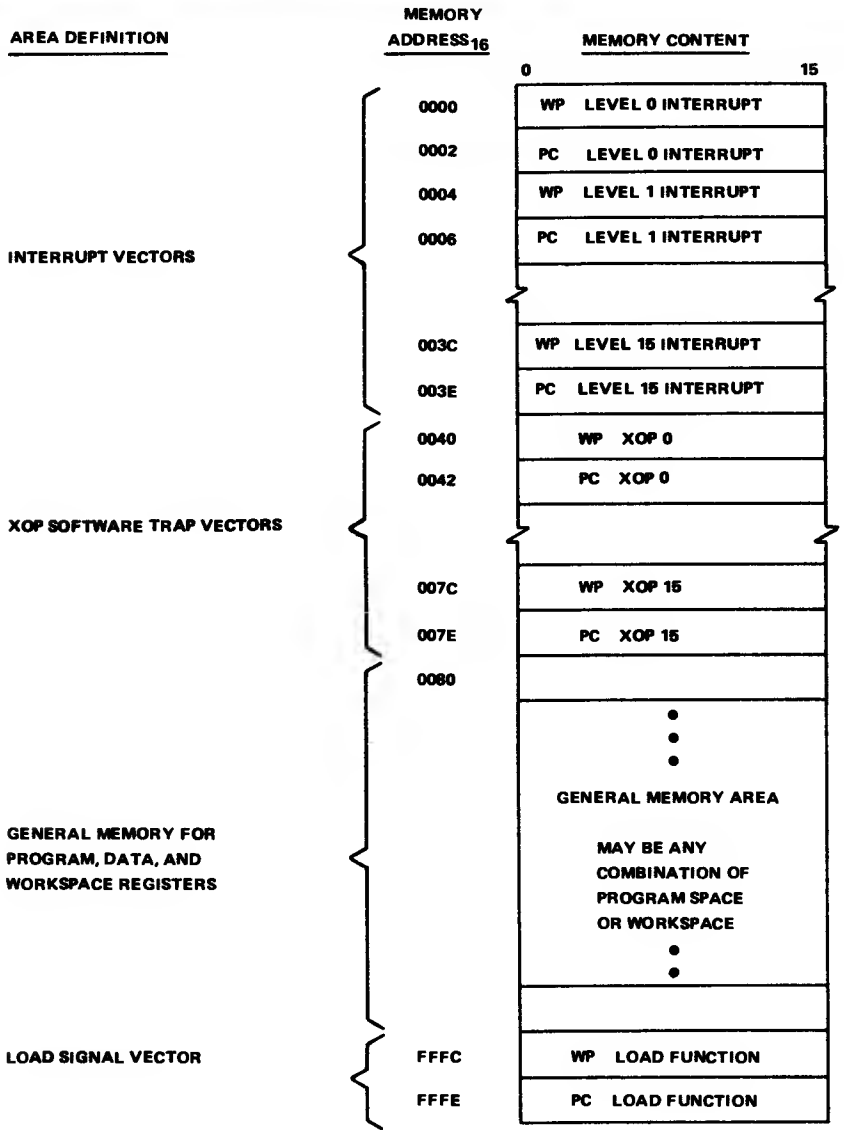
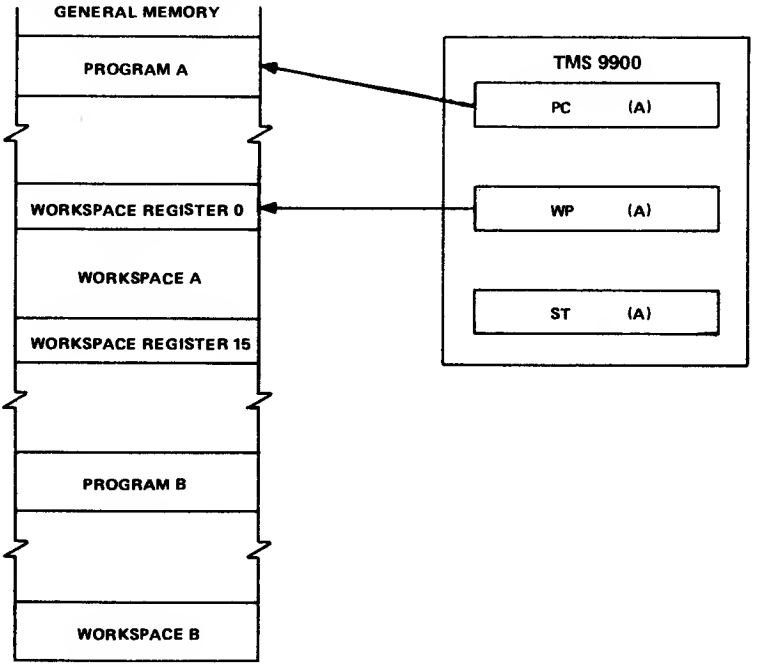


FIGURE 2 – MEMORY MAP

Three internal registers are accessible to the user. The program counter (PC) contains the address of the instruction following the current instruction being executed. This address is referenced by the processor to fetch the next instruction from memory and is then automatically incremented. The status register (ST) contains the present state of the processor and will be further defined in the Instruction Set section. The workspace pointer (WP) contains the address of the first word in the currently activa sat of workspace registers.

A workspace-register file occupies 16 contiguous memory words in the general memory area (see Figure 2). Each workspace register may hold data or addresses and function as operand registers, accumulators, address registers, or

index registers. During instruction execution, the processor addresses any register in the workspace by adding the register number to the contents of the workspace pointer and initiating a memory request for the word. The relationship between the workspace pointer and its corresponding workspace is shown below.



The workspace concept is particularly valuable during operations that require a context switch, which is a change from one program environment to another (as in the case of an interrupt) or to a subroutine. Such an operation, using a conventional multi-register arrangement, requires that at least part of the contents of the register file be stored and reloaded. A memory cycle is required to store or fetch each word. By exchanging the program counter, status register, and workspace pointer, the TMS 9900 accomplishes a complete context switch with only three store cycles and three fetch cycles. After the switch the workspace pointer contains the starting address of a new 16-word workspace in memory for use in the new routine. A corresponding time saving occurs when the original context is restored. Instructions in the TMS 9900 that result in a context switch include:

1. Branch and Load Workspace Pointer (BLWP)
2. Return from Subroutine (RTWP)
3. Extended Operation (XOP).

Device interrupts, $\overline{\text{RESET}}$, and $\overline{\text{LOAD}}$ also cause a context switch by forcing the processor to trap to a service subroutine.

2.2 INTERRUPTS

The TMS 9900 employs 16 interrupt levels with the highest priority level 0 and lowest level 15. Level 0 is reserved for the $\overline{\text{RESET}}$ function and all other levels may be used for external devices. The external levels may also be shared by several device interrupts, depending upon system requirements.

The TMS 9900 continuously compares the interrupt code (IC0 through IC3) with the interrupt mask contained in status-register bits 12 through 15. When the level of the pending interrupt is less than or equal to the enabling mask level (higher or equal priority interrupt), the processor recognizes the interrupt and initiates a context switch following

completion of the currently executing instruction. The processor fetches the new context WP and PC from the interrupt vector locations. Then, the previous context WP, PC, and ST are stored in workspace registers 13, 14, and 15, respectively, of the new workspace. The TMS 9900 then forces the interrupt mask to a value that is one less than the level of the interrupt being serviced, except for level-zero interrupt, which loads zero into the mask. This allows only interrupts of higher priority to interrupt a service routine. The processor also inhibits interrupts until the first instruction of the service routine has been executed to preserve program linkage should a higher priority interrupt occur. All interrupt requests should remain active until recognized by the processor in the device-service routine. The individual service routines must reset the interrupt requests before the routine is complete.

If a higher priority interrupt occurs, a second context switch occurs to service the higher priority interrupt. When that routine is complete, a return instruction (RTWP) restores the first service routine parameters to the processor to complete processing of the lower-priority interrupt. All interrupt subroutines should terminate with the return instruction to restore original program parameters. The interrupt-vector locations, device assignment, enabling-mask value, and the interrupt code are shown in Table 1.

TABLE 1
INTERRUPT LEVEL DATA

Interrupt Level	Vector Location (Memory Address In Hex)	Device Assignment	Interrupt Mask Values To Enable Respective Interrupts (ST12 thru ST15)	Interrupt Codes IC0 thru IC3
(Highest priority) 0	00	Reset	0 through F*	0000
1	04	External device	1 through F	0001
2	08	↓	2 through F	0010
3	0C	↓	3 through F	0011
4	10	↓	4 through F	0100
5	14	↓	5 through F	0101
6	18	↓	6 through F	0110
7	1C	↓	7 through F	0111
8	20	↓	8 through F	1000
9	24	↓	9 through F	1001
10	28	↓	A through F	1010
11	2C	↓	8 through F	1011
12	30	↓	C through F	1100
13	34	↓	D through F	1101
14	38	↓	E and F	1110
(Lowest priority) 15	3C	External device	F only	1111

*Level 0 can not be disabled.

The TMS 9900 interrupt interface utilizes standard TTL components as shown in Figure 3. Note that for eight or less external interrupts a single SN74148 is required and for one external interrupt $\overline{\text{INTREQ}}$ is used as the interrupt signal with a hard-wired code IC0 through IC3.

8◀

2.3 INPUT/OUTPUT

The TMS 9900 utilizes a versatile direct command-driven I/O interface designated as the communications-register unit (CRU). The CRU provides up to 4096 directly addressable input bits and 4096 directly addressable output bits. Both input and output bits can be addressed individually or in fields of from 1 to 16 bits. The TMS 9900 employs three dedicated I/O pins (CRUIN, CRUOUT, and CRUCLK) and 12 bits (A3 through A14) of the address bus to interface with the CRU system. The processor instructions that drive the CRU interface can set, reset, or test any bit in the CRU array or move between memory and CRU data fields.

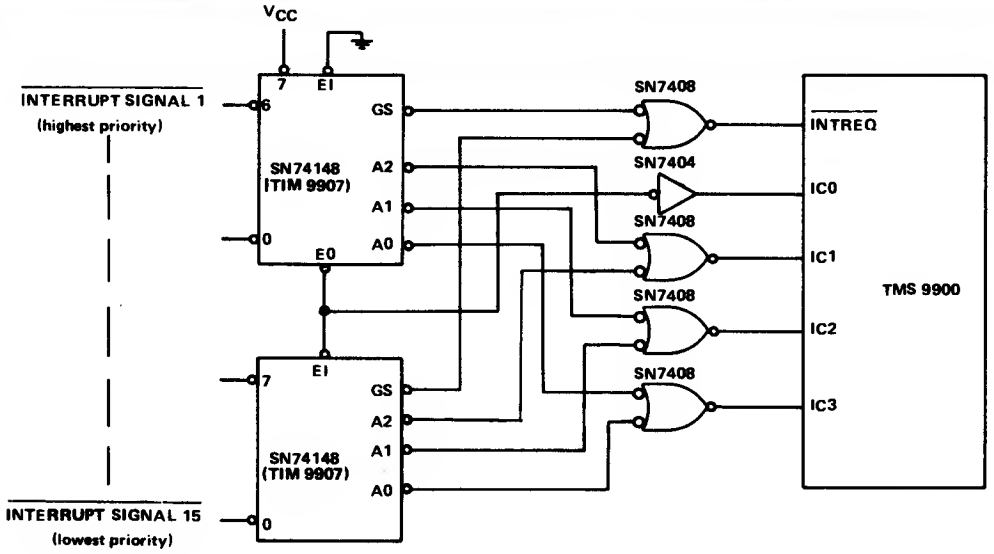


FIGURE 3 – TMS 9900 INTERRUPT INTERFACE

2.4 SINGLE-BIT CRU OPERATIONS

The TMS 9900 performs three single-bit CRU functions: test bit (TB), set bit to one (SBO), and set bit to zero (SBZ). To identify the bit to be operated upon, the TMS 9900 develops a CRU-bit address and places it on the address bus, A3 to A14.

For the two output operations (SBO and SBZ), the processor also generates a CRUCLK pulse, indicating an output operation to the CRU device, and places bit 7 of the instruction word on the CRUOUT line to accomplish the specified operation (bit 7 is a one for SBO and a zero for SBZ). A test-bit instruction transfers the addressed CRU bit from the CRUIN input line to bit 2 of the status register (EQUAL).

The TMS 9900 develops e CRU-bit address for the single-bit operetions from the softwere base eddress conteined in workspece register 12 end the signed displecement count conteined in bits 8 through 15 of the instruction. The displecement ellows two's complement eddressing from base minus 128 bits through bese plus 127 bits. The hardwere bese eddress, bits 3 through 14 of W12, is edded to the signed displecement specified in the instruction end the result is loaded onto the eddress bus. *Figure 4* illustrates the development of a single-bit CRU address.

2.5 MULTIPLE-BIT CRU OPERATIONS

▶8

The TMS 9900 performs two multiple-bit CRU operations: store communications register (STCR) and load communications register (LDCR). Both operations perform a data transfer from the CRU-to-memory or from memory-to-CRU as illustrated in Figure 5. Although the figure illustrates a full 16-bit transfer operation, any number of bits from 1 through 16 may be involved. The LDCR instruction fetches a word from memory and right-shifts it to serially transfer it to CRU output bits. If the LDCR involves eight or fewer bits, those bits come from the right-justified field within the addressed byte of the memory word. If the LDCR involves nine or more bits, those bits come from the right-justified field within the whole memory word. When transferred to the CRU interface, each successive bit receives an address that is sequentially greater than the address for the previous bit. This addressing mechanism results in an order reversal of the hits; that is, bit 15 of the memory word (or bit 7) becomes the lowest addressed bit in the CRU and bit 0 becomes the highest addressed bit in the CRU field.

An STCR instruction transfers data from the CRU to memory. If the operation involves a byte or less transfer, the transferred data will be stored right-justified in the memory byte with leading bits set to zero. If the operation involves from nine to 16 bits, the transferred data is stored right-justified in the memory word with leading bits set to zero.

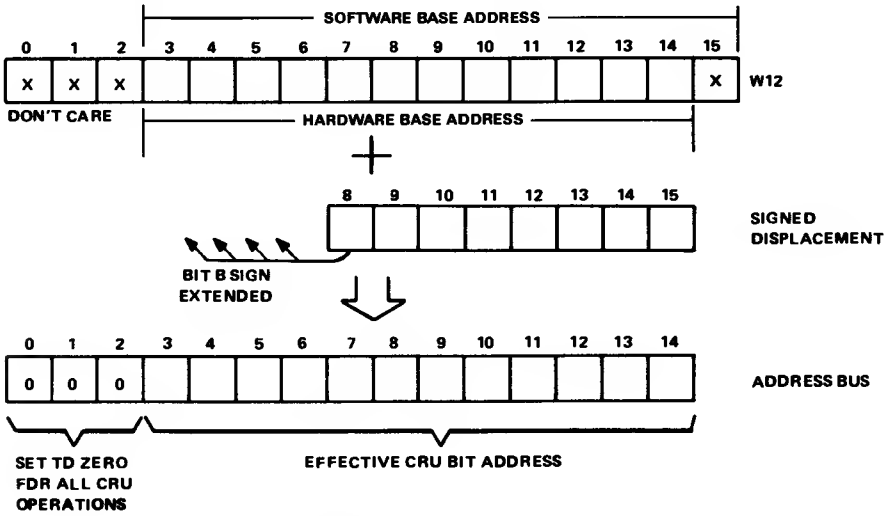
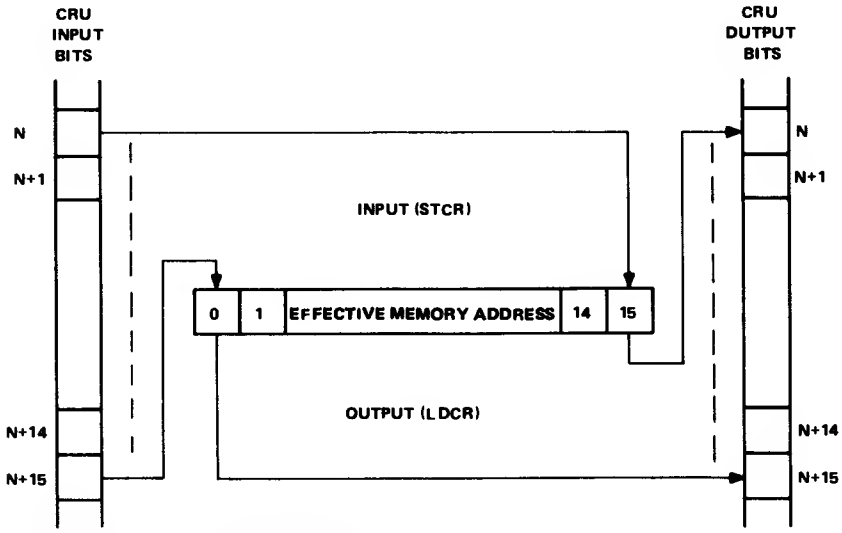


FIGURE 4 – TMS 9900 SINGLE-BIT CRU ADDRESS DEVELOPMENT



N = BIT SPECIFIED BY CRU BASE REGISTER

FIGURE 5 – TMS 9900 LDCR/STCR DATA TRANSFERS

When the input from the CRU device is complete, the first bit from the CRU is the least-significant-bit position in the memory word or byte.

Figure 6 illustrates how to implement a 16-bit input and a 16-bit output register in the CRU interface. CRU addresses are decoded as needed to implement up to 256 such 16-bit interface registers. In system application, however, only the exact number of interface bits needed to interface specific peripheral devices are implemented. It is not necessary to have a 16-bit interface register to interface an 8-bit device.

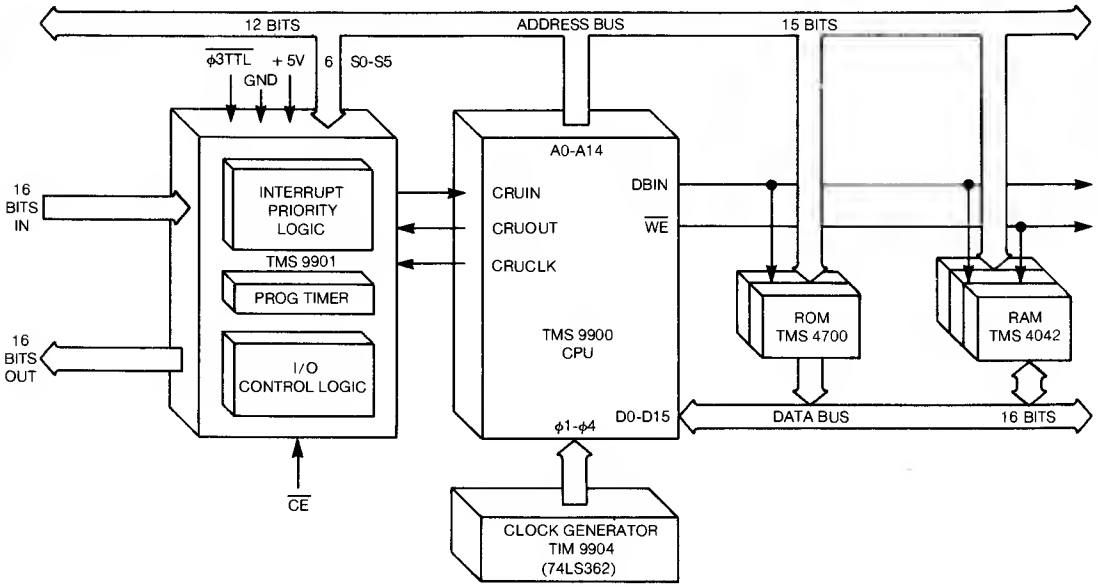


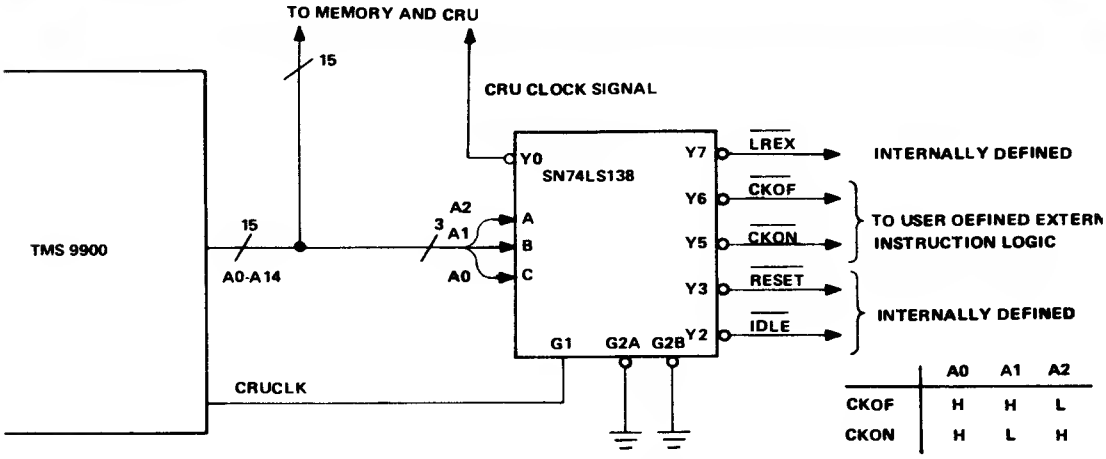
FIGURE 6 – TMS 9900 16-BIT INPUT/OUTPUT INTERFACE

2.6 EXTERNAL INSTRUCTIONS

The TMS 9900 has five external instructions that allow user-defined external functions to be initiated under program control. These instructions are CKON, CKOF, RSET, IDLE, and LREX. These mnemonics, except for IDLE, relate to functions implemented in the 990 minicomputer and do not restrict use of the instructions to initiate various user-defined functions. IDLE also causes the TMS 9900 to enter the idle state and remain until an interrupt, $\overline{\text{RESET}}$, or $\overline{\text{LOAD}}$ occurs. When any of these five instructions are executed by the TMS 9900, a unique 3-bit code appears on the most-significant 3 bits of the address bus (A0 through A2) along with a CRUCLK pulse. When the TMS 9900 is in an idle state, the 3-bit code and CRUCLK pulses occur repeatedly until the idle state is terminated. The codes are:

EXTERNAL INSTRUCTION	A0	A1	A2
LREX	H	H	H
CKOF	H	H	L
CKON	H	L	H
RSET	L	H	H
IDLE	L	H	L

Figure 7 illustrates typical external decode logic to implement these instructions. Note that a signal is generated to inhibit CRU decodes during external instructions.



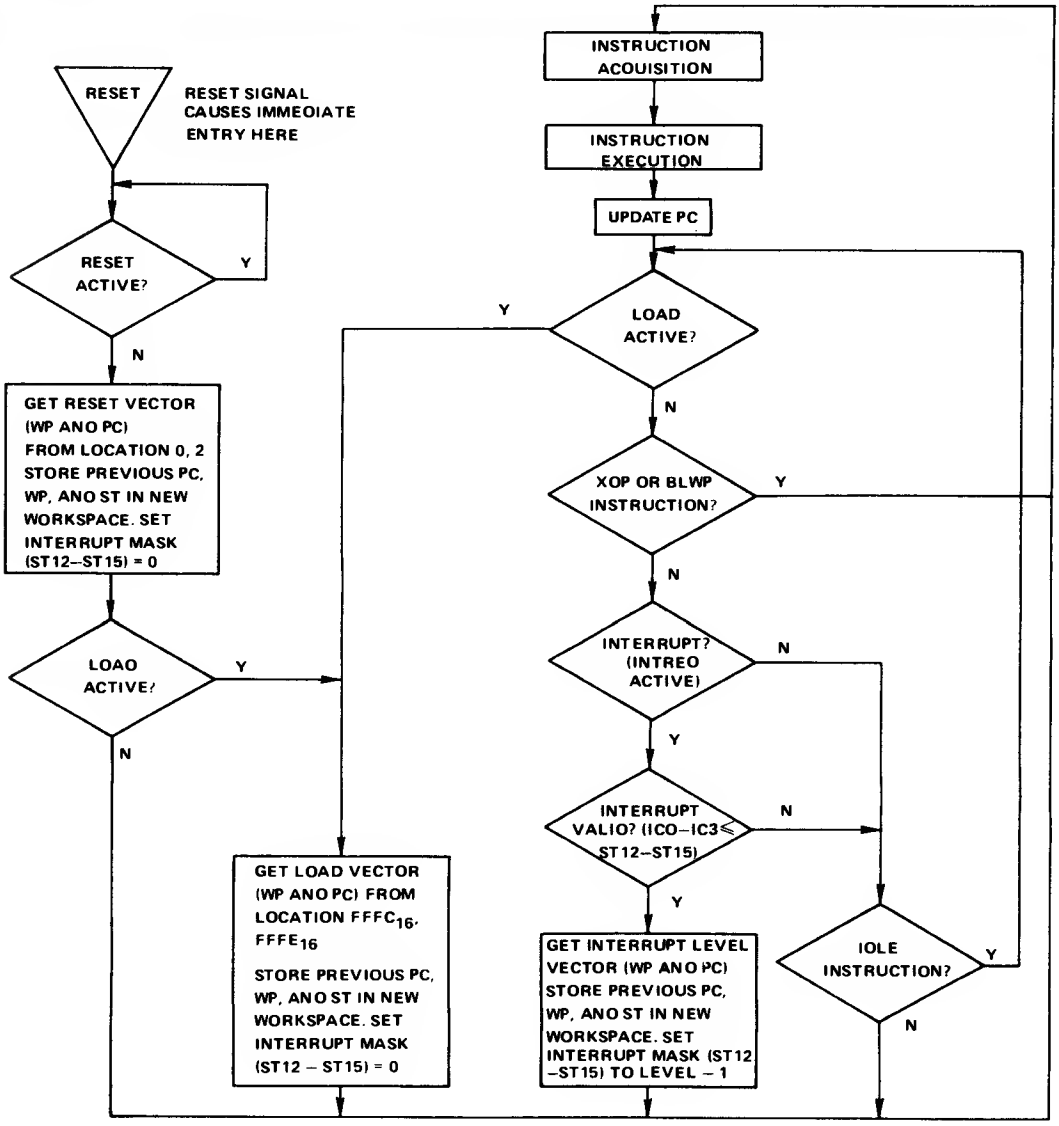
	A0	A1	A2
CKOF	H	H	L
CKON	H	L	H

FIGURE 7 – EXTERNAL INSTRUCTION DECODE LOGIC

2.7 LOAD FUNCTION

The $\overline{\text{LOAD}}$ signal allows cold-start ROM loaders and front panels to be implemented for the TMS 9900. When active, $\overline{\text{LOAD}}$ causes the TMS 9900 to initiate an interrupt sequence immediately following the instruction being executed. Memory location FFFC is used to obtain the vector (WP and PC). The old PC, WP and ST are loaded into the new workspace and the interrupt mask is set to 0000. Then, program execution resumes using the new PC and WP.

8◀



▶8

FIGURE 8 - TMS 9900 CPU FLOW CHART

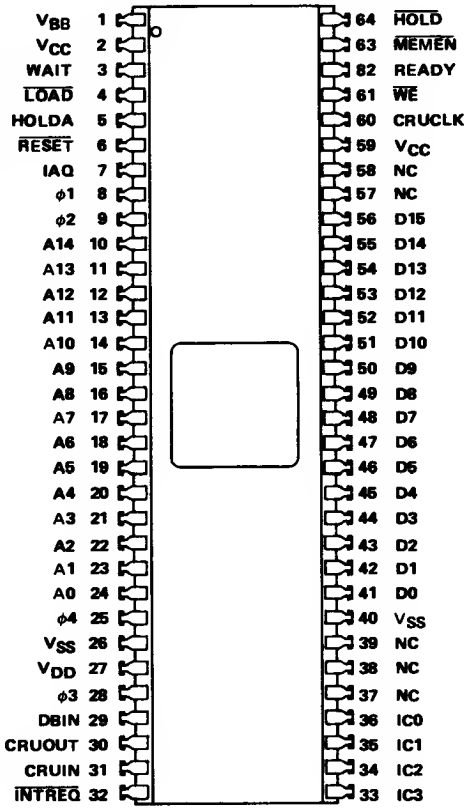
2.8 TMS 9900 PIN DESCRIPTION

Table 2 defines the TMS 9900 pin assignments and describes the function of each pin.

TABLE 2
TMS 9900 PIN ASSIGNMENTS AND FUNCTIONS

SIGNATURE	PIN	I/O	DESCRIPTION
			ADDRESS BUS
A0 (MSB)	24	OUT	A0 through A14 comprise the address bus.
A1	23	OUT	This 3-state bus provides the memory-
A2	22	OUT	address vector to the external-memory
A3	21	OUT	system when $\overline{\text{MEMEN}}$ is active and I/O-bit
A4	20	OUT	addresses and external-instruction addresses
A5	19	OUT	to the I/O system when $\overline{\text{MEMEN}}$ is inactive.
A6	18	OUT	The address bus assumes the high-impedance
A7	17	OUT	state when HOLDA is active.
A8	16	OUT	
A9	15	OUT	
A10	14	OUT	
A11	13	OUT	
A12	12	OUT	
A13	11	OUT	
A14 (LSB)	10	OUT	
			DATA BUS
D0 (MSB)	41	I/O	D0 through D15 comprise the bidirectional
D1	42	I/O	3-state data bus. This bus transfers memory
D2	43	I/O	data to (when writing) and from (when
D3	44	I/O	reading) the external-memory system when
D4	45	I/O	$\overline{\text{MEMEN}}$ is active. The data bus assumes the
D5	46	I/O	high-impedance state when HOLDA is
D6	47	I/O	active.
D7	48	I/O	
D8	49	I/O	
D9	50	I/O	
D10	51	I/O	
D11	52	I/O	
D12	53	I/O	
D13	54	I/O	
D14	55	I/O	
D15 (LSB)	56	I/O	
			POWER SUPPLIES
V_{BB}	1		Supply voltage (−5 V NOM)
V_{CC}	2,59		Supply voltage (5 V NOM). Pins 2 and 59 must be connected in parallel.
V_{DD}	27		Supply voltage (12 V NOM)
V_{SS}	26,40		Ground reference. Pins 26 and 40 must be connected in parallel.
			CLOCKS
$\phi1$	8	IN	Phase-1 clock
$\phi2$	9	IN	Phase-2 clock
$\phi3$	28	IN	Phase-3 clock
$\phi4$	25	IN	Phase-4 clock

TMS 9900 PIN ASSIGNMENTS



NC — No Internal connection

8

TABLE 2 (CONTINUED)

SIGNATURE	PIN	I/O	DESCRIPTION
			BUS CONTROL
DBIN	29	OUT	Data bus in. When active (high), DBIN indicetes that the TMS 9900 has disabled its output buffers to allow tha memory to place memory-raad data on the data bus during $\overline{\text{MEMEN}}$. DBIN remeins low in all othar cases except when HOLDA is ective.
$\overline{\text{MEMEN}}$	63	OUT	Memory enable. When active (low), $\overline{\text{MEMEN}}$ indicates that the eddrass bus contains a memory address.
$\overline{\text{WE}}$	61	OUT	Write anable. Whan active (low), $\overline{\text{WE}}$ indicates that memory-writa data is available from tha TMS 9900 to be written into memory.
CRUCLK	60	OUT	CRU clock. Whan active (high), CRUCLK indicetes that external interface logic should sample the output data on CRUOUT or should decode axternal instructions on A0 through A2.
CRUIN	31	IN	CRU data in. CRUIN, normally driven by 3-stete or open-collector davices, receives input deta from externel interface logic. Whan the processor executes a STCR or TB instruction, it samples CRUIN for tha lavel of tha CRU input bit specified by the addrass bus (A3 through A14).
CRUOUT	30	OUT	CRU data out. Serial I/O data appears on the CRUOUT line when an LDCR, SBZ, or SBO instruction is axecuted. The data on CRUOUT should be samplad by external I/O interface logic when CRUCLK goes active (high).
			INTERRUPT CONTROL
$\overline{\text{INTREQ}}$	32	IN	Interrupt request. When active (low), $\overline{\text{INTREQ}}$ indicates thet an external intarrupt is requested. If $\overline{\text{INTREQ}}$ is active, the processor loads the data on the interrupt-coda-input lines IC0 through IC3 into the internal intarrupt-code-storage registar. The code is compared to the intarrupt mask bits of the status register. If equal or higher priority than the enabled interrupt level (interrupt code equal or less than status register bits 12 through 15) the TMS 9900 interrupt sequance is initiated. If the comperison feils, the processor ignores the request. $\overline{\text{INTREQ}}$ should remain active and the processor will continue to sample IC0 through IC3 until the program enables a sufficiently low priority to accapt the request intarrupt.
IC0 (MSB)	36	IN	Interrupt codes. IC0 is the MSB of the interrupt code, which is sampled when $\overline{\text{INTREQ}}$ is active. When IC0 through IC3 ara LLLH, the highest external-priority intarrupt is being requested and when HHHH, tha lowest-priority intarrupt is being requested.
IC1	35	IN	
IC2	34	IN	
IC3 (LSB)	33	IN	
			MEMORY CONTROL
$\overline{\text{HOLD}}$	64	IN	Hold. Whan active (low), $\overline{\text{HOLD}}$ indicates to tha processor that an external controller (e.g., DMA devica) desires to utilize the address and data buses to transfar data to or from memory. The TMS 9900 antars the hold state following a hold signal when it has completed its prasent memory cycle.* The processor than places tha eddress and data buses in the high-impedance state (along with $\overline{\text{WE}}$, $\overline{\text{MEMEN}}$, and DBIN) end responds with e hold-acknowladge signel (HOLDA). When $\overline{\text{HOLD}}$ is removed, the procassor returns to normal oparation.
HOLDA	5	OUT	Hold acknowledga. Whan activa (high), HOLDA indicatas that the processor is in the hold state and tha address and data busas and memory control outputs ($\overline{\text{WE}}$, $\overline{\text{MEMEN}}$, and DBIN) are in the high-impedance state.
READY	62	IN	Ready. When active (high), READY indicatas that memory will be ready to reed or write during the next clock cycle. When not-ready is indicated during a memory operation, the TMS 9900 enters a wait state and suspends intarnal operation until tha memory systems indicate ready.
WAIT	3	OUT	Wait. When active (high), WAIT indicates thet the TMS 9900 has entered a wait state because of a not-ready condition from memory.

*If the cycle following the present memory cycle is also a memory cycle, it, too, is completed before the TMS9900 enters the hold state. The maximum number of consecutive memory cycles is three.

▶8

TABLE 2 (CONCLUDED)

SIGNATURE	PIN	I/O	DESCRIPTION
			TIMING AND CONTROL
IAO	7	OUT	Instruction acquisition. IAO is active (high) during eny memory cycle when the TMS 9900 is acquiring an instruction. IAQ can be used to detect illegal op codes.
$\overline{\text{LOAD}}$	4	IN	Load. When ective (lowl, $\overline{\text{LOAD}}$ causes the TMS 9900 to axecute a nonmaskable interrupt with memory address $FFFC_{16}$ containing the trap vector (WP and PCl. The load sequence begins after the instruction being executed is completed. $\overline{\text{LOAO}}$ will also terminate an idle state. If $\overline{\text{LOAO}}$ is active during the time $\overline{\text{RESET}}$ is relaased, then tha $\overline{\text{LOAD}}$ trap will occur after the $\overline{\text{RESET}}$ function is completed. $\overline{\text{LOAO}}$ should remain active for one instruction period. IAO can be used to determine instruction boundaries. This signal can be used to implament cold-stert ROM loaders. Additionelly, front-panel routines can be implemented using CRU bits as front-panel-interface signals and softwara-control routines to control the panel operations.
$\overline{\text{RESET}}$	6	IN	Reset. When active (low), $\overline{\text{RESET}}$ causes the processor to be reset and inhibits $\overline{\text{WE}}$ and CRUCLK. When $\overline{\text{RESET}}$ is released, the TMS 9900 then initiates a level-zero interrupt sequence that acquires WP end PC from locations 0000 end 0002, sets ell status register bits to zero, and sterts execution. $\overline{\text{RESET}}$ will also terminata an idle state. $\overline{\text{RESET}}$ must be held active for a minimum of three clock cycles.

2.9 TIMING

2.9.1 MEMORY

A basic memory read and write cycle is shown in Figure 9. The read cycle is shown with no wait states and the write cycle is shown with one wait state.

$\overline{\text{MEMEN}}$ goes active (low) during each memory cycle. At the same time that $\overline{\text{MEMEN}}$ is active, the memory address appears on the address bus bits A0 through A14. If the cycle is a memory-read cycle, DBIN will go active (high) at the same time $\overline{\text{MEMEN}}$ and A0 through A14 become valid. The memory-write signal $\overline{\text{WE}}$ will remain inactive (high) during a read cycle. If the read cycle is also an instruction acquisition cycle, IAQ will go active (high) during the cycle.

The READY signal, which allows extended memory cycles, is shown high during $\phi 1$ of the second clock cycle of the read operation. This indicates to the TMS 9900 that memory-read data will be valid during $\phi 1$ of the next clock cycle. If READY is low during $\phi 1$, then the TMS 9900 enters a wait state suspending internal operation until a READY is sensed during a subsequent $\phi 1$. The memory read data is then sampled by the TMS 9900 during the next $\phi 1$, which completes the memory-read cycle.

At the end of the read cycle, $\overline{\text{MEMEN}}$ and DBIN go inactive (high and low, respectively). The address bus may also change at this time, however, the data bus remains in the input mode for one clock cycle after the read cycle.

A write cycle is similar to the read cycle with the exception that $\overline{\text{WE}}$ goes active (low) as shown and valid write data appears on the data bus at the same time the address appears. The write cycle is shown as an example of a one-wait-state memory cycle. READY is low during $\phi 1$ resulting in the WAIT signal shown.

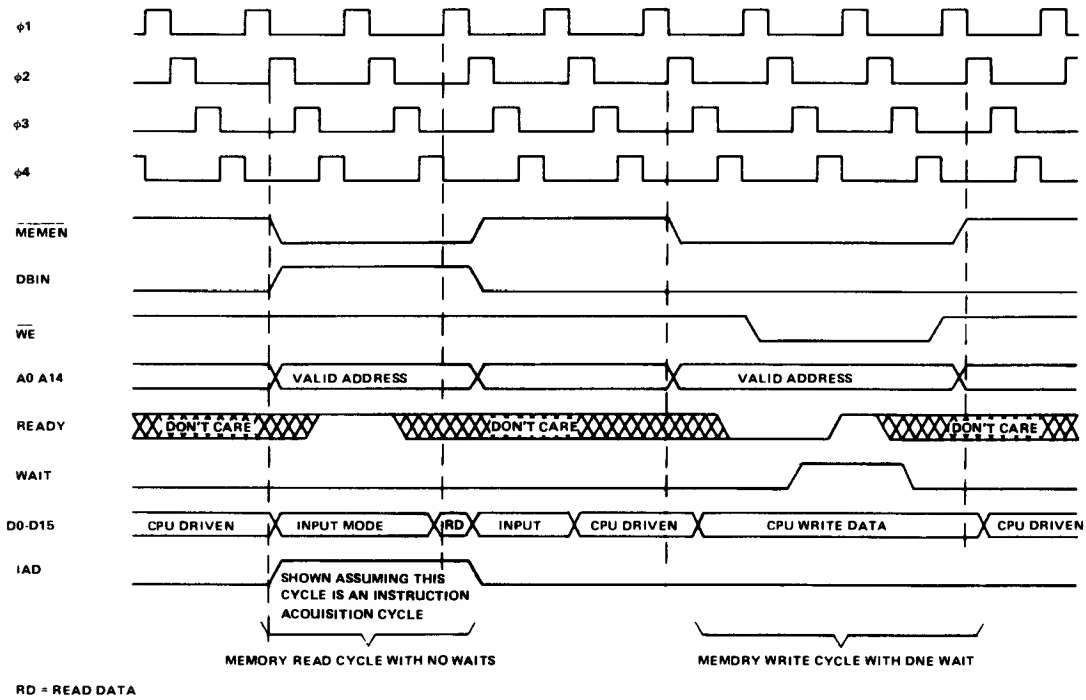
2.9.2 HOLD

Other interfaces may utilize the TMS 9900 memory bus by using the hold operation (illustrated in Figure 10) of the TMS 9900. When $\overline{\text{HOLD}}$ is active (low), tha TMS 9900 entars the hold state at the next available non-memory cycla. Considering that there can be a maximum of three consecutive memory cycles, the maximum delay between $\overline{\text{HOLD}}$ going active to HOLDA going active (high) could be $t_{c(\phi)}$ (for setup) + (6 + 3W) $t_{c(\phi)}$ + $t_{c(\phi)}$ (delay for HOLDA), where W is the number of wait statas per memory cycle and $t_{c(\phi)}$ is the clock cycle time. When the TMS 9900 has entered the hold state, HOLDA goes active (high) and A0 through A15, D0 through D15 DBIN, $\overline{\text{MEMEN}}$, and $\overline{\text{WE}}$ go into a high-impedance state to allow other devices to use the memory buses. When $\overline{\text{HOLD}}$ goes inactive (high), tha TMS 9900 resumes processing as shown. If hold occurs during a CRU operation, the TMS 9900 uses an extra clock cycle (after the removal of the $\overline{\text{HOLD}}$ signal) to reassert the CRU address providing the normal setup times for the CRU bit transfer that was interrupted.

2.9.3 CRU

CRU interface timing is shown in Figure 11. The timing for transferring two bits out and one bit in is shown. These transfers would occur during the execution of a CRU instruction. The other cycles of the instruction execution are not illustrated. To output a CRU bit, the CRU-bit address is placed on the address bus A0 through A14 and the actual bit data on CRUOUT. During the second clock cycle a CRU pulse is supplied by CRUCLK. This process is repeated until the number of bits specified by the instruction are completed.

The CRU input operation is similar in that the bit address appears on A0 through A14. During the subsequent cycle the TMS 9900 accepts the bit input data as shown. No CRUCLK pulses occur during a CRU input operation.



▶8

FIGURE 9 – TMS 9900 MEMORY BUS TIMING

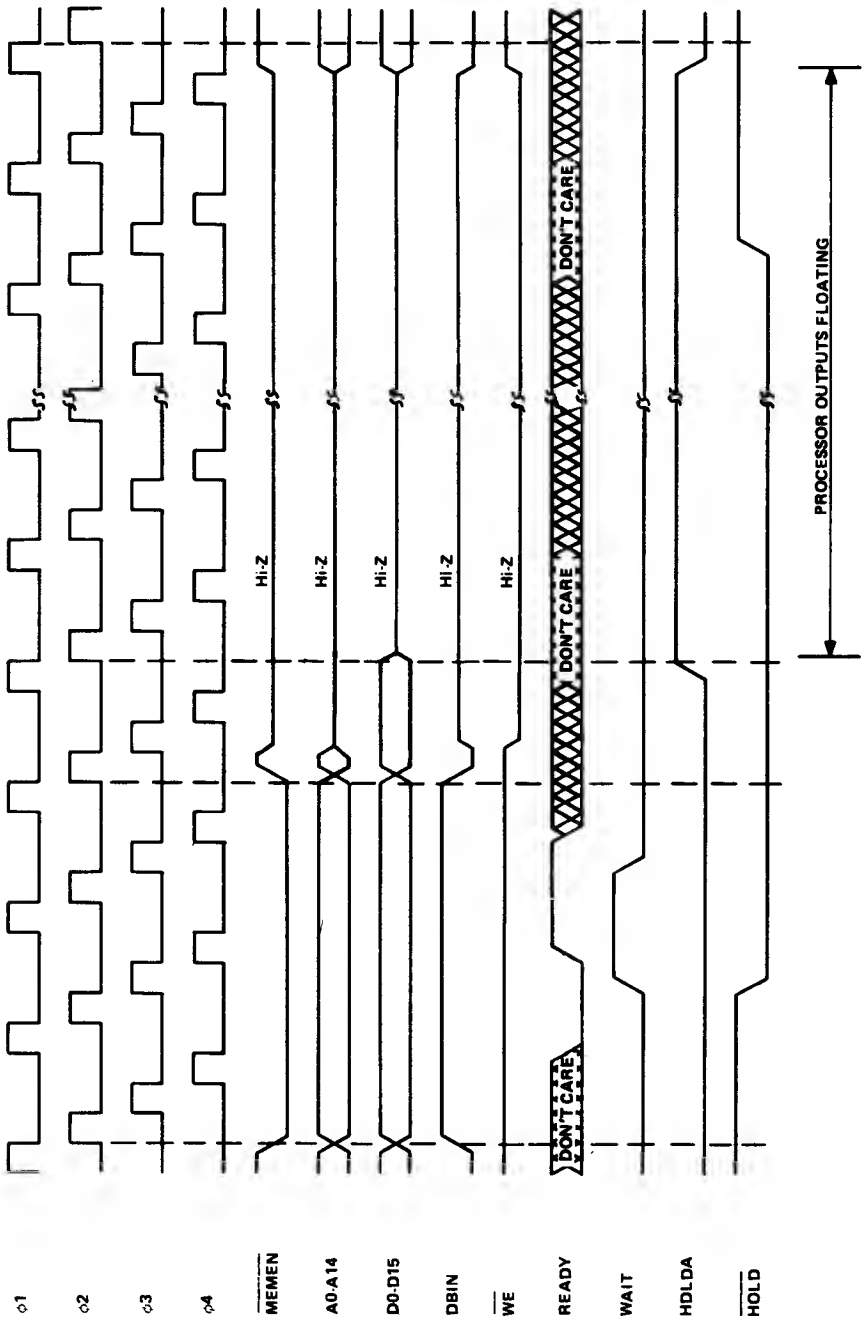


FIGURE 10 — TMS 9900 HOLD TIMING

8

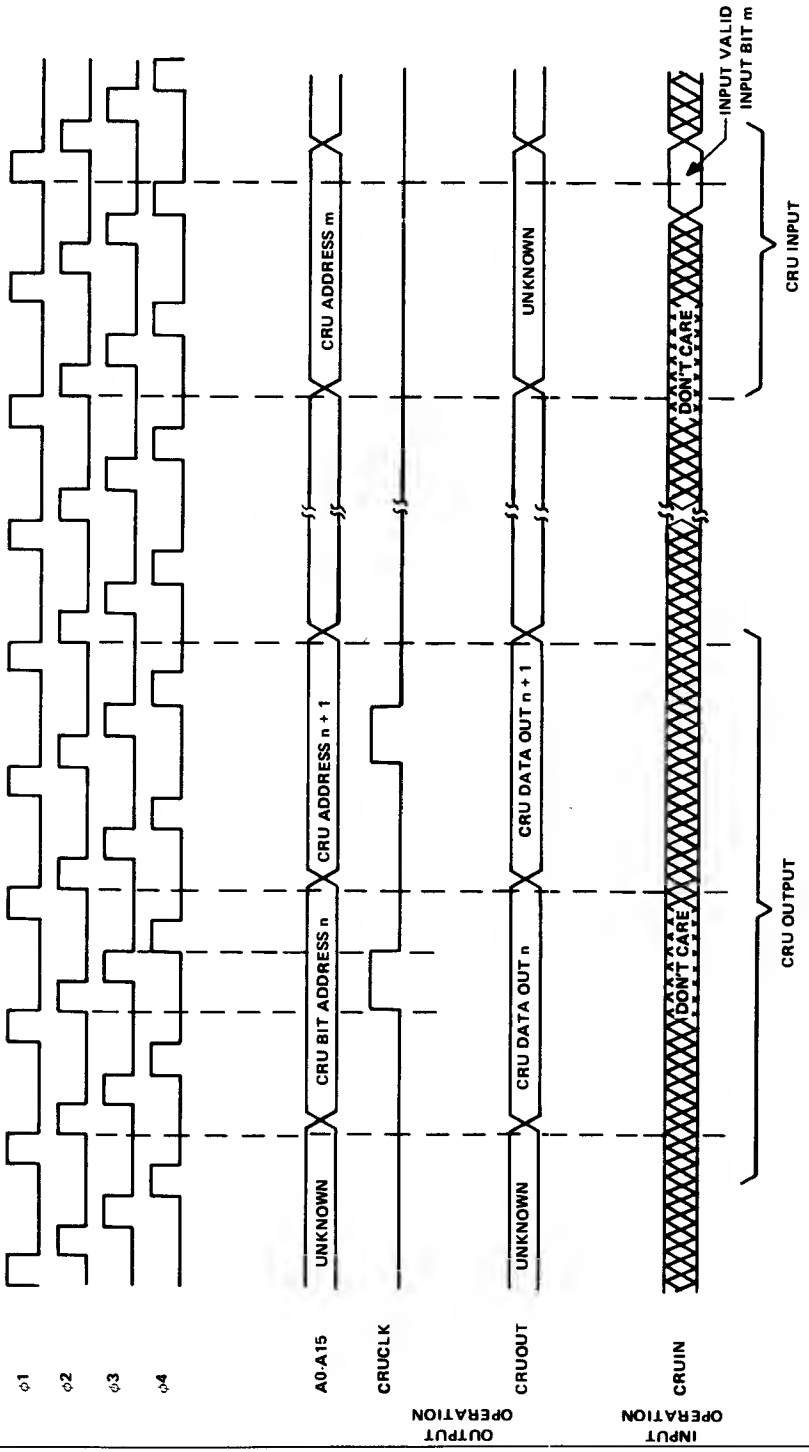


FIGURE 11 – TMS 9900 CRU INTERFACE TIMING

▶8

3.6 TMS 9900 INSTRUCTION EXECUTION TIMES

Instruction execution times for the TMS 9900 are a function of:

1) Clock cycle time, $t_{c(\phi)}$

2) Addressing mode used where operands have multiple addressing mode capability

3) Number of wait states required per memory access.

Table 3 lists the number of clock cycles and memory accesses required to execute each TMS 9900 instruction. For instructions with multiple addressing modes for either or both operands, the table lists the number of clock cycles and memory accesses with all operands addressed in the workspace-register mode. To determine the additional number of clock cycles and memory accesses required for modified addressing, add the appropriate values from the referenced tables. The total instruction-execution time for an instruction is:

$$T = t_c(\phi)(C+(W*M))$$

where:

T = total instruction execution time;
$t_{c(\phi)}$ = clock cycle time;
C = number of clock cycles for instruction execution plus address modification;
W = number of required wait states per memory access for instruction execution plus address modification;
M = number of memory accesses.

TABLE 3
INSTRUCTION EXECUTION TIMES

INSTRUCTION	CLOCK CYCLES C	MEMORY ACCESS M	ADDRESS MODIFICATION† SOURCE	ADDRESS MODIFICATION† DEST
A	14	4	A	A
AB	14	4	B	B
ABS (MSB = 0)	12	2	A	–
(MSB = 1)	14	3	A	–
AI	14	4	–	–
ANDI	14	4	–	–
B	8	2	A	–
BL	12	3	A	–
BLWP	26	5	A	–
C	14	3	A	A
CB	14	3	B	B
CI	14	3	–	–
CKDF	12	1	–	–
CKDN	12	1	–	–
CLR	10	3	A	–
COC	14	3	A	–
CZC	14	3	A	–
DEC	10	3	A	–
DECT	10	3	A	–
DIV (ST4 is set)	16	3	A	–
DIV (ST4 is reset)*	92-124	6	A	–
IDLE	12	1	–	–
INC	10	3	A	–
INCT	10	3	A	–
INV	10	3	A	–
Jump (PC is changed)	10	1	–	–
(PC is not changed)	8	1	–	–
LDCR (C = 0)	52	3	A	–
(1≤C≤8)	20+2C	3	B	–
(9≤C≤15)	20+2C	3	A	–
LI	12	3	–	–
LIMI	16	2	–	–
LREX	12	1	–	–
$\overline{\text{RESET}}$ function	26	5	–	–
$\overline{\text{LOAD}}$ function	22	5	–	–
Interrupt context switch	22	6	–	–

INSTRUCTION	CLOCK CYCLES C	MEMORY ACCESS M	ADDRESS MODIFICATION† SOURCE	ADDRESS MODIFICATION† DEST
LWPI	10	2	–	–
MOV	14	4	A	A
MOVB	14	4	B	B
MPY	52	5	A	–
NEG	12	3	A	–
ORI	14	4	–	–
RSET	12	1	–	–
RTWP	14	4	–	–
S	14	4	A	A
SB	14	4	B	B
SBO	12	2	–	–
SBZ	12	2	–	–
SETD	10	3	A	–
Shift (C≠0)	12+2C	3	–	–
(C=0, Bits 12–15 of WRD=0)	52	4	–	–
(C=0, Bits 12–15 of WRP=N≠0)	20+2N	4	–	–
SOC	14	4	A	A
SOCB	14	4	B	B
STCR (C=0)	60	4	A	–
(1≤C≤7)	42	4	B	–
(C=8)	44	4	B	–
(9≤C≤15)	58	4	A	–
STST	8	2	–	–
STWP	8	2	–	–
SWPB	10	3	A	–
SZC	14	4	A	A
SZCB	14	4	B	B
TB	12	2	–	–
X **	8	2	A	–
XDP	36	8	A	–
XOR	14	4	A	–
Undefined op codes: 0000-01FF, 0320-033F, 0C00-0FFF, 0780-07FF	6	1	–	–

*Execution time is dependent upon the partial quotient after each clock cycle during execution.
**Execution time is added to the execution time of the instruction located at the source address minus 4 clock cycles and 1 memory access time.
†The letters A and B refer to the respective tables that follow.

8

ADDRESS MODIFICATION – TABLE A

ADDRESSING MODE	CLOCK CYCLES C	MEMORY ACCESSES M
WR (T_S or T_D = 00)	0	0
WR indirect (T_S or T_D = 01)	4	1
WR indirect auto-increment (T_S or T_D = 11)	8	2
Symbolic (T_S or T_D = 10, S or D = 0)	8	1
Indexed (T_S or T_D = 10, S or D ≠ 0)	8	2

ADDRESS MODIFICATION – TABLE B

ADDRESSING MODE	CLOCK CYCLES C	MEMORY ACCESSES M
WR (T_S or T_D = 00)	0	0
WR indirect (T_S or T_D = 01)	4	1
WR indirect auto-increment (T_S or T_D = 11)	6	2
Symbolic (T_S or T_D = 10, S or D = 0)	8	1
Indexed (T_S or T_D = 10, S or D ≠ 0)	8	2

As an example, the instruction MOVB is used in a system with $t_{c(\phi)} = 0.333\ \mu s$ and no wait states are required to access memory. Both operands are addressed in the workspace register mode:

$$T = t_c(\phi)(C+(W*M)) = 0.333\ (14+(0*4)) = 4.662\ \mu s$$

If two wait states per memory access were required, the execution time is:

$$T = 0.333\ (14+(2*4))\ \mu s = 7.326\ \mu s.$$

If the source operand was addressed in the symbolic mode and two wait states were required:

$$T = t_c(\phi)(C+(W*M))$$
$$C = 14 + 8 = 22$$
$$M = 4 + 1 = 5$$
$$T = 0.333\ (22+(2*5))\ \mu s = 10.656\ \mu s.$$

4. TMS 9900 ELECTRICAL SPECIFICATIONS

4.1 ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Supply voltage, V_{CC} (see Note 1)	–0.3 to 20 V
Supply voltage, V_{DD} (see Note 1)	–0.3 to 20 V
Supply voltage, V_{SS} (see Note 1)	–0.3 to 20 V
All input voltages (see Note 1)	–0.3 to 20 V
Output voltage (with respect to V_{SS})	–2 V to 7 V
Continuous power dissipation	1.2 W
Operating free-air temperature range	0°C to 70°C
Storage temperature range	–55°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: Under absolute maximum ratings voltage values are with respect to the most negative supply, V_{BB} (substrate), unless otherwise noted. Throughout the remainder of this section, voltage values are with respect to V_{SS}.

4.2 RECOMMENDED OPERATING CONDITIONS

	MIN	NOM	MAX	UNIT
Supply voltage, V_{BB}	−5.25	−5	−4.75	V
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply voltage, V_{DD}	11.4	12	12.6	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH} (all inputs except clocks)	2.2	2.4	V_{CC}+1	V
High-level clock input voltage, $V_{IH(\phi)}$	V_{DD}−2		V_{DD}	V
Low-level input voltage, V_{IL} (all inputs except clocks)	−1	0.4	0.8	V
Low-level clock input voltage, $V_{IL(\phi)}$	−0.3	0.3	0.6	V
Operating free-air temperature, T_A	0		70	°C

4.3 ELECTRICAL CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (UNLESS OTHERWISE NOTED)

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
I_I	Input current	Data bus during DBIN	$V_I = V_{SS}$ to V_{CC}		±50	±100	μA
		$\overline{WE}$, $\overline{MEMEN}$, DBIN, Address bus, Data bus during HOLDA	$V_I = V_{SS}$ to V_{CC}		±50	±100	
		Clock	V_I = −0.3 to 12.6 V		±25	±75	
		Any other inputs	$V_I = V_{SS}$ to V_{CC}		±1	±10	
V_{OH}	High-level output voltage		I_O = −0.4 mA	2.4		V_{CC}	V
V_{OL}	Low-level output voltage		I_O = 3.2 mA			0.65	V
			I_O = 2 mA			0.50	
$I_{BB(av)}$	Supply current from V_{BB}				0.1	1	mA*
$I_{CC(av)}$	Supply current from V_{CC}				50	75	mA*
$I_{DD(av)}$	Supply current from V_{DD}				25	45	mA*
C_i	Input capacitance (any inputs except clock and data bus)		V_{BB} = −5, f = 1MHz, unmeasured pins at V_{SS}		10	15	pF
$C_{i(\phi1)}$	Clock-1 input capacitance		V_{BB} = −5, f = 1MHz, unmeasured pins at V_{SS}		100	150	pF
$C_{i(\phi2)}$	Clock-2 input capacitance		V_{BB} = −5, f = 1MHz, unmeasured pins at V_{SS}		150	200	pF
$C_{i(\phi3)}$	Clock-3 input capacitance		V_{BB} = −5, f = 1MHz, unmeasured pins at V_{SS}		100	150	pF
$C_{i(\phi4)}$	Clock-4 input capacitance		V_{BB} = −5, f = 1MHz, unmeasured pins at V_{SS}		100	150	pF
C_{DB}	Data bus capacitance		V_{BB} = −5, f = 1MHz, unmeasured pins at V_{SS}		15	25	pF

8

†All typical values are at T_A = 25°C and nominal voltages.
*D.C. Component of Operating Clock

4.4 TIMING REQUIREMENTS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS

PARAMETER		MIN	NOM	MAX	UNIT
$t_{c(\phi)}$	Clock cycle time	300	333	500	ns
$t_{r(\phi)}$	Clock rise time	5	12		ns
$t_{f(\phi)}$	Clock fall time	10	12		ns
$t_{w(\phi)}$	Clock pulse width, high level	40	45	100	ns
$t_{s(\phi)}$	Clock spacing, time between any two adjacent clock pulses	0	5		ns
$t_{d(\phi)}$	Time between rising edge valid any two adjacent clock pulses	73	83		ns
t_{su}	Data or control setup time before clock 1	30			ns
t_h	Data hold time after clock 1	10			ns

4.5 SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH} (B) or t_{PHL} (B)	All other outputs	C_L = 200 pF		20	40	ns
t_{PLH} (C) or t_{PHL} (C)	Propagation delay CRUCLK, WE, MEMEN, WAIT, DBIN				30	ns

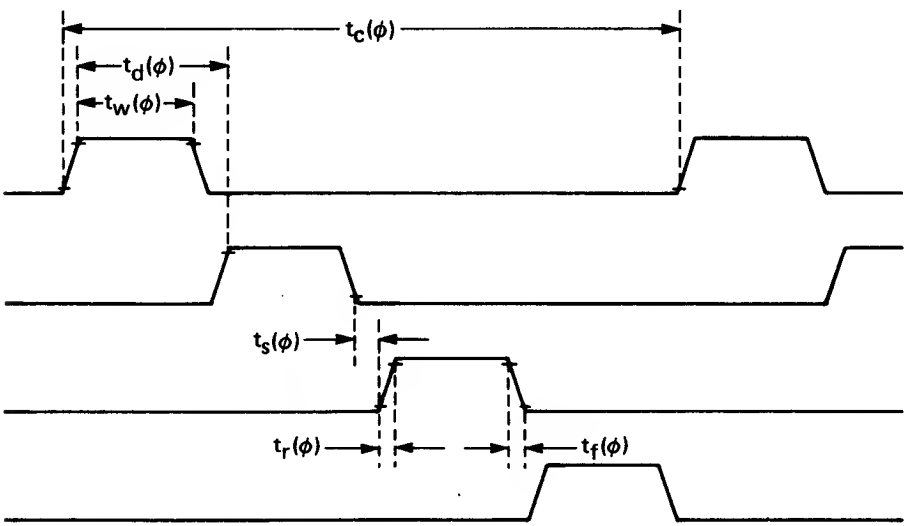


FIGURE 12 – CLOCK TIMING

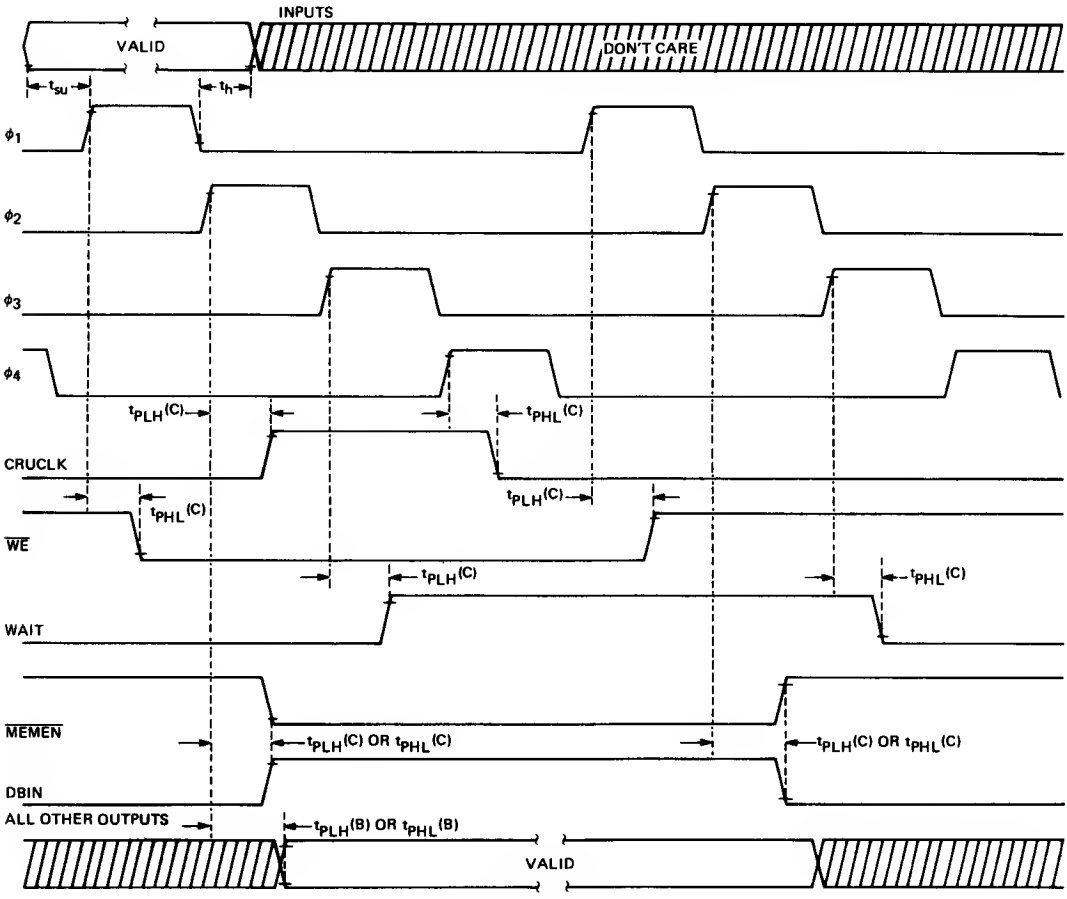


FIGURE 13—SIGNAL TIMING

8

TMS 9900-40 ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Supply voltage, V_{CC} (see Note 1) . . . . . −0.3 to 20 V
Supply voltage, V_{DD} (see Note 1) . . . . . −0.3 to 20 V
Supply voltage, V_{SS} (see Note 1) . . . . . −0.3 to 20 V
All input voltages (see Note 1) . . . . . −0.3 to 20 V
Output voltage (with respect to V_{SS}) . . . . . −2 V to 7 V
Continuous power dissipation . . . . . 1.2 W
Operating free-air temperature range . . . . . 0°C to 70°C
Storage temperature range . . . . . −55°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may effect device reliability.

NOTE 1: Under absolute maximum ratings voltage values are with respect to the most negative supply, V_{BB} (substrate), unless otherwise noted. Throughout the remainder of this section, voltage values are with respect to V_{SS}.

RECOMMENDED OPERATING CONDITIONS

	MIN	NOM	MAX	UNIT
Supply voltage, V_{BB}	−5.25	−5	−4.75	V
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply Voltage, V_{DD}	11.4	12	12.6	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH} (all inputs except clocks)	2.2	2.4	$V_{CC}+1$	V
High-level clock input voltage, $V_{IH(\phi)}$	$V_{DD}-2$		V_{DD}	V
Low-level input voltage, V_{IL} (all inputs except clocks)	−1	0.4	0.8	V
Low-level clock input voltage, $V_{IL(\phi)}$	−0.3	0.3	0.6	V
Operating free-air temperature, T_A	0		70	°C

▶8

DESIGN GOAL

This document describes the design specifications for a product under development. Texas Instruments reserves the right to change these specifications in any manner, without notice.

ELECTRICAL CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (UNLESS OTHERWISE NOTED)

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
I_I	Input current	Data bus during OBIN	$V_I = V_{SS}$ to V_{CC}		±50	±100	μA
		WE, MEMEN, OBIN, Address bus, Data bus during HOLDA	$V_I = V_{SS}$ to V_{CC}		±50	±100	
		Clock*	$V_I = -0.3$ to 12.6 V		±25	±75	
		Any other inputs	$V_I = V_{SS}$ to V_{CC}		±1	±10	
V_{OH}	High-level output voltage		$I_O = -0.4$ mA	2.4		V_{CC}	mA
V_{OL}	Low-level output voltage		$I_O = 3.2$ mA			0.65	mA
			$I_O = 2$ mA			0.50	mA
I_{BB}(av)	Supply current from V_{BB}				0.1	1	pF
I_{CC}(av)	Supply current from V_{CC}				50	75	
I_{DD}(av)	Supply current from V_{DD}				25	45	pF
C_I	Input Capacitance (any inputs except clock and data bus)		f = 1MHz, V_{BB} = -5 V, unmeasured pins at V_{SS}		10	15	pF
$C_{I(\phi 1)}$	Clock-1 input capacitance		f = 1 MHz, V_{BB} = -5 V, unmeasured pins at V_{SS}		100	150	pF
$C_{I(\phi 2)}$	Clock-2 input capacitance		f = 1 MHz, V_{BB} = -5 V, unmeasured pins at V_{SS}		150	200	pF
$C_{I(\phi 3)}$	Clock-3 input capacitance		f = 1 MHz, V_{BB} = -5 V, unmeasured pins at V_{SS}		100	150	pF
$C_{I(\phi 4)}$	Clock-4 input capacitance		f = 1 MHz, V_{BB} = -5 V, unmeasured pins at V_{SS}		100	150	
C_{DB}	Data bus capacitance		f = 1 MHz, V_{BB} = -5 V, unmeasured pins at V_{SS}		15	25	pF

†All typical values are at $T_A = 25°C$ and nominal voltages.
*O.C. component of operating clock.

TIMING REQUIREMENTS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS

PARAMETER		MIN	NOM	MAX	UNIT
$t_c(\phi)$	Clock cycle time	2.40	2.50		ns
$t_r(\phi)$	Clock rise time	5	12		ns
$t_f(\phi)$	Clock fall time	10	12		ns
$t_w(\phi)$	Pulse width, high level	33			ns
$t_s(\phi)$	Clock spacing, time between any two adjacent clock pulses	0	45		ns
$t_D(o)$	Time between rising edges, valid between any two adjacent clock pulses	55	63		ns
t_{su}	Data or control setup time before clock $\phi 1$	25			ns
t_h	Data hold time after clock $\phi 1$	10			ns

8

SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}(C) or t_{PHL}(C) propagation delay CRUCLK, $\overline{WE}$, $\overline{MEMEN}$, WAIT, OBIN	C_L = 200pF		20	20	ns
t_{PLH}(B) or t_{PHL}(B) all other outputs			30	30	ns

DESIGN GOAL
See page 8-28 for Design Goal

SBP 9900A

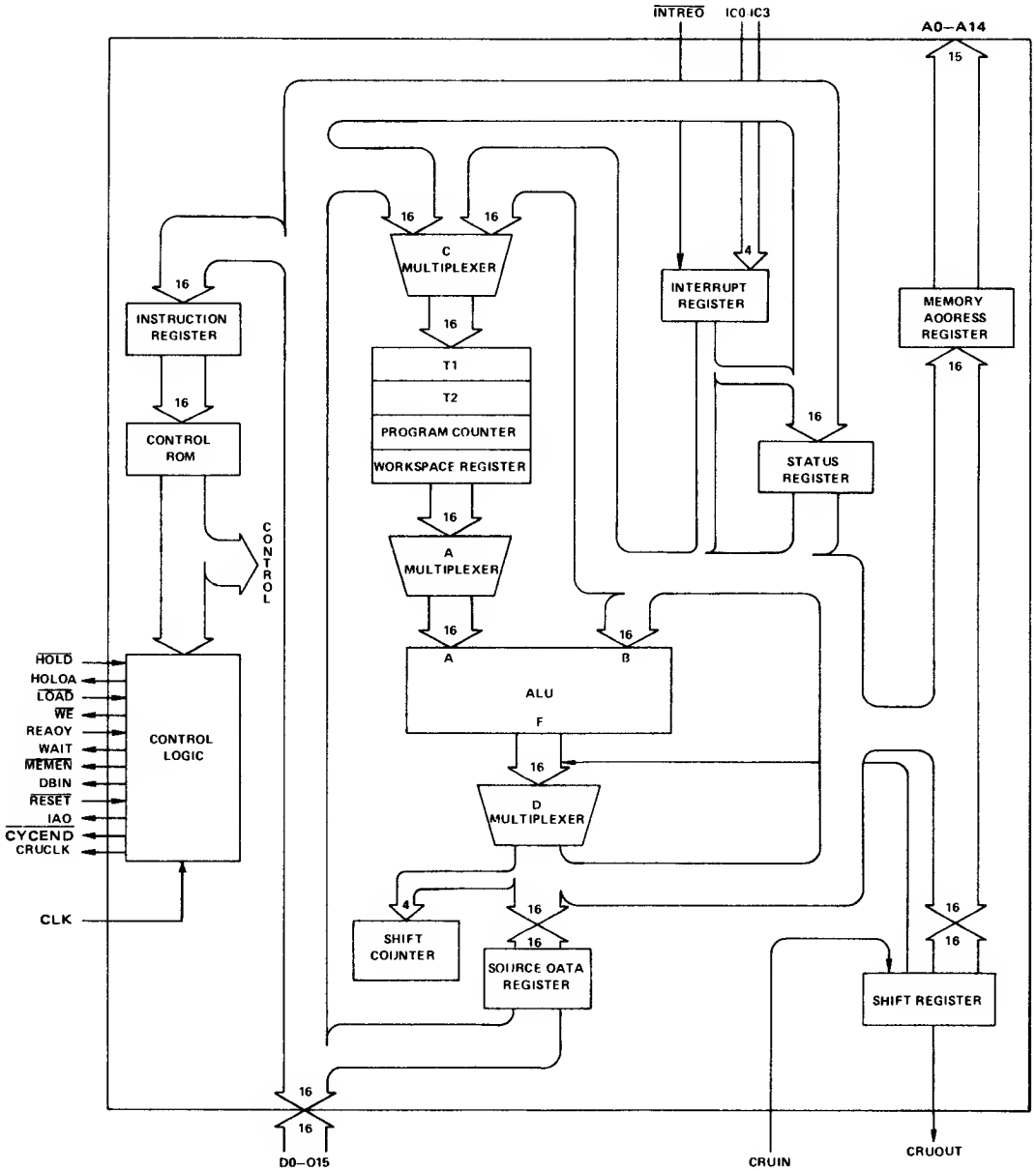


Figure 1. SBP 9900A Architecture.

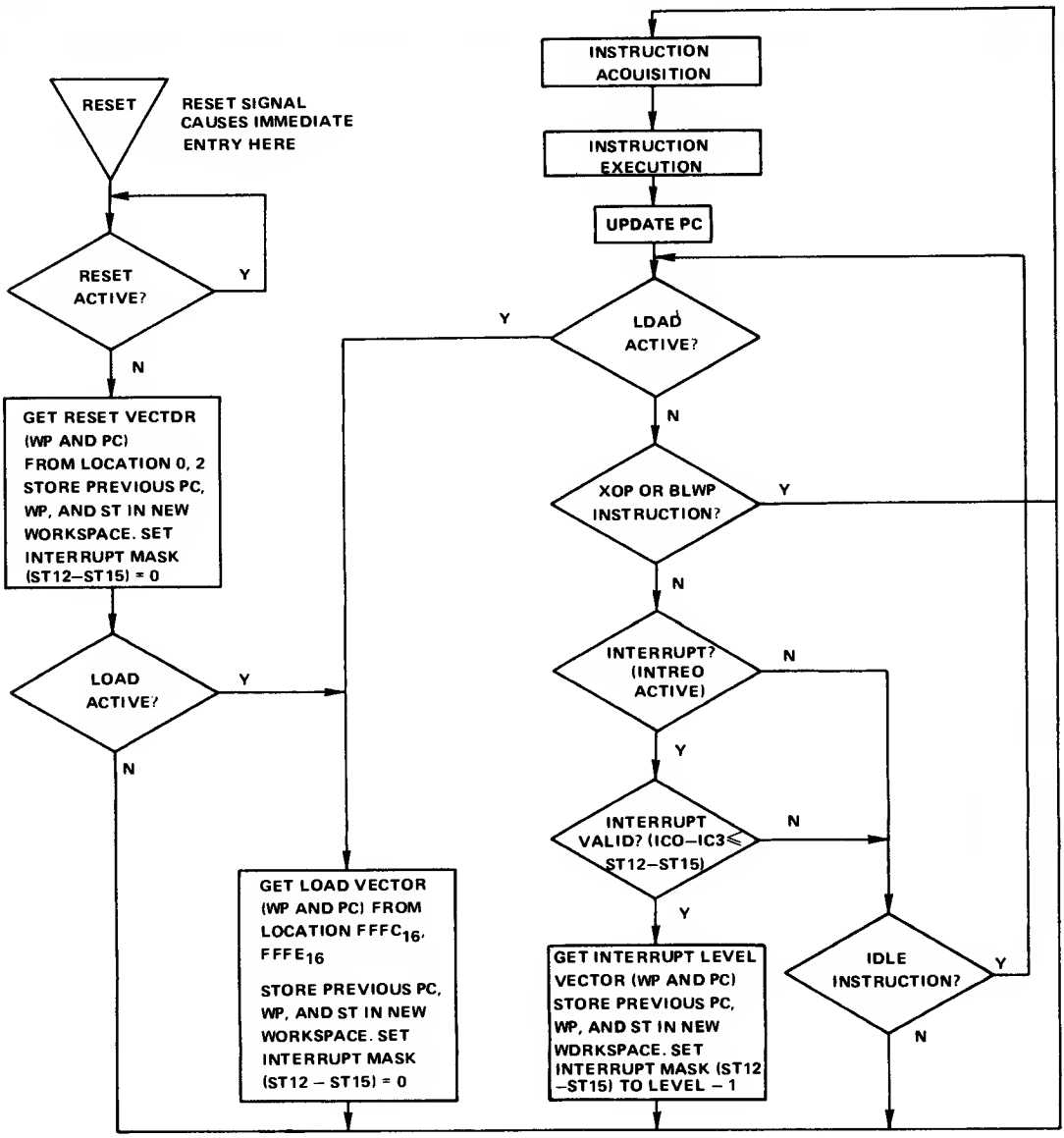
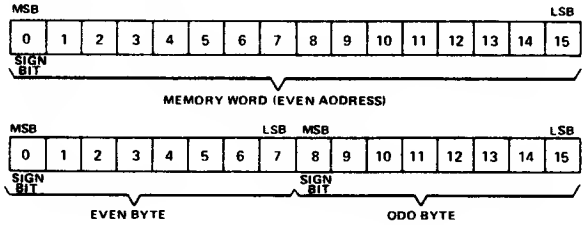


Figure 2. 9900 CPU Flow Chart

ARCHITECTURE

The Memory word of the 9900 is 16 bits long. Each word is also defined as 2 bytes of 8 bits. The instruction set of the 9900 allows both word and byte operands. Thus, all memory locations are on even address boundaries and byte instructions can address either the even or odd byte. The memory space is 65,536 bytes or 32,768 words. The word and byte formats are shown below.

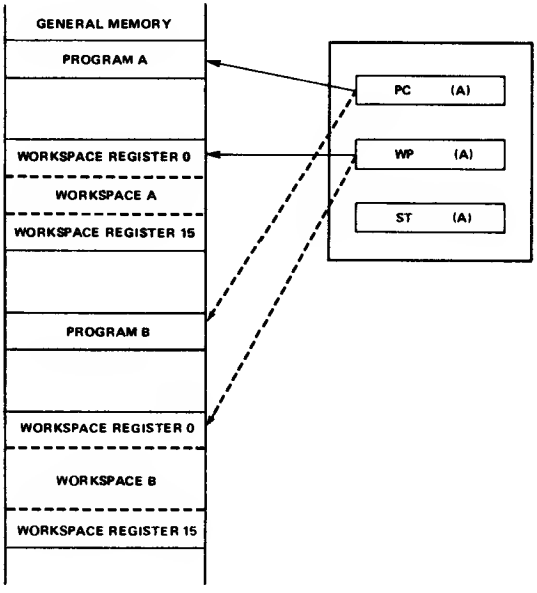


Registers and Memory

The 9900 family employs an advanced memory-to-memory architecture. Blocks of memory designated as workspace replace internal-hardware registers with program-data registers.

Three internal registers are accessible to the user. The program counter (PC) contains the address of the instruction following the current instruction being executed. This address is referenced by the processor to fetch the next instruction from memory and is then automatically incremented. The status register (ST) contains the present state of the processor. The workspace pointer (WP) contains the address of the first word in the currently active set of workspace registers.

A workspace-register file occupies 16 contiguous memory words in the general memory area (see *Figure 3).* Each workspace register may hold data or addresses and function as operand registers, accumulators, address registers, or index registers. During instruction execution, the processor addresses any given register in the workspace by adding the register number to the contents of the workspace pointer and initiating a memory request for the word. The relationship between the workspace pointer and its corresponding workspace is shown below.



8◀

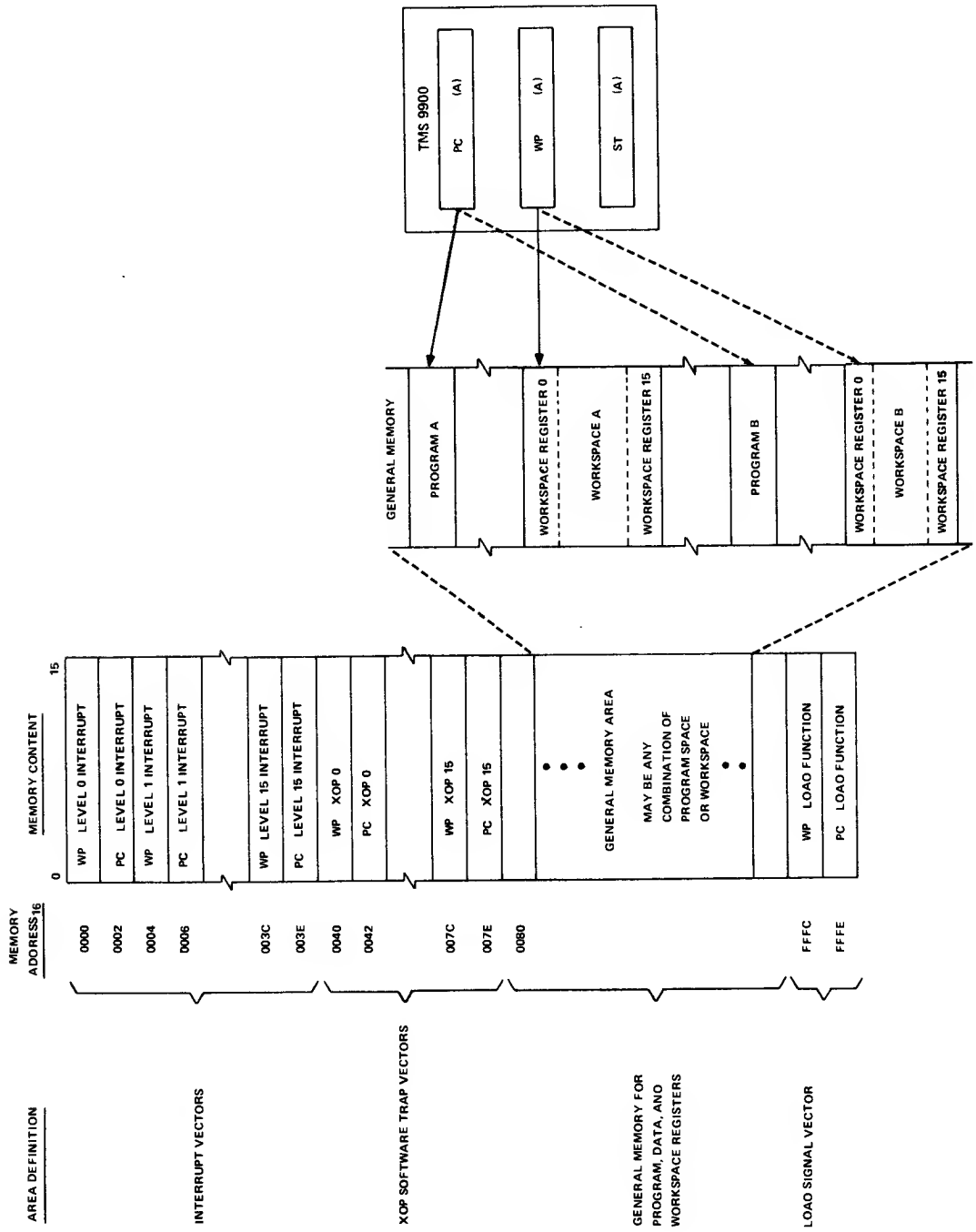


Figure 3. Memory Map

INTERRUPTS

The architecture of the 9900 family allows vectoring of 16 interrupts. These interrupts are assigned levels from 0 to 15. The interrupt at level 0 has the highest priority and the interrupt at level 15 has the lowest priority. The 9900 implements all 16 interrupt levels. Level 0 is reserved for $\overline{\text{RESET}}$ function.

The 9900 continuously compares the interrupt code with the interrupt mask contained in the status-register. When the level of the pending interrupt is less than or equal to the enabling mask level (higher or equal priority interrupt), the processor recognizes the interrupt and initiates a context switch following completion of the currently executing instruction. The processor fetches the new context WP and PC from the interrupt vector locations. Then, the previous context WP, PC, and ST are stored in workspace registers 13, 14, and 15, respectively, of the new workspace. The 9900 then forces the interrupt mask to a value that is one less than the level of the interrupt being serviced, except for level-zero interrupt, which loads zero into the mask. This allows only interrupts of higher priority to interrupt a service routine. The processor also inhibits interrupts until the first instruction of the service routine has been executed to preserve program linkage should a higher priority interrupt occur. All interrupt requests should remain active until recognized by the processor in the device-service routine. The individual service routines must reset the interrupt requests before the routine is complete.

If a higher priority interrupt occurs, a second context switch occurs to service the higher priority interrupt. When that routine is complete, a return instruction (RTWP) restores the first service routine parameters to the processor to complete processing of the lower-priority interrupt. All interrupt subroutines should terminate with the return instruction to restore original program parameters. The interrupt-vector locations, device assignment, enabling-mask value, and the interrupt code are shown in *Table 1.*

Table 1. Interrupt Level Data

Interrupt Level	Vector Location (Memory Address In Hex)	Device Assignment	Interrupt Mask Values To Enable Respective Interrupts (ST12 thru ST15)	Interrupt Codes IC0 thru IC3
(Highest priority) 0	00	Reset	0 through F*	0000
1	04	External device	1 through F	0001
2	08	↓	2 through F	0010
3	0C	↓	3 through F	0011
4	10	↓	4 through F	0100
5	14	↓	5 through F	0101
6	18	↓	6 through F	0110
7	1C	↓	7 through F	0111
8	20	↓	8 through F	1000
9	24	↓	9 through F	1001
10	28	↓	A through F	1010
11	2C	↓	B through F	1011
12	30	↓	C through F	1100
13	34	↓	D through F	1101
14	38	↓	E and F	1110
(Lowest priority) 15	3C	External device	F only	1111

* Level 0 cen not be disebled.

The 9900 interrupt interface utilizes the TMS 9901 Programmable Systems Interface as shown in *Figure 4.*

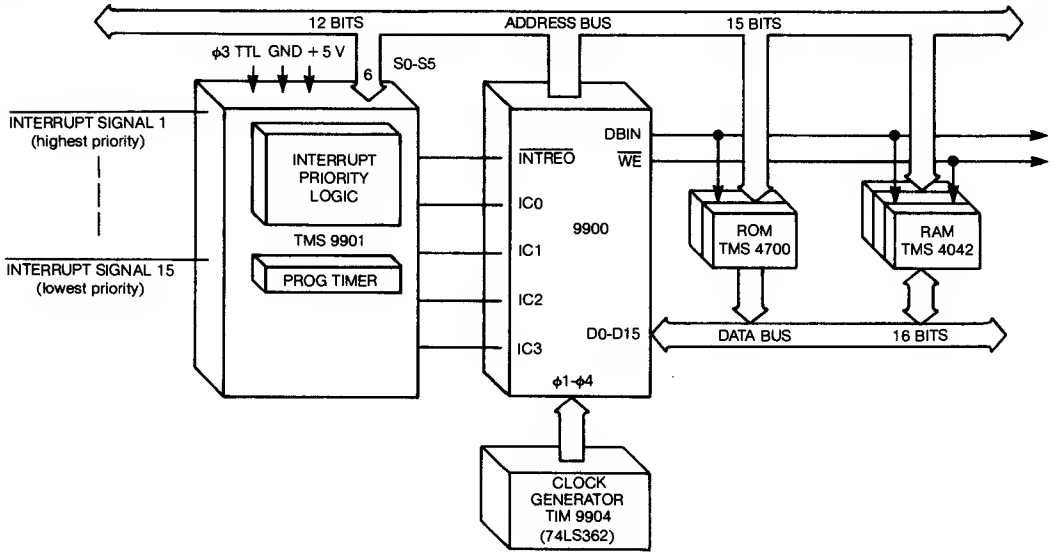


Figure 4. 9900 Interrupt Interface

Input/Output

The 9900 Utilizes a versatile direct command-driven I/O interface designated as the communications-register unit (CRU). The CRU provides up to 4096 directly addressable input bits and 4096 directly addressable output bits. Both input and output bits can be addressed individually or in fields of from 1 to 16 bits. The 9900 employs three dedicated I/O pins (CRUIN, CRUOUT, and CRUCLK) and certain bits of the address bus to interface with the CRU system. The processor instructions that drive the CRU interface can set, reset, or test any bit in the CRU array or move between memory and CRU data fields.

▶8

Single-Bit CRU Operations

The 9900 performs three single-bit CRU functions: test bit (TB), set bit to one (SBO), and set bit to zero (SBZ). To identify the bit to be operated upon, the 9900 develops a CRU-bit address and places it on the address bus, A3 to A14.

For the two output operations (SBO and SBZ), the processor also generates a CRUCLK pulse, indicating an output operation to the CRU device, and places bit 7 of the instruction word on the CRUOUT line to accomplish the specified operation (bit 7 is a one for SBO and a zero for SBZ). A test-bit instruction transfers the addressed CRU bit from the CRUIN input line to bit 2 of the status register (EQUAL).

The 9900 develops a CRU-bit address for the single-bit operations from the software base address contained in workspace register 12 and the signed displacement count contained in bits 8 through 15 of the instruction. The displacement allows two's complement addressing from base minus 128 bits through base plus 127 bits. The hardware base address, bits 3 through 14 of WR12, is added to the signed displacement specified in the instruction and the result is loaded onto the address bus. *Figure 5* illustrates the development of a single-bit CRU address.

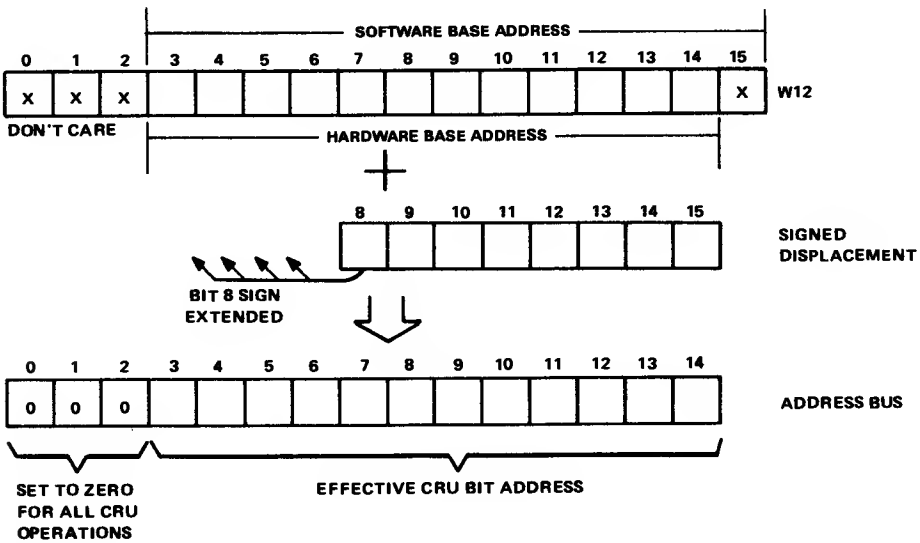


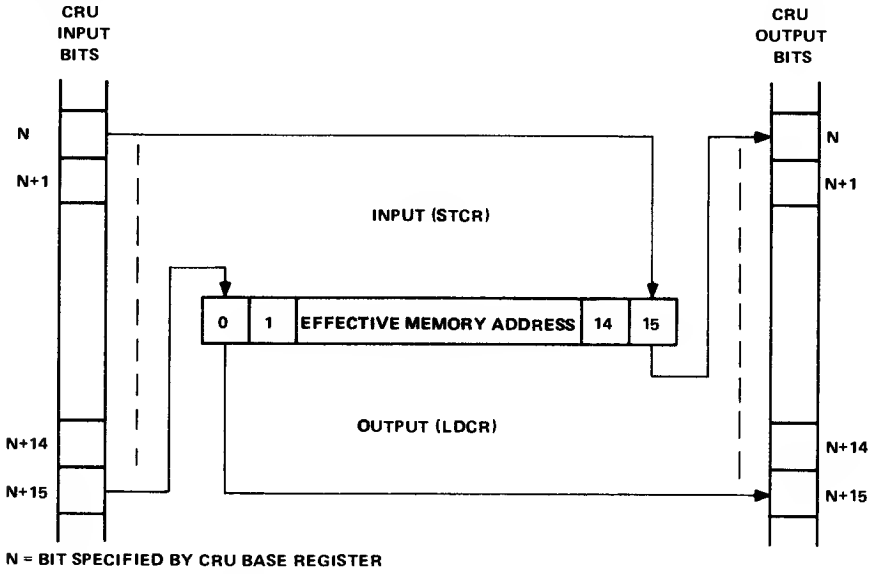
Figure 5. Single-Bit CRU Address Development

8

Multiple-Bit CRU Operations

The 9900 performs two multiple-bit CRU operations: store communications register (STCR) and load communications register (LDCR). Both operations perform a data transfer from the CRU-to-memory or from memory-to-CRU as illustrated in *Figure 6.* Although the figure illustrates a full 16-bit transfer operation, any number of bits from 1 to 16 may be involved. The LDCR instruction fetches a word from memory and right-shifts it to serially transfer it to CRU output bits. If the LDCR involves eight or fewer bits, those bits come from the right-justified field within the addressed byte of the memory word. If the LDCR involves nine or more bits, those bits come from the right-justified field within the whole memory. When transferred to the CRU interface, each successive bit receives an address that is sequentially greater than the address for the previous bit. This addressing mechanism results in an order reversal of the bits, that is, bit 15 of the memory word (or bit 7) becomes the lowest addressed bit in the CRU and bit 0 becomes the highest addressed bit in the CRU field.

An STRC instruction transfers data from the CRU to memory. If the operation involves a byte or less transfer, the transferred data will be stored right-justified in the memory byte with leading bits set to zero. If the operation involves from nine to 16 bits, the transferred data is stored right-justified in the memory word with leading bits set to zero.



▶8

Figure 6. 9900 LDCR/STRC Data Transfers

When the input from the CRU device is complete, the first bit from the CRU is the least-significant-bit position in the memory word or byte.

Figure 7 illustrates how to implement a 16-bit input and a 16-bit output register in the CRU interface using the TMS 9901. CRU addresses are decoded as needed to implement up to 256 such 16-bit interface registers. In system application, however, only the exact number of interface bits needed to interface specific peripheral devices are implemented. It is not necessary to have a 16-bit interface register to interface an 8-bit device.

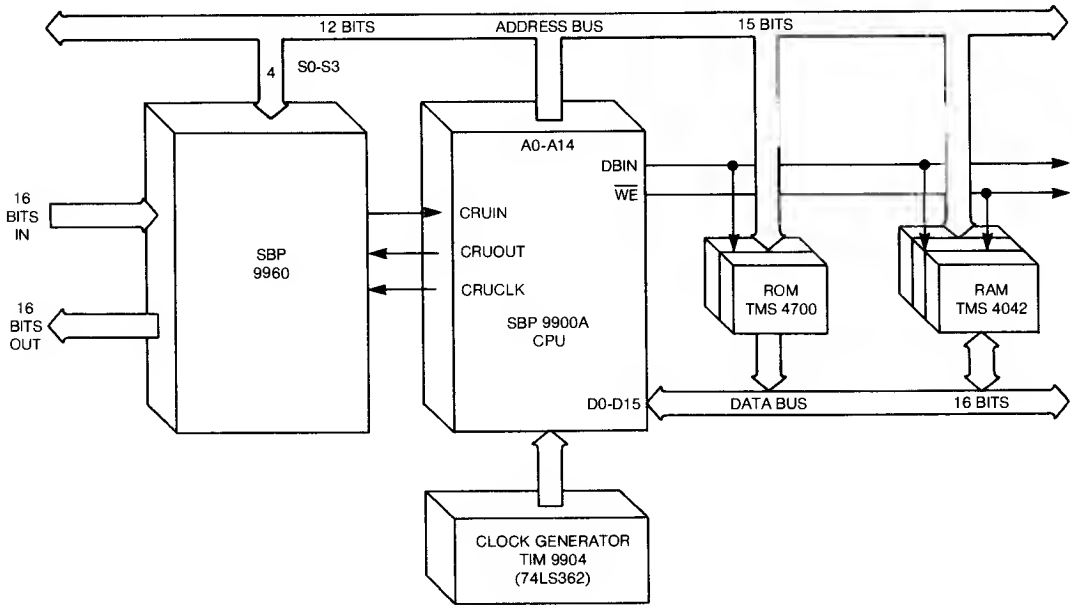


Figure 7. SBP 9900A 16-bit Input/Output Interface

External Instructions

The 9900 has five external instructions that allow user-defined external functions to be initiated under program control. These instructions are CKON, CKOF, RSET, IDLE, and LREX. These mnemonics, except for IDLE, relate to functions implemented in the 9900 minicomputer and do not restrict use of the instructions to initiate various user-defined functions. IDLE also causes the 9900 to enter the idle state and remain until an interrupt, $\overline{\text{RESET}}$, or $\overline{\text{LOAD}}$ occurs. When any of these five instructions are executed by the 9900, a unique 3-bit code appears on the address bus along with a CRUCLK pulse. The user must provide external hardware to decode this 3 bit code and implement this external function. When the 9900 is in an idle state, the 3-bit code and CRUCLK pulses occur repeatedly until the idle state is terminated. The codes are:

EXTERNAL INSTRUCTION	A0	A1	A2
LREX	H	H	H
CKOF	H	H	L
CKON	H	L	H
RSET	L	H	H
IDLE	L	H	L

Figure 8 illustrates typical external decode logic to implement these instructions. Note CRUCLK to the CRU is inhibited during external instructions.

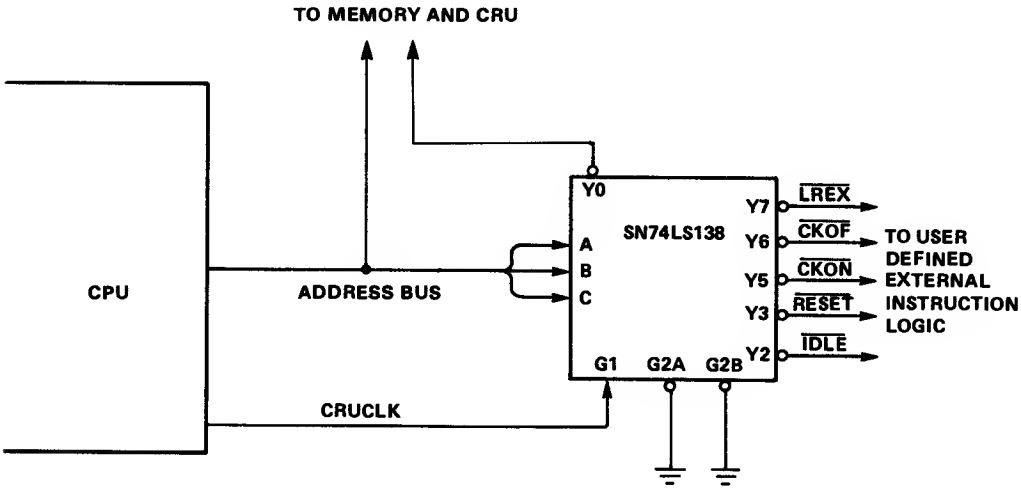


Figure 8. External Instruction Decode Logic

Load Function

The $\overline{\text{LOAD}}$ signal allows cold-start ROM loaders and front panels to be implemented for the 9900. When active, $\overline{\text{LOAD}}$ causes the 9900 to initiate an interrupt sequence immediately following the instruction being executed. A memory location is used to obtain the vector (WP and PC). The old PC, WP and ST are loaded into the new workspace and the interrupt mask is set to 0000. Then, program execution resumes using the new PC and WP.

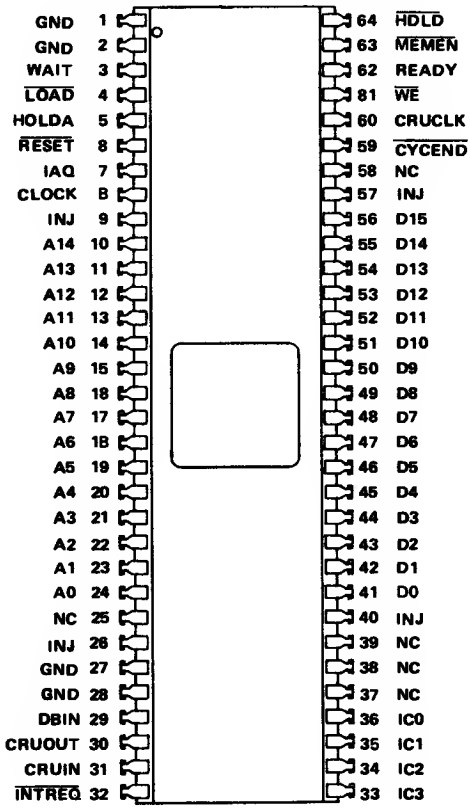
SBP 9900A PIN DESCRIPTION

Table 2 describes the function of each SBP 9900A pin, and *Figure 9* illustrates their assigned locations.

Table 2. 9900 Pin Assignments and Functions

SIGNATURE	PIN	I/O	DESCRIPTION
			ADDRESS BUS
A0 (MSB)	24	DUT	A0 (MSB) through A14 (LSB) comprise the address bus. This open-collector bus provides the memory-address vector to the external-memory system when $\overline{\text{MEMEN}}$ is active, and I/O-bit addresses to the I/O system when $\overline{\text{MEMEN}}$ is inactive. When HOLDA is active, the address bus is pulled to the logic level HIGH state by the individual pull-up resistors tied to each respective open-collector output.
A14 (LSB)	10	DUT	
			DATA BUS
D0 (MSB)	41	I/O	D0 (MSB) through D15 (LSB) comprise the bidirectional open-collector data bus. This bus transfers memory data to (when writing) and from (when reading) the external-memory system when $\overline{\text{MEMEN}}$ is active. When HOLDA is active, the data bus is pulled to the logic level HIGH state by the individual pull-up resistors tied to each respective open-collector output.
D15 (LSB)	56	I/O	
			POWER SUPPLY
INJ	9		Injector-Supply-Current
INJ	26		Injector-Supply-Current
INJ	40		Injector-Supply-Current
INJ	57		Injector-Supply-Current
GND	1		Ground Reference
GND	2		Ground Reference
GND	27		Ground Reference
GND	28		Ground Reference
			CLOCK
CLOCK	8	IN	CLOCK
			BUS CONTROL
DBIN	29	OUT	DATA BUS IN. When active (pulled to logic level HIGH), DBIN indicates that the SBP 9900A has disabled its output buffers to allow the memory to place memory-read data on the data bus during $\overline{\text{MEMEN}}$. DBIN remains at logic level LOW in all other cases except when HOLDA is active (pulled to logic level HIGH).
$\overline{\text{MEMEN}}$	63	DUT	MEMORY ENABLE. When active (logic level LOW), $\overline{\text{MEMEN}}$ indicates that the address bus contains a memory address.
$\overline{\text{WE}}$	61	OUT	WRITE ENABLE. When active (logic level LOW), $\overline{\text{WE}}$ indicates that the SBP 9900A data bus is outputting data to be written into memory.

Figure 9. SBP 9900A Pin Assignments.



NC—No internal connection

8

Table 2. (Continued)

SIGNATURE	PIN	I/O	DESCRIPTION
CRUCLK	60	OUT	COMMUNICATIONS-REGISTER-UNIT (CRU) CLOCK. When active (pullad to logic level HIGH), CRUCLK indicates to the axternal interfaca logic the presence of output deta on CRUOUT, or the prasance of en encoded external instruction on A0 through A2.
CRUIN	31	IN	CRU DATA IN. CRUIN, normally driven by 3-stata or open-collector devicas, receives input data from the external interface logic. When the SBP 9900A executes a STCR or TB instruction, it samples CRUIN for the level of the CRU input bit spacifiad by the address bus (A3 through A14).
CRUOUT	30	OUT	CRU DATA OUT. CRUOUT outputs seriel date when the SBP 9900A executes a LDCR, SBZ, SBO instruction. The data on CRUOUT should be sampled by the external intarface logic when CRUCLK goes active (pullad to logic level HIGH).
			INTERRUPT CONTROL
$\overline{\text{INTREQ}}$	32	IN	INTERRUPT REQUEST. Whan ective (logic level LOW), $\overline{\text{INTREQ}}$ indicates thet an external interrupt is requesting service. If $\overline{\text{INTREQ}}$ is activa, the SBP 9900A loads the data on tha interrupt-coda input-lines IC0 through IC3 into tha internel interrupt-coda storage register. Tha code is then compered to the interrupt mask bits of the status register. If equal or higher priority then the anablad interrupt level (intarrupt code equal or less than status register bits 12 through 15), the SBP 99DDA initiatas the intarrupt sequence. If tha comparison fails, tha SBP 9900A ignores the intarrupt request. In that case, $\overline{\text{INTREQ}}$ should be held active. The SBP 9900A will continua to sample IC0 through IC3 until the progrem enables a sufficiently low interrupt-leval to eccept the requasting intarrupt.
IC0 (MSB)	36	IN	INTERRUPT COOES IC0 (MSB) through IC3 (LSB), receiving an interrupt idantify code, are samplad by the SBP 99DDA when $\overline{\text{INTREQ}}$ is active (logic laval LOW). Whan IC0 through IC3 are LLLH, tha higher priority *external* interrupt is requesting service; when HHHH, the lowest priority external interrupt is requesting service.
IC0 (LSB)	33	IN	
			MEMORY CONTROL
$\overline{\text{HOLD}}$	64	IN	When activa (logic level LOW), $\overline{\text{HOLD}}$ indicates to tha SBP 9900A thet an external controller (e.g., DMA device) desires to use both the address bus and data bus to transfer dete to or from memory. In rasponse, the SBP 9900A entars the hold state after completion of its present mamory cycle. Tha SBP 9900A than allows its address bus, data bus, $\overline{\text{WE}}$, $\overline{\text{MEMEN}}$, 0BIN, and HOLDA facilities to be pulled to the logic level HIGH state. When $\overline{\text{HOL0}}$ is deectivated, the SBB 9900A returns to normal operation from the point et which it was stopped.
HOLDA	5	OUT	HOLD ACKNOWLEOGE. When ective (pulled to logic lavel HIGH), HOL0A indicates thet the SBP 9900A is in the hold stete and that its address bus, dete bus, $\overline{\text{WE}}$, $\overline{\text{MEMEN}}$, and DBIN facilities ere pullad to the logic leval HIGH state.
READY	62	IN	When active (logic lavel HIGH), READY indicates that the memory will be ready to read or write during the next clock cycla. When not-raady is indicated during a memory operetion, tha SBP 9900A enters a wait state end suspends internal operation until the memory systems activata READY.
WAIT	3	OUT	Whan active (pullad to logic level HIGH), WAIT indicatas that tha SBP 9900A has entered a weit state in response to a not-reedy condition from memory.
			TIMING AND CONTROL
IAQ	7	IN	INSTRUCTION ACQUISITION. IAQ is active (pulled to logic level HIGH) during any SBP 990DA initiatad instruction acquisition memory cycla. Consequently, IAQ may be usad to facilitete datection of illegal op codes.
$\overline{\text{CYCEND}}$	59	OUT	CYCLE END. When ective (logic level LOW), $\overline{\text{CYCEND}}$ indicates thet the SBP 9900A will initiata a new microinstruction cycle on tha low-to-high transition of the next CLOCK.
$\overline{\text{LOAD}}$	4	IN	When active (logic level LOW), $\overline{\text{LOAD}}$ causes the SBP 99D0A to axecute e nonmaskabla interrupt with mamory addresses $FFFC_{16}$ and $FFFE_{16}$ containing the associatad trap vectors (WP and PC). The load sequence is initieted after tha instruction being executed is completed. $\overline{\text{LOA0}}$ will also terminate an idle stata. If $\overline{\text{LOAD}}$ is active during tha time $\overline{\text{RESET}}$ is activa, the $\overline{\text{LOAD}}$ trap will occur aftar tha $\overline{\text{RESET}}$ function is completed. $\overline{\text{LOAD}}$ should ramein active for one instruction execution period (IAQ may be used to monitor instruction bounderies). $\overline{\text{LOA0}}$ may be

Table 2. (Continued)

SIGNATURE	PIN	I/O	DESCRIPTION
$\overline{\text{LOAD}}$ (Cont.)			used to implement cold-start ROM loaders. Additionally, front-panel routinas may be implamantad using CRU bits as front-panel-interfaca signals, and software-control routines to direct the panel operations.
$\overline{\text{RESET}}$	6	IN	Whan active (logic level LOW), $\overline{\text{RESET}}$ ceuses the SBP 9900A to reset itsalf and inhibit $\overline{\text{WE}}$ and CRUCLK. Whan $\overline{\text{RESET}}$ is relaasad, the SBP 9900A initiates a level-zero intarrupt sequenca acquiring tha WP end PC trep vactors from memory locations 0000_{16} and 0002_{16}, sats all status ragister bits to logic level LOW, and than fetchas tha first instruction of the reset program environment. $\overline{\text{RESET}}$ must be held active for a minimum of threa CLOCK cyclas.

TIMING

SBP 9900A Memory

The SBP 9900A basic memory timing for a memory-read cycle with no wait states, and a memory-write cycle with one wait state, is as shown in *Figure 10.* During each memory-read or memory-write cycle, $\overline{\text{MEMEN}}$ becomes active (logic level LOW) along with valid memory-address data appearing on the address bus (A0 through A14).

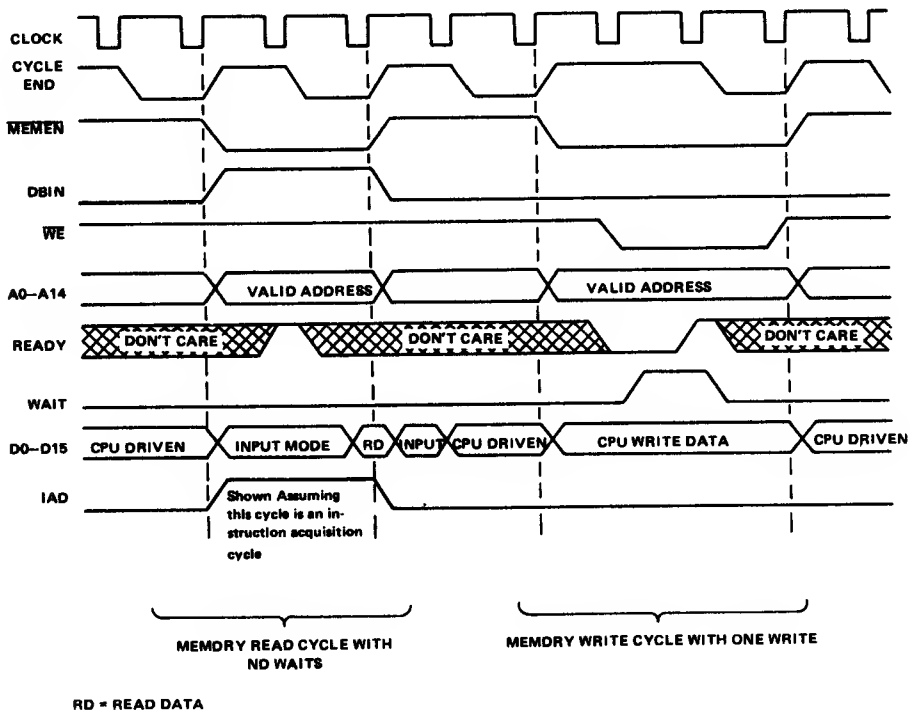


Figure 10. SBP 9900A Memory Bus Timing

In the case of a memory-read cycle, DBIN becomes active (pulled to logic level HIGH) at the same time memory-address data becomes valid; the memory write strobe $\overline{\text{WE}}$ remains inactive (pulled to logic level HIGH). If the memory-read cycle is initiated for acquisition of an instruction, IAQ becomes active (pulled to logic level HIGH) at the same time $\overline{\text{MEMEN}}$ becomes active. At the end of a memory-read cycle, $\overline{\text{MEMEN}}$ and DBIN together become inactive. At this time, though the address may change, the data bus remains in the input mode until terminated by the next high-to-low transition of the clock.

In the case of a memory-write cycle, $\overline{WE}$ becomes active (logic level LOW) with the first high-to-low transition of the clock after $\overline{MEMEN}$ becomes active; DBIN remains inactive. At the end of a memory-write cycle, $\overline{WE}$ and $\overline{MEMEN}$ together become inactive.

During either a memory-read or a memory-write operation, READY may be used to extend the duration of the associated memory cycle such that the speed of the memory system may be coordinated with the speed of the SBP 9900A. If READY is inactive (logic level LOW) during the first low-to-high transition of the clock after $\overline{MEMEN}$ becomes active, the SBP 9900A will enter a wait state suspending further progress of the memory cycle. The first low-to-high transition of the clock after READY becomes active terminates the wait state and allows normal completion of the memory cycle.

SBP 9900A Hold

The SBP 9900A hold facilities allow both the '9900A and external devices to share a common memory. To gain memory-bus control, an external device requiring direct memory access (DMA) sends a hold request ($\overline{HOLD}$) to the SBP 9900A. When the next available non-memory cycle occurs, the SBP 9900A enters a hold state and signals its surrender of the memory-bus to the external device via a hold acknowledge (HOLDA). Receiving the hold acknowledgement, the external device proceeds to utilize the common memory. After its memory requirements have been satisfied, the external device returns memory-bus control to the SBP 9900A by releasing $\overline{HOLD}$.

When $\overline{HOLD}$ becomes active (logic level LOW), the SBP 9900A enters a hold state at the beginning of the next available non-memory cycle as shown below. Upon entering a hold state, HOLDA becomes active (pulled to logic level HIGH) with the following signals pulled to a HIGH logic level by the individual pull-up resistors tied to each respective open-collector output: DBIN, $\overline{MEMEN}$, $\overline{WE}$, A0 through A14, and D0 through D15. When $\overline{HOLD}$ becomes inactive, the SPB 9900A exits the hold state and regains memory-bus control. If $\overline{HOLD}$ becomes active during a CRU operation, the SBP 9900A uses an extra clock cycle after the deactivation of $\overline{HOLD}$ to reassert the CRU address thereby providing the normal setup time for the CRU-bit transfer.

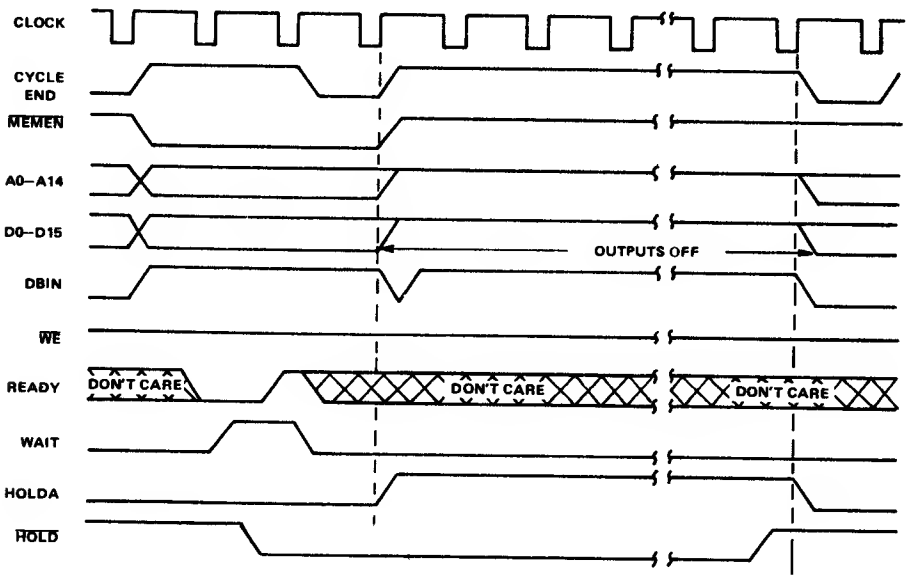


Figure 11. SBP 9900A Hold Timing

SBP 9900A CRU

The transfer of two data-bits from memory to a peripheral CRU device, and the transfer of one data-bit from a peripheral CRU device to memory, is shown in *Figure 12.* To transfer a data-bit to a peripheral CRU device, the SBP 9900A outputs the corresponding CRU-bit-address on address bus bits A3 through A14 and the respective data-bit on CRUOUT. During the second clock cycle of the operation, the SBP 9900A outputs a pulse, on CRUCLK, indicating to the peripheral CRU device the presence of a data-bit. This process is repeated until transfer of the entire field of data-bits specified by the CRU instruction has been accomplished. To transfer a data-bit from a peripheral CRU device, the SBP 9900A outputs the corresponding CRU-bit-Address on address bus bits A3 through A14 and receives the respective data-bit on CRUIN. No CRUCLK pulses occur during a CRU input operation.

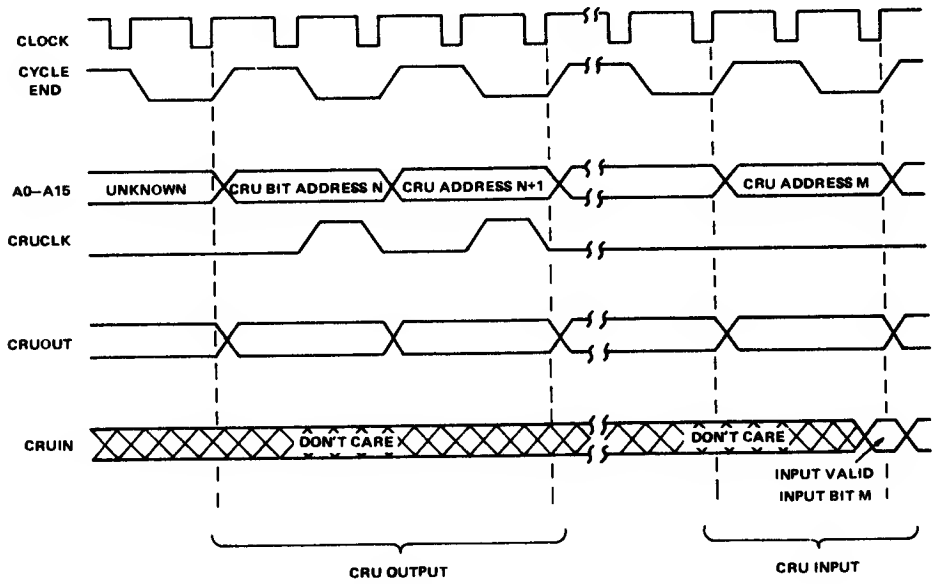


Figure 12. SBP 9900A CRU Interface Timing

MICROINSTRUCTION CYCLE

The SBP 9900 includes circuitry which will indicate the completion of a microinstruction cycle. Designated as the $\overline{\text{CYCEND}}$ function, it provides CPU status that can simplify system design. The $\overline{\text{CYCEND}}$ output will go to a low logic level as a result of the low-to-high transition of each clock pulse which initiates the last clock of a microinstruction.

SBP 9900A INSTRUCTION EXECUTION TIMES

Instruction execution times for the SBP 9900A are a function of:

1) Clock cycle time, $t_c(\phi)$
2) Addressing mode used where operands have multiple addressing mode capability
3) Number of wait states required per memory access.

Table 3 lists the number of clock cycles and memory accesses required to execute each SBP 9900A instruction. For instructions with multiple addressing modes for either or both operands, the table lists the number of clock cycles and memory accesses with all operands addressed in the workspace-register mode. To determine the additional number of clock cycles and memory accesses required for modified addressing, add the appropriate values from the referenced tables. The total instruction-execution time for an instruction is:

$$T = t_c(\phi)\,(C + W \cdot M)$$

where:

T = total instruction execution time;
$t_c(\phi)$ = clock cycle time;
C = number of clock cycles for instruction execution plus address modification;
W = number of required wait states per memory access for instruction execution plus address modification;
M = number of memory accesses.

Table 3. Instruction Execution Times

INSTRUCTION	CLOCK CYCLES C	MEMORY ACCESS M	ADDRESS MODIFICATION† SOURCE	ADDRESS MODIFICATION† DEST
A	14	4	A	A
AB	14	4	B	B
ABS (MSB = 0)	12	2	A	–
(MSB = 1)	14	3	A	–
AI	14	4	–	–
ANDI	14	4	–	–
B	8	2	A	–
BL	12	3	A	–
BLWP	26	6	A	–
C	14	3	A	A
CB	14	3	B	B
CI	14	3	–	–
CKOF	12	1	–	–
CKON	12	1	–	–
CLR	10	3	A	–
COC	14	3	A	–
CZC	14	3	A	–
DEC	10	3	A	–
DECT	10	3	A	–
DIV (ST4 is set)	18	3	A	–
DIV (ST4 is reset)*	92-124	8	A	–
IDLE	12	1	–	–
INC	10	3	A	–
INCT	10	3	A	–
INV	10	3	A	–
Jump (PC is changed)	10	1	–	–
(PC is not changed)	8	1	–	–
LDCR (C = 0)	52	3	A	–
(1≤C≤8)	20+2C	3	B	–
(9≤C≤15)	20+2C	3	A	–
LI	12	3	–	–
LIMI	14	2	–	–
LREX	12	1	–	–
$\overline{\text{RESET}}$ function	26	5	–	–
$\overline{\text{LOAD}}$ function	22	5	–	–
Interrupt context switch	22	5	–	–

INSTRUCTION	CLOCK CYCLES C	MEMORY ACCESS M	ADDRESS MODIFICATION† SOURCE	ADDRESS MODIFICATION† DEST
LWPI	10	2	–	–
MOV	14	4	A	A
MOVB	14	4	B	B
MPY	52	5	A	–
NEG	12	3	A	–
ORI	14	4	–	–
RSET	12	1	–	–
RTWP	14	4	–	–
S	14	4	A	A
SB	14	4	B	B
SBO	12	2	–	–
SBZ	12	2	–	–
SETO	10	3	A	–
Shift (C≠0)	12+2C	3	–	–
(C=0, Bits 12–15 of WR0=0)	52	4	–	–
(C=0, Bits 12–15 of WRP=N≠0)	20+2N	4	–	–
SOC	14	4	A	A
SOCB	14	4	B	B
STCR (C=0)	60	4	A	–
(1≤C≤7)	42	4	B	–
(C=8)	44	4	B	–
(9≤C≤15)	58	4	A	–
STST	8	2	–	–
STWP	8	2	–	–
SWPB	10	3	A	–
SZC	14	4	A	A
SZCB	14	4	B	B
TB	12	2	–	–
X **	8	2	A	–
XOP	36	8	A	–
XOR	14	4	A	–
Undefined op codes: 0000-01FF, 0320-033F, 0C00-0FFF, 0780-07FF	6	1	–	–

*Execution time is dependent upon the partial quotient after each clock cycle during execution.
**Execution time is added to the execution time of the instruction located at the source address minus 4 clock cycles and 1 memory access time.
†The letters A and B refer to the respective tables that follow.

Table A. Address Modification

ADDRESSING MODE	CLOCK CYCLES C	MEMORY ACCESSES M
WR (T_S or T_D = 00)	0	0
WR indirect (T_S or T_D = 01)	4	1
WR indirect auto-increment (T_S or T_D = 11)	8	2
Symbolic (T_S or T_D = 10, S or D = 0)	8	1
Indexed (T_S or T_D = 10, S or D ≠ 0)	8	2

Table B. Address Modification

ADDRESSING MODE	CLOCK CYCLES C	MEMORY ACCESSES M
WR (T_S or T_D = 00)	0	0
WR indirect (T_S or T_D = 01)	4	1
WR indirect euto-increment (T_S or T_D = 11)	6	2
Symbolic (T_S or T_D = 10, S or D = 0)	8	1
Indexed (T_S or T_D = 10, S or D ≠ 0)	8	2

As an example, the instruction MOVB is used in a system with $t_c(\phi) = 0.333$ μs and no wait states are required to access memory. Both operands are addressed in the workspace register mode:

$$T = t_c(\phi)\,(C + W \cdot M) = 0.333\,(14 + 0 \cdot 4)\ \mu s = 4.662\ \mu s.$$

If two wait states per memory access were required, the execution time is:

$$T = 0.333\,(14 + 2 \cdot 4)\ \mu s = 7.326 = \mu s.$$

If the source operand was addressed in the symbolic mode and two wait states were required:

$$T = t_c(\phi)\,(C + W \cdot M)$$
$$C = 14 + 8 = 22$$
$$M = 4 + 1 = 5$$
$$T = 0.333\,(22 + 2 \cdot 5)\ \mu s = 10.656\ \mu s.$$

INTERFACING

The input/output (I/O) accommodations have been designed for TTL compatibility. Direct interfacing, supportable by the entire families of catalog devices, is shown in *Figure 13.*

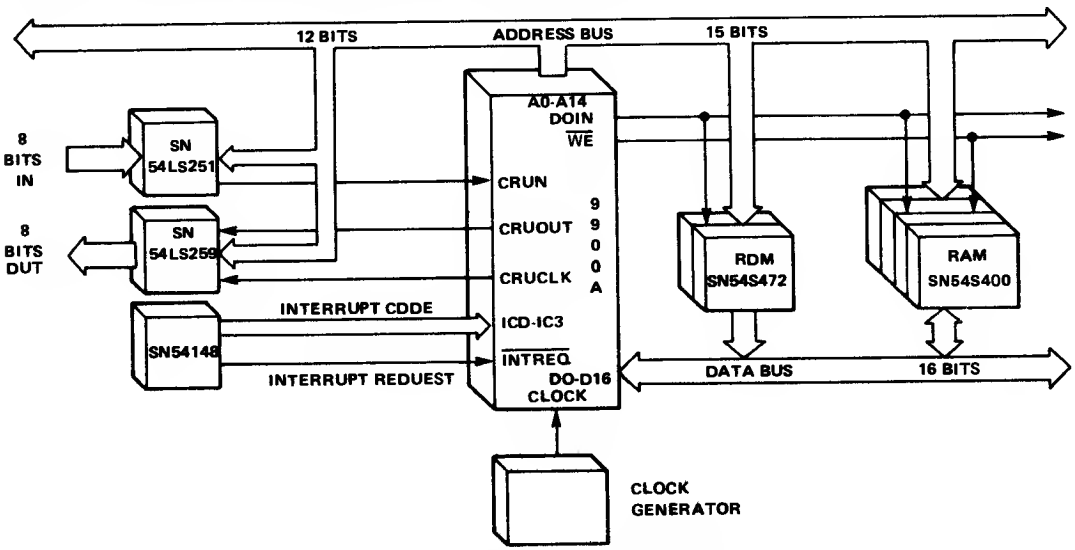


Figure 13. Typical SBP 9900A System

Input Circuit

The input circuit used on the SBP 9900A is basically an RTL configuration which has been modified for TTL compatibility as shown in *Figure 14A.* An input-clamping diode is incorporated to limit negative excursions (ringing) when the SBP 9900A is on the receiving end of a transmission line; an input switching threshold of nominally +1.5 volts has been specified for improved noise immunity. This threshold is achieved via two resistors which function as a voltage divider to increase the one V_{BE} threshold of the I^2L input transistor to +1.5 volts. Since this input circuit is independent of injector current, input threshold compatibility is maintained over the entire speed/power performance range.

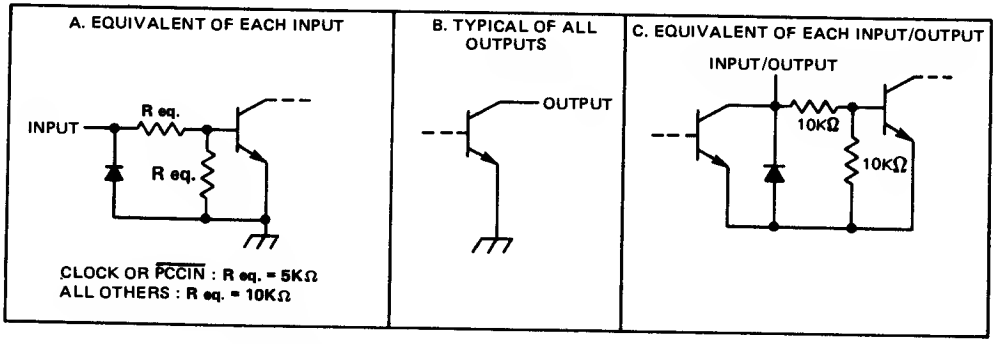


Figure 14. Schematics of Equivalent Inputs, Outputs, Inputs/Outputs

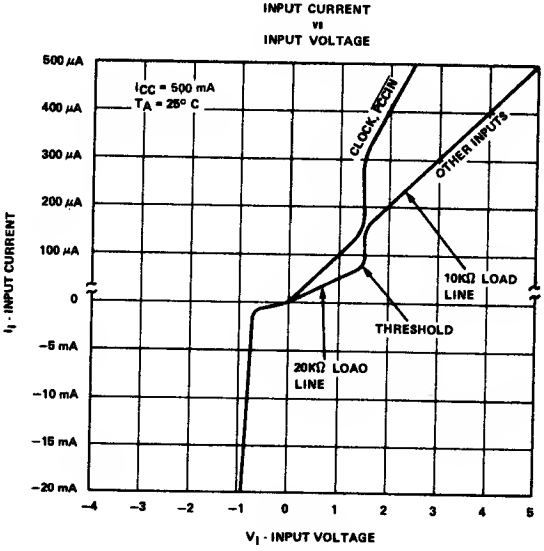


Figure 15. Typical Input Characteristics

The input circuit characteristics for input current versus input voltage are shown in *Figure 15.* The 10K and 20K ohm load lines and threshold knee at +1.5 volts provide a high-impedance characteristic to reduce input loading and improve the low-logic level input noise immunity over some standard TTL inputs. Full compatibility is maintained with virtually all 5 volt logic families even when the SBP 9900A is powered down (injector current reduced).

Sourcing Inputs

The inputs may be sourced directly by most 5 volt logic families. Five volt functions which feature internal pull-up resistors at their outputs require no external interface components; five volt functions which feature open-collector outputs generally require external pull-up resistors.

Terminating Unused Inputs

Inputs which are selected to be hardwired to a logic-level low may be connected directly to ground. Inputs which are selected to be hardwired to a logic-level high must be tied, via a current limiting (pull-up) resistor, to a logic-level-high low-impedance voltage source such as V_{CC}. A single transient protecting resistor may be utilized common to (N) inputs.

OUTPUT CIRCUIT

The output circuit selected for the SBP 9900A is an injected open-collector transistor shown in *Figure 14B.* Since this transistor is injected, output sourcing capability is directly related to injector current. In other words, the number of loads which may be sourced by an SBP 9900A output is directly reduced as injector current is reduced.

The output circuit characteristic for logic-level low output voltage (V_{OL}) versus logic-level low output current (I_{OL}) is shown in *Figure 16.* At rated injector current, the SBP 9900A output circuit offers a low-level output voltage of typically 220 mV.

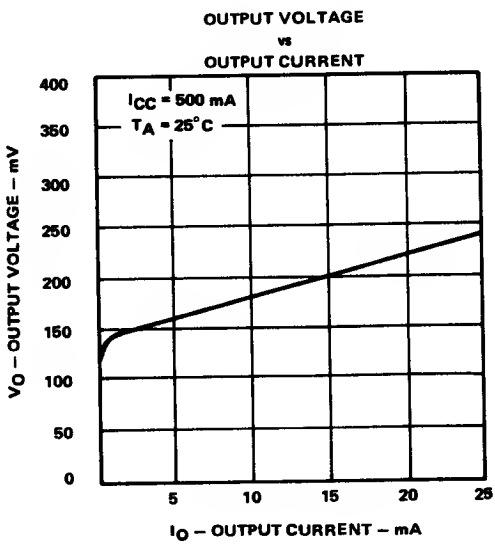


Figure 16. Typical Output Characteristics

The output circuit characteristics for 1) logic-level high output voltage (V_{OH}) and current (I_{OH}), 2) rise times, and 3) next stage input noise immunity, are a function of the load circuit being sourced. The load circuit may be either:

A) the direct input, if no source current is required, of a five-volt logic family function,

or, for greater noise immunity and improved rise times,

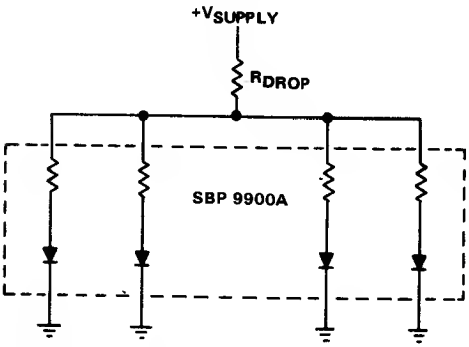
B) the direct input of a five-volt logic family function in conjunction with a discrete pull-up resistor.

When a discrete pull-up resistor (R_L) is utilized, the fanout requirements placed on a particular SBP 9900A output restrict both the maximum and minimum value of R_L.

POWER SOURCE

I^2L is a current-injected logic. When placed across a curve tracer, the processor will resemble a silicon switching diode. Any voltage or current source capable of supplying the desired current at the injector mode voltage required will suffice. A dry-cell battery, a 5-volt TTL power supply, a programmable current supply (for power-up/power-down operation) — literally whatever power source is convenient can be used for most cases. For example, if a 5-volt TTL power supply is to be used, a series dropping resistor would be connected between the 5-volt supply and the injector pins of the I^2L device, as illustrated in *Figure 17,* to select the desired operating current.

An alternate solution utilizes the Texas Instruments TL497 switching-regulator as illustrated in *Figure 18.*



GENERAL FORMULA (OHM'S LAW)

$$R_{DROP} = \frac{V_{SUPPLY} - V_{CC}}{I_{CC}}$$

EXAMPLE FOR V_{SUPPLY} = 5V, AND I_{CC} = 500 mA:

$$R_{DROP} = \frac{5 - 1.05}{0.5} = \frac{3.95}{0.5} = 7.9 \text{ OHMS}$$

Figure 17. Injector Current Calculations

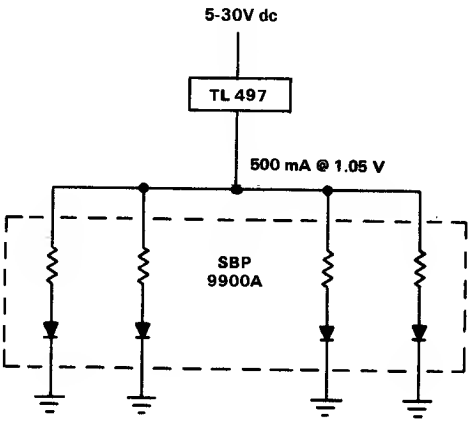


Figure 18. Switching-Regulator Injector Source

Operating from a constant current power source, the SBP 9900A may be powered-up/powered-down with complete maintenance of data integrity to execute instructions over a speed/power range spanning several orders of user-selectable injector-supply-current range as illustrated in *Figure 19.*

Figures 20 and *21* show the typical injector node voltages which occur across the temperature and injector current ranges.

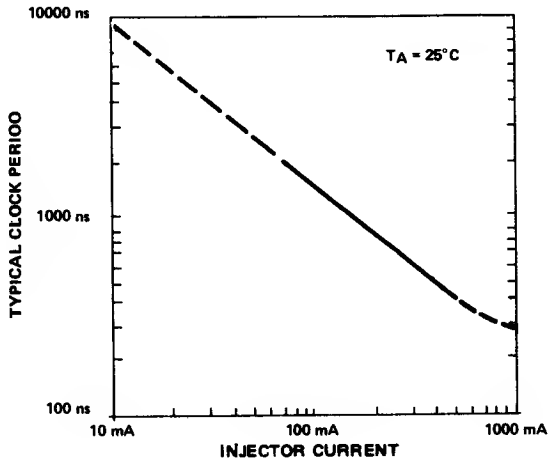


Figure 19. SBP 9900A Clock Period vs. Injector Current

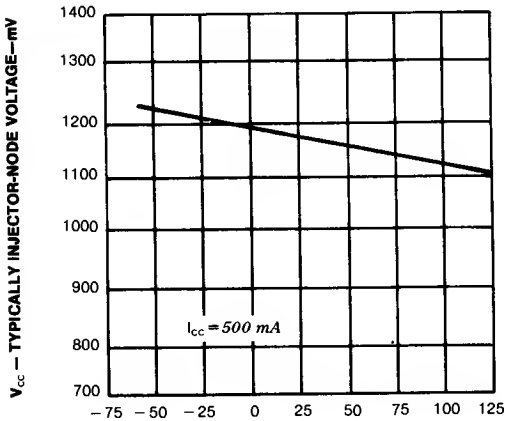


Figure 20. SBP 9900A Injector Node Voltage vs. Free-Air Temperature

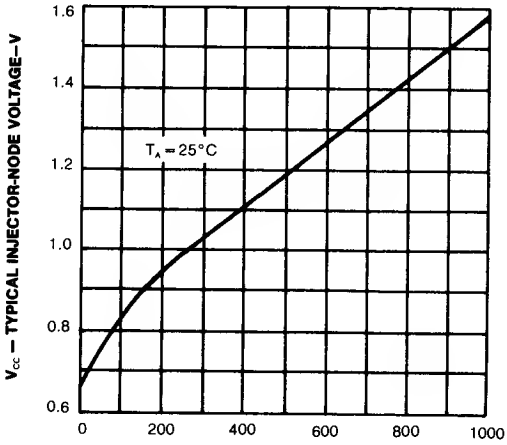


Figure 21. SBP 9900A Injector Node Voltage vs. Injector Current

ELECTRICAL SPECIFICATIONS

SBP 9900A Recommended Operating Conditions, Unless Otherwise Noted I_{CC} = 500 mA

		MIN	NOM	MAX	UNIT
Supply current, I_{CC}		450	500	550	mA
High-level output voltage, V_{DH}				5.5	V
Low-level output current, I_{DL}				20	mA
Clock frequency, f_{clock}		0		2	MHz
Width of clock pulse, t_w	High (67%) (V_{IH} = 2.5 V max)	330			ns
	Low (33%)	170			
Clock rise time, t_r			10		ns
Clock fall time, t_f			10		ns
Setup time, t_{su} (see Figure 24)	$\overline{HDLD}$	210↑			ns
	READY	140↑			
	D0 – D15	85↑			
	CRUIN	65↑			
	$\overline{INTREQ}$	25↑			
	IC0 – IC3	25↑			
Hold time, t_h (see Figure 24)	$\overline{HOLD}$	25↑			ns
	READY	65↑			
	D0 – D15	65↑			
	CRUIN	55↑			
	$\overline{INTREQ}$	90↑			
	IC0 – IC3	90↑			
Operating free-air temperature, T_A	SBP 9900 M/N	−55		125	°C
	SBP 9900E	−40		85	

↑Rising edge of clock pulse is reference.

SBP 9900A ELECTRICAL CHARACTERISTICS (OVER RECOMMENDED OPERATING FREE-AIR TEMPERATURE RANGE, UNLESS OTHERWISE NOTED)

PARAMETER			TEST CONDITIONS†		MIN	TYP‡	MAX	UNIT
V_{IH}	High-level input voltage				2			V
V_{IL}	Low-level input voltage						0.8	V
V_{IK}	Input clamp voltage		I_{CC} = MIN,	I_I = −12 mA			−1.5	V
I_{OH}	High-level output current		I_{CC} = 500 mA, V_{IL} = 0.8 V,	V_{IH} = 2 V, V_{OH} = 5.5 V			400	μA
V_{OL}	Low-level output voltage		I_{CC} = 500 mA, V_{IL} = 0.8 V,	V_{IH} = 2 V, I_{OL} = 20 mA			0.4	V
I_I	Input current	Clock	I_{CC} = 500 mA,	V_I = 2.4 V		480	600	μA
		All other inputs				240	300	

†For conditions shown as MAX, use the appropriate value specified under recommended operating conditions.
‡All typical values are at I_{CC} = 500 mA, T_A = 25°C.

SBP 9900A SWITCHING CHARACTERISTICS, (I_{CC} = NOM, T_A = RECOMMENDED OPERATING FREE-AIR TEMPERATURE RANGE UNLESS OTHERWISE NOTED) SEE *Figures 22* AND *23*.

PARAMETER	FROM	TO	TEST CONDITIONS	MIN	TYP‡	MAX	UNIT
f_{max}	MAXIMUM CLOCK FREQUENCY		C_L = 150 pF, R_L = 280 Ω	2			MHz
t_{PLH} or t_{PHL}	CLOCK	ADDRESS BUS (A0 – A14)			170	225	ns
t_{PLH} or t_{PHL}	CLOCK	DATA BUS (D0 – D15)			170	265	ns
t_{PLH} or t_{PHL}	CLOCK	WRITE ENABLE ($\overline{WE}$)			220	295	ns
t_{PLH} or t_{PHL}	CLOCK	CYCLE END ($\overline{CYCEND}$)			170	225	ns
t_{PLH} or t_{PHL}	CLOCK	DATA BUS IN (DBIN)			190	250	ns
t_{PLH} or t_{PHL}	CLOCK	MEMORY ENABLE ($\overline{MEMEN}$)			155	205	ns
t_{PLH} or t_{PHL}	CLOCK	CRU CLOCK (CRUCLK)			187	280	ns
t_{PLH} or t_{PHL}	CLOCK	CRU DATA OUT (CRUOUT)			210	265	ns
t_{PLH} or t_{PHL}	CLOCK	HOLD ACKNOWLEDGE (HLDA)			320	410	ns
t_{PLH} or t_{PHL}	CLOCK	WAIT			155	210	ns
t_{PLH} or t_{PHL}	CLOCK	INSTRUCTION ACQUISITION (IAQ)			155	210	ns

‡All typical values are at 25°C.

8

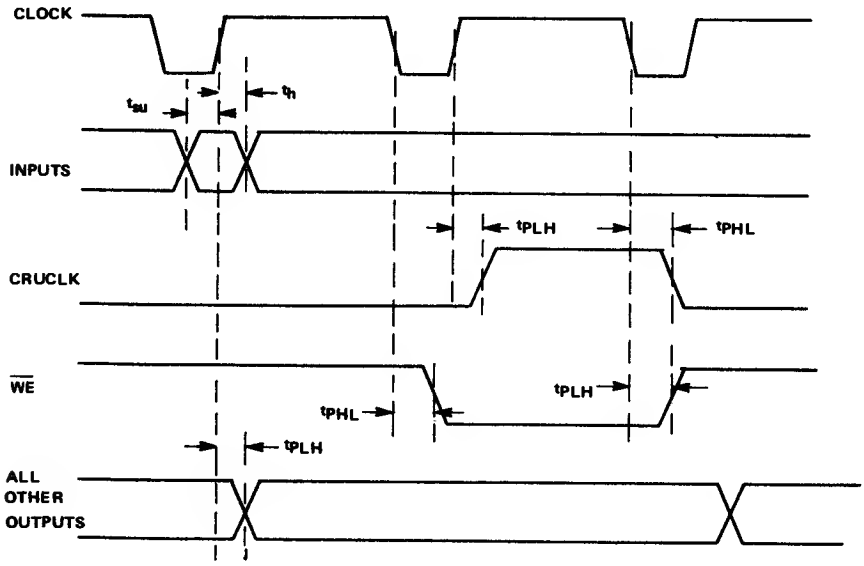


Figure 22. Switching Times — Voltage Waveforms

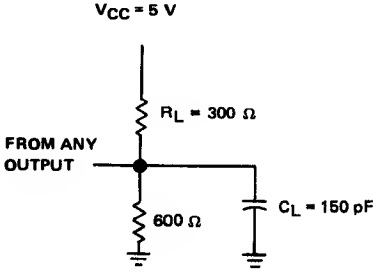


Figure 23. Switching Times Load Circuits

Clock Frequency vs. Temperature

Stability of the operational frequency over the full temperature range of −55°C to 125°C is illustrated in *Figure 24*. The effects of temperature on clock frequency are nil above 0°C. Below 0°C the effects are typically less than −8% with respect to the typical performance.

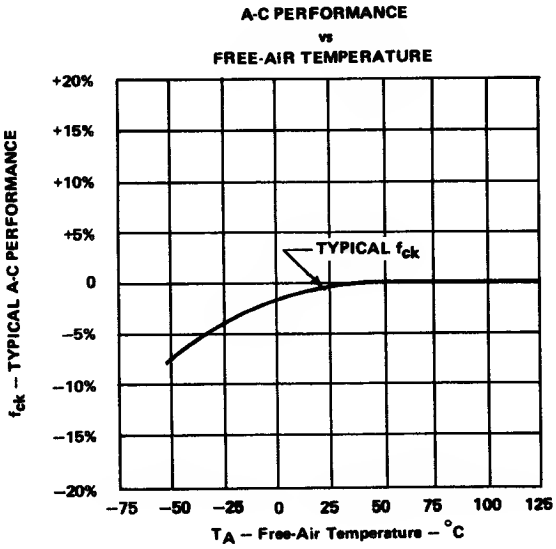


Figure 24. SBP 9900A A-C Performance vs. Temperature

8◀

TMS 9980A/
TMS 9981

8

1. INTRODUCTION

1.1 DESCRIPTION

The TMS 9980A/TMS 99B1 is a software-compatible member of TI's 9900 family of microprocessors. Designed to minimize the system cost for smaller systems, the TMS 99B0A/TMS 99B1 is a singla-chip 16-bit central processing unit (CPU) which has an 8-bit data bus, on-chip clock, and is packaged in a 40-pin package (see Figure 1). The instruction set of the TMS 9980A/TMS 9981 includes the capabilities offered by full minicomputers and is exactly the same as the 9900's. The uniqua mamory-to-mamory architecture features multiple register files, resident in memory, which allow faster response to interrupts and incraased programming flexibility. The separate bus structure simplifies the system design affort. Texas Instruments provides a compatible set of MOS and TTL memory and logic function circuits to be used with a TMS 9980A/TMS 9981 system.

1.2 KEY FEATURES

- 16-8it Instruction Word
- Full Minicomputer Instruction Set Capability Including Multiply and Divide
- Up to 16,384 Bytes of Mamory
- 8-Bit Memory Data 8us
- Advanced Mamory-to-Memory Architecture
- Separate Memory, I/O, and Interrupt-Bus Structures
- 16 General Registers
- 4 Prioritized Interrupts
- Programmed and DMA I/O Capability
- On-Chip 4-Phase Clock Generator
- 40-Pin Package
- N-Channel Silicon-Gate Technology

1.3 TMS 9980A/TMS 99B1 DIFFERENCES

The TMS 9980A and the TMS 9981 although very similar, have several differences which user should be aware.

1. The TMS 9980A requires a V_{BB} supply (pin 21) while the TMS 9981 has an internal charge pump to generate V_{BB} from V_{CC} and V_{DD}. 8◀
2. The TMS 9981 has an optional on-chip crystal oscillator in addition to the external clock mode of the TMS 9980A.
3. The pin-outs are not compatible for D0-D7, INT0-INT2, and $\overline{\phi}3$.

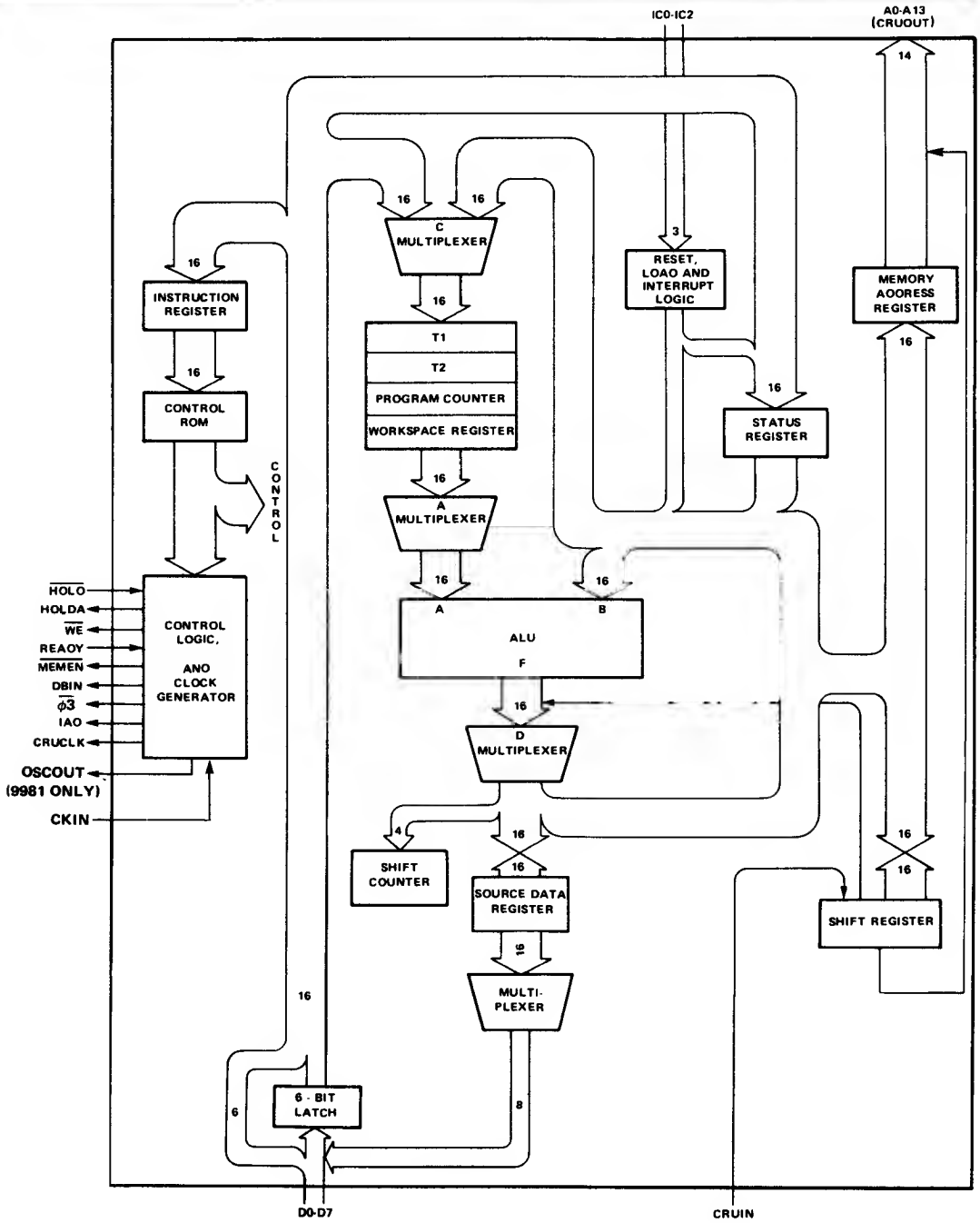
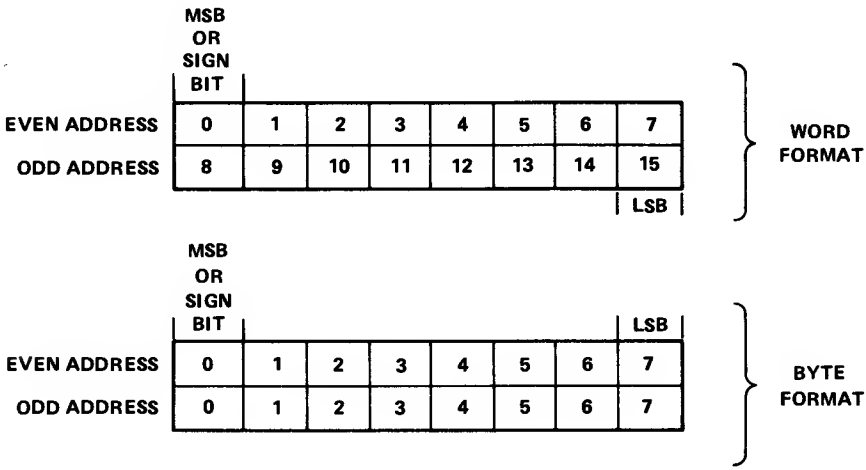


FIGURE 1 – ARCHITECTURE

2. ARCHITECTURE

The memory for the TMS 9980A/TMS 9981 is addressable in 8-bit bytes. A word is defined as 16 bits or 2 consecutive bytes in memory. The words are restricted to be on even address boundaries, i.e., the most-significant half (8 bits) resides at even address and the least-significant half resides at the subsequent odd address. A byte can reside at even or odd address. The word and byte formats are shown below.



2.1 REGISTERS AND MEMORY

The TMS 9980A/TMS 9981 employs an advanced memory-to-memory architecture. Blocks of memory designated as workspace replace internal hardware registers with program-data registers. The TMS 9980A/TMS 9981 memory map is shown in Figure 2. The first two words (4 bytes) are used for RESET trap vector. Addresses 0004_{16} through 0013_{16} are used for interrupt vectors. Addresses 0040 through 007F are used for the extended operation (XOP) instruction trap vectors. The last four bytes at address $3FFC_{16}$ to 3FFF are used for trap vector for the LOAD function.

The remaining memory is available for programs, data, and workspace registers. If desired, any of the special areas may also be used as general memory.

Three internal registers are accessible to the user. The program counter (PC) contains the address of the instruction following the current instruction being executed. This address is referenced by the processor to fetch the next instruction from memory and is then automatically incremented. The status register (ST) contains the present state of the processor and will be further defined in Section 3.4. The workspace pointer (WP) contains the address of the first word in the currently active set of workspace registers.

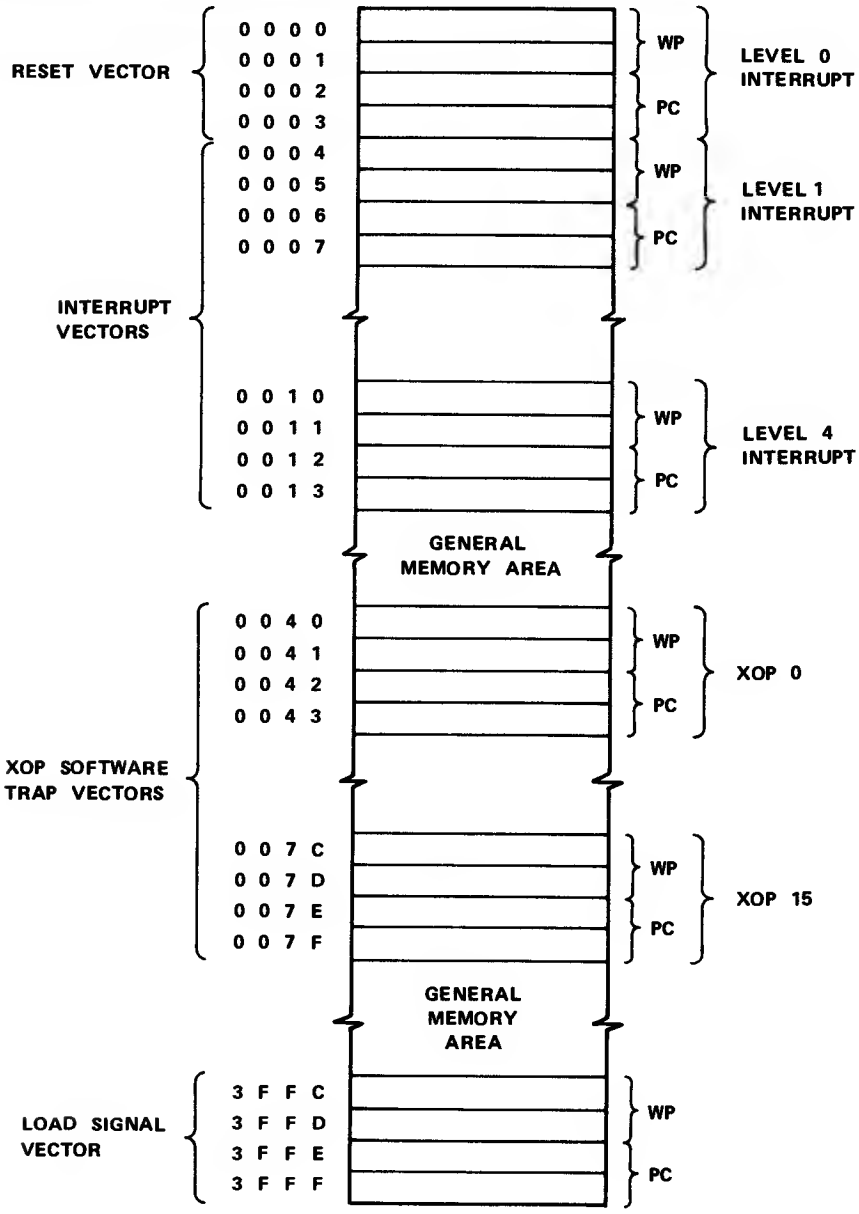
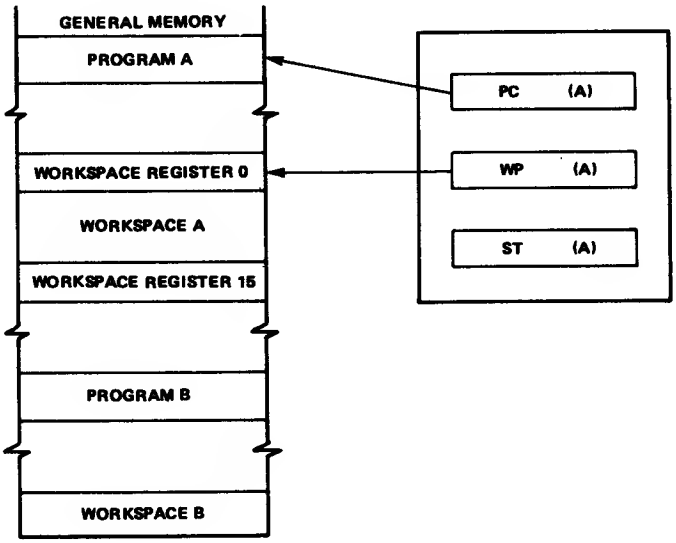


FIGURE 2 – MEMORY MAP

A workspace-register file occupies 16 contiguous memory words in the general memory area. Each workspace register may hold data or addresses and function as operand registers, accumulators, address registers, or index registars. During instruction execution, the processor addresses any register in the workspace by adding the register number to the contents of the workspace pointar and initiating a memory request for the word. The relationship betwaen the workspace pointer and its corresponding workspace is shown below.



The workspace concept is particularly valuabla during operations that require a context switch, which is a change from one program environment to another (as in the case of an interrupt or call to a subroutine). Such an operation, using a conventional multi-register arrangement, requires that at least part of the contents of the register file be stored and reloaded. A memory cycla is required to store or fetch each word. By exchanging the program counter, status register, and workspace pointer, the TMS 9980A/TMS 9981 accomplishes a complete context switch with only six stora cycles and six fetch cycles. After the switch the workspace pointer contains the starting address of a new 16-word workspace in memory for use in the new routine. A corresponding time saving occurs when the original context is restored. Instructions in the TMS 9980A/TMS 9981 that result in a context switch include:

1. Branch and Load Workspace Pointer (BLWP)
2. Return from Subroutine (RTWP)
3. Extanded Operation (XOP)

Device interrupts, $\overline{\text{RESET}}$, and $\overline{\text{LOAD}}$ also cause a context switch by forcing the processor to trap to a service subroutine.

2.2 INTERRUPTS

The architecture of the 9900 family allows vectoring of 16 interrupts. These interrupts are assigned levels from 0 to 15. The interrupt at level 0 has the highest priority and the interrupt at level 15 has the lowest priority. The TMS 9900 implements all 16 interrupt levels. The TMS 9980A/TMS 9981 implements only 5 levels (level 0 and levels 1 through 4). Level 0 is reserved for $\overline{\text{RESET}}$ function.

Levels 1 through 4 may be used for external devices. The external levels may also be shared by several device interrupts, depending upon system requirements. The TMS 9980A/TMS 9981 continuously compares the interrupt code (IC0 through IC2) with the interrupt mask contained in status-register bits 12 through 15. When the level of the pending interrupt is less than or equal to the enabling mask level (higher or equal priority interrupt), the processor recognizes the interrupt and initiates a context switch following completion of the currently executing instruction. The processor fetches the new context WP and PC from the interrupt vector locations. Then, the previous context WP, PC, and ST are stored in workspace registers 13, 14, and 15, respectively, of the new workspace. The TMS 9980A/TMS 9981 then forces the interrupt mask to a value that is one less than the level of the interrupt being serviced. This allows only interrupts of higher priority to interrupt a service routine. The processor also inhibits interrupts until the first instruction of the service routine has been executed to allow modification of interrupt mask if needed (to mask out certain interrupts). All interrupt requests should remain active until recognized by the processor in the device-service routine. The individual service routines must reset the interrupt requests before the routine is complete. The interrupt code (IC0-IC2) may change asynchronously within the constraints specified in Section 2.10.4.

If a higher priority interrupt occurs, a second context switch occurs to service the higher-priority interrupt. When that routine is complete, a return instruction (RTWP) restores the first service routine parameters to the processor to complete processing of the lower-priority interrupt. All interrupt subroutines should terminate with the return instruction to restore original program parameters. The interrupt-vector locations, device assignment, enabling mask value and the interrupt code are shown in Table 1.

TABLE 1
INTERRUPT LEVEL DATA

INTERRUPT CODE (IC0–IC2)	FUNCTION	VECTOR LOCATION (MEMORY ADDRESS IN HEX)	DEVICE ASSIGNMENT	INTERRUPT MASK VALUES TO ENABLE (ST12 THROUGH ST15)
1 1 0	Level 4	0 0 1 0	External Device	4 Through F
1 0 1	Level 3	0 0 0 C	External Device	3 Through F
1 0 0	Level 2	0 0 0 8	External Device	2 Through F
0 1 1	Level 1	0 0 0 4	External Device	1 Through F
0 0 1	Reset	0 0 0 0	Reset Stimulus	Don't Care
0 1 0	Load	3 F F C	Load Stimulus	Don't Care
0 0 0	Reset	0 0 0 0	Reset Stimulus	Don't Care
1 1 1	No-Op	-----	-----	-----

▶8

Note that $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ functions are also encoded on the interrupt code input lines. Figure 3 illustrates some of the possible configurations. To realize $\overline{\text{RESET}}$ and one interrupt no external component is needed. If $\overline{\text{LOAD}}$ is also needed, a three input AND gate is wired as shown. If the system requires more than one interrupt, a single SN74148 (TIM 9907) is required.

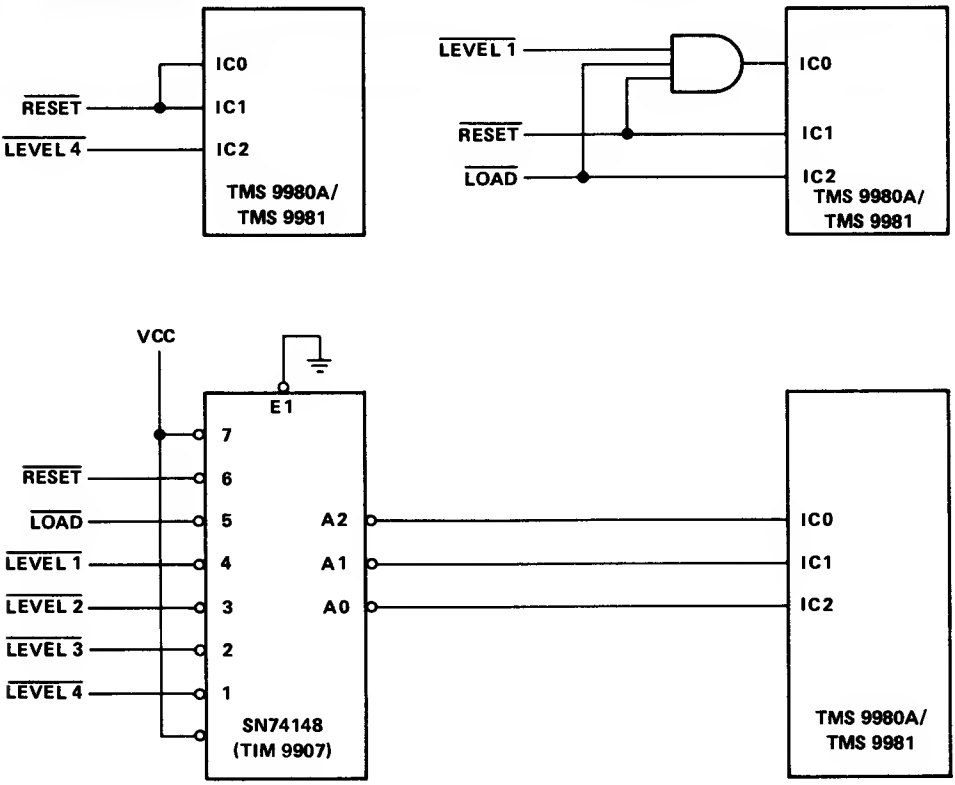


FIGURE 3 – TMS 9980A/TMS 9981 INTERRUPT INTERFACE

2.3 INPUT/OUTPUT

The TMS 9980A/TMS 9981 utilizes a versatile direct command-driven I/O interface designated as the communications-register unit (CRU). The CRU provides up to 2,048 directly addressable output bits. Both input and output bits can be addressed individually or in fields of from 1 to 16 bits. The TMS 9980A/TMS 9981 employs CRUIN, CRUCLK, and A13 (for CRUOUT) and 11 bits (A2-A12) of the address bus to interface with the CRU system. The processor instructions that drive the CRU interface can set, reset, or test any bit in the CRU array or move between memory and CRU data fields.

8◀

2.4 SINGLE-BIT CRU OPERATIONS

The TMS 9980A/TMS 9981 performs three single-bit CRU functions: test bit (TB), set bit to one (SBO), and set bit to zero (SBZ). To identify the bit to be operated upon, the TMS 9980A/TMS 9981 develops a CRU-bit address and places it on the address bus, A2 to A12.

For the two output operations (SBO and SBZ), the processor also generates a CRUCLK pulse, indicating an output operation to the CRU device and places bit 7 of the instruction word on the A13 line to accomplish the specified

operation (bit 7 is a one for S8O and a zero for S8Z). A test-bit instruction transfers the addressed CRU bit from the CRUIN input line to bit 2 of the status register (EQUAL).

The TMS 9980A/TMS 9981 develops e herdwere bese address for the single-bit operetions from the softwere base eddress conteined in workspece register 12 and the signed displecement count conteined in bits 8 through 15 of the instruction. The displecement allows two's complement eddressing from bese minus 128 bits through base plus 127 bits. The herdware bese eddress (bits 4 through 14 of WR12) is added to the signed displecement specified in the instruction end the result is loaded into the eddress bus. *Figure 4* illustrates the development of e single-bit CRU address.

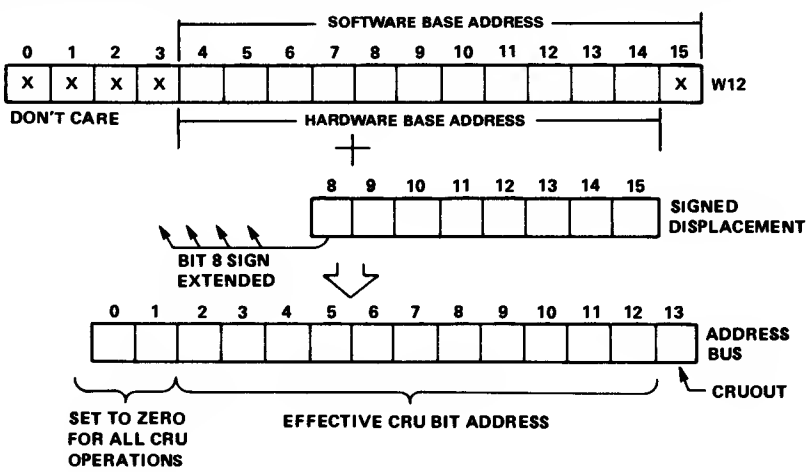


FIGURE 4 – TMS 9980A/TMS 9981 SINGLE-BIT CRU ADDRESS DEVELOPMENT

2.5 MULTIPLE-BIT CRU OPERATIONS

The TMS 9980A/TMS 9981 performs two multiple-bit CRU operations: store communications register (STCR) and load communications register (LDCR). 8oth operations perform a data transfer from the CRU-to-memory or from memory-to-CRU as illustrated in Figure 5. Although the figure illustrates a full 16-bit transfer operation, any number of bits from 1 through 16 may be involved. The LDCR instruction fetches a word from memory and right-shifts it to serially transfer it to CRU output bits. If the LDCR involves eight or fewer bits, those bits come from the right-justified field within the addressed byte of the memory word. If the LDCR involves nine or more bits, those bits come from the right-justified field within the whole memory word. When transferred to the CRU interface, each successive bit receives an address that is sequentially greater than the address for the previous bit. This addressing mechanism results in an order reversal of the bits; that is, bit 15 of the memory word (or bit 7) becomes the lowest addressed bit in the CRU and bit 0 becomes the highest addressed bit in the CRU field.

▶8

An STCR instruction trensfers data from the CRU to memory. If the operation involves a byte or less transfer, the transferred data will be stored right-justified in the memory byte with leading bits set to zero. If the operation involves from nine to 16 bits, the transferred data is stored right-justified in the memory word with leading bits set to zero.

When the input from the CRU device is complete, the first bit from the CRU is the least-significant-bit position in the memory word or byte.

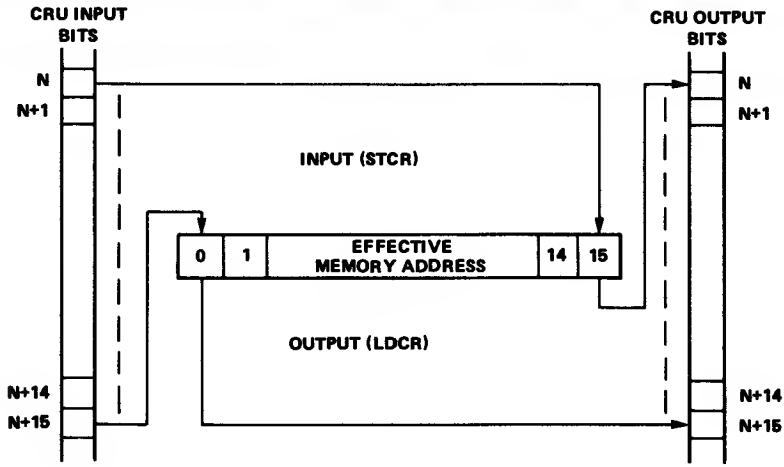


FIGURE 5 – TMS 9980A/TMS 9981 LDCR/STCR DATA TRANSFERS

Figure 6 illustrates how to implemant a 16-bit input and a 16-bit output register in the CRU interface. CRU addresses ara decoded as needed to implement up to 128 such 16-bit interface registers. In system application, however, only the exact number of interface bits needed to interfaca specific peripheral devices are implemented. It is not necessary to have a 16-bit interface register to interface an 8-bit device.

2.6 EXTERNAL INSTRUCTIONS

The TMS 9980A/TMS 9981 has five external instructions that allow user-defined external functions to be initiated under program control. These instructions ara CKON, CKOF, RSET, IDLE, and LREX. These mnemonics, except for IDLE, relata to functions implemented in the 990 minicomputer and do not restrict use of tha instructions to initiate various user-defined functions. IDLE also causes tha TMS 9980A/TMS 9981 to enter the idle state and remain until an interrupt, RESET, or LOAD occurs. When any of these five instructions are executed by the TMS 9980A/TMS 9981, a unique 3-bit coda appears on the address bus, bits A13, A0, and A1, along with a CRUCLK pulse. When the TMS 9980A/TMS 9981 is in an idle state, the 3-bit coda and CRUCLK pulses occur repeatedly until the idle stata is terminated. Tha codes ara:

EXTERNAL INSTRUCTION	A13	A0	A1
LREX	H	H	H
CKOF	H	H	L
CKON	H	L	H
RSET	L	H	H
IDLE	L	H	L
CRU INSTRUCTIONS	H/L	L	L

8

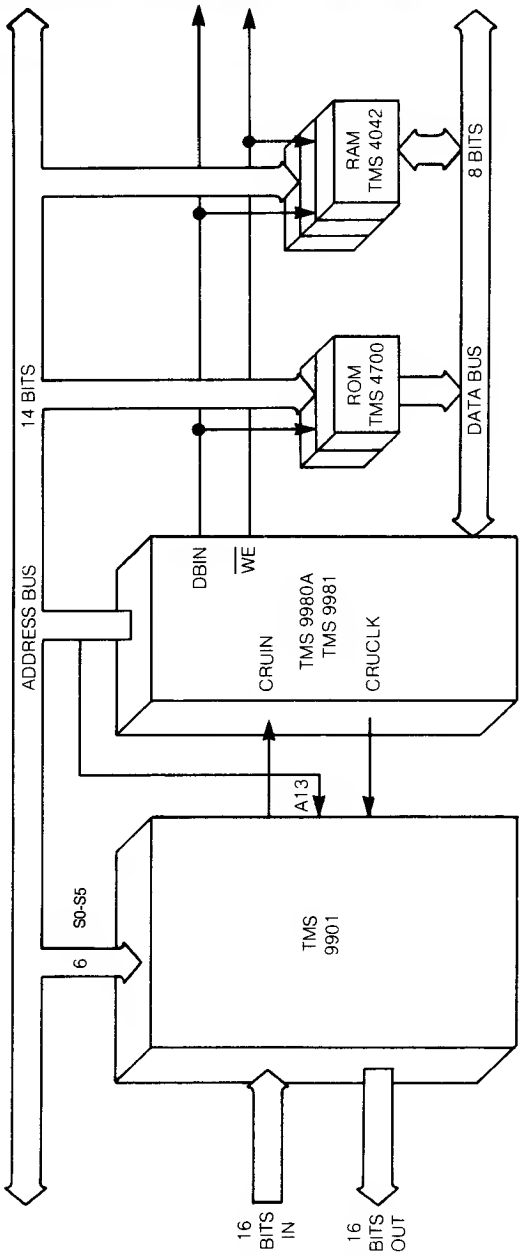


FIGURE 6 – TMS 9980A/9981 16-BIT INPUT/OUTPUT INTERFACE

Note that during external instructions bits (A2-A12) of the address bus may have any of the possible binary patterns. Since these bits (A2-A12) are used as CRU addresses, CRUCLK to the CRU must be gated with a decode of 0 on A0 and A1 to avoid erroneous strobe to CRU bits during external instruction execution.

Figure 7 illustrates typical external decode logic to implement these instructions. Note CRUCLK to the CRU is inhibited during external instructions.

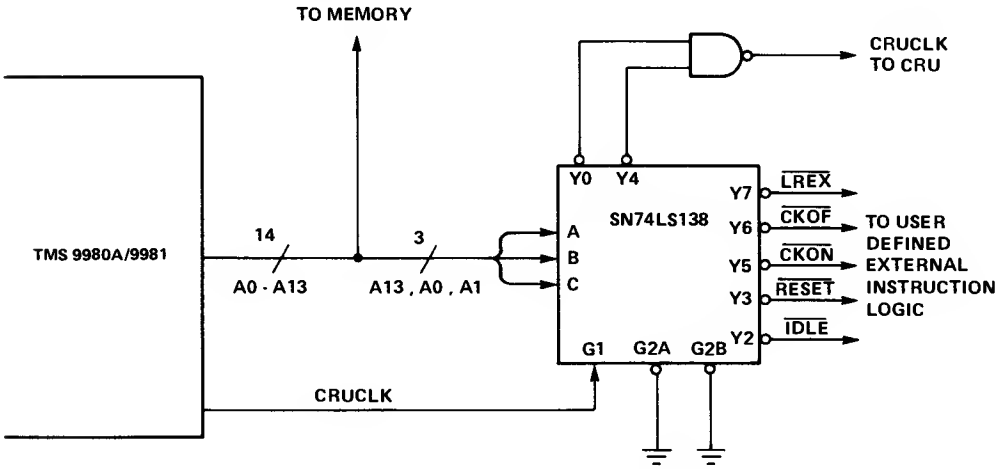


FIGURE 7 – EXTERNAL INSTRUCTION DECODE LOGIC

2.7 NON-MASKABLE INTERRUPTS

2.7.1 LOAD Function

The LOAD stimulus is an unmaskable interrupt that allows cold-start ROM loaders and front panels to be implemented for the TMS 9980A/TMS 9981. When the TMS 9980A/TMS 9981 decodes LOAD on IC0-IC2 lines, it initiates an interrupt sequence immediately following the instruction being executed. Memory location 3FFC is used to obtain the vector (WP and PC). The old PC, WP, and ST are loaded into the new workspace and the interrupt mask is set to 0000. Then the program execution resumes using the new PC and WP. Recognition of LOAD by the processor will also terminate the idle condition. External stimulus for LOAD must be held active (on IC0-IC2) for one instruction period by using IAQ signal.

2.7.2 RESET

When the TMS 9980A/TMS 9981 recognizes a RESET on IC0-IC2, it resets and inhibits $\overline{WE}$ and CRUCLK. Upon removal of the RESET code, the TMS 9980A/TMS 9981 initiates a level-zero interrupt sequence that acquires WP and PC from location 0000 and 0002, sets all status register bits to zero and starts execution. Recognition of RESET by the processor will also terminate an idle state. External stimulus for RESET must be held active for a minimum of three clock cycles.

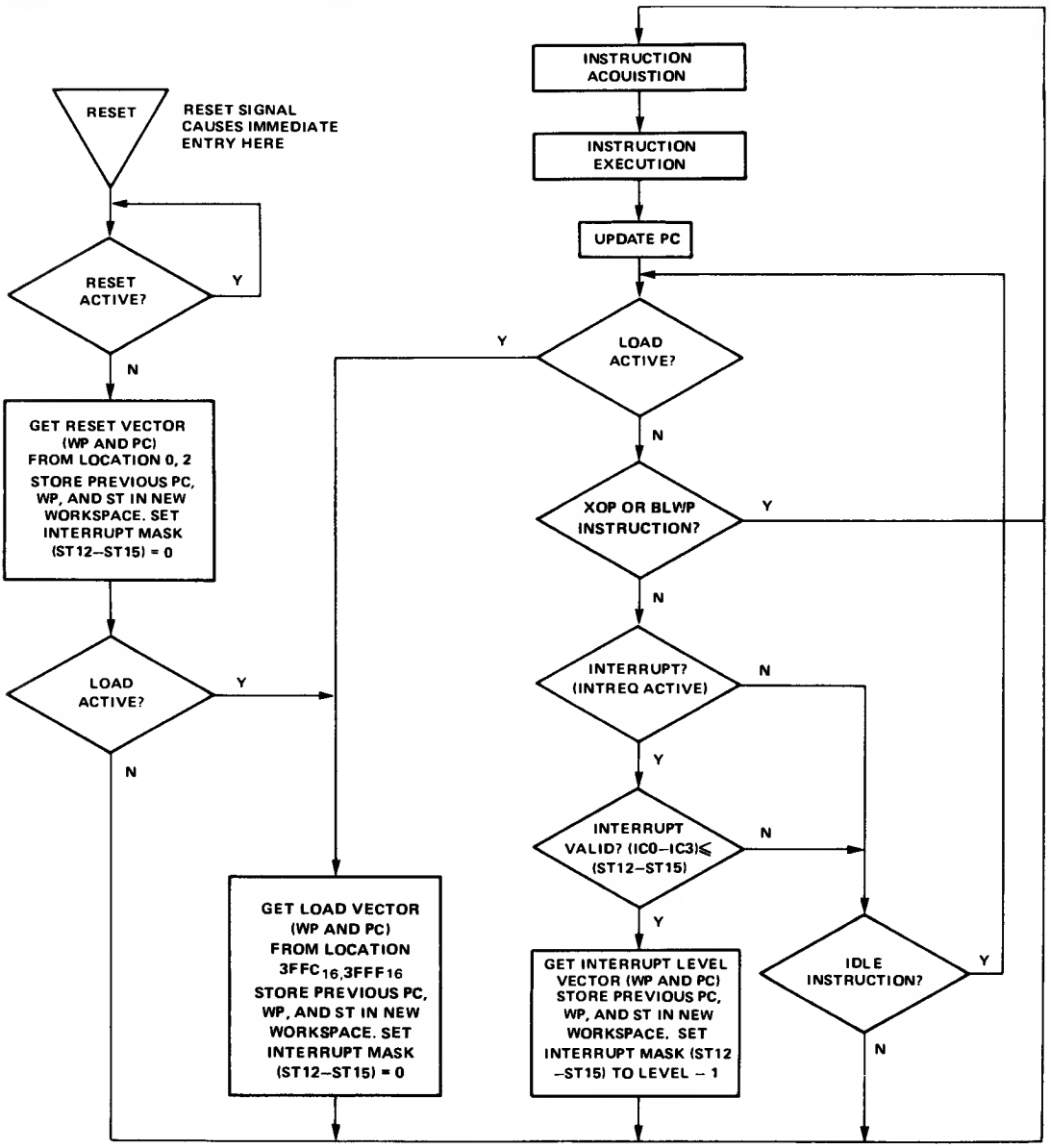


FIGURE 8 – TMS 9980A/TMS 9981 CPU FLOW CHART

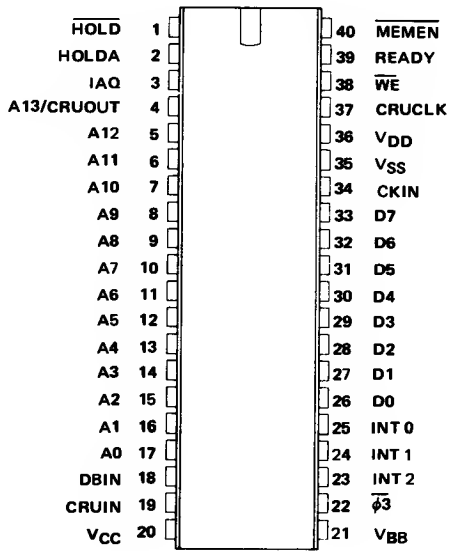
2.8 TMS 9980A PIN DESCRIPTION

Table 2 defines the TMS 9980A pin assignments and describes the function of each pin.

TABLE 2
TMS 9980A PIN ASSIGNMENTS AND FUNCTIONS

SIGNATURE	PIN	I/O	DESCRIPTION
			ADDRESS BUS
A0 (MSB)	17	OUT	A0 through A13 comprise the address bus. This 3-state bus provides the memory-address vector to the external-memory system when MEMEN is ective and I/O-bit addresses and external-instruction addresses to the I/O system when $\overline{\text{MEMEN}}$ is inactive. The address bus assumes the high-impedance state when HOLDA is active.
A1	16	OUT	
A2	15	OUT	
A3	14	OUT	
A4	13	OUT	
A5	12	OUT	
A6	11	OUT	
A7	10	OUT	
AB	9	OUT	
A9	B	OUT	
A10	7	OUT	
A11	6	OUT	
A12	5	OUT	
A13/CRUOUT	4	OUT	**CRUOUT** Seriel I/O data appears on A13 when an LDCR, SBZ end SBO instruction is executed. This dete should be sampled by the I/O interfece logic when CRUCLK goes active (high). One bit of the externel instruction code appeers on A13 during external instruction execution.
			DATA BUS
D0 (MSB)	26	I/O	D0 through D7 comprise the bidirectional 3-state data bus. This bus transfers memory date to (when writing) end from (when reading) the external-memory system when $\overline{\text{MEMEN}}$ is active. The data bus assumes the high-impedance state when HOLDA is active.
D1	27	I/O	
D2	28	I/O	
D3	29	I/O	
D4	30	I/O	
D5	31	I/O	
D6	32	I/O	
D7 (LSB)	33	I/O	
			POWER SUPPLIES
V_{BB}	21		Supply voltage (−5V NOM)
V_{CC}	20		Supply voltage (5 V NOM)
V_{DD}	36		Supply voltage (12 V NOM)
V_{SS}	35		Ground reference
			CLOCKS
CKIN	34	IN	Clock In. A TTL compatible input used to generete the internal 4-phase clock. CKIN frequency is 4 times the desired system frequency.
$\overline{\phi 3}$	22	OUT	Clock phase 3 ($\phi 3$) inverted; used as a timing reference.
			BUS CONTROL
DBIN	1B	OUT	Deta bus in. When active (high), DBIN indicetes that the TMS 9980A hes disabled its output buffers to allow the memory to place memory-read deta on the data bus during MEMEN. DBIN remains low in all other cases except when HOLDA is active et which time it is in the high-impedance stete.

TMS 9980A PIN ASSIGNMENTS



8

TABLE 2 (CONTINUED)

SIGNATURE	PIN	I/O	DESCRIPTION
$\overline{\text{MEMEN}}$	40	DUT	Memory enable. When active (low), $\overline{\text{MEMEN}}$ indicates that the address bus contains a memory address. When HDLDA is activa, $\overline{\text{MEMEN}}$ is in the high impedance stete.
$\overline{\text{WE}}$	38	DUT	Write enable. When active (low), $\overline{\text{WE}}$ indicates that memory-write data is available from the TMS 9980 to be written into memory. When HOLDA is active, $\overline{\text{WE}}$ is in the high-impadence stete.
CRUCLK	37	OUT	CRU clock. When active (high), CRUCLK indicates that external interface logic should sample the output data on CRUOUT or should decode external instructions on A0, A1, A13.
CRUIN	19	IN	CRU data in. CRUIN, normally driven by 3-state or open-collector devices, receives input data from external interface logic. When the processor executes a STCR or TB instruction, it samples CRUIN for the level of the CRU input bit specified by the address bus (A2 through A12).
INT2	23	IN	Interrupt code. Refer to Section 2.2 for detailed description.
INT1	24	IN	
INT0	25	IN	
			MEMORY CONTROL
$\overline{\text{HOLD}}$	1	IN	Hold. When active (low), $\overline{\text{HOLD}}$ indicates to the processor that an external controller (e.g., DMA device) desires to utilize the address and deta buses to trensfer deta to or from memory. The TMS 9980A enters the hold state following a hold signal when it has completed its present memory cycle.* The processor then places the address and data buses in the high-impedance state (along with $\overline{\text{WE}}$, $\overline{\text{MEMEN}}$, and DBIN) and responds with a hold-acknowledge signal (HOLDA). When $\overline{\text{HDLD}}$ is removed, the processor returns to normal operation.
HOLDA	2	DUT	Hold acknowledge. When active (high), HOLDA indicates that the processor is in the hold state and the address and data buses and memory control outputs ($\overline{\text{WE}}$, $\overline{\text{MEMEN}}$, and OBIN) are in the high-impedance state.
READY	39	IN	Ready. When active (high), READY indicates that memory will be ready to read or write during the next clock cycle. Whan not-reedy is indicated during a memory operation, the TMS 9980A enters a wait state and suspends internal operation until the memory systems indicated ready.
			TIMING AND CONTROL
IAQ	3	OUT	Instruction acquisition. IAQ is active (high) during any mamory cycle when the TMS 9980A is acquiring an instruction. IAQ can be used to detect illegal op codes. It may also be used to synchronize LOAD stimulus.

*If the cycle following the present memory cycle is also a memory cycle it, too, is completed before TMS 9980 enters hold state.

▶8

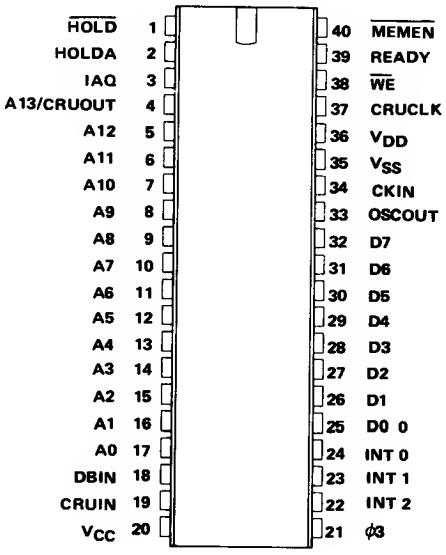
2.9 TMS 9981 PIN DESCRIPTION

Table 3 defines the TMS 9981 pin assignments and describes the function of each pin.

TABLE 3
TMS 9981 PIN ASSIGNMENTS AND FUNCTIONS

SIGNATURE	PIN	I/O	DESCRIPTION
A0(MSB)	17	OUT	**ADDRESS BUS**
A1	16	OUT	A0 through A13 comprise the address bus.
A2	15	OUT	This 3-state bus provides the memory-
A3	14	OUT	address vector to the external-memory sys-
A4	13	OUT	tem when MEMEN is active and I/O-bit
A5	12	OUT	addresses and external-instruction eddresses
A6	11	OUT	to the I/O system when $\overline{\text{MEMEN}}$ is inactive.
A7	10	OUT	The address bus assumes the high-impedance
A8	9	OUT	state when HOLDA is active.
A9	8	OUT	
A10	7	OUT	
A11	6	OUT	
A12	5	OUT	
A13/CRUOUT	4	OUT	**CRUOUT**
			Serial I/O data appears on A13 when an LOCR, SBZ and SBO instruction is executed. This data should be sampled by the I/O interface logic when CRUCLK goes active (high). One bit of the external instruction code appeers on A13 during external instruction execution.
D0 (MSB)	25	I/O	**DATA BUS**
D1	26	I/O	D0 through D7 comprise the bidirectional
D2	27	I/O	3-state deta bus. This bus transfers memory
D3	28	I/O	data to (when writing) and from (when
D4	29	I/O	reading) the external-memory system when
D5	30	I/O	$\overline{\text{MEMEN}}$ is active. The data bus essumes the
D6	31	I/O	high-impedance state when HOLDA is
D7 (LSB)	32	I/O	active.
			POWER SUPPLIES
V_{CC}	20		Supply voltage (5 V NOM)
V_{DD}	36		Supply voltage (12 V NOM)
V_{SS}	35		Ground reference
CKIN	34	IN	**CLOCKS**
OSCOUT	33	OUT	Clock In and Oscillator Out. These pins may be used in either of two modes to generate the internal 4-phese clock. In mode 1 a crystal of 4 times the desired system frequency is connected between CKIN and OSCOUT (see Figure 13). In mode 2 OSCOUT is left floating and CKIN is driven by a TTL compatible source whose frequency is 4 times the desired system frequency.
$\overline{\phi 3}$	21	OUT	Clock phase 3 ($\phi 3$) inverted; used es a timing reference.
DBIN	18	OUT	**BUS CONTROL**
			Data bus in. When active (high), OBIN indicates that the TMS 9981 has disabled its output buffers to allow the memory to place memory-read data on the data bus during $\overline{\text{MEMEN}}$. DBIN remains low in all other cases except when HOLDA is active et which time it is in the hlgh-impedance state.

TMS 9981 PIN ASSIGNMENTS



8

TABLE 3 (CONTINUED)

SIGNATURE	PIN	I/O	DESCRIPTION
$\overline{\text{MEMEN}}$	40	OUT	Memory enable. When active (low), $\overline{\text{MEMEN}}$ indicates that the address bus contains a memory address. When HOLDA is active, $\overline{\text{MEMEN}}$ is in the high-impedance state.
$\overline{\text{WE}}$	38	OUT	Write enable. When active (low), $\overline{\text{WE}}$ indicates that memory-write data is available from the TMS 9981 to be written into memory. When HOLDA is active, $\overline{\text{WE}}$ is in the high-impedance state.
CRUCLK	37	OUT	CRU clock. When active (high), CRUCLK indicates that external interface logic should sample the output data on CRUOUT or should decode external instructions on A0, A1, A13.
CRUIN	19	IN	CRU data in. CRUIN, normally driven by 3-state or open-collector devices, receives input data from external interface logic. When the processor executes a STCR or TB instruction, it samples CRUIN for the level of the CRU input bit specified by the address bus (A2 through A12).
INT2	22	IN	Interrupt code. Refer to Section 2.2 for detailed description.
INT1	23	IN	
INT0	24	IN	
$\overline{\text{HOLD}}$	1	IN	**MEMORY CONTROL** Hold. When active (low), HOLD indicates to the processor that an external controller (e.g., DMA device) desires to utilize the address and data buses to transfer data to or from memory. The TMS 9981 enters the hold state following a hold signal when it has completed its present memory cycle.* The processor then places the address and data buses in the high-impedance state (along with $\overline{\text{WE}}$, $\overline{\text{MEMEN}}$, and DBIN) and responds with a hold-acknowledge signal (HOLDA). When $\overline{\text{HOLD}}$ is removed, the processor returns to normal operation.
HOLDA	2	OUT	Hold acknowledge. When active (high), HOLDA indicates that the processor is in the hold state and the address and data buses and memory control outputs ($\overline{\text{WE}}$, $\overline{\text{MEMEN}}$, and DBIN) are in the high-impedance state.
READY	39	IN	Ready. When active (high), READY indicates that memory will be ready to read or write during the next clock cycle. When not-ready is indicated during a memory operation, the TMS 9981 enters a wait state and suspends internal operation until the memory systems indicated ready.
IAQ	3	OUT	**TIMING AND CONTROL** Instruction acquisition. IAQ is active (high) during any memory cycle when the TMS 9981 is acquiring an instruction. IAQ can be used to detect illegal op codes. It may also be used to synchronize LOAD stimulus.

*If the cycle following the present memory cycle is also a memory cycle it, too, is completed before TMS 9981 enters hold state.

2.10 TIMING

2.10.1 Memory

Basic memory read and write cycles are shown in Figures 9a and 9b. Figure 9a shows a read and a write cycle with no wait states while Figure 9b shows a read and a write cycle for a memory requiring one wait state.

▶8

$\overline{\text{MEMEN}}$ goes active (low) during each memory cycle. At the same time that $\overline{\text{MEMEN}}$ is active, the memory address appears on the address bits A0 through A13. Since the TMS 9980A/TMS 9981 has an 8-bit data bus, every memory operation consists of two consecutive memory cycles. Address bit A13 is 0 for the first of the two cycles and goes to 1 for the second. If the cycle is a memory-read cycle, DBIN will go active (high) at the same time $\overline{\text{MEMEN}}$ and A0 through A13 become valid. The memory-write ($\overline{\text{WE}}$) signal remains inactive during a read cycle.

The READY signal allows extended memory cycle as shown in Figure 9b.

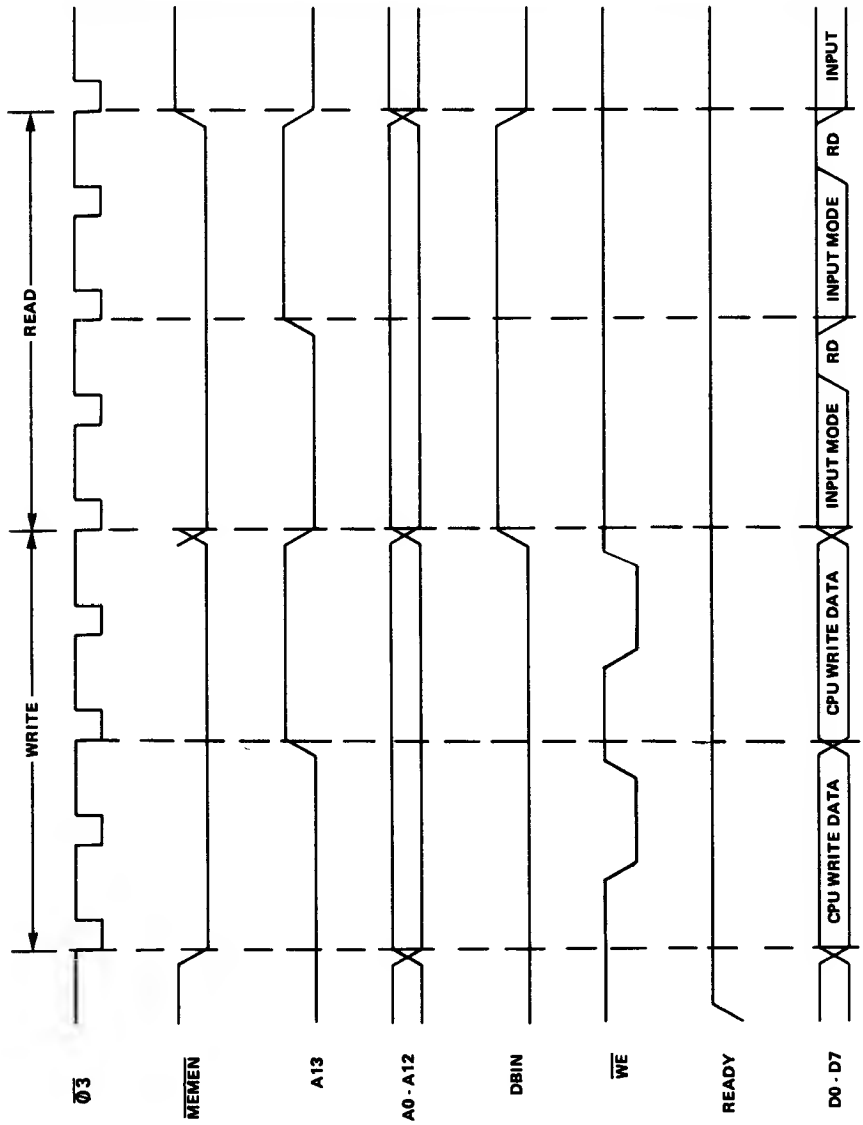


FIGURE 9a – TMS 9980A/TMS 9981 MEMORY BUS TIMING (NO WAIT STATES)

8◀

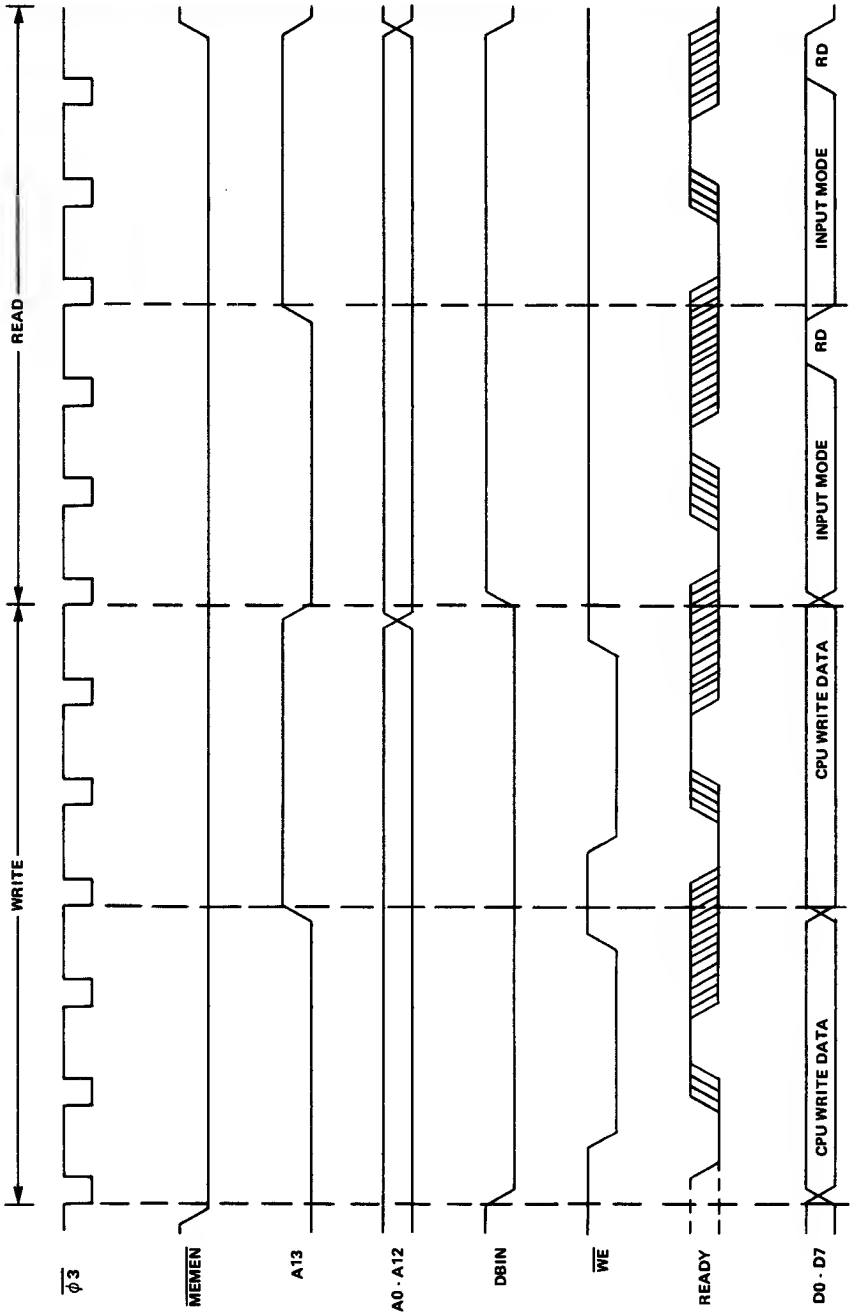


FIGURE 9b – TMS 9980A/TMS 9981 MEMORY BUS TIMING (ONE WAIT STATE)

▶8

At the end of the read cycle, $\overline{MEMEN}$ and DBIN go inactive (high and low respectively). The address bus also changes at this time, however, the data bus remains in the input mode for one clock cycle after the read cycle.

A write cycle is similar to read cycle except that $\overline{WE}$ goes active (low) as shown and valid write data appears on the data bus at the same time the address appears.

2.10.2 HOLD

Other interfaces may utilize the TMS 9980A/TMS 9981 memory bus by using the hold operation (illustrated in Figure 10) of the TMS 9980A/TMS 9981. When $\overline{HOLD}$ is active (low), the TMS 9980A/TMS 9981 enters the hold state at the next available non-memory cycle clock period. When the TMS 9980A/TMS 9981 has entered the hold state HOLDA goes active (high), A0 through A13, D0 through D7, DBIN, $\overline{MEMEN}$, and $\overline{WE}$ go into high-impedance state to allow other devices to use the memory buses. When $\overline{HOLD}$ goes inactive, TMS 9980A/TMS 9981 resumes processing as shown. Considering that there can be a maximum of 6 consecutive memory operations, the maximum delay between $\overline{HOLD}$ going active to HOLDA going active (high) could be $t_{c(\phi)}$ (for set up) + (12 + 6 W) $t_{c(\phi)}$ (delay for HOLDA), where W is the number of wait states per memory cycle and $t_{c(\phi)}$ is the clock cycle time. If hold occurs during a CRU operation, the TMS 9980A/TMS 9981 uses an extra clock cycle (after the removal of the $\overline{HOLD}$ signal) to reassert the CRU address providing the normal setup times for the CRU bit transfer that was interrupted.

2.10.3 CRU

CRU interface timing is shown in Figure 11. The timing for transferring two bits out and one bit in is shown. These transfers would occur during the execution of a CRU instruction. The other cycles of the instruction execution are not illustrated. To output a CRU bit, the CRU-bit address is placed on the address bus A2 through A12 and the actual bit data on A13. During the second clock cycle a CRU pulse is supplied by CRUCLK. This process is repeated until the number of bits specified by the instruction are completed.

The CRU input operation is similar in that the bit address appears on A2 through A12. During the subsequent cycle, the TMS 9980A/TMS 9981 accepts the bit input data as shown. No CRUCLK pulses occur during a CRU input operation.

2.10.4 Interrupt Code (IC0-IC2)

The TMS 9980A/TMS 9981 uses 4 phase clock ($\phi1$, $\phi2$, $\phi3$, and $\phi4$) for timing and control of the internal operations. IC0-IC2 are sampled during $\phi4$ and then during $\phi2$.

If these two successive samples are equal, the code is accepted and latched for internal use on the subsequent $\phi1$. In systems with simple interrupt structures this allows the interrupt code to change asynchronously without the TMS 9980A/TMS 9981 accepting erroneous codes. Figure 3 shows systems with a single level of external interrupt implemented that would require no external timing. When implementing multiple external interrupts, as in the bottom diagram of Figure 3, external synchronization of interrupt requests is required. See Figure 12 for a timing diagram. In systems with more than one external interrupt, the interrupts should be synchronized with the $\overline{\phi3}$ output of the TMS 9980A/TMS 9981 to avoid code transitions on successive sample cycles. This synchronization ensures that the TMS 9980A/TMS 9981 will service only the proper active interrupt level.

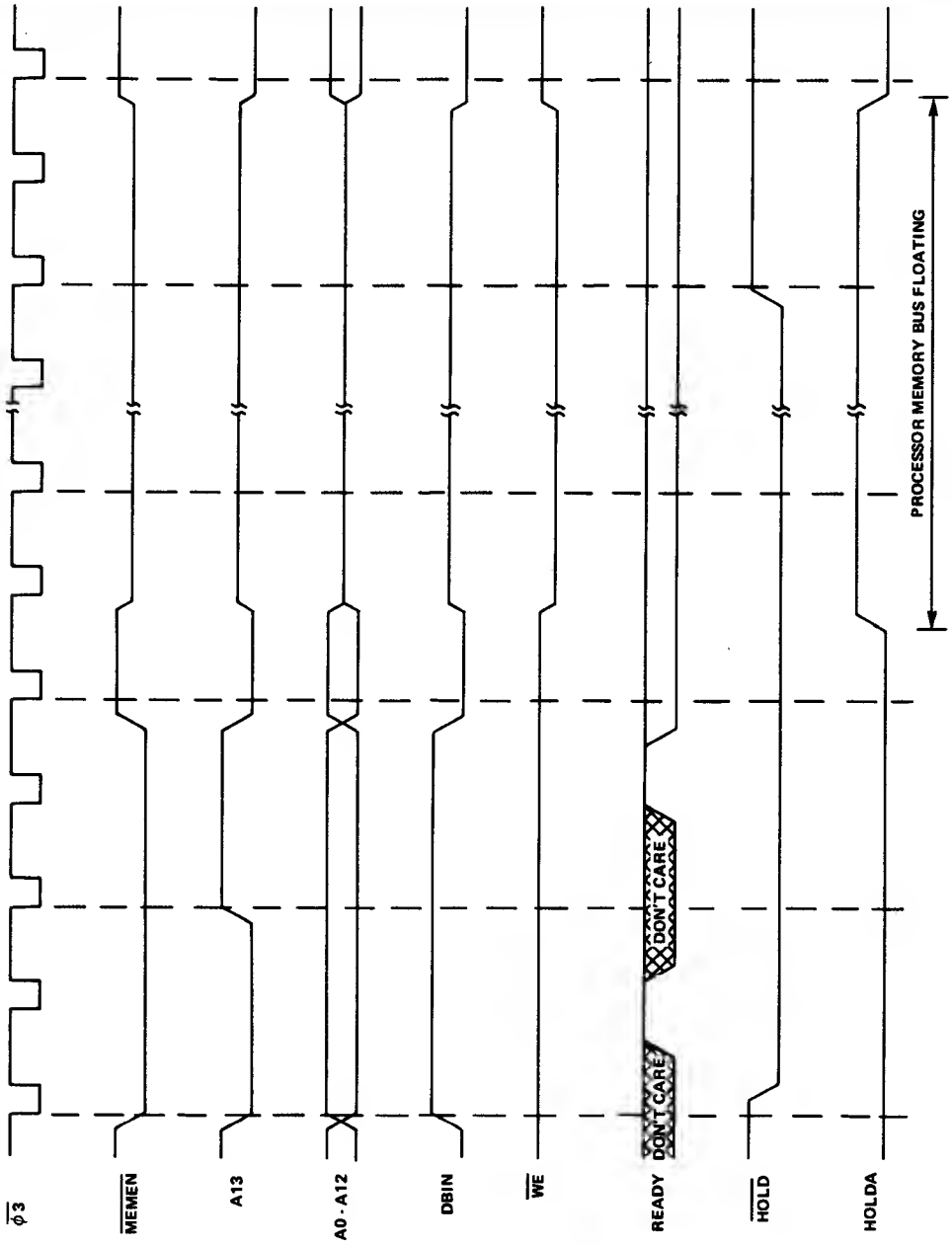


FIGURE 10 – TMS 9980A/TMS 9981 HOLD TIMING

▶8

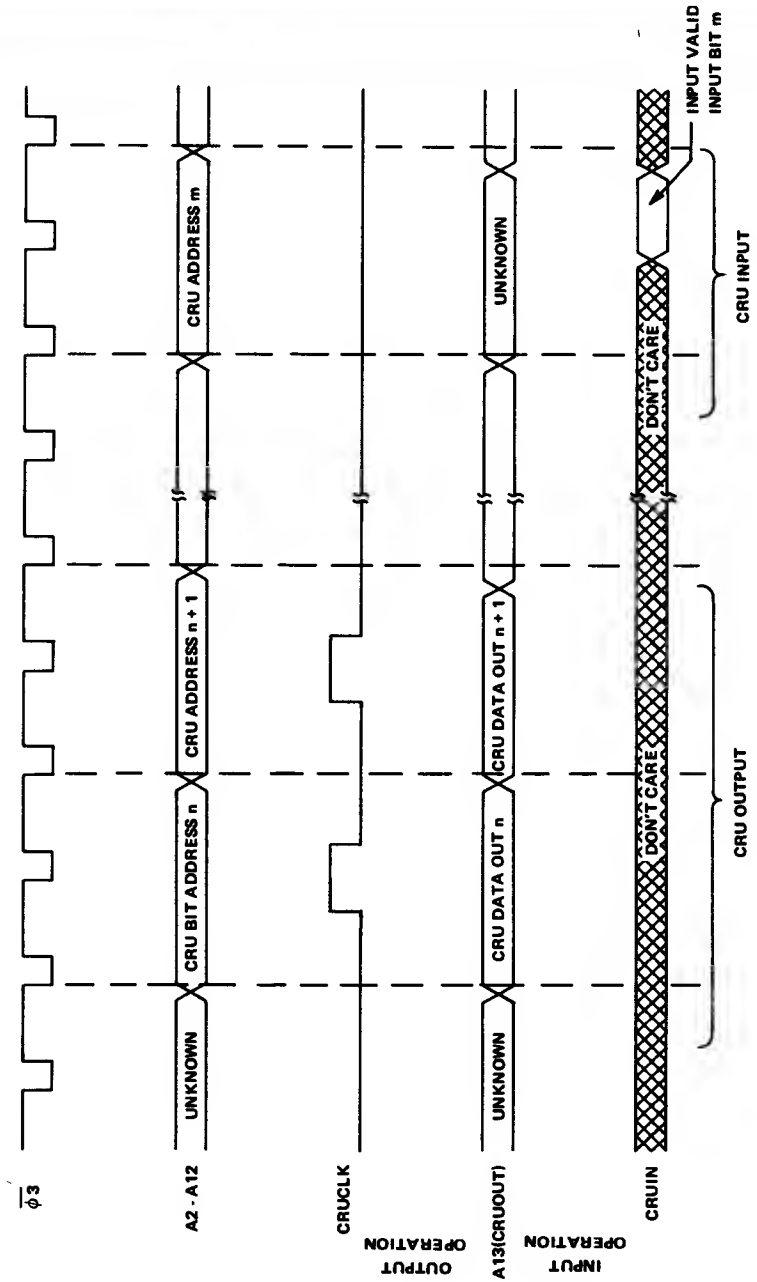


FIGURE 11 – TMS 9980A/TMS 9981 CRU INTERFACE TIMING

8

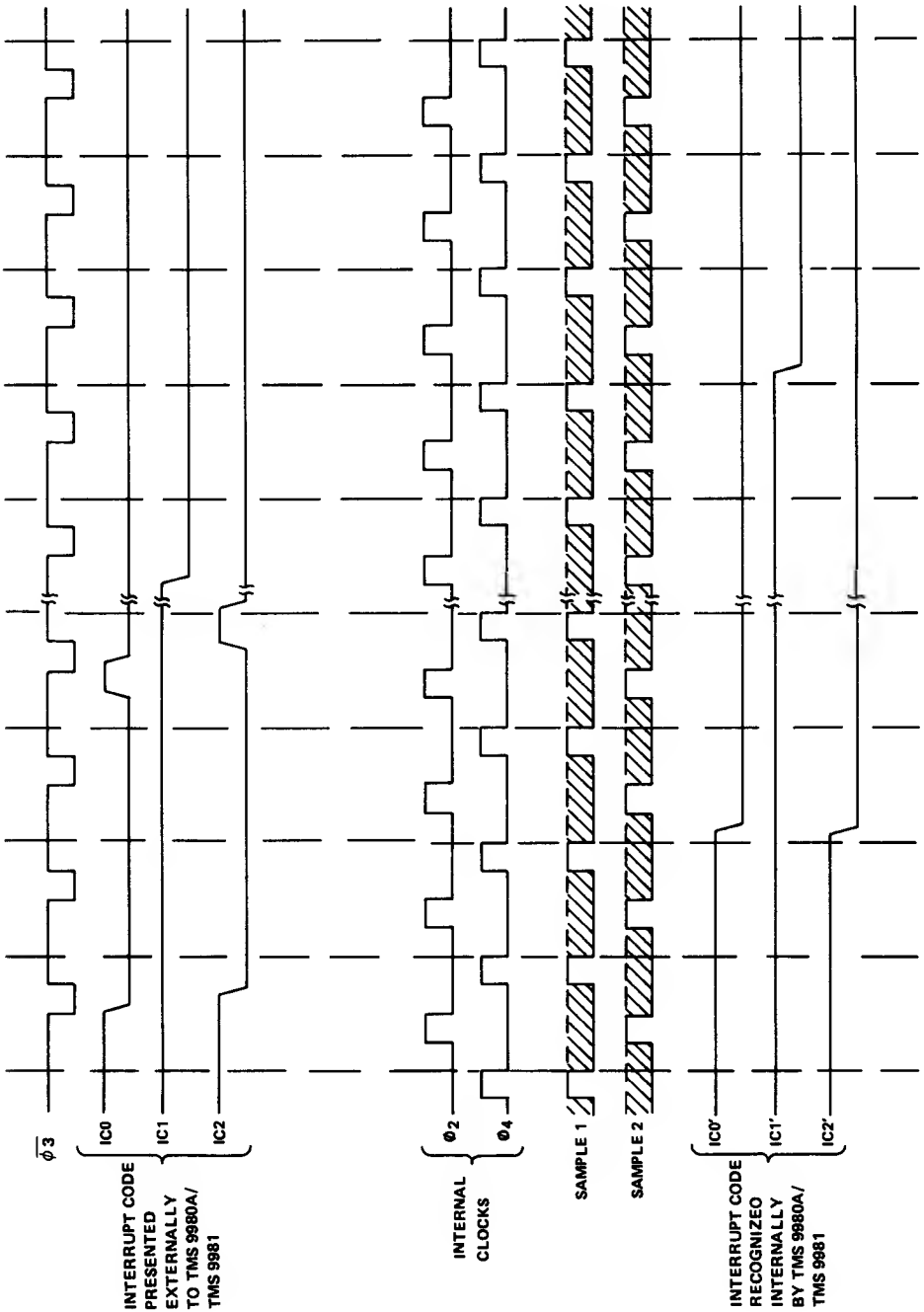


FIGURE 12 – INTERRUPT CODE TIMING

▶8

3.6 TMS 9980A/TMS 9981 INSTRUCTION EXECUTION TIMES

Instruction execution times for the TMS 9980A/TMS 9981 are a function of:

1) Clock cycle time, $t_{c(\phi)}$

2) Addressing mode used where operands have multiple addressing mode capability

3) Number of wait states required per memory access.

Table 4 lists the number of clock cycles and memory accesses required to execute each TMS 9980A/TMS 9981 instruction. For instructions with multiple addressing modes for either or both operands, Table 4 lists the number of clock cycles and memory accesses with all operands addressed in the workspace-register mode. To determine the additional number of clock cycles and memory accesses required for modified addressing, add the appropriate values from the referenced tables. The total instruction-execution time for an instruction is:

$$T = t_{c(\phi)} (C+W \cdot M)$$

where:

T = total instruction time;

$t_{c(\phi)}$ = clock cycle time;

C = number of clock cycles for instruction execution plus address modification;

W = number of required wait states per memory access for instruction execution plus address modification;

M = number of memory accesses.

As an example, the instruction MOVB is used in a system with $t_{c(\phi)}$= 0.400μs and no wait states are required to access memory. Both operands are addressed in the workspace register mode:

$$T = t_{c(\phi)} (C+W \cdot M) = 0.400\ (22+0 \cdot 8) = 8.8\ \mu s.$$

If two wait states per memory access were required, the execution time is:

$$T = 0.400\ (22 + 2 \cdot 8)\ \mu s = 15.2\ \mu s.$$

If the source operand was addressed in the symbolic mode and two wait states were required:

$$T = t_{c(\phi)} (C+W \cdot M)$$

$$C = 22 + 10 = 32$$

$$M = 8 + 2 = 10$$

$$T = 0.400\ (32 + 2 \cdot 10) = 20.8\ \mu s.$$

8

TABLE 4
INSTRUCTION EXECUTION TIMES

INSTRUCTION	CLOCK CYCLES C	MEMORY ACCESS M	ADDRESS MODIFICATION***	
			SOURCE	DESTINATION
A	22	8	A	A
AB	22	8	B	8
ABS (MS8 = 0)	16	4	A	–
(MS8 = 1)	20	6	A	–
AI	22	8	–	–
ANDI	22	8	–	–
B	12	4	A	–
BL	18	6	A	–
BLWP	38	12	A	–
C	20	6	A	A
CB	20	6	B	8
CI	20	6	–	–
CKOF	14	2	–	–
CKON	14	2	–	–
CLR	16	6	A	–
COC	20	6	A	–
CZC	20	6	A	–
DEC	16	6	A	–
OECT	16	6	A	–
OIV (ST4 is set)	22	6	A	–
DIV (ST4 is reset)*	104-136	12	A	–
IDLE	14	2	–	–
INC	16	6	A	–
INCT	16	6	A	–
INV	16	6	A	–
Jump (PC is changed)	12	2	–	–
(PC is not changed)	10	2	–	–
LOCR (C = 0)	58	6	A	–
(1<C<8)	26+2C	6	8	–
(9<C<15)	26+2C	6	A	–
LI	18	6	–	–
LIMI	22	6	–	–
LREX	14	2	–	–
LWPI	14	4	–	–
MOV	22	8	A	A
MOVB	22	8	B	B
MPY	62	10	A	–
NEG	18	6	A	–
ORI	22	8	–	–
RSET	14	2	–	–
RTWP	22	8	–	–
S	22	8	A	A
SB	22	8	8	8
SBO	16	4	–	–
SBZ	16	4	–	–
SETO	16	6	A	–
Shift (C ≠ 0)	18+2C	6	–	–
(C ≠0, Bits 12–15 of WRO = 0)	60	B	–	–
(C = 0, Bits 12–15 of WRP = N ≠ 0)	28+2N	8	–	–
SOC	22	8	A	A
SOCB	22	8	B	B
STCR (C = 0)	68	8	A	–
(1<C<7)	50	8	B	–
(C = 8)	52	8	B	–
(9<C<15)	66	8	A	–

*Execution time is dependent upon the partial quotient after each clock cycle during execution.

***The letters A and B refer to the respective tables that follow.

TABLE 4 (CONTINUED)

INSTRUCTION	CLOCK CYCLES C	MEMORY ACCESS M	ADDRESS MODIFICATION*** SOURCE	DESTINATION
STST	12	4	–	–
STWP	12	4	–	–
SWPB	16	6	A	–
SZC	22	8	A	A
SZCB	22	8	B	B
TB	16	4	–	–
X**	12	4	A	–
XOP	52	16	A	–
XOR	22	8	A	–
RESET function	36	10	–	–
LOAD function	32	10	–	–
Interrupt context switch	32	10	–	–
Undefined op codes: 0000-01FF, 0320 033F, 0C00-0FFF, 0780-07FF	B	2	–	–

**Execution time is added to the execution time of the instruction located at the source address.
***The letters A and B refer to the respective tables that follow.

ADDRESS MODIFICATION – TABLE A

ADDRESSING MODE	CLOCK CYCLES C	MEMORY ACCESSES M
WR (T_S or T_D = 00)	0	0
WR indirect (T_S or T_D = 01)	6	2
WR indirect auto-increment (T_S or T_D = 11)	12	4
Symbolic (T_S or T_D = 10, S or D = 0)	10	2
Indexed (T_S or T_D = 10, S or D ≠ 0)	12	4

ADDRESS MODIFICATION – TABLE B

ADDRESSING MODE	CLOCK CYCLES C	MEMORY ACCESSES M
WR (T_S or T_D = 00)	0	0
WR indirect (T_S or T_D = 01)	6	2
WR indirect auto-increment (T_S or T_D = 11)	10	4
Symbolic (T_S or T_D = 10, S or D = 0)	10	2
Indexed (T_S or T_D = 10, S or D ≠ 0)	12	4

4. TMS 9980A/TMS 9981 ELECTRICAL SPECIFICATIONS

4.1 ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Supply voltage, V_{CC} (see Note 1)	−0.3 to 15 V
Supply voltage, V_{DD} (see Note 1)	−0.3 to 15 V
Supply voltage, V_{BB} (see Note 1) (9980A only)	−5.25 to 0 V
All input voltages (see Note 1)	−0.3 to 15 V
Output voltage (see Note 1)	−2 V to 7 V
Continuous power dissipation	1.4 W
Operating free-air temperature range	0°C to 70°C
Storage temperature range	−55°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" mey cause permenent damege to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: Under ebsolute meximum ratings voltege values are with respect to V_{SS}.

4.2 RECOMMENDED OPERATING CONDITIONS

	MIN	NOM	MAX	UNIT
Supply voltage, V_{BB} (9980A only)	−5.25	−5	−4.75	V
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply voltage, V_{DD}	11.4	12	12.6	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH}	2.2	2.4	V_{CC}+1	V
Low-level input voltage, V_{IL}	−1	0.4	0.8	V
Operating free-air temperature, T_A	0	20	70	°C

4.3 ELECTRICAL CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (UNLESS OTHERWISE NOTED)

PARAMETER			TEST CONDITIONS	MIN	TYP*	MAX	UNIT
I_I	Input current	Data bus during DBIN	V_I = V_{SS} to V_{CC}			±75	μA
		$\overline{WE}$, $\overline{MEMEN}$, $\overline{DBIN}$ during HOLDA	V_I = V_{SS} to V_{CC}			±75	μA
		Any other inputs	V_I = V_{SS} to V_{CC}			±10	μA
V_{OH}	High-level output voltage		I_O = −0.4 mA	2.4			V
V_{OL}	Low-level output voltage		I_O = 2 mA			0.5	V
			I_O = 3.2 mA			0.65	V
I_{BB}	Supply current from V_{BB} (9980A Only)					1	mA
I_{CC}	Supply current from V_{CC}		0°C		50	60	mA
			70°C		40	50	mA
I_{DD}	Supply current from V_{DD}		0°C		70	80	mA
			70°C		65	75	mA
C_I	Input capacitance (any inputs except data bus)		f = 1 MHz, unmeasured pins at V_{SS}		15		pF
C_{DB}	Data bus capacitance		f = 1 MHz, unmeasured pins at V_{SS}		25		pF
C_O	Output capacitance (any output except data bus)		f = 1 MHz, unmeasured pins at V_{SS}		15		pF

*All typical values are at T_A = 25°C and nominal voltages

4.4 CLOCK CHARACTERISTICS

The TMS 9980A and TMS 9981 have an internal 4-phase clock generator/driver. This is driven by an external TTL compatible signal to control the phase generation. In addition, the TMS 9981 provides an output (OSCOUT) that in conjunction with CKIN forms an on-chip crystal oscillator. This oscillator requires an external crystal and two capacitors as shown in Figure 13. The external signal or crystal must be 4 times the desired system frequency.

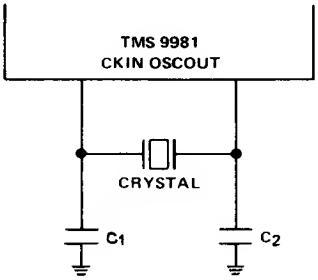


FIGURE 13 – CRYSTAL OSCILATOR CIRCUIT

4.4.1 Internal Crystal Oscillator (9981 Only)

The internal crystal oscillator is used as shown in Figure 13. The crystal should be a fundamental series resonant type. C_1 and C_2 represent the total capacitance on these pins including strays and parasitics.

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Crystal frequency	0°C-70°C	6		10	MHZ
C_1, C_2	0°C-70°C	10	15	25	pf

4.4.2 External Clock

The external clock on the TMS 9980A and optional on the TMS 9981, uses the CKIN pin. In this mode the OSCOUT pin of the TMS 9981 must be left floating. The external clock source must conform to the following specifications.

PARAMETER		MIN	TYP	MAX	UNIT
f_{ext}	External source frequency*	6		10	MHz
V_H	External source high level	2.2			V
V_L	External source low level			0.8	V
T_r/T_f	External source rise/fall time		10		ns
T_{WH}	External source high level pulse width	40			ns
T_{WL}	External source low level pulse width	40			ns

*This allows a system speed of 1.5 MHz to 2 MHz.

8

4.5 SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS

The timing of all the inputs and outputs are controlled by the internal 4 phase clock; thus all timings are based on the width of one phase of the internal clock. This is $1/f_{(CKIN)}$ (whether driven or from a crystal). This is also $\frac{1}{4}f_{system}$. In the following table this phase time is denoted t_w.

All external signals are with reference to $\phi 3$ (see Figure 14).

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_r(\phi 3)$	Rise time of $\phi 3$		3	5	10	ns
$t_f(\phi 3)$	Fall time of $\phi 3$		5	7.5	15	ns
$t_w(\phi 3)$	Pulse width of $\phi 3$	$t_w = 1/f(CKIN) = \frac{1}{4}f_{system}$	t_w-15	t_w-10	t_w+10	ns
t_{su}	Data or control setup time*		t_w-30			ns
t_h	Data hold time*		$2t_w+10$			ns
$t_{PHL}(\overline{WE})$	Propagation delay time WE high to low	$C_L = 200pf$	t_w-10	t_w	t_w+20	ns
$t_{PLH}(\overline{WE})$	Propagation delay time WE low to high		t_w	t_w+10	t_w+30	ns
t_{PHL}(CRUCLK)	Propagation delay time, CRUCLK high to low		−20	−10	+10	ns
t_{PLH}(CRUCLK)	Propagation delay time, CRUCLK low to high		$2t_w-10$	$2t_w$	$2t_w+20$	ns
t_{OV}	Delay time from output valid to $\phi 3$ low		t_w-50	t_w-30		ns
t_{OX}	Delay time from output invalid to $\phi 3$ low			t_w-20	t_w	ns

*All inputs except IC0-IC2 must be synchronized to meet these requirements. IC0-IC2 may change asynchronously. See section 2.10.4.

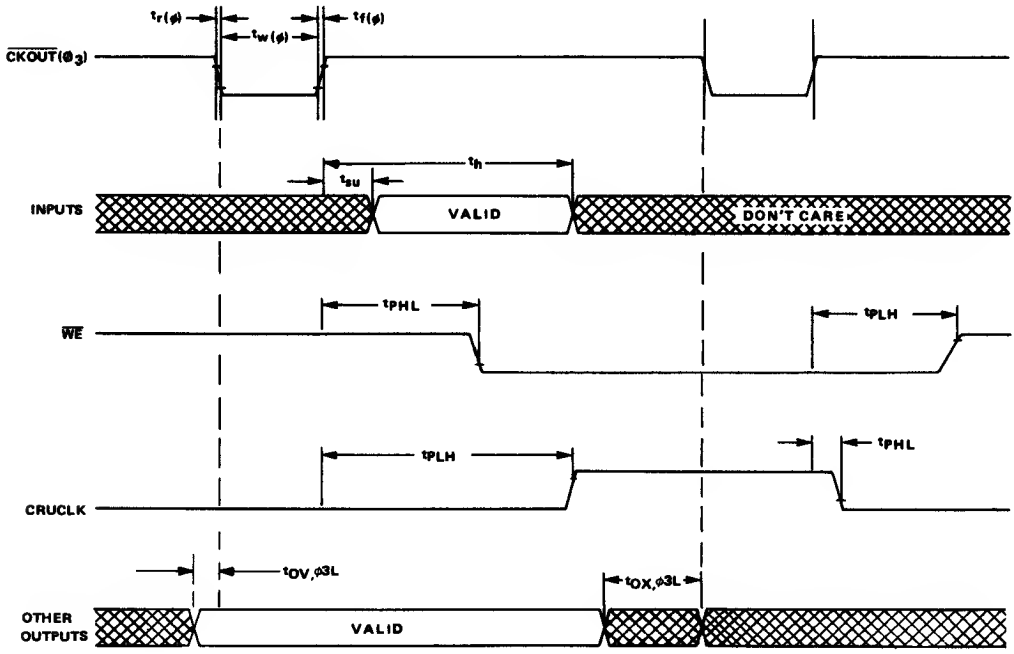


FIGURE 14 – EXTERNAL SIGNAL TIMING DIAGRAM

TMS 9940

8

INTRODUCTION

DESCRIPTION

The TMS 9940 is a single-chip, 16-bit microcomputer containing a CPU, memory (RAM and EPROM/ROM), and extensive I/O. Except for four instructions that do not apply to the TMS 9940 microcomputer configuration, the TMS 9940 instruction set matches that of the TMS 9900 and includes capabilities offered by minicomputers. In addition, the TMS 9940 instruction set includes two instructions that facilitate manipulation of binary coded decimal (BCD) data, and a single-word load-interrupt-mask (LIIM) instruction.

The unique memory-to-memory architecture features multiple register files, resident in the RAM, which allow faster response to interrupts and increased programming flexibility. The memory consists of 128 bytes of RAM and 2048 bytes of EPROM/ROM. The TMS 9940 implements four levels of interrupts, including an internal decrementer which can be programmed as a timer or an event counter. All members of the TMS 9900 family of peripheral circuits are compatible with the TMS 9940. The TMS 9940 is fully supported by software and hardware development systems. The TMS 9940 is fully supported by factory applications engineers and technical answering services.

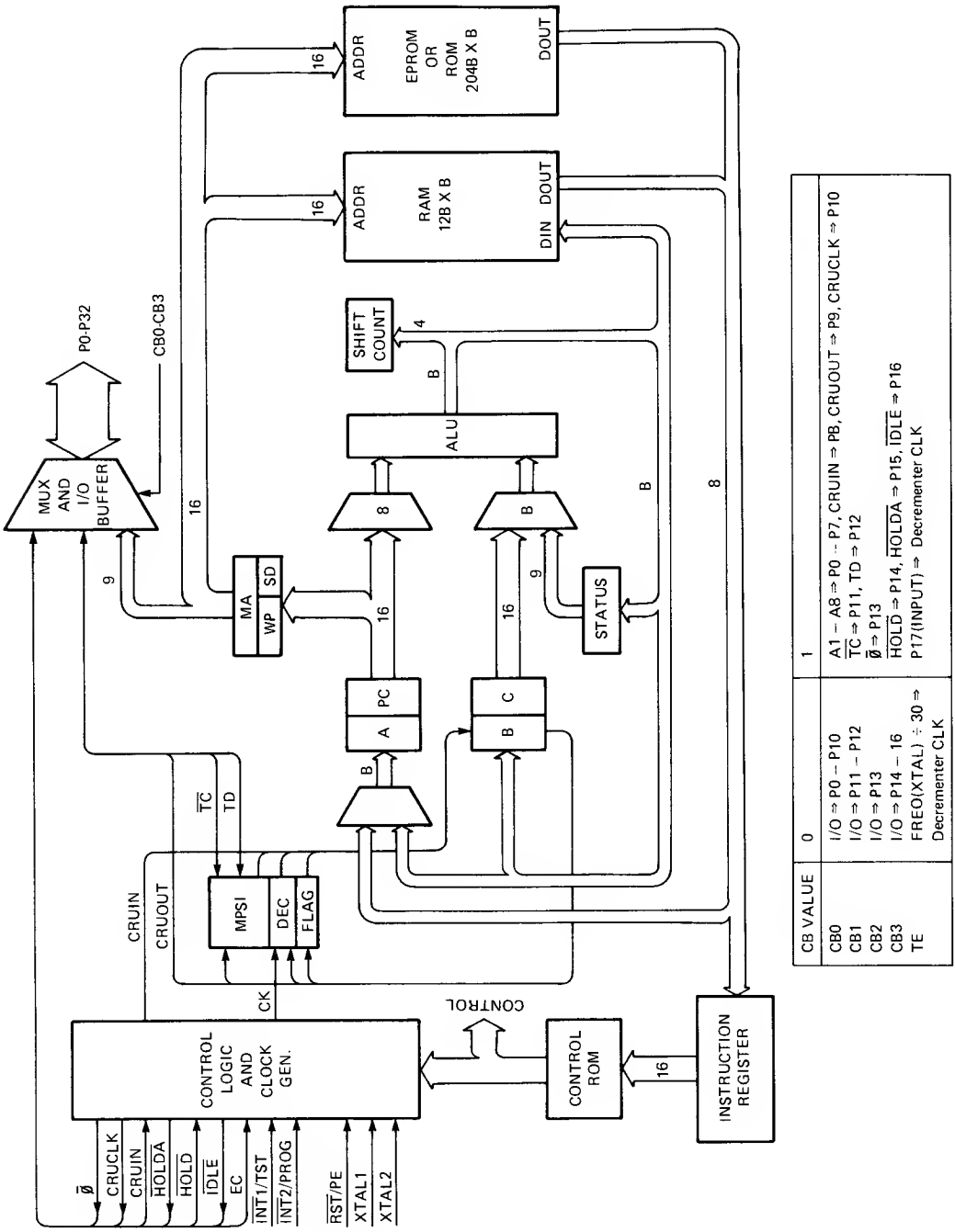
KEY FEATURES

- 16-bit instruction word;
- Minicomputer instruction set including multiply and divide;
- 2048 bytes of EPROM (TMS 9940E)/ROM (TMS 9940M) on chip;
- 128 bytes of RAM on chip;
- 16 general purpose registers;
- 4 prioritized interrupts;
- On-chip timer/event counter;
- 32 bits general purpose I/O Ports;
- 256 bits I/O expansion;
- Easy test function ;
- Multiprocessor system interface;
- Power down capability for low stand-by power;
- Five speed ranges for maximum performance;
- N-channel silicon gate MOS, 5 volt power supply;
- An EPROM device, the TMS 9940E, is contained in a 40-pin, 600-mil, dual-in-line ceramic package with quartz lid;
- A mask ROM device, the TMS 9940M, is contained in a 40-pin, 600-mil, dual-in-line plastic or ceramic package.

PARTS IDENTIFICATION

EPROM DEVICE	MASK-ROM DEVICE	OSCILLATOR FREQUENCY MHz (NOM)
TMS 9940E	TMS 9940M	5
TMS 9940E-40	TMS 9940M-40	4
TMS 9940E-30	TMS 9940M-30	3
TMS 9940E-20	TMS 9940M-20	2
TMS 9940E-10	TMS 9940M-10	1

NOTE: An additional MPXXXX number is used to identify custom ROM codes for TMS 9940M devices.

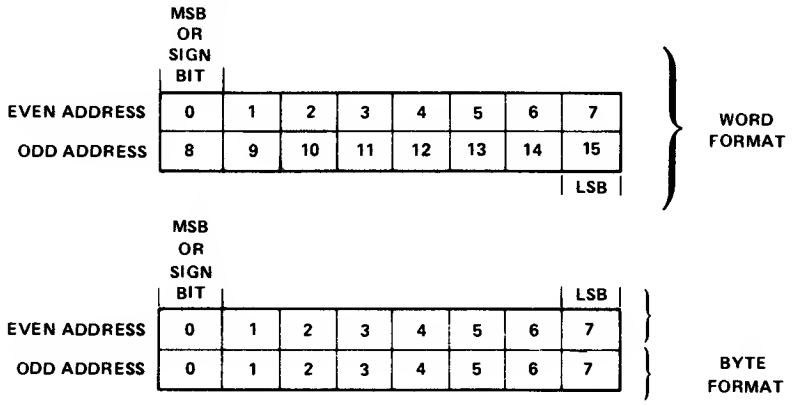


CB VALUE	0	1
CB0	I/O ⇒ P0 – P10	A1 – A8 ⇒ P0 – P7, CRUIN ⇒ PB, CRUOUT ⇒ P9, CRUCLK ⇒ P10
CB1	I/O ⇒ P11 – P12	$\overline{TC}$ ⇒ P11, TD ⇒ P12
CB2	I/O ⇒ P13	Ø ⇒ P13
CB3	I/O ⇒ P14 – 16	$\overline{HOLD}$ ⇒ P14, $\overline{HOLDA}$ ⇒ P15, $\overline{IDLE}$ ⇒ P16
TE	FREQ(XTAL) ÷ 30 ⇒ Decrementer CLK	P17(INPUT) ⇒ Decrementer CLK

Figure 1. TMS 9940 Architecture.

ARCHITECTURE

Memory for the TMS 9940 is organized in 8-bit bytes. The processors are nevertheless 16-bit processors requiring two memory accesses for each 16-bit word. A word is defined as 16 bits or two consecutive bytes in memory. The words are restricted to be on even address boundaries, i.e., the most significant half (8 bits) resides at even address and the least significant half resides at the subsequent odd address. A byte can reside at even or odd addresses. The word and byte formats are shown below.



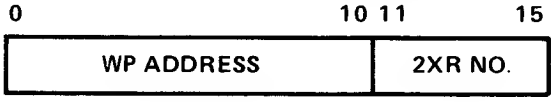
Registers and Memory

The TMS 9940 employs an advanced memory-to-memory architecture where blocks of memory designated as workspaces replace dedicated hardware registers with program-data registers. The TMS 9940 memory map is shown in *Figure 2.* The 2k x 8 EPROM/ROM is assigned memory addresses 0000_{16} through $07FF_{16}$, and the 128 x 8 RAM is assigned memory addresses 8300_{16} through $837F_{16}$.

The first eight words in the EPROM/ROM (addresses 0000_{16} through $000F_{16}$) are used for the interrupt vectors, and 24 words (addresses 0050_{16} through $007F_{16}$) are used for the extended operation (XOP) instruction trap vectors. The remaining memory is available for programs, data, and workspace registers. If desired, any of the special areas may also be used as general EPROM/ROM memory.

Three machine registers are accessible to the user. The 15-bit program counter (PC) contains the address of the instruction following the current instruction being executed. This address is referenced by the processor to fetch the next instruction from memory and is then automatically incremented. The 16-bit status register (ST) contains the present state of the processor. The 11-bit workspace register (WP) points to the first word in the currently active set of workspace registers.

The workspace-register files are nonoverlapping and contain 16 contiguous memory words. Each workspace register may hold data or an address, and function as an operand register, accumulator, address register, or index register. During instruction execution, the processor addresses any register in the workspace by concatenating the 11-bit WP value (bits 0 to 10) with two times the specified register number (bits 11 to 15) as shown below. WP addresses in RAM will be one of four values: 8300_{16}, 8320_{16}, 8340_{16}, and 8360_{16}.



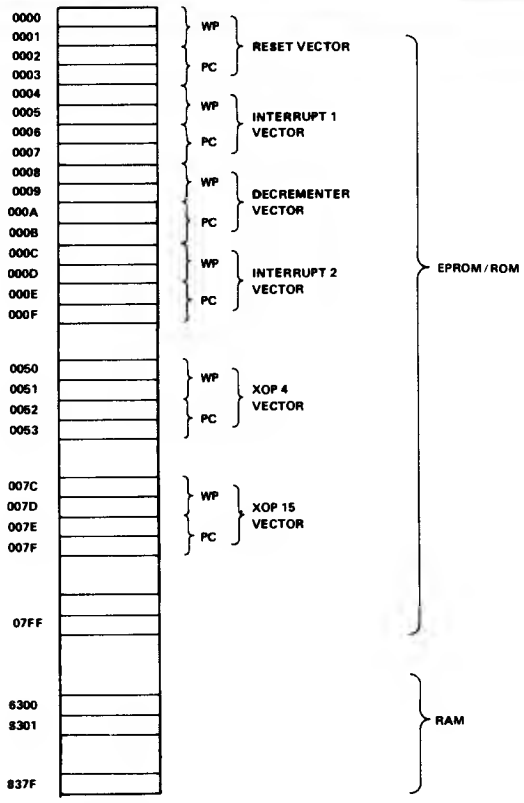
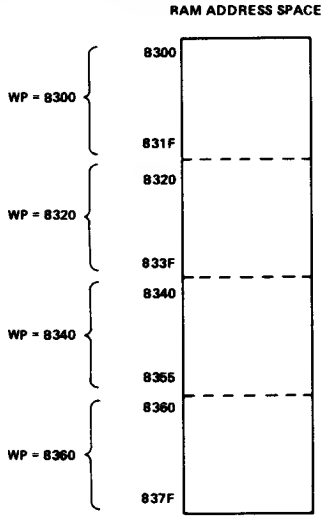


Figure 2. TMS 9940 Memory Map

Up to four nonoverlapping workspaces can be defined in the RAM. The relationship between the workspace pointer value and its corresponding workspace are shown below:



The workspace concept is particularly valuable during operations that require a context switch, which is a change from one program environment to another (as in the case of an interrupt or call to a subroutine). Such an operation, using a conventional multi-register arrangement, requires that at least part of the contents of the register file be stored and reloaded. The TMS 9940, however, can accomplish a complete context switch simply by exchanging the values in the PC, ST, and WP. Instructions in the TMS 9940 that result in a context switch include:

1. Branch and Load Workspace Pointer (BLWP);
2. Return from Subroutine (RTWP);
3. Extended Operation (XOP).

$\overline{\text{RESET}}$, the decrementer interrupt, and the external device interrupts ($\overline{\text{INT}}$1 and $\overline{\text{INT}}$2) also cause a context switch by forcing the processor to trap to a service subroutine.

INTERRUPTS

The TMS9940 implements four hardware interrupt levels. The highest priority interrupt level (level 0) is reserved for the $\overline{\text{RESET}}$ function followed by a user defined external interrupt $\overline{\text{INT1}}$ (level 1), the decrementer (level 2), and the second user defined external interrupt $\overline{\text{INT2}}$ (level 3). The $\overline{\text{RESET}}$ function will be accepted whenever it goes active (e.g., in the middle of an instruction), whereas all other levels are accepted at the end of the presently executing instruction.

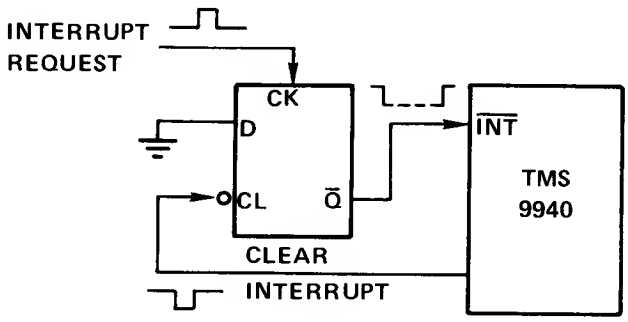
The TMS9940 external interrupt interface consists of three discrete input lines ($\overline{\text{RESET}}$, $\overline{\text{INT1}}$, $\overline{\text{INT2}}$). The input levels are standard TTL levels and the signals require no external synchronization.

The TMS9940 continuously compares the value of the highest priority active interrupt level with the interrupt mask contained in status register bits 14 and 15. When the level of the pending interrupt is less than or equal to the enabling mask value (higher or equal priority interrupt), the processor recognizes the interrupt and initiates a context switch. The processor fetches the new context WP and PC from the interrupt vector locations and stores the previous context WP into R13, PC into R14, and ST into R15, of the new workspace. The interrupt mask is loaded with a value that is one less than the interrupt level being serviced (NOTE: $\overline{\text{RESET}}$ forces the mask value to zero) so that only higher priority interrupts will be recognized during the service routine. The processor also inhibits interrupts until the first instruction of the service routine has been executed to preserve program linkage should a higher priority interrupt occur.

$\overline{\text{RESET}}$ must be held active for a minimum of five clock cycles to guarantee recognition. When $\overline{\text{RESET}}$ is removed the status register and configuration word are set to zero and a level zero interrupt is initiated.

The decrementer interrupt is discussed in detail in a later section. If the decrementer is programmed as an external event counter with a start value of 1, P17/EC will function as a positive edge-triggered interrupt input.

External device interrupt requests are priority level sensitive and, if masked out, must remain active until recognized by the processor executing in the device service routine. The individual service routines must reset the interrupt mask and request before the service routine is complete. A typical schematic to latch in an interrupt requests is shown below:



If a higher priority interrupt becomes active during a service routine, a second context switch occurs to service the higher priority interrupt. When that routine is complete, a return instruction (RTWP) restores the first service routine parameters to the processor to complete processing of the lower priority interrupt. All interrupt service routines should end with the return instruction to restore original program parameters. The interrupt-vector locations, device assignments, and enabling-mask values are shown on *Table 1.*

Table 1. Interrupt Level Data

Interrupt Level		Vector Location (Memory Address In Hex)	Device Assignment	Interrupt Mask Value To Enable Respective Interrupts ST14 & ST15
(Highest Priority)	0	0000	Reset	0 through 3*
	1	0004	External Device	1 through 3
	2	0008	Decrementer	2 and 3
(Lowest Priority)	3	000C	External Device	3 only

*Level 0 cannot be disabled.

As shown in *Table 3,* the positive-logic value of the interrupt pins may be read by CRU instructions even though the interrupt may be masked.

INPUT/OUTPUT

The TMS9940 has a communications register unit (CRU) drive I/O interface. The I/O features implemented on a single 32-bit channel (P0 to P31) are:

- General Purpose I/O: The 32-bits (P0 to P31) of individually controlled I/O data.
- I/O Expansion Via CRU: 256 user configured external I/O bits (P0 to P10).
- Multiprocessor System Interface (P11 and P12): A register for passing commands/data between processors.
- External event counter (P17).
- Power Down

The system engineer's flexibility in the TMS9940 applications is extended greatly by software I/O structuring. The key element is only four bits, called configuration bits (CB), contained in the configuration control register. This register holds the multiplexer control that selects optional modes for 17 of the 32 I/O terminals. The configuration bits controlling these modes are as follows:

Configuration Bit	*Function Controllers*
CB 0	CRU I/O Expansion
CB 1	Multiprocessor System Interface
CB 2	External synchronization (Clock output)
CB 3	Power down and hold Logic

The decrementer used as an event counter has its input available from P17 continuously and does not need a configuration bit to control it.

Generally, a dedicated control system will need only one configuration set up; however, the flexibility allows for multiple configurations dynamically changing for more I/O capacity.

The TMS9940 allows the user to configure part of the I/O pins as special functions for system applications. The configurable pins are shown in *Table 2.*

Table 2. Configuration Bit Effects

MODE	PIN NAME	PIN NO.	CONFIGURATION BIT (CB) NO.	CRU BIT (HEX)	EFFECT OF CB BIT 0	EFFECT OF CB BIT 1	I/O CB=1
CRU Expansion	P0/A1 (MSB)	23	0	183	P0	A1	OUT
	P1/A2	24	0	183	P1	A2	OUT
	P2/A3	25	0	183	P2	A3	OUT
	P3/A4	26	0	183	P3	A4	OUT
	P4/A5	27	0	183	P4	A5	OUT
	P5/A6	28	0	183	P5	A6	OUT
	P6/A7	29	0	183	P6	A7	OUT
	P7/A8	30	0	183	P7	A8	OUT
	P8/CRUIN	18	0	183	P8	CRUIN	IN
	P9/CRUOUT	17	0	183	P9	CRUOUT	OUT
	P10/CRUCLK	16	0	183	P10	CRUCLK	OUT
MPSI	P11/$\overline{TC}$	14	1	184	P11	$\overline{TC}$	I/O
	P12/TD	11	1	184	P12	TD	I/O
SYNC	P13/$\overline{\phi}$	15	2	185	P13	$\overline{\phi}$	OUT
Power Down & Hold	P14/$\overline{HLD}$	10	3	186	P14	$\overline{HLD}$	IN
	P15/$\overline{HLDA}$	9	3	186	P15	$\overline{HLDA}$	OUT
	P16/$\overline{IDLE}$	8	3	186	P16	$\overline{IDLE}$	OUT

Note: P17 is continuously available if the decrementer is used as an event counter.

That is, CB0 controls the I/O Expansion Channel, CB1 controls the MPSI, CB2 allows a clock output, and CB3 configures $\overline{HLD}$, $\overline{HLDA}$, and $\overline{IDLE}$ for power down. Application of $\overline{RESET}$ forces the configuration bits to zero, the all I/O line condition. The configuration can then be changed by outputting the desired bit value to the designated CRU address. (See *Table 3.*)

Communications Register Unit (CRU)

The CRU is a bit-oriented I/O interface through which both input and output bits can be directly addressed individually, or in fields of from 1 to 16 bits. The processor instructions that drive the CRU interface can set, reset, or test any bit in the CRU array or move between memory and CRU data fields. The CRU bit address assignments for all I/O and dedicated functions are shown in *Table 3.*

▶8

CRU instructions that manipulate external data are the only instructions which send CRUCLK pulses out of terminal 16.

Table 3. CRU Bit Address Assignments

CRU Bit Address	CRU Read Data	CRU Write Data
000	I/O Expansion	I/O Expansion
↓	↓	↓
0FF	I/O Expansion	I/O Expansion
180	INT1	—
181	Decrementer Interrupt	Clear Decrementer Interrupt
182	INT2	—
183	—	Configuration Bit 0
184	—	Configuration Bit 1
185	—	Configuration Bit 2
186	—	Configuration Bit 3
190	Decrementer (LSB)	Decrementer (LSB)
↓	↓	↓
19D	Decrementer (MSB)	Decrementer (MSB)
19E	—	T/C (See Decrementer) 1 = Timer, 0 = Counter
1A0	MPSI (LSB)	MPSI (LSB)
↓	↓	↓
1AF	MPSI (MSB)	MPSI (MSB)
1B0	FLAG 0	FLAG 0
	↓	↓
1BF	FLAG F	FLAG F
1C0	—	P0 Direction (1 = OUT, 0 = IN)
↓		↓
1DF	—	P31 Direction (1 = OUT, 0 = IN)
1E0	P0 DATA	P0 DATA
↓	↓	↓
1FF	P31 DATA	P31 DATA

Note: CRU addresses not listed above are not usable.

Single-Bit CRU Operations

The TMS9940 performs three single-bit CRU functions: test bit (TB), set bit to one (SBO), and set bit to zero (SBZ). To identify the bit to be operated upon, the TMS9940 develops a CRU-bit address and places it on the address bus.

For the two output operations (SBO and SBZ) the processor also generates a CRUCLK pulse, indicating an output operation to the CRU device, and places bit 7 of the instruction word on the CRUOUT line to accomplish the specified operation (bit 7 is a one for SBO and a zero for SBZ). A test-bit instruction transfers the addressed CRU bit from the CRUIN input line to bit 2 of the status register (EQUAL).

The TMS 9940 develops a hardware base address for the single-bit operations from the software base address contained in workspace register 12 and the signed displacement count contained in bits 8 through 15 of the instruction. The displacement allows two's complement addressing from base minus 128 bits through base plus 127 bits. The hardware base address (bits 6 through 14 of WR12) is added to the signed displacement specified in the instruction, and the result is loaded onto the address bus. *Figure 3* illustrates the development of a single-bit CRU address for the SBO, SBZ, and TB instruction.

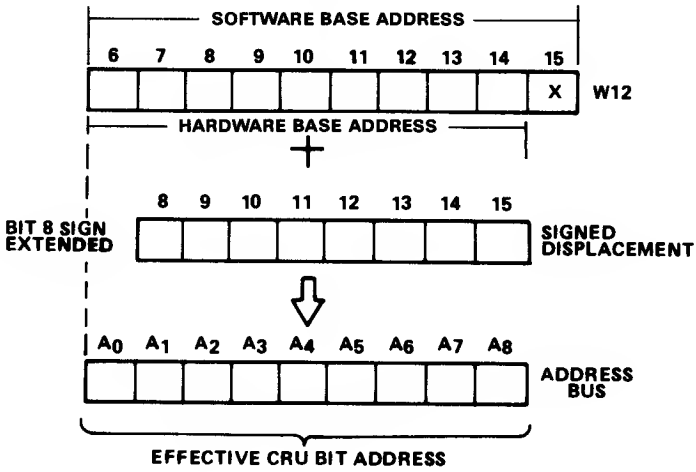


Figure 3. TMS9940 Single-Bit CRU Address Development

Multiple-Bit CRU Operations

The TMS9940 performs two multiple-bit CRU operations: store communications register (STCR) and load communications register (LDCR). Both operations perform a data transfer from the CRU-to-memory or from memory-to-CRU as illustrated in *Figure 4.* Although the figure illustrates a full 16-bit transfer operation, any number of bits from 1 through 16 may be involved. The LDCR instruction fetches a word from memory and right-shifts it to transfer it serially to CRU output bits. If the LDCR involves eight or fewer bits, those bits come from the right-justified field within the addressed byte of the memory word. If the LDCR involves nine or more bits, those bits come from the right-justified field within the whole memory word. When transferred to the CRU interface, each successive bit receives an address that is sequentially greater than the address for the previous bit. This addressing mechanism results in an order reversal of the bits; this is, bit 15 of the memory word (or bit 7) becomes the lowest addressed bit in the CRU, and bit 0 becomes the highest bit in the CRU field.

An STCR instruction transfers data from the CRU to memory. If the operation involves a byte or less transfer, the transferred data will be stored right-justified in the memory byte with leading bits set to zero. if the operation involves from 9 to 16 bits, the transferred data is stored right-justified in the memory word with leading bits set to zero.

When the input from the CRU device is complete, the first bit from the CRU is in the least-significant-bit position in the memory word or byte.

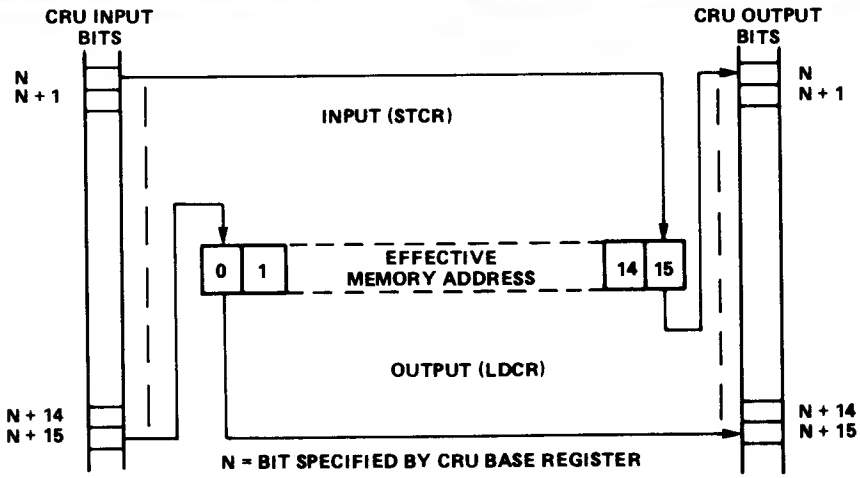
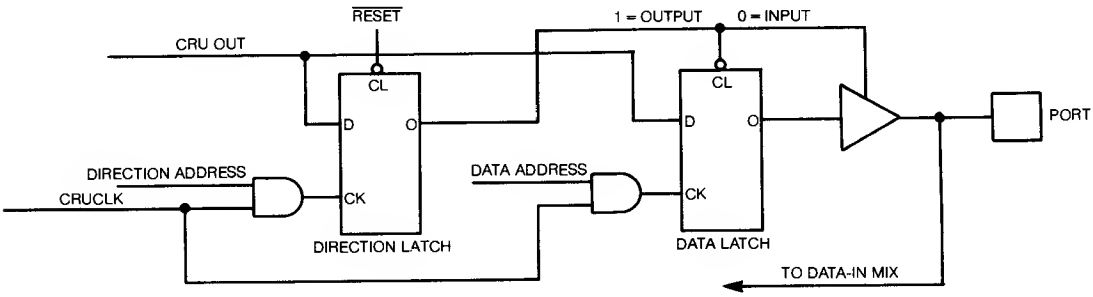


Figure 4. TMS9940 LDCR/STCR Data Transfers

General Purpose I/O

The TMS9940 contains 32 I/O pins which can be used as individually controlled I/O lines with each line independently programmed as an input or output. $\overline{\text{RESET}}$ forces all I/O lines to the input mode until programmed, by an I/O command. Once programmed, the line will stay in the designated state until it is reprogrammed or $\overline{\text{RESET}}$ again becomes active. Reading a line will input data present on the pin without affecting its direction. The lines can be accessed individually by the single bit CRU instructions (SBO, SBZ, TB) or in groups of 1 to 16 by the multiple bit CRU instructions (STCR, LDCR). The I/O data and direction bits are accessed through dedicated bit addresses as shown in *Table 3.*

When an I/O port is programmed to be an input, the previous output data is reset to zero. Thus the data direction and configuration bits should be set first, then the desired output data is set by the appropriate CRU instruction. The equivalent logic for the output control of ports 17 to 31 is shown below.



I/O Expansion

The TMS9940 allows direct I/O expansion for up to 256 bits by use of a standard 9900 family CRU interface. I/O lines P0-P10 can be configured as an 8-bit address bus (A1-A8), CRUIN, CRUOUT, and CRUCLK to interface to any CRU based peripheral. (See the configuration section for details.)

Figure 5 illustrates how to implement a system containing a TMS9901 programmable system interface, a TMS9902 asynchronous communications controller, and a TMS9903 synchronous communications controller.

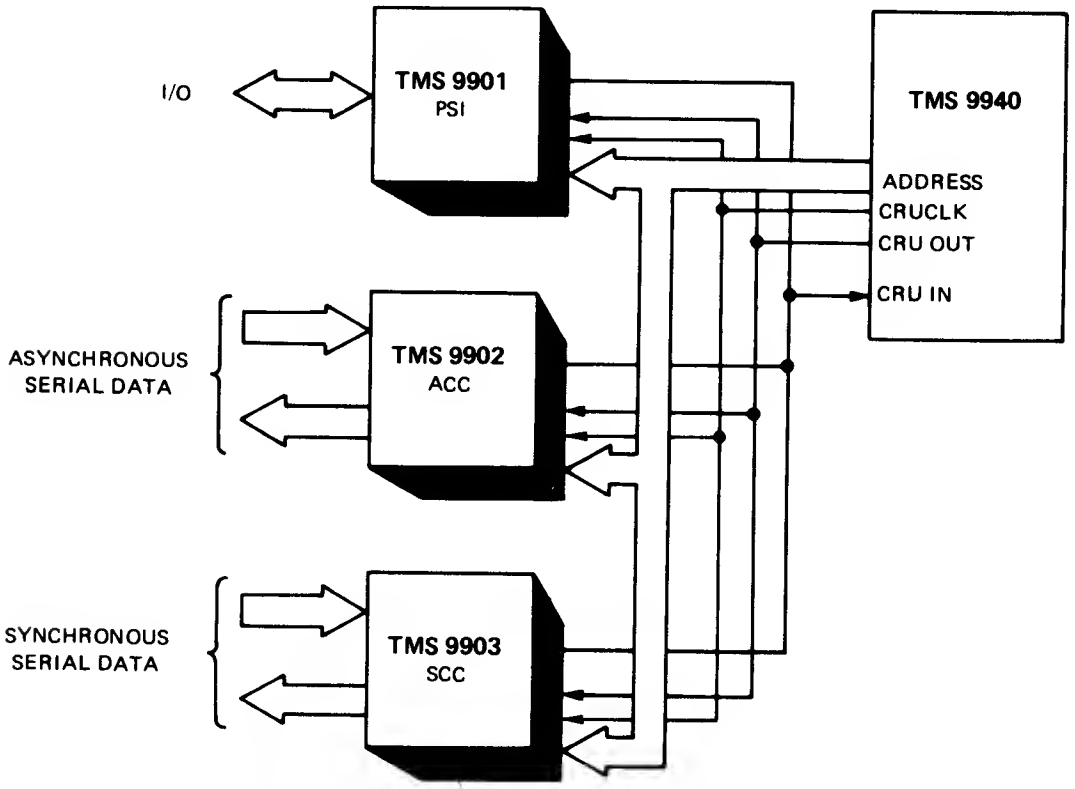
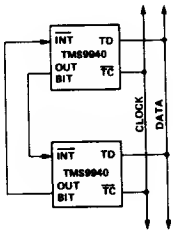


Figure 5. TMS 9940 Input/Output Expansion Interface

8

Multiprocessor System Interface (MPSI)

The MPSI is a two-wire interface for transferring data in a multiple processor system. Since the TMS9940 can execute instructions out of its RAM, the MPSI allows the capability of efficiently downloading instruction sequences which can then be executed. Thus, multiple processor systems can reconfigure themselves in system applications. The MPSI can also be used to transfer data to be operated on, such as in a master-slave situation with the master distributing tasks to the slaves.



For multiple TMS9940 systems the MPSI is connected as shown. Additional CPU's can be connected simply by "wire ORing" to the MPSI signals.

A block diagram of the internal MPSI logic is shown in *Figure 6.*

The protocol of the system is such that all devices are "receivers" except when actually transmitting data (the "sender" mode). The TD input signal feeds a 16-bit shift register that is clocked by the TC input to allow 16 bits of data to be shifted into the shift register completely transparent to the rest of the CPU operation. After the data has been sent, the "sender" interrupts the "receiver" (through a normal interrupt input) so that the "receiver" can execute an STCR instruction to input its MPSI data from its dedicated MPSI CRU bit addresses (see *Table 3*). As needed, the "receiver" can then interrupt the "sender" to acknowledge receipt and/or request new data.

To become a "sender" the TMS9940 executes an LDCR instruction to the dedicated MPSI CRU addresses. Automatically, the TD signal switches to the output mode to send data, and the TC signal sends out the CRUCLK strobe. After completion of the instruction, TD and TC again revert to the input mode to switch the device back to "receiver" status.

The MPSI is compatible with the standard 9900 family CRU interface. An example illustrating the TMS9940 and TMS9900 communicating through the MPSI is shown in *Figure 7.*

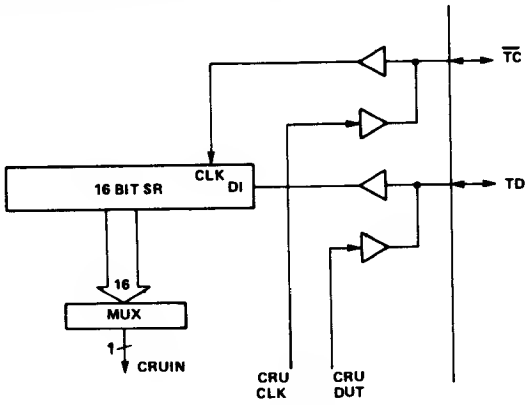
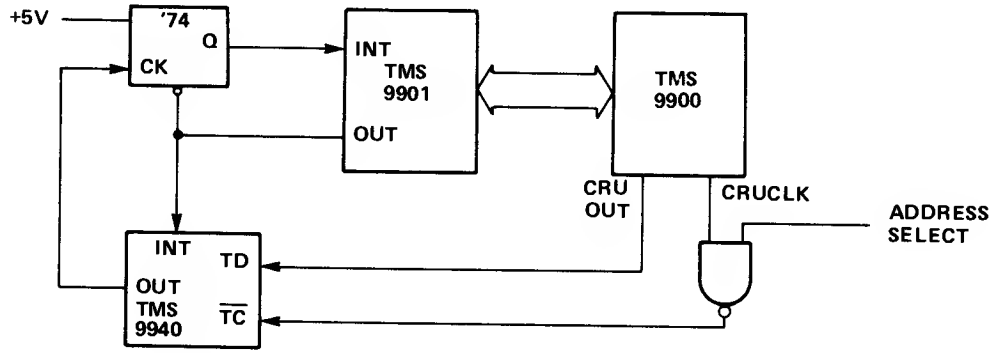
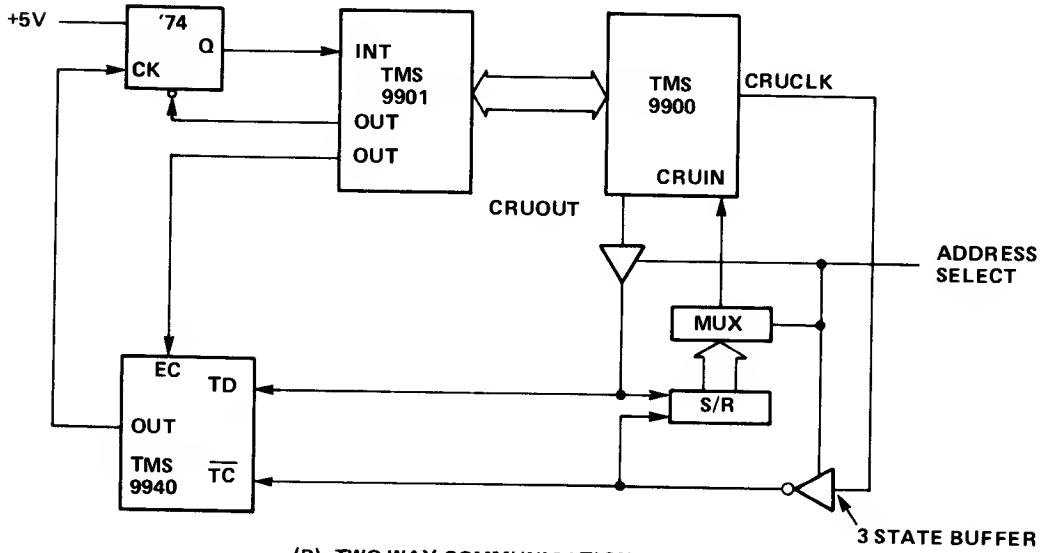


Figure 6. MPSI Block Diagram



(A) ONE-WAY COMMUNICATION; TMS 9900 DOWNLOADS TO TMS 9940

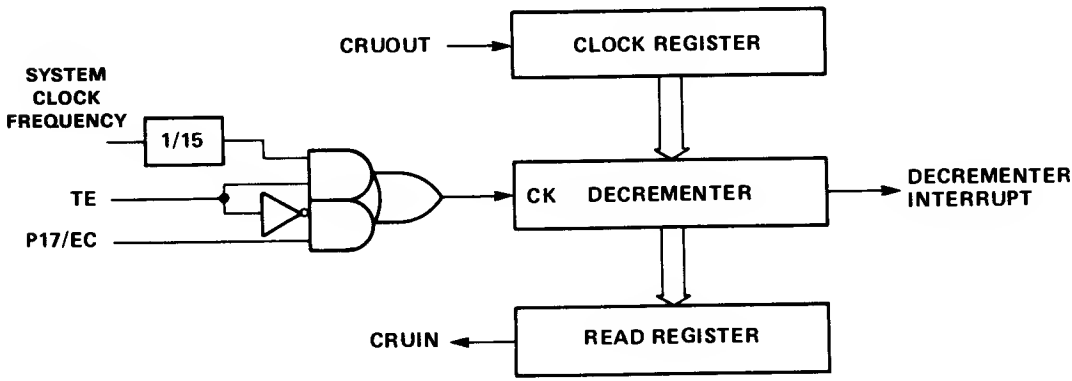


(B) TWO-WAY COMMUNICATION

Figure 7. TMS 9900 and TMS9940 Communication through MPSI

DECREMENTER (TIMER/EVENT COUNTER)

The TMS9940 contains a 14-bit decrementing register which can function as a programmable real-time clock, an event timer, or an external event counter. A block diagram of the timer/counter is shown below.



When $\overline{\text{RESET}}$ is active, a zero value is forced into the clock register to disable the decrementer. Writing a non-zero value into the clock register through the dedicated CRU bit addresses (bits 190_{16} to $19D_{16}$ as shown in *Table 3)* enables the decrementer to start at the programmed value, count down to zero at a rate equal to system oscillator x 1/30, issue an interrupt, and restart at the programmed value. The interrupt is then automatically cleared by the interrupt context switch.

The decrementer is programmed to function as a timer or event counter by a dedicated Timer Enable CRU bit, TE (see *Table 3).* Writing a one (1) into TE will program the decrementer as a timer, and a zero will program the decrementer as an event counter.

When programmed as a timer, the decrementer can function as an interval timer simply by loading the proper start value in CRU bits 190_{16} to $19D_{16}$. The decrementer will then issue interrupts at the chosen interval.

The decrementer can also be used as an event timer when programmed as a timer by reading the timer values through the dedicated CRU bit addresses at the start and stop points of the event of interest and comparing the two values. The difference will be a direct measurement of the elapsed time.

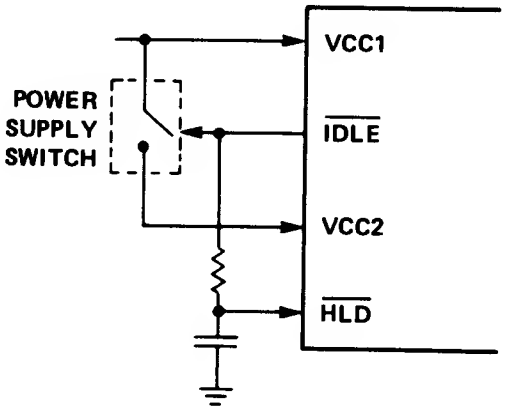
When programmed as an event counter, the decrementer functions as above except that pin P17/EC is the clock input instead of the system clock. A positive edge transition on P17/EC will decrement the count. When the count reaches zero, the decrementer is reloaded with the programmed start value and an interrupt is issued. Note that P17/EC can function as a positive edge-triggered interrupt by loading a start value of one. 8◀

FLAG REGISTER

The TMS9940 incorporates a 16-bit flag register internally. Each of the bits is under program control and can be SET, RESET, and TESTED. The bits are accessed through dedicated CRU bit addresses and utilize the CRU instructions (LDCR, STCR, TB, SBO, SBZ) or control. The CRU bit addresses assignments for the flag register are shown in *Table 3.*

Power Down

Applications which have low duty cycles (for example, those which are human interactive) and/or require low power dissipation, can make use of the power down capability to lower average power. The TMS9940 is powered by two separate power supplies: (1) V_{CC1}, which powers the RAM and interrupt logic, and (2) V_{CC2}, which powers the rest of the circuitry. A diagram showing the way to connect power to use the power down feature is shown below.



In the above circuit when the IDLE instruction is executed, a low value will be output on $\overline{IDLE}$ to open the power supply switch. Inputting an interrupt into the CPU will force the processor out of IDLE and drive $\overline{IDLE}$ HIGH, which will close the switch and power up the rest of the circuitry. The HLD input is a Schmitt Trigger input which will keep the CPU stopped until V_{CC2} has settled. The particular values of R and C chosen are system dependent.

To use the power down feature configuration bit (CB) 3 must be set to a 1 to enable (low) $\overline{HLD}$, $\overline{HLDA}$, and $\overline{IDLE}$. Execution of the IDLE instruction will disable the decrementer interrupt (level 2) and $\overline{INT2}$ (level 3) and the processor can be powered down with the circuit shown in the figure. External lows to either $\overline{RST}$ or $\overline{INT1}$ can be used to force the CPU out of the IDLE state; however, if the processor was powered down, $\overline{INT1}$ must be used to maintain RAM data integrity as the RAM write latches temporarily float during $\overline{RESET}$. If decrementer CB is equal to a zero when IDLE is executed, the decrementer interrupt and $\overline{INT2}$ are not disabled and the CPU cannot be powered down.

Hold and Hold Acknowledge

▶8 Multiple processor operation may require temporary suspension of operation of one microcomputer (e.g., where both microcomputers access common devices through the general purpose I/O lines). This could entail a "master/slave" situation. One microcomputer (master) can place the other (slave) on hold by activating the slaves' $\overline{HLD}$ line. When in the hold state, the slave issues $\overline{HLDA}$. When the master deactivates the slaves' $\overline{HLD}$ line, the slave leaves the hold state. Configuration bit 3 must be a one at the slave so that its $\overline{HLD}$ pin will be active.

Synchronization Mode ($\overline{\phi}$)

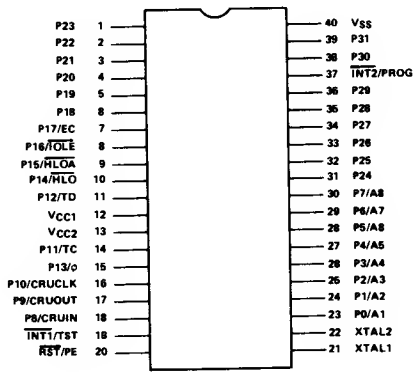
A clock output for use with external hardware is available on terminal 15, P13/$\overline{\phi}$. When configured in the sync mode (see *Table 2*), P13/$\overline{\phi}$ sends out the internal clock that is half of the oscillator frequency.

TMS9940 Terminal Assignments

Table 4 defines the TMS9940 pin assignments and describes the function of each pin.

Table 4. TMS9940 Pin Assignments and Functions

SIGNATURE	PIN	I/O	DESCRIPTION
XTAL1	21	IN	Crystal input pin for control of internal oscilletor.
XTAL2	22	IN	Crystel input pin for control of internal oscillator. Also input pin for external oscillator.
V_{CC1}	12		Supply voltage (+5 V). The internal RAM and interrupt logic are powered by this supply.
V_{CC2}	13		Supply voltege (+5V). All logic except the RAM end interrupt logic are powered by this supply.
V_{SS}	40		Ground reference.
$\overline{RST}$/PE	20	IN	$\overline{RESET}$/Progrem Enable. When ective low (Schmitt Triggar Input, V_{IL}) the $\overline{RESET}$ sequence is initiated. $\overline{RESET}$ must be held ective for a minimum of fiva clock cycles. When active high (V_{IP}) the EPROM programming function is enabled. (See EROM Programming Section for detailed description.)
$\overline{INT1}$/TST	19	IN	$\overline{Interrupt\ 1}$/TEST. When ective low (V_{IL}) external device interrupt 1 is active. When active high (V_{IP}) the device is switched into the test mode (see TEST FUNCTION Section for detailed description).
$\overline{INT2}$/PROG	37	IN	$\overline{Interrupt\ 2}$/PROGRAM PULSE. Whan active low (V_{IL}) and $\overline{RST}$/PE is not ective high, externel device intarrupt 2 is active. When $\overline{RST}$/PE is active high (V_{IP}), $\overline{INT2}$/PROG becomes the programming pulse input for EPROM programming. (See Programming Section for description.)
P0/A1	23	I/O	General Purpose I/O lines. P0-P7 can also be configured as the address bus (A1 is MS8) of the I/O expansion chennel (see Configuration Section for details).
P1/A2	24		
P2/A3	25		
P3/A4	26		
P4/A5	27		
P5/A6	28		
P6/A7	29		
P7/A8	30		



8◂

Table 4. TMS 9940 Pin Assignments and Functions (Continued)

SIGNATURE	PIN	I/O	DESCRIPTION
P8/CRUIN	18	I/O	General Purpose I/O Line. P8 can also be configured as the CRUIN data input signal for the I/O expension channel (see I/O Section for configuration details).
P9/CRUOUT	17	I/O	Generel Purpose I/O Line. P9 can also be configured as the CRUOUT data output signal for the I/O expansion channel (see I/O Section for configuration details).
P10/CRUCLK	16	I/O	General Purpose I/O Line. P10 can elso be configured as the CRUCLK data strobe output signel for the I/O expension channel (see I/O Section for configuration deteils).
P11/$\overline{TC}$	14	I/O	Generel Purpose I/O Line. P11 can also be configured es the transfer clock for the Multiprocessor System Interface (see I/O Section for configuration deteils).
P12/TD	11	I/O	General Purpose I/O Line. P12 can also be configured as the transfer date signal for the Multiprocessor System Interface (see I/O Section for configuration details).
P13/$\overline{\phi}$	15	I/O	General Purpose I/O Line. P13 cen also be configured as a clock output signel (see I/O Section for configuration deteils).
P14/$\overline{HLD}$	10	I/O	General Purpose I/O Line. P14 can elso be configured as the $\overline{HOLD}$ (ective low) Schmitt Trigger input to force the processor to stop until $\overline{HOLD}$ returns to the inactive state. (See I/O Section for configuration details.)
P15/$\overline{HLDA}$	9	I/O	General Purpose I/O Line. P15 can also be configured as the Hold Acknowledge output (ective low). When the processor enters the HOLD state, $\overline{HOLDA}$ becomes active. (See I/O Section for configuretion details.)
P16/$\overline{IDLE}$	8	I/O	Generel Purpose I/O. P16 can also be configured as the $\overline{IDLE}$ output signal (active low) for power down. (See I/O Section for configuration details.)
P17/EC	7	I/O	Generel Purpose I/O Line. P17 can elso be progremmed as the event counter input. The decrementer will decrement on eech positive transition of EC. (See Decrementer Section for programming details.)
P18	6	I/O	General Purpose I/O Line
P19	5	I/O	General Purpose I/O Line
P20	4	I/O	General Purpose I/O Line
P21	3	I/O	General Purpose I/O Line
P22	2	I/O	General Purpose I/O Line
P23	1	I/O	Generel Purpose I/O Line
P24	31	I/O	General Purpose I/O Line

Table 4. TMS 9940 Pin Assignments and Functions (Concluded)

SIGNATURE	PIN	I/O	DESCRIPTION
P25	32	I/O	General Purpose I/O Line
P26	33	I/O	General Purpose I/O Line
P27	34	I/O	General Purpose I/O Line
P28	35	I/O	General Purpose I/O Line
P29	36	I/O	General Purpose I/O Line
P30	38	I/O	General Purpose I/O Line
P31	39	I/O	General Purpose I/O Line

TMS9940 INSTRUCTION SET

Definition

Each instruction of the TMS9940 set performs one of the following operations:

- Arithmetic, logical, comparison, or manipulation operations on data;
- Loading or storage of machine registers (program counter, workspace pointer, or status);
- Data transfer between memory and external devices via the CRU;
- Control functions.

This instruction set is identical to that of the TMS9900 with the following exceptions:

- Instructions added (as dedicated XOP's 0 to 3);
  —DCA —LIIM —DCS
- Instructions deleted
  —RSET —CKOF
  —CKON —LREX

A complete listing of the instructions and addressing modes is found in a later section.

TMS9940 Instruction Execution Times

Instruction execution times for the TMS9940 are a function of:

1. Clock cycle time, $t_c(\phi) \equiv 2 \cdot t_{cy}$ where $t_{cy} \equiv 1/\text{Oscillator Frequency } (f_{osc})$
2. Addressing mode used where operands have multiple addressing mode capability.

Table 5 lists the number of clock cycles required to execute each TMS9940 instruction. For instructions with multiple addressing modes for either or both operands, the table lists the number of clock cycles with all operands addressed in the workspace register mode. To determine the additional number of clock cycles required for modified addressing, add the appropriate values from Table A. The total instruction execution time for an instruction is

$$T = t_c(\phi) \cdot C$$

where,

T = total instruction time
$t_c(\phi)$ = clock cycle time
C = number of clock cycles required for instruction execution plus address modification.

8◀

Table 5. TMS 9940 Instruction Execution Times

INSTRUCTION	CLOCK CYCLES	ADDRESS MODIFICATION
A	10	See Table A
AB	7	See Table A
ABS (MSB = 0)	12	See Table A
(MSB = 1)	12	See Table A
AI	12	
ANDI	12	
B	8	See Table A
BL	10	See Table A
BLWP	20	See Table A
C	10	See Table A
CB	7	See Table A
CI	12	
CLR	8	See Table A
COC	10	See Table A
CZC	10	See Table A
DCA	7	See Table A
DCS	7	See Table A
DEC	8	See Table A
DECT	8	See Table A
DIV (ST 4 is SET)	14	See Table A
DIV (ST 4 is RESET)*	128	See Table A
IDLE	10	
INC	8	See Table A
INCT	8	See Table A
INV	8	See Table A
JUMP	6	
LDCR (C = 0)	42	See Table A
(1≤C≤8)	8 + 2C	See Table A
(9≤C≤15)	10 + 2C	See Table A
LI	12	
LIIM	10	

▶8

Table 5. TMS 9940 Instruction Execution Times (Continued)

INSTRUCTION	CLOCK CYCLES	ADDRESS MODIFICATION
LIMI	14	
LWPI	12	
MOV	8	See Table A
MOVB	6	See Table A
MPY	82	See Table A
NEG	10	See Table A
ORI	12	
RTWP	14	See Table A
S	10	See Table A
SB	7	See Table A
SBO	10	
SBZ	10	
SETO	8	See Table A
SHIFT (C = 0)	12 + 2N	
(C = 0, BITS 12-15 of WR0 = 0)	46	
(C = 0, BITS 12-15 of WR0 = N ≠ 0)	14 + 2N	
SOC	10	See Table A
SOCB	7	See Table A
STRC (C = 0)	46	See Table A
(1 ≤ C ≤ 8)	29	See Table A
(9 ≤ C ≤ 15)	46	See Table
STST	8	
STWP	8	
SWPB	8	See Table A
SZC	10	See Table A
SZCB	7	See Table A
TB	10	
X**	6	See Table A
XOP	26	See Table A
XOR	10	See Table A
Reset Function	16	
Interrupt Context Switch	16	

*Execution time is dependent on the partial quotient after each clock cycle during execution.
**Execution time is added to the execution time of the instruction located at the source address.

8

TABLE A
ADDRESS MODIFICATION

ADDRESSING MODE	CLOCK CYCLES (C)
WR (T_s or T_d = 00)	0
WR indirect (T_s or T_d = 01)	2
WR indirect auto increment (T_s or T_d = 11)	4
Symbolic (T_s or T_d =10, S or D = 0)	6
Indexed (T_s or T_d =10, S or D ≠ 0)	8

TMS9940E EPROM PROGRAMMING

Erasure

Before programming, the TMS9940E is erased by exposing the chip through the transparent lid to high intensity ultraviolet light (wavelength: 2537 angstroms). The recommended exposure is 10 watt-seconds per square centimeter. This can be obtained by, for instance, 20 to 30 minutes exposure of a filterless Model S52 shortwave UV lamp about 2.5 centimeters above the EPROM. After exposure all bits are in the "0" state.

Programming

The TMS9940E should be initialized by RESET before the programming sequence begins. The EPROM consists of 16K bits of program memory organized as 2K bytes (8 bits) located at (starting) address 0000_{16}. Data is transferred into the CPU for programming through P24(MSB)—P31 (LSB). Taking the PE signal active high (V_{IP}) initializes the internal address pointer of 0000_{16} and inputs the first byte of data (see *Figure 8*). After a minimum delay of 40 clock cycles, PROG can be applied (V_{IP}, 50 ms) and the data present on P24-P31 updated to the next byte. Tha falling edge of PROG inputs the new byte of data to the next location and after a minimum delay of 25 clock cycles the PROG pulse can be applied again. This sequence is continued until the entire 2K bytes have been programmed. Note that the memory is programmed in sequence starting at 0000_{16}, and the input data must be valid at the rising edge of PE or falling edge of PROG.

Programming/Test Function Electrical Characteristics

▶8

PARAMETER		MIN	NOM	MAX	UNIT
t_r	TST, PE, PROG input rise time		100		ns
t_f	TST, PE, PROG input fall time		100		ns
t_{su}	Input data setup time to rising edge of PE, TST or to falling edge of PROG		0		ns
t_h	Input data hold time past rising edge of PE, TST		80 $t_{c(\phi)}$		ns
t_h(P-da)	Input data hold time past falling edge of PROG		50 $t_{c(\phi)}$		ns
t_h(P-PE,T)	PE, TST input hold time past falling edge of PROG		0		ns
t_{su}(P-PE,T)	PROG input setup time to rising edge of PE, TST		0		ns
t_h(T-PL)	PROG input pulse low past rising edge of TST, PE		80 $t_{c(\phi)}$		ns
t_w(PL)	PROG input pulse width low		50 $t_{c(\phi)}$		ns
t_w(PHP)	PROG input pulse width high in the programing mode		50		ms
t_w(PHT)	PROG input pulse width high in the test mode		4 $t_{c(\phi)}$		ns

NOTE: Timing diagrams in Figure 8.

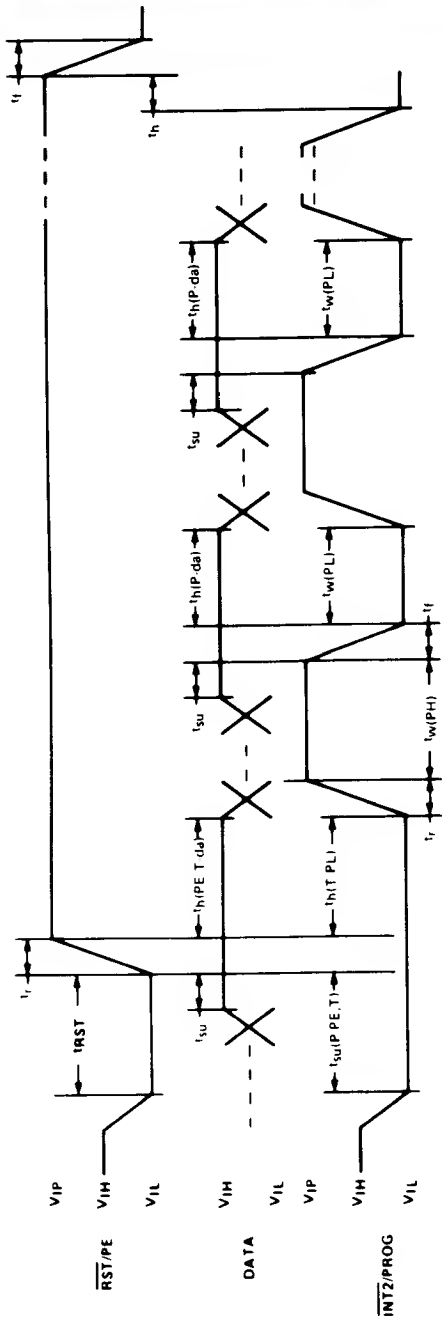


Figure 8. EPROM Programming Timing Diagram

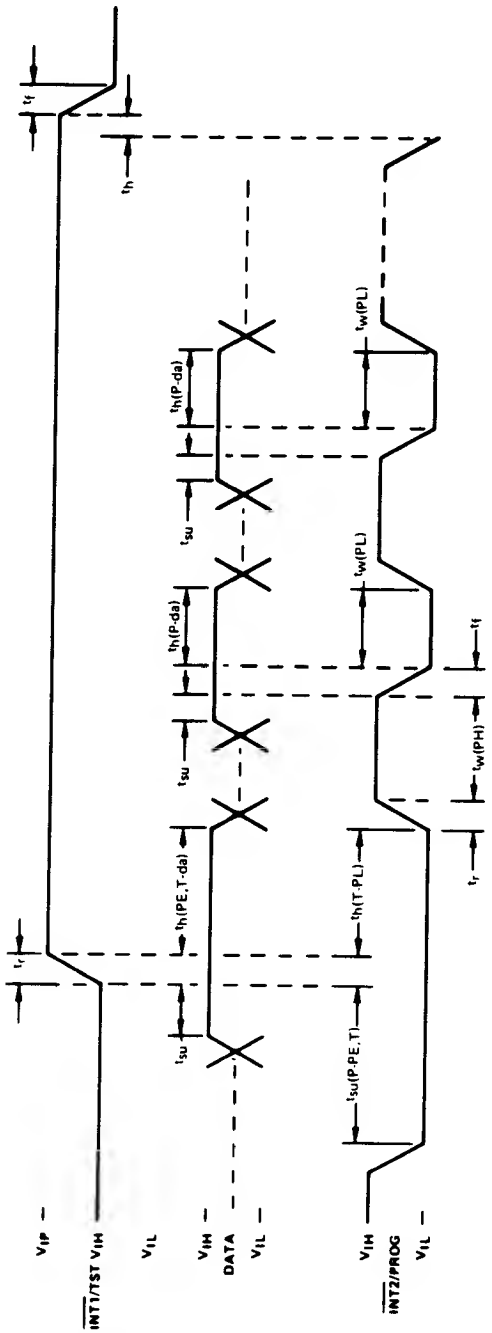


Figure 9. Test Function Timing Diagram

TEST FUNCTION

This test function allows loading a program into the RAM area of the TMS9940 through pins P24 through P31. This program can then be executed, and the results of this execution used to verify operation of the TMS9940. The program could include error messages as well as a successful completion message sent to a peripheral device accessed through the CRU.

The processor should be initialized by $\overline{\text{RESET}}$ before any test sequence begins. Data is directly loaded in sequence into the RAM through P24 (MSB)—P31 (LSB). Taking the TEST signal active high (V_{IP}) initializes the internal address pointer to 8300_{16} (starting address of RAM) and inputs the first byte of data (see *Figure 9*). After a minimum delay of 40 clock cycles PROG can be applied (V_{IH}, 4 clock cycles minimum) and the data present on P24—P31 updated to the next byte. The falling edge of PROG inputs the new byte of data to the next location and, after a minimum delay of 25 clock cycles, PROG can be applied again. This sequence is continued until the desired data has been loaded into the RAM. Taking TEST inactive will then jump the processor to the address specified by the last 16 bits loaded. Note that the RAM is loaded in sequence starting at 8300_{16}, and the input data must be valid at the rising edge of TST or on the falling edge of PROG.

TMS9940 ELECTRICAL SPECIFICATIONS

Absolute Maximum Ratings Over Operating Free-Air Temperature Range (Unless Otherwise Noted)*

Supply Voltage, V_{CC1}†	−0.3 to 20 V
Supply Voltage, V_{CC2}	−0.3 to 20 V
Programming Voltage, PE	−0.3 to 35 V
All Input Voltages	−0.3 to 20 V
Output Voltage	−2 to 7 V
Continuous Power Dissipation	1.5 watt
Operating Free-Air Temperature Range	0°C to 70°C
Storage Temperature Range	−55°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

†All voltage values are with respect to V_{SS}.

Recommended Operating Conditions

PARAMETER	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC1}		5		V
Supply voltage, V_{CC2}		5		V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH}	2.0			V
Low-level input voltage, V_{IL}			0.8	V
Program/test input voltage, V_{IP}		26		V
Operating free-air temperature, T_A	0		+70	°C

8◂

Electrical Characteristics

PARAMETER	TEST CONDITIONS	MIN	NOM	MAX	UNITS
I_I input current, all inputs	$V_I = V_{SS}$ to V_{CC}		±10		μA
V_{OH}, high-level output voltage, all outputs	$I_O = -0.4$ mA		2.4		V
V_{OL}, low-level output voltage, all outputs	$I_O = 2$ mA		0.4		V
I_{CC1}, supply current from V_{CC1}			10		mA
I_{CC2}, supply current from V_{CC2}			150		mA
C_I, input capacitance, all inputs	f = 1 MHz unmeasured pins at V_{SS}		15		pF
C_O, output capacitance, all outputs	f = 1 MHz unmeasured pins at V_{SS}		15		pF

Clock Characteristics

The TMS 9940 has an internal oscillator and a two-phase clock generator controlled by an external or crystal. The user may also disable the oscillator and directly inject a frequency source into the XTAL2 input. The crystal frequency and the external frequency source must be double the desired system CLOCK frequency.

Internal Oscillator

The internal oscillator is enabled by connecting a crystal across XTAL 1 and XTAL 2. The system CLOCK frequency $1/t_{c(\phi)}$, is one-half the crystal oscillator frequency, f_{osc}.

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNITS
f_{osc}, TMS 9940E, TMS 9940M		.5	5.0	5.12	MHz
f_{osc}, TMS 9940E-40, TMS 9940M-40		.5	4.0	4.10	MHz
f_{osc}, TMS 9940E-30, TMS 9940M-30	$0°C \leq T \leq +70°C$	.5	3.0	3.07	MHz
f_{osc}, TMS 9940E-20, TMS 9940M-20		.5	2.0	2.05	MHz
f_{osc}, TMS 9940E-10, TMS 9940M-10		.5	1.0	1.02	MHz

Note: $t_{cy} \equiv 1/f_{osc}$

$t_{c(\phi)} \equiv 2 \cdot t_{cy}$

▸8

TMS 9940 DESIGN SUPPORT

Program Support Tools

Table 6 defines four of the major program development methods available for the TMS 9940 microcomputer. Each program development product supports assembly language programming, options 1 and 2 support hardware emulation for target system evaluation.

Table 6. TMS 9940 Program Support

OPTION	SYSTEM	HOST COMPUTER	ASSEMBLER	EMULATION	SUPPLIER
1	TM 990/40DS Low Cost Standalone Development System	16-Bit TI Microcomputer TM 990/40DS User Supplied Terminal (TTY, TI 733, or TI 745)	Yes, Line-By-Line (Non Symbolic)	Yes, Non-Real Time	Semiconductor Group -TI-
2	AMPL System (Advanced Microprocessor Prototyping Laboratory)	16-Bit TI Minicomputer, FS 990/4 (Includes Terminal)	Yes, TXMIRA, Full Assembly	Yes, Real-Time	Digital Systems Group -TI-
3	TMSWIOIT Transportable-FORTRAN Source, Cross-Support Software Package	User-Supplied 16-Bit Minicomputer or larger (eg. 32-Bit Mainframe) User Supplied I/O	Yes, Full Assembly	No	Semiconductor Group -TI-
4	Timeshare	Timeshare 32-Bit Mainframe User Supplied I/O	Yes, Macro Assembly And Full Assembly	No	G.E. N.C.S.S. TYMSHARE

Options 3 and 4 support compatible computer simulation based on the TMS 9900 microprocessor; thus, functional instruction simulation without TMS 9400 timing or I/O data is possible.

To determine the most cost-effective tool, the second column relates to the computer equipment required. Timeshare has a repeating cost to consider, whereas, the remainder are one-time investments.

To assemble short program modules via option 1, the Line-By-Line Assembler is the fastest method available. It interactively provides mnemonic-to-object assembly, excluding symbolic addressing references. Standalone program debug is then performed through the terminal keyboard.

The bulk of the TMS 9940 instruction set is identical to the TMS 9900 assembly language. Available cross assemblers are:

- PX9 ASM on the CS 990/4 cassette development system
- TXMIRA on the FS 990/4 floppy disk development system
- SYSMAC on the DS 990/10
- TMS 9900 cross-assembler (available on several timesharing networks).

Three instructions on the TMS 9940 are not found on TMS 9900 assemblers: DCA, DCS, and LIIM. However, these mnemonics are made acceptable by the 'DXOP' assembler directive. The 'DXOP' assembler directive is available on all of the above mentioned assemblers. The DXOP function is to define a label for a specific XOP value. The directive should appear at the beginning of the source life. The following listing shows how the DCA, DCS, and LIIM mnemonics are defined using the DXOP directive, and a short sample program using the three instructions. Note that the DXOP directives are used prior to using the instructions.

```
SAMPLE          TXMIRA      936227**                    PAGE 0001
9940 SAMPLE PROGRAM

0002                        DXOP   DCA,0        DEFINE XOP 0 AS DCA
0003                        DXOP   DCS,1        DEFINE XOP 1 AS DCS.
0004                        DXOP   LIIM,2       DEFINE XOP 2 AS LIIM
0005                        IDT    'SAMPLE'
0006 0000 A081  START       AB     R1, R2       ADD REGISTERS 1 AND 2 TOGETHER
0007 0002 2C02              DCA    2            CORRECT THE RESULT FOR BCD.
0008 0004 1002              JMP    QUIT         GO TO CLEAN UP.
0009 0006 6081  STARTS      SB     R1, R2       SUBTRACT REGISTER 1 FROM 2.
0010 0008 2C42              DC8    R2           CORRECT THE RESULT FOR BCD.
0011 000A 2C82  QUIT        LIIM   R2           LOAD NEW INTERRUPT MASK.
0012 000C 0380              RTWP   GO HOME
0013                        END    START

0000 ERRORS
```

Factory Programming — TMS 9940M

Produced from any of the program support tools, a TI standard TMS 9900 family object format is accepted for factory programming. The absolute object form with a custom MPXXXX number in the program identifier field are acceptable. The object file can be sent and subsequently verified through a timeshare transmission or TI 733 — compatible digital cassettes (punched cards, paper tape, FS 990 floppy discs are also accepted) when developed on the TM 990/40DS or non-TI designed support tools, a user can send a master TMS 9940E device containing the code to be produced in volume.

User Programming — TMS 9940E

The TM 990/40DS low-cost development system can program, verify, and download TMS 9940E devices. A TI designed test program or user defined programs (modifying the TIBUGII resident monitor) also provide functional testing on the development system. Refer to the Test Function section for a detailed description.

A programmer module is an accessory to FS 990 minicomputers. The program, verify and download functions work together with the sophisticated AMPL package in FS 990/4 systems.

Custom Applications

Through a staff of experienced application programmers and microprocessor specialists, Texas Instruments will, upon request, assist customers in evaluating applications, in training designers to program the TMS 9940, and in simulating programs. TI will also contract to write programs to customer's specifications.

TMS 9985

INTRODUCTION

Description

The TMS 9985 is a software compatible member of TI's 9900 family of microprocessors and microcomputers and contains a 16-bit CPU, 256 bytes of RAM, on chip timer/event counter, external 16-bit address bus and 8-bit data bus, and is packaged in a 40-pin package. The instruction set of the TMS 9985 includes the capabilities offered by full minicomputers and is exactly the same as the TMS 9940 microcomputer's. The unique memory-to-memory architecture features multiple register files, resident in memory, which allows faster response to interrupts and increased programming flexibility. The separate bus structure (see *Figure 8-46*) simplifies the system design effort. All members of the TMS 9900 family of peripheral circuits are compatible with the TMS 9985. The TMS 9985 is fully supported by software and hardware development systems.

Key Features Different from the TMS 9900

- 5-MHz Speed
- 8-Bit Memory Data Bus
- 5 Prioritized Interrupts
- 40-Pin Package
- On Chip Timer/Event Counter
- 256 Bits of RAM on Chip
- Separate Memory, I/O and Interrupt Bus Structures
- On Chip Programmable Flags (16)
- Multiprocessor System Interface
- Single 5-Volt Supply
- Speed Selected Versions

Differences Between the TMS 9985 and the TMS 9940

The TMS 9985 is so similar to the TMS 9940 that only the differences are described here.

Key Features Different from the TMS 9940

- 5-MHz Speed
- Up to 65,536 Bytes of Memory
- 256 Bytes of RAM On Chip
- 8-Bit Memory Data Bus
- Separate Memory, I/O and Interrupt Bus Structures
- 5 Prioritized Interrupts

ARCHITECTURE

Registers and Memory

See the TMS 9940.

Interrupts

The TMS 9985 implements five hardware interrupt level. Interrupt level data is shown in *Table 1.*

Table 1. *Interrupt Level Data*

▶8

INTERRUPT LEVEL	VECTOR LOCATION (MEMORY ADDRESS IN HEX)	DEVICE ASSIGNMENT	INT. MASK VALUE TO ENABLE RESPECTIVE INTERNAL STATUS REGISTERS 14 AND 15
(Highest Priority) 0	0000	RESET	0 THROUGH 3*
1	FFFC	LOAD	0 THROUGH 3*
2	0004	EXTERNAL DEV	1 THROUGH 3
3	0008	DECREMENTER	2 AND 3
4	000C	EXTERNAL DEV	3 ONLY

* $\overline{\text{RESET}}$ AND $\overline{\text{LOAD}}$ CAN NOT BE DISABLED

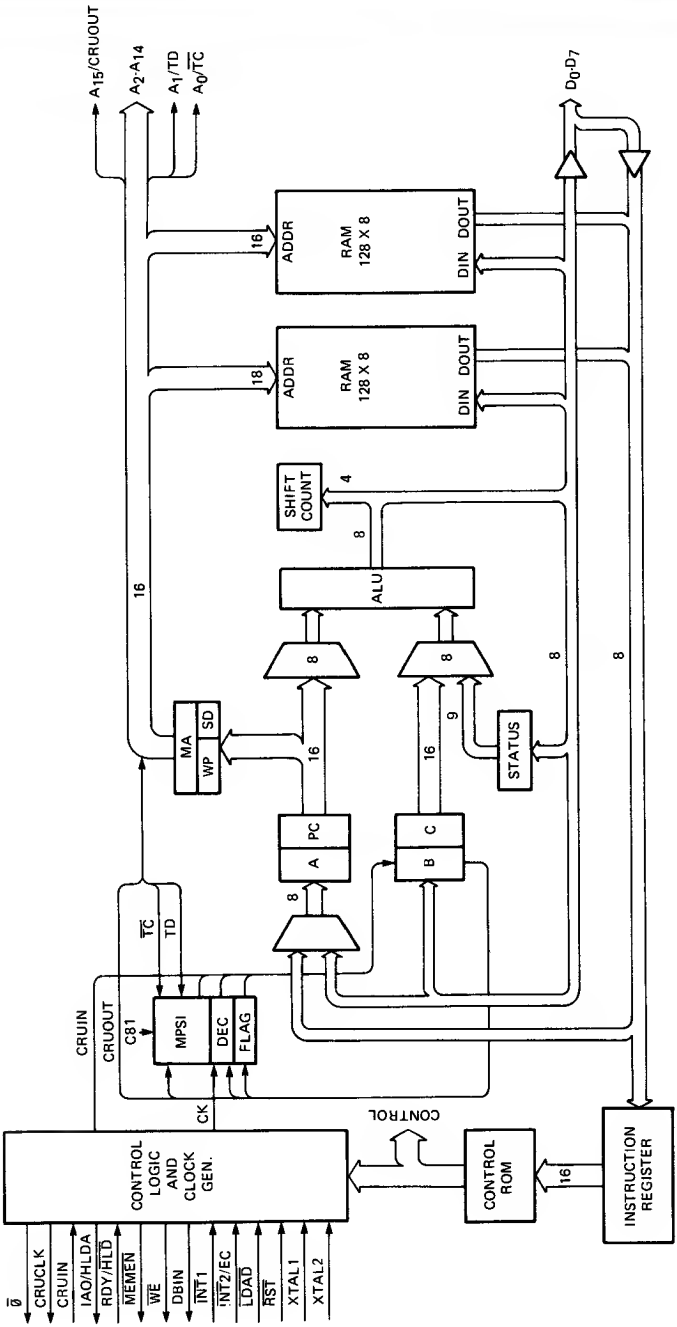


Figure 1. TMS 9985 Architecture

INPUT/OUTPUT

The TMS 9985 supports four types of I/O channels:

1. Communications Register Unit (CRU)
2. Memory Mapped (MM)
3. Direct Memory Access (DMA)
4. The Multiprocessor System Interface (MPSI)

The CRU and MPSI are much the same as those in the TMS 9940. See the TMS 9940 for a discussion of the decrementer and flag register.

Memory Mapped I/O Channel

Memory Mapped I/O is a byte oriented I/O interface through which input or output bytes can be directly addressed. The interface is defined to exist in memory address space and is accessed as if it were a memory location. All processor instructions that access memory can be used to drive the Memory Mapped interface, and thus, arithmetic and logical operations can be performed directly on MM I/O. *Figure 1* illustrates how to implement a I byte input and I byte output MM register.

Direct Memory Access (DMA)

Direct Memory Access (DMA) is a block oriented I/O interface through which blocks of data can be moved into and out of the system memory under external control. The external controller applies $\overline{\text{HOLD}}$ to initiate a DMA request. $\overline{\text{HOLD}}$ is sampled during nonmemory cycles and, when detected, forces the TMS 9985 to enter a hold state. The processor places the address and data buses into the high impedance state and responds with a hold acknowledge signal (HOLDA). When $\overline{\text{HOLD}}$ is removed the TMS 9985 will then return to normal operation. *Figure 2* shows how to implement a DMA system using the TMS 9911 DMA controller. The maximum latency time between a $\overline{\text{HOLD}}$ request and a HOLDA response is equal to 37 clock cycles. The DMA channel cannot be used to move data into or out of the internal RAM.

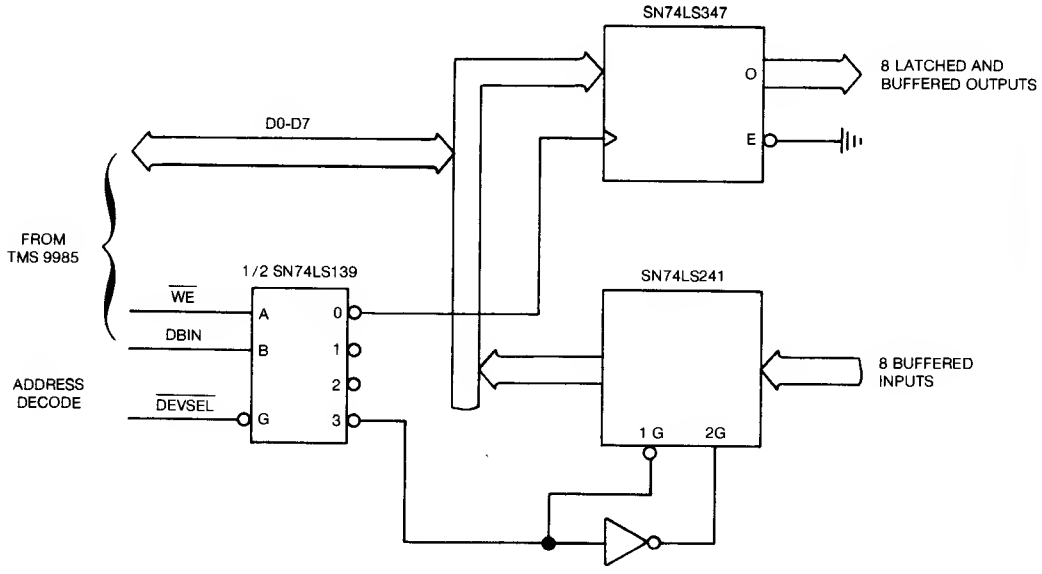


Figure 2. 8-Bit Memory mapped I/O Interface

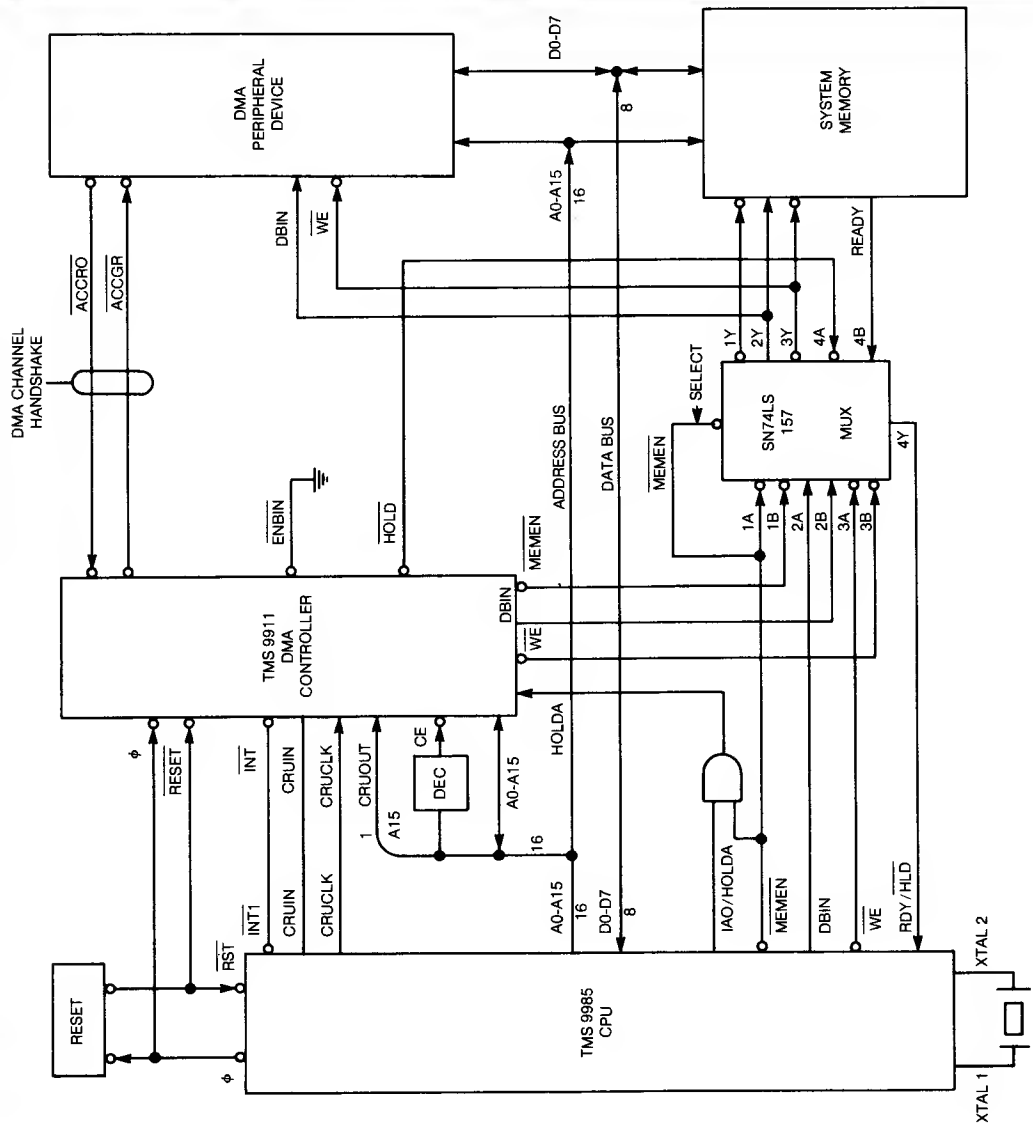


Figure 3. TMS 9985 DMA Interface Using the TMS 9911 DMA Controller

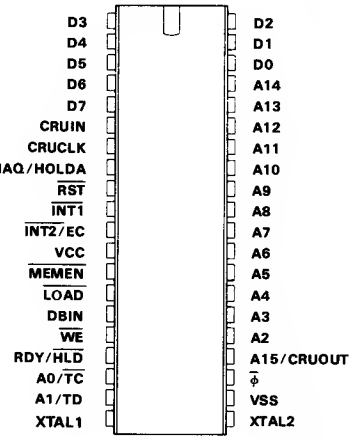
System Configuration

The TMS 9985 allows the user to configure part of the pins as special functions for system applications. The configurable pins are shown below.

Pin Name	*Configuration Bit*	*Configuration*	
		Output	Input
A0/$\overline{TC}$	1	A0	$\overline{TC}$
A1/TD	1	A1	TD

TMS 9985 Pin Description

SIGNATURE	I/O	DESCRIPTION
XTAL 1	IN	Crystal input pin for control of internel oscilletor.
XTAL 2	IN	Crystal input pin for control of internal oscillator. Also input pin for externel oscillator.
VCC		Supply voltege (+5 volts)
VSS		Ground reference
$\overline{\phi}$	OUT	Clock output signel. The frequency of $\overline{\phi}$ is ½ of the oscillator input frequency.
$\overline{RST}$	IN	$\overline{RESET}$. When active (LOW), (Schmitt Trigger input) the RESET sequence is initiated. RESET must be held active for a minimum of five clock cycles.
$\overline{INT1}$	IN	$\overline{INTERRUPT}$ 1. When active (LOW), external device interrupt 1 is active.
$\overline{INT2}$/EC	IN	$\overline{INTERRUPT}$ 2/EVENT COUNTER. When active (LOW), external device interrupt 2 is ective. When the decrementer is progremmed as en event counter (T/$\overline{C}$=0), a positive transition on $\overline{INT2}$/EC will decrement the count.
$\overline{LOAD}$	IN	$\overline{LOAD}$. when active (LOW), $\overline{LOAD}$ causes the TMS 9985 to execute a non maskable interrupt with memory address $FFFC_{16}$ conteining the trap vector (WP and PC). The loed sequence begins after the instruction being executed is completed. $\overline{LOAD}$ will also terminete an idle state. If $\overline{LOAD}$ is active during the time $\overline{RESET}$ is releesed, then the LOAD trap will occur efter the $\overline{RESET}$ function is completed. LOAD should remain active for one instruction period. IAQ can be used to determine instruction boundaries.
A0/$\overline{TC}$	I/O	ADDRESS BIT 0/$\overline{TRANSFER\ CLOCK}$ when configured as address, A0 is the MSB of the 16 bit memory address bus and the 15 bit CRU address bus. When configured for the MPSI, $\overline{TC}$ is the trensfer clock Line (See I/O Section for configuration details).
A1/TD	I/O	ADDRESS 8IT 1/TRANSFER DATA. When configured as eddress, A1 is the 2nd most significant bit of the 16 bit memory address bus and the 15 bit CRU address bus. When configured for the MPSI, TD is the transfer date line (See I/O Section for configuretion details).



SIGNATURE	I/O	DESCRIPTION
A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14	OUT	ADDRESS BUS. A2 is the 3rd most significant bit of the 16 bit memory address bus and the 15 bit CRU address bus. A14 is the 2nd least significant bit of the 16 bit memory address bus and the LSB of the 15 bit CRU address bus. The address bus assumes the high impedance state when holda is active.
A15/ CRUOUT	OUT	ADDRESS BIT 15/CRU OUTPUT DATA. A15/CRUOUT is the LSB of the 16 bit memory address bus and the output data line for CRU output instructions. A15/CRUOUT assumes the high impedance state when HOLDA is active.
D0 D1 D2 D3 D4 D5 D6 D7	I/O	DATA BUS. D0 is the MSB of the B bit data bus, and D7 is the LSB. This bus transfers data to (when writing) and from (when reading) the external memory system when $\overline{\text{MEMEN}}$ is active. The data bus assumes the high impedance state when HOLDA is active.
DBIN	OUT	DATA 8US IN. When active (HIGH), D8IN indicates that the TMS 9985 has disabled its output buffers to allow the external memory to place memory-read data on the data bus during $\overline{\text{MEMEN}}$. DBIN remains low in all other cases.
$\overline{\text{MEMEN}}$	OUT	MEMORY ENABLE. When active (LOW), $\overline{\text{MEMEN}}$ indicates that the address bus contains an external memory address, the READY/$\overline{\text{HOLD}}$ input is sampling READY, and the IAQ/HOLDA output is outputting IAQ.
$\overline{\text{WE}}$	OUT	WRITE ENABLE. When active (LOW), $\overline{\text{WE}}$ indicates that memory write data is available from the TMS 9985 to be written into external memory during $\overline{\text{MEMEN}}$. $\overline{\text{WE}}$ remains high in all other cases.
CRUIN	IN	CRU DATA IN. CRUIN is the input data line for CRU input instructions and is sampled during the TB instruction.
CRUCLK	OUT	CRU CLOCK. When active (HIGH), CRUCLK indicates that the external interface logic should sample the output data on CRUOUT.
RDY/$\overline{\text{HLD}}$	IN	READY/$\overline{\text{HOLD}}$. When $\overline{\text{MEMEN}}$ is active the RDY/$\overline{\text{HLD}}$ input sample READY; when $\overline{\text{MEMEN}}$ is inactive, $\overline{\text{HOLD}}$ is sempled. When active (HIGH), READY indicates that externel memory will be ready to read or write with no additional wait states. When not-ready is indicated during an external memory operation the TMS 9985 enters a wait state and suspends internal operation until the memory systems indicate ready. When active (LOW), $\overline{\text{HOLD}}$ indicates to the processor that an external controller desires to utilize the address and data buses to transfer data to or from external memory. Following a hold signal, the TMS 9985 enters a hold state at the next memory cycle. The processor places the address end data buses in the high impedance state and responds with a hold acknowledge signal (HOLDA). When $\overline{\text{HOLD}}$ is removed, the TMS 9985 raturns to normal operation.
IAQ/HOLDA	OUT	INSTRUCTION ACQUISITION/HOLD ACKNOWLEDGE. When $\overline{\text{MEMEN}}$ is ective the IAQ/HLDA line outputs IAQ; when $\overline{\text{MEMEN}}$ is inactive HOLDA is output. IAQ is active (HIGH), during any memory cycle when the TMS 9985 is acquiring an instruction. HOLDA is active (HIGH) when the processor is in the hold state and the address and data buses are in the high impedance state.

TIMING

Memory

Basic memory read and write cycles are shown in *Figures 4* and *5*. *Figure 4* shows read and write cycles with no wait states while *Figure 5* shows read and write cycles for a memory requiring one or two wait states.

$\overline{\text{MEMEN}}$ goes active (LOW) during each memory. At the same time that $\overline{\text{MEMEN}}$ is active, the memory address appears on the address bus (A0 through A15). If the cycle is a memory read cycle, DBIN will be active (HIGH) and the data present on the data bus (D0 through D7) will be input into the processor. If the cycle is a memory write cycle, $\overline{\text{WE}}$ will go active (LOW) and data will be input by the CPU onto the data bus. At the end of the cycle $\overline{\text{MEMEN}}$ and DBIN or $\overline{\text{WE}}$ will go inactive.

Hold

Other interfaces may utilize the TMS 9985 memory bus by using the hold operation (illustrated in *Figure 6*). The external $\overline{\text{HOLD}}$ input is sampled during nonmemory cycles and when active (LOW), forces the TMS 9985 to enter the hold state. The processor places the address and data buses into the high impedance state to allow other devices to use the memory buses, and outputs the hold acknowledge signal (HOLDA, active HIGH). When $\overline{\text{HOLD}}$ goes inactive, the TMS 9985 resumes processing as shown. The maximum latency time between a $\overline{\text{HOLD}}$ request and a HOLDA response is equal to 37 clock cycles.

CRU

CRU interface timing is shown in *Figure 7*. The timing for transferring two bits out and one bit in is shown. These transfers would occur during the execution of a CRU instruction. The other cycles of the instruction execution are not illustrated. To output a CRU bit, the CRU-bit address is placed on the address bus A0 through A14 and the actual bit data on A15/CRUOUT. During the second clock cycle a CRU pulse is supplied by CRUCLK. This process is repeated until the number of bits specified by the instruction are completed.

The CRU input operation is similar in that the bit address appears on A0 through A14.

During the subsequent cycle, the TMS 9985 accepts the bit input data as shown. No CRUCLK pulses occur during a CRU input operation.

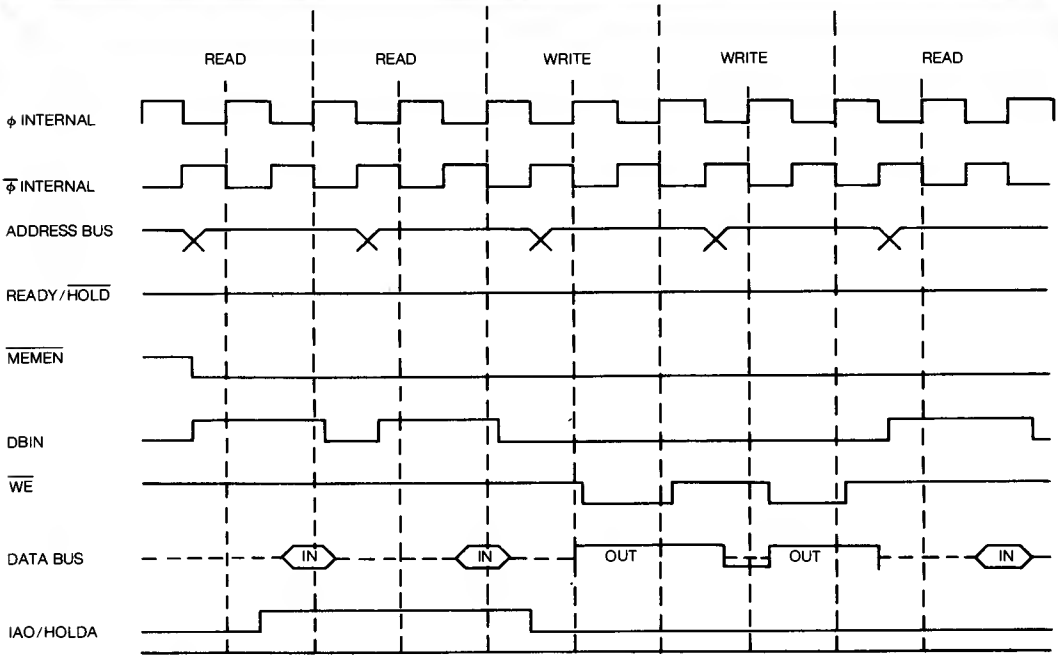


Figure 4. TMS 9985 Memory Bus Timing (no wait states)

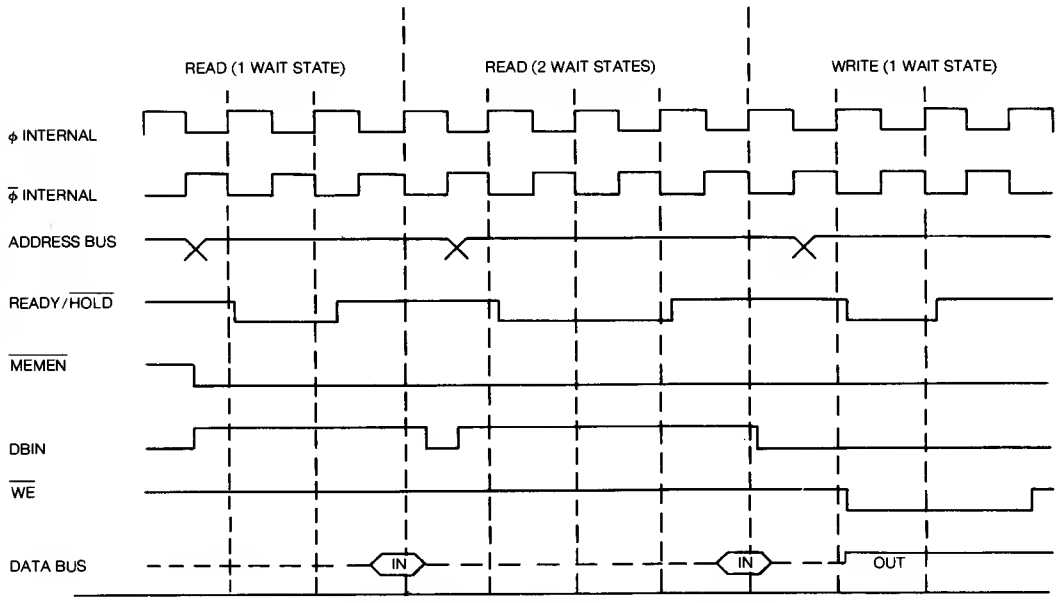


Figure 5. TMS 9985 Memory Bus Timing (with wait states)

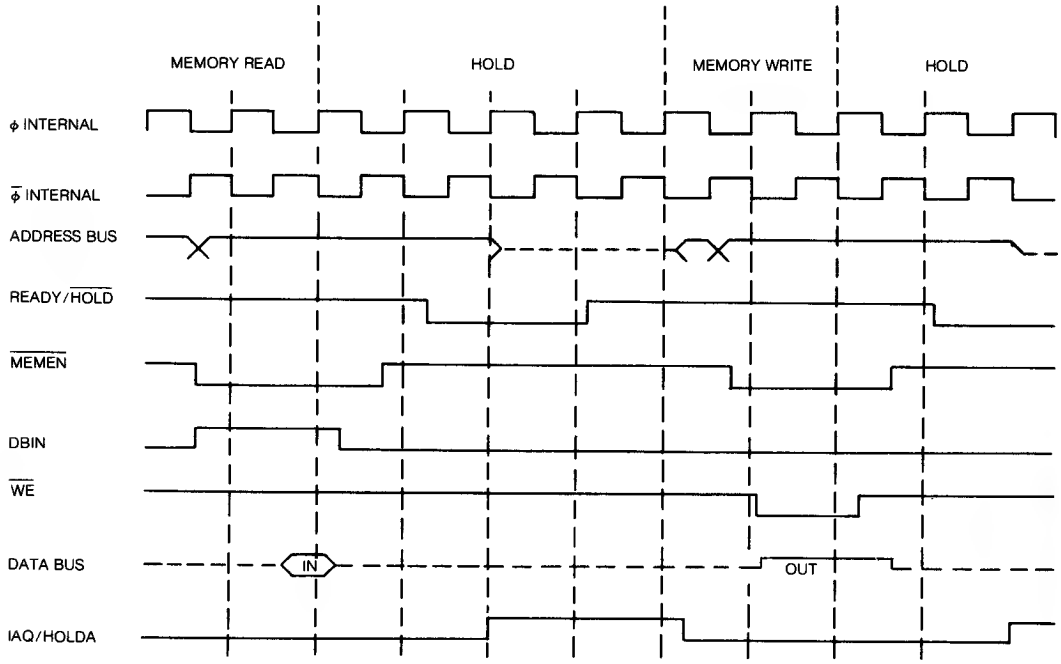


Figure 6. TMS 9985 Hold Timing

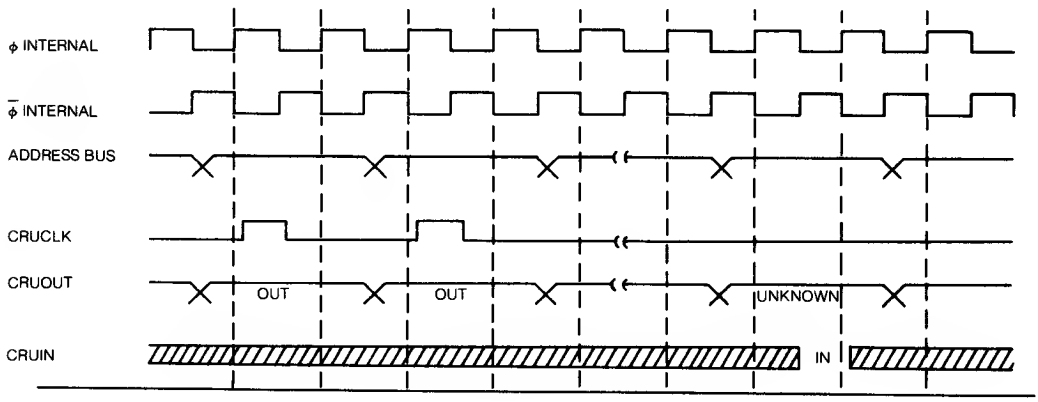


Figure 7. TMS 9985 CRU Interface Timing

9900 Instruction Set

8

9900 INSTRUCTION SET

Definition

Each 9900 instruction performs one of the following operations:

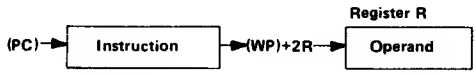
- Arithmetic, logical, comparison, or manipulation operations on data
- Loading or storage of internal registers (program counter, workspace pointer, or status)
- Data transfer between memory and external devices via the CRU
- Control functions.

Addressing Modes

The 9900 instructions contain a variety of available modes for addressing random-memory data (e.g., program parameters and flags), or formatted memory data (character strings, data lists, etc.). The following figures graphically describe the derivation of the effective address for each addressing mode. The applicability of addressing modes to particular instructions is described in the Instructions Section along with the description of the operations performed by the instruction. The symbols following the names of the addressing modes [R, *R, *R+, @ LABEL, or @ TABLE (R)] are the general forms used by 9900 assemblers to select the addressing mode for register R.

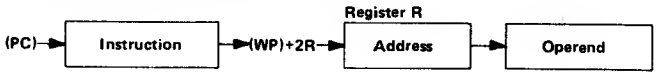
Workspace Register Addressing R

Workspace Register R contains the operand.



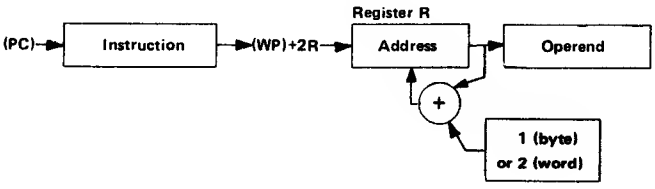
Workspace Register Indirect Addressing *R

Workspace Register R contains the address of the operand.



Workspace Register Indirect Auto Increment Addressing *R+

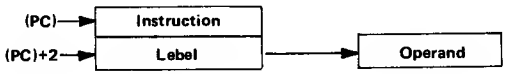
Workspace Register R contains the address of the operand. After acquiring the operand, the contents of workspace register R are incremented.



▶8

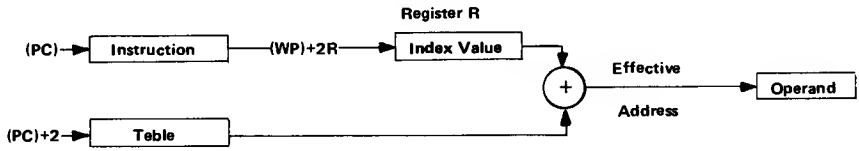
SYMBOLIC (DIRECT) ADDRESSING @LABEL

The word following the instruction contains the address of the operand.



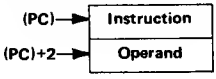
INDEXED ADDRESSING @ TABLE (R)

The word following the instruction contains the base address. Workspace register R conatins the index value. The sum of the base address and the index value results in the effective address of the operand.



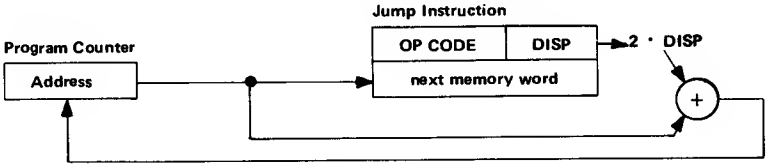
IMMEDIATE ADDRESSING

The word following the instruction contains the operand.



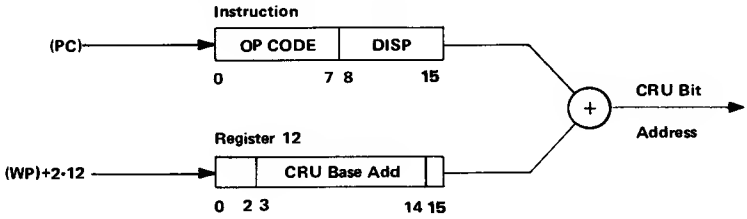
PROGRAM COUNTER RELATIVE ADDRESSING

The 8-bit signed displacement in the right byte (bits 8 through 15) of the instruction is multiplied by 2 and added to the updated contents of the program counter. The result is placed in the PC.



CRU RELATIVE ADDRESSING

The 8-bit signed displacement in the right byte of the instruction is added to the CRU base address (bits 3 through 14 of the workspace register 12). The result is the CRU address of the selected CRU bit.



Terms and Definitions

The following terms are used in describing the instructions of the 9900:

TERM	DEFINITION
B	Byte Indicator (1=byte, 0 = word)
C	Bit count
D	Destination address register
DA	Destination address
IOP	Immediate operand
LSB(n)	Least significant (right most) bit of (n)
MSB(n)	Most significant (left most) bit of (n)
N	Don't care
PC	Program counter
Result	Result of operation performed by instruction
S	Source address register
SA	Source address
ST	Status register
STn	Bit n of status register
T_D	Destination address modifier
T_S	Source address modifier
W	Workspace register
WRn	Workspace register n
(n)	Contents of n
a→b	a is transferred to b
|n|	Absolute value of n
+	Arithmetic addition
−	Arithmetic subtraction
AND	Logical AND
OR	Logical OR
⊕	Logical exclusive OR
$\overline{n}$	Logical complement of n

Status Register

The status register contains the interrupt mask level and information pertaining to the instruction operation.

0	1	2	3	4	5	6	7 8 9 10 11	12 13 14 15
ST0	ST1	ST2	ST3	ST4	ST5	ST6	not used (=0)	ST12 ST13 ST14 ST15
L>	A>	=	C	O	P	X		Interrupt Mask

▶8

BIT	NAME	INSTRUCTION	CONDITION TO SET BIT TO 1
ST0	LOGICAL GREATER THAN	C,CB	If MSB(SA) = 1 and MSB(DA) = 0, or if MSB(SA) = MSB(DA) and MSB of [(DA)-(SA)] = 1
		CI	If MSB(W) = 1 and MSB of IOP = 0, or if MSB(W) = MSB of IOP and MSB of [IOP-(W)] = 1
		ABS	If (SA) ≠ 0
		All Others	If result ≠ 0
ST1	ARITHMETIC GREATER THAN	C,CB	If MSB(SA) = 0 and MSB(DA) = 1, or if MSB(SA) = MSB(DA) and MSB of [(DA)-(SA)] = 1
		CI	If MSB(W) = 0 and MSB of IOP = 1, or if MSB(W) = MSB of IOP and MSB of [IOP-(W)] = 1
		ABS	If MSB(SA) = 0 and (SA) ≠ 0
		All Others	If MSB of result = 0 and result ≠ 0

BIT	NAME	INSTRUCTION	CONDITION TO SET BIT TO 1
ST2	EQUAL	C, CB	If (SA) = (DA)
		C1	If (W) = IOP
		COC	If (SA) and $(\overline{DA})$ = 0
		CZC	If (SA) and (DA) = 0
		TB	If CRUIN = 1
		ABS	If (SA) = 0
		All others	If result = 0
ST3	CARRY	A, AB, ABS, AI, DEC, DECT, INC, INCT, NEG, S, SB	If CARRY OUT = 1
		SLA, SRA, SRC, SRL	If lest bit shifted out = 1
ST4	OVERFLOW	A, AB	If MSB(SA) = MSB(DA) end MSB of result ≠ MSB(DA)
		AI	If MSB(W) = MSB of IOP end MSB of result ≠ MSB(W)
		S, SB	If MSB(SA) ≠ MSB(DA) and MSB of result ≠ MSB(DA)
		DEC, DECT	If MSB(SA) = 1 end MSB of result = 0
		INC, INCT	If MSB(SA) = 0 and MSB of result = 1
		SLA	If MSB changes during shift
		DIV	If MSB(SA) = 0 and MSB(DA) = 1, or if MSB(SA) = MSB(DA) end MSB of [(DA)-(SA)] = 0
		ABS, NEG	If (SA) = $B000_{16}$
ST5	PARITY	CB, MOVB	If (SA) has odd number of 1's
		LDCR, STCR	If 1 ≤ C ≤ B and (SA) hes odd number of 1's
		AB, SB, SOCB, SZCB	If result has odd number of 1's
ST6	XOP	XOP	If XOP instruction is executed
ST12–ST15	INTERRUPT MASK	LIMI	If corresponding bit of IOP is 1
		RTWP	If corresponding bit of WR15 is 1

The TMS 9940 has a slightly different arrangement of its status register. Note that the first six status bits are the same as for the TMS 9900.

0	1	2	3	4	5	6	7	8 9 10 11 12 13	14 15
ST0 L>	ST1 A>	ST2 =	ST3 C	ST4 O	ST5 P	not used (= 0)	ST7 DC	not used (=0)	ST14 ST15 INTERRUPT MASK

ST7	DIGIT CARRY	A,ABS,AI,DEC, DECT,INC,INCT NEG,S	If carry out of least significatn BCD Digit of most significant byte = 1
		AB,DCA,DCS,SB	If carry out of least significant BCD Digit = 1
ST14-ST15	INTERRUPT MASK	LIIM	If corresponding bit of S is 1
		LIMI	If corresponding bit of IOP is 1
		RTWP	If corresponding bit of WR 13 is 1

8◂

INSTRUCTIONS

DUAL OPERAND INSTRUCTIONS WITH MULTIPLE ADDRESSING MODES FOR SOURCE AND DESTINATION OPERAND

General format:

0 1 2	3	4 5	6 7 8 9	10 11	12 13 14 15
OP CODE	B	T_D	D	T_S	S

If B = 1 the operands are bytes and the operand addresses are byte addresses. If B = 0 the operands are words and the operand addresses are word addresses.

The addressing mode for each operand is determined by the T field of that operand.

T_S OR T_D	S OR D	ADDRESSING MODE	NOTES
00	0, 1, . . . 15	Workspace register	1
01	0, 1, . . . 15	Workspaca register indirect	
10	0	Symbolic	4
10	1, 2, . . . 15	Indexed	2,4
11	0, 1, . . . 15	Workspace register indirect auto-increment	3

Notes: 1. When a workspace register is the operand of a byte instruction (bit 3 = 1), the left byte (bits 0 through 7) is the operand and the right byte (bits 8 through 15) is unchanged.
2. Workspace register 0 may not be used for indexing.
3. The workspace register is incremented by 1 for byte instructions (bit 3 = 1) and is incremented by 2 for word instructions (bit 3 = 0).
4. When $T_S = T_D = 10$, two words are required in addition to the instruction word. The first word is the source operand base address and the second word is the destination operand base address.

MNEMONIC	OP CODE 0 1 2	B 3	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
A	1 0 1	0	Add	Yes	0-4	(SA)+(DA) → (DA)
AB	1 0 1	1	Add bytas	Yas	0-5	(SA)+(DA) → (DA)
C	1 0 0	0	Compare	No	0-2	Compare (SA) to (DA) and set appropriate status bits
CB	1 0 0	1	Compare bytes	No	0-2,5	Compare (SA) to (DA) and set appropriata status bits
S	0 1 1	0	Subtract	Yas	0-4	(DA) – (SA) → (DA)
SB	0 1 1	1	Subtract bytes	Yes	0-5	(DA) – (SA) → (DA)
SOC	1 1 1	0	Set ones corresponding	Yes	0-2	(DA) OR (SA) → (DA)
SOCB	1 1 1	1	Set ones corresponding bytes	Yes	0-2,5	(DA) OR (SA) → (DA)
SZC	0 1 0	0	Set zeroes corrasponding	Yas	0-2	(DA) AND ($\overline{SA}$) → (DA)
SZCB	0 1 0	1	Set zeroes corresponding bytes	Yas	0-2,5	(DA) AND ($\overline{SA}$) → (DA)
MOV	1 1 0	0	Move	Yes	0-2	(SA) → (DA)
MOVB	1 1 0	1	Move bytes	Yas	0-2,5	(SA) → (DA)

▶8

DUAL OPERAND INSTRUCTIONS WITH MULTIPLE ADDRESSING MODES FOR THE SOURCE OPERAND AND WORKSPACE REGISTER ADDRESSING FOR THE DESTINATION

General format:

0 1 2 3 4 5	6 7 8 9	10 11	12 13 14 15
OP CODE	D	T_S	S

The addressing mode for the source operand is determined by the T_S field.

T_S	S	ADDRESSING MODE	NOTES
00	0, 1, . . . 15	Workspace register	
01	0, 1, . . . 15	Workspace register indirect	
10	0	Symbolic	
10	1, 2, . . . 15	Indexed	1
11	0, 1, . . . 15	Workspace register indirect auto increment	2

Notes: 1. Workspace register 0 may not be used for indexing.
2. The workspace register is incremented by 2.

MNEMONIC	OP CODE 0 1 2 3 4 5	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
COC	0 0 1 0 0 0	Compere ones corresponding	No	2	Test (D) to determine if 1's are in each bit position where 1's are in (SA). If so, set ST2.
CZC	0 0 1 0 0 1	Compare zeros corresponding	No	2	Test (D) to determine if 0's ere in eech bit position where 1's are in (SA). If so, set ST2.
XOR	0 0 1 0 1 0	Exclusive OR	Yes	0-2	(D) ⊕ (SA) → (D)
MPY	0 0 1 1 1 0	Multiply	No		Multiply unsigned (D) by unsigned (SA) and plece unsigned 32-bit product in D (most significant) end D+1 (least significant). If WR15 is D, the next word in memory efter WR15 will be used for the least significant half of the product.
DIV	0 0 1 1 1 1	Divide	No	4	If unsigned (SA) is less than or equal to unsigned (D), perform no operation end set ST4. Otherwise, divide unsignad (D) and (D+1) by unsigned (SA). Quotient → (D), remainder → (D+1). If D = 15, the next word in memory after WR 15 will be used for the remainder.

8◀

EXTENDED OPERATION (XOP) INSTRUCTION

General format:

0 1 2 3 4 5	6 7 8 9	10 11	12 13 14 15
OP CODE	D	T_S	S

The T_S and S fields provide multiple mode addressing capability for the source operand. When the XOP is executed, ST6 is set and the following transfers occur:

$(40_{16} + 4D) \rightarrow (WP)$
$(42_{16} + 4D) \rightarrow (PC)$
$SA \rightarrow$ (new WR11)
(old WP) $\rightarrow$ (new WR 13)
(old PC) $\rightarrow$ (new WR 14)
(old ST) $\rightarrow$ (new WR 15)

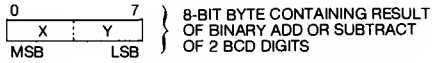
The TMS 9900 does not test interrupt requests ($\overline{INTREQ}$) upon completion of the XOP instruction. The TMS 9980A/TMS 9981 tests for reset and load but does not test for interrupt requests ($\overline{INTREQ}$) upon completion of the XOP instruction.

The TMS 9940 has the same general format for extended operations as the TMS 9900 with the differences described below.

MNEMONIC	D FIELD 6789	MEANING	RESULT COMPARED TO ZERO?	STATUS BITS AFFECTED	DESCRIPTION
DCA	0000	Decimal Correct Addition	Yes	0-3,5,7	The byte specified by SA is corrected to form 2 BCD digits as shown in Table 4
DCS	0001	Decimal Correct Subtraction	Yes	0-3,5,7	The byte specified by SA is corrected to form 2 BCD digits as shown in Table 4
LIIM	001X	Load Interrupt Mask	No	14,15	Ts must equal 0. S, Bits 14 and 15 → ST 14 and ST 15.
XOP	01XX 10XX 11XX	General XOP	No	–	$(40_{16} + 4D) \rightarrow (WP)$ $(42_{16} + 4D) \rightarrow (PC)$; SA → (New WR11); (Old WP) → (New WR 13); (Old PC) → (New WR 14); (Old ST) → (New WR 15); Following execution of an XOP instruction, the TMS 9940 inhibits interrupt levels 1,2, and 3 until one more instruction is executed.

▶8

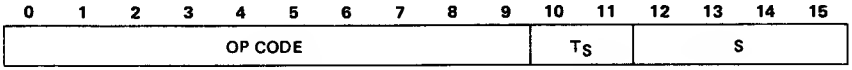
RESULT OF DCA AND DCS INSTRUCTIONS



BYTE BEFORE EXECUTION				BYTE AFTER DCA				BYTE AFTER DCS			
C	X	DC	Y	C	X	DC	Y	C	X	DC	Y
0	X<10	0	Y<10	0	X	0	Y	—	—	—	—
0	X<10	1	Y<10	0	X	0	Y+6	—	—	—	—
0	X<9	0	Y≥10	0	X+1	1	Y+6	—	—	—	—
1	X<10	0	Y<10	1	X+6	0	Y	—	—	—	—
1	X<10	1	Y<10	1	X+6	0	Y16	—	—	—	—
1	X<10	0	Y≥10	1	X+7	1	Y+6	—	—	—	—
0	X≥10	0	Y<10	1	X+6	0	Y	—	—	—	—
0	Z≥10	1	Y<10	1	X+6	0	Y+6	—	—	—	—
0	X≥9	0	Y≥10	1	X+7	1	Y+6	—	—	—	—
0	X	0	Y	—	—	—	—	0	X+10	1	Y+10
0	X	1	Y	—	—	—	—	0	X+10	0	Y
1	X	0	Y	—	—	—	—	1	X	1	Y+10
1	X	1	Y	—	—	—	—	1	X	0	Y

Single Operand Instructions

General format:



The T_S and S fields provide multiple mode addressing capability for the source operand.

8

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7 8 9	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
B	0 0 0 0 0 1 0 0 0 1	Branch	No	–	SA → (PC)
BL	0 0 0 0 0 1 1 0 1 0	Branch and link	No	–	(PC) → (WR11); SA → (PC)
BLWP	0 0 0 0 0 1 0 0 0 0	Branch and load workspace pointer	No	–	(SA) → (WP); (SA+2) → (PC); (old WP) → (new WR13); (old PC) → (new WR14); (old ST) → (new WR15); the interrupt input ($\overline{\text{INTREQ}}$) is not tested upon completion of the BLWP instruction. The TMS 9980A/TMS 99B1 tests for reset and load but does not test for interrupt requests ($\overline{\text{INTREQ}}$) upon completion of the XOP instruction.
CLR	0 0 0 0 0 1 0 0 1 1	Clear operand	No	–	0 → (SA)
SETO	0 0 0 0 0 1 1 1 0 0	Set to ones	No	–	$FFFF_{16}$ → (SA)
INV	0 0 0 0 0 1 0 1 0 1	Invert	Yes	0-2	($\overline{\text{SA}}$) → (SA)
NEG	0 0 0 0 0 1 0 1 0 0	Negate	Yes	0-4	–(SA) → (SA)
ABS	0 0 0 0 0 1 1 1 0 1	Absolute value*	No	0-4	|(SA)| → (SA)
SWPB	0 0 0 0 0 1 1 0 1 1	Swap bytes	No	–	(SA), bits 0 thru 7 → (SA), bits 8 thru 15; (SA), bits 8 thru 15 → (SA), bits 0 thru 7.
INC	0 0 0 0 0 1 0 1 1 0	Increment	Yes	0-4	(SA) + 1 → (SA)
INCT	0 0 0 0 0 1 0 1 1 1	Increment by two	Yes	0-4	(SA) + 2 → (SA)
DEC	0 0 0 0 0 1 1 0 0 0	Decrement	Yes	0-4	(SA) – 1 → (SA)
DECT	0 0 0 0 0 1 1 0 0 1	Decrement by two	Yes	0-4	(SA) – 2 → (SA)
X†	0 0 0 0 0 1 0 0 1 0	Execute	No	–	Execute the instruction at SA.

*Operand is compered to zero for stetus bit.

†If edditional memory words for the axecute instruction era required to define the operends of the instruction loceted et SA, these words will be eccessed from PC and tha PC will be updeted eccordingly. The instruction acquisition signal (IAQ) will not be true when the 9900 eccasses the instruction at SA. Status bits ere affected in the normal menner for the instruction executed.

CRU Multiple-Bit Instructions

General format:

0 1 2 3 4 5	6 7 8 9	10 11	12 13 14 15
OP CODE	C	T_S	S

▸8

The C field specifies the number of bits to be transferred. If C = 0, 16 bits will be transferred. The CRU base register (WR 12, bits 3 through 14) defines the starting CRU bit address. The bits are transferred serially and the CRU address is incremented with each bit transfer, although the contents of WR 12 is not affected. T_S and S provide multiple mode addressing capability for the source operand. If 8 or fewer bits are transferred (C = 1 through 8), the source address is a byte address. If 9 or more bits are transferred (C = 0, 9 through 15), the source address is a word address. If the source is addressed in the workspace register indirect auto increment mode, the workspace register is incremented by 1 if C = 1 through 8, and is incremented by 2 otherwise.

MNEMONIC	OP CODE 0 1 2 3 4 5	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
LDCR	0 0 1 1 0 0	Load communcation register	Yes	0-2,5†	Beginning with LSB of (SA), transfer the specified number of bits from (SA) to the CRU.
STCR	0 0 1 1 0 1	Store communcation register	Yes	0-2,5†	Beginning with LSB of (SA), transfer the specified number of bits from the CRU to (SA). Load unfilled bit positions with 0.

†ST5 is affected only if $1 \leq C \leq 8$.

CRU Single-Bit Instructions

General format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE								SIGNED DISPLACEMENT							

CRU relative addressing is used to address the selected CRU bit.

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7	MEANING	STATUS BITS AFFECTED	DESCRIPTION
SBO	0 0 0 1 1 1 0 1	Set bit to one	–	Set the selected CRU output bit to 1.
SBZ	0 0 0 1 1 1 1 0	Set bit to zero	–	Set the selected CRU output bit to 0.
TB	0 0 0 1 1 1 1 1	Test bit	2	If the selected CRU input bit = 1, set ST2.

Jump Instructions

General format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE								DISPLACEMENT							

Jump instructions cause the PC to be loaded with the value selected by PC relative addressing if the bits of ST are at specified values. Otherwise, no operation occurs and the next instruction is executed since PC points to the next instruction. The displacement field is a word count to be added to PC. Thus, the jump instruction has a range of –128 to 127 words from memory-word address following the jump instruction. No ST bits are affected by jump instruction.

8◀

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7	MEANING	ST CONDITION TO LOAD PC
JEQ	0 0 0 1 0 0 1 1	Jump equal	ST2 = 1
JGT	0 0 0 1 0 1 0 1	Jump greater than	ST1 = 1
JH	0 0 0 1 1 0 1 1	Jump high	ST0 = 1 and ST2 = 0
JHE	0 0 0 1 0 1 0 0	Jump high or equal	ST0 = 1 or ST2 = 1
JL	0 0 0 1 1 0 1 0	Jump low	ST0 = 0 and ST2 = 0
JLE	0 0 0 1 0 0 1 0	Jump low or equal	ST0 = 0 or ST2 = 1
JLT	0 0 0 1 0 0 0 1	Jump less than	ST1 = 0 and ST2 = 0
JMP	0 0 0 1 0 0 0 0	Jump unconditional	unconditional
JNC	0 0 0 1 0 1 1 1	Jump no carry	ST3 = 0
JNE	0 0 0 1 0 1 1 0	Jump not equal	ST2 = 0
JNO	0 0 0 1 1 0 0 1	Jump no overflow	ST4 = 0
JOC	0 0 0 1 1 0 0 0	Jump on carry	ST3 = 1
JOP	0 0 0 1 1 1 0 0	Jump odd parity	ST5 = 1

Shift Instructions

General format:

0 1 2 3 4 5 6 7	8 9 10 11	12 13 14 15
OP CODE	C	W

If C=0, bits 12 through 14 of WR0 contain the shift count. If C=0 and bits 12 through 15 of WR0=0, the shift count is 16.

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
SLA	0 0 0 0 1 0 1 0	Shift left arithmetic	Yes	0-4	Shift (W) left. Fill vacated bit positions with 0.
SRA	0 0 0 0 1 0 0 0	Shift right arithmetic	Yes	0-3	Shift (W) right. Fill vacated bit positions with original MSB of (W).
SRC	0 0 0 0 1 0 1 1	Shift right circular	Yes	0-3	Shift (W) right. Shift previous LSB into MSB.
SRL	0 0 0 0 1 0 0 1	Shift right logical	Yes	0-3	Shift (W) right. Fill vacated bit positions with 0's.

Immediate Register Instructions

8 General format:

0 1 2 3 4 5 6 7 8 9 10	11	12 13 14 15
OP CODE	N	W
IOP		

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7 8 9 10	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
AI	0 0 0 0 0 0 1 0 0 0 1	Add immediate	Yes	0-4	(W) + IOP → (W)
ANDI	0 0 0 0 0 0 1 0 0 1 0	AND immediate	Yes	0-2	(W) AND IOP → (W)
CI	0 0 0 0 0 0 1 0 1 0 0	Compare immediate	Yes	0-2	Compare (W) to IOP and set appropriate status bits
LI	0 0 0 0 0 0 1 0 0 0 0	Load immediate	Yes	0-2	IOP → (W)
ORI	0 0 0 0 0 0 1 0 0 1 1	OR immediate	Yes	0-2	(W) OR IOP → (W)

Internal Register Load Immediate Instructions

General format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE											N				
IOP															

MNEMONIC	OP CODE 0	1	2	3	4	5	6	7	8	9	10	MEANING	DESCRIPTION
LWPI	0	0	0	0	0	0	1	0	1	1	1	Load workspace pointer immediate	IOP → (WP), no ST bits affected
LIMI	0	0	0	0	0	0	1	1	0	0	0	Load interrupt mask	IOP, bits 12 thru 15 → ST12 thru ST15

Internal Register Store Instructions

General format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE											N	W			

No ST bits are affected.

MNEMONIC	OP CODE 0	1	2	3	4	5	6	7	8	9	10	MEANING	DESCRIPTION
STST	0	0	0	0	0	0	1	0	1	1	0	Store status register	(ST) → (W)
STWP	0	0	0	0	0	0	1	0	1	0	1	Store workspace pointer	(WP) → (W)

Return Workspace Pointer (RTWP) Instruction

General format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	1	1	1	0	0	N				

The RTWP instruction causes the following transfers to occur:

(WR 15)→(ST)
(WR 14)→(PC)
(WR 13)→(WP)

8◂

External Instructions

General format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE											N				

External instructions cause the three most-significant address lines (A0 through A2) to be set to the below-described levels and the CRUCLK line to be pulsed, allowing external control functions to be initiated.

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7 8 9 10	MEANING	STATUS BITS AFFECTED	DESCRIPTION	ADDRESS BUS A0	A1	A2
IDLE	0 0 0 0 0 0 1 1 0 1 0	Idle		Suspend TMS 9900 instruction execution until en interrupt, $\overline{\text{LOAD}}$, or $\overline{\text{RESET}}$ occurs	L	H	L
RSET	0 0 0 0 0 0 1 1 0 1 1	Reset	12–15	0 → ST12 thru ST15	L	H	H
CKOF	0 0 0 0 0 0 1 1 1 1 0	User defined		——	H	H	L
CKON	0 0 0 0 0 0 1 1 1 0 1	User defined		——	H	L	H
LREX	0 0 0 0 0 0 1 1 1 1 1	User defined		——	H	H	H

IDLE INSTRUCTION — TMS 9940

General format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE											N				

The IDLE instruction stops the TMS 9940 until an interrupt or $\overline{\text{RESET}}$ occurs. See the *Power Down* section for use of the IDLE instruction.

▸8

Peripheral and Interface Circuits

8

1. INTRODUCTION

1.1 DESCRIPTION

The TMS 9901 Programmable Systems Interface (PSI) is a multifunctional component designed to provide low cost interrupt and I/O ports and an interval timer for TMS 9900-family microprocessor systems. The TMS 9901 is fabricated using N-channel silicon-gate MOS technology. The TMS 9901 is TTL-compatible on all inputs and outputs, including the power supply (+5 V) and single-phase clock.

1.2 KEY FEATURES

- Low Cost
- 9900-Family Peripheral
- Performs Interrupt and I/O Interface functions:
  - — Six Dedicated Interrupt Lines
  - — Seven Dedicated I/O Lines
  - — Nine Programmable Lines as I/O or Interrupt
  - — Up to 15 Interrupt Lines
  - — Up to 22 Input Lines
  - — Up to 16 Output Lines
- Easily Cascaded for Expansion
- Interval or Event Timer
- Single 5 V Power Supply
- All Inputs and Outputs TTL-Compatible
- Standard 40-Pin Plastic or Ceramic Package
- N-Channel Silicon-Gate MOS Technology.

1.3 APPLICATION OVERVIEW

The following example of a typical application may help introduce the user to the TMS 9901 PSI. Figure 1 is a block diagram of a typical application. Each of the ideas presented below is described in more detail in later sections of this manual.

The TMS 9901 PSI interfaces to the CPU through the Communications Register Unit (CRU) and the interrupt control lines as shown in Figure 1. The TMS 9901 occupies 32 bits of CRU input and output space. The five least significant bits of address bus are connected to the S lines of the PSI to address one of the 32 CRU bits of the TMS 9901. The most significant bits of the address bus are decoded on CRU cycles to select the PSI by taking its chip enable ($\overline{CE}$) line active (LOW).

Interrupt inputs to the TMS 9901 PSI are synchronized with $\bar{\phi}$, inverted, and then ANDed with the appropriate mask bit. Once every $\bar{\phi}$ clock time, the prioritizer looks at the 15 interrupt input AND gates and generates the interrupt control code. The interrupt control code and the interrupt request line constitute the interrupt interface to the CPU.

After reset all I/O ports are programmed as inputs. By writing to any I/O port, that port will be programmed as an output port until another reset occurs, either software or hardware. Data at the input pins is buffered on to the TMS 9901. Data to the output ports is latched and then buffered off-chip by the PSI's MOS-to-TTL buffers.

The interval timer on the TMS 9901 is accessed by writing a ONE to select bit zero, (control bit) which puts the PSI CRU interface in the clock mode. Once in the clock mode the 14-bit clock contents can be read or written. Writing to the clock register will reinitialize the clock and cause it to start decrementing. When the clock counts to zero, it will cause an interrupt and reload to its initial value. Reading the clock contents permits the user to see the decrementer contents at that point in time just before entering the clock mode. The clock read register is not updated when the PSI is in the clock mode.

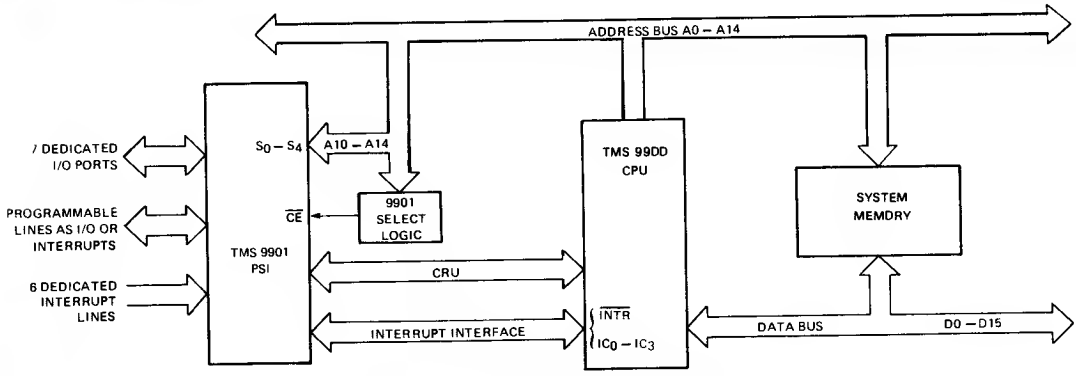


FIGURE 1– TYPICAL TMS 9901 PROGRAMMABLE SYSTEM INTERFACE (PSI) APPLICATION

2. ARCHITECTURE

The architecture of the TMS 9901 Programmable Systems Interface (PSI) is designed to provide the user maximum flexibility when designating system I/O ports and interrupts. The TMS 9901 can be divided into four subsystems: CRU interface, interrupt interface, input/output interface, and interval timer. Figure 2 is a general block diagram of the TMS 9901 internal architecture. Each of the subsystems of the PSI is discussed in detail in subsequent paragraphs.

2.1 CRU Interface

The CPU communicates with the TMS 9901 PSI via the CRU. The TMS 9901 occupies 32 bits in CRU read space and 32 bits in CRU write space. Table 1 shows the mapping for CRU bit addresses to TMS 9901 functions.

The CRU interface consists of five address select lines (S0-S4), chip enable ($\overline{CE}$), and the three CRU lines (CRUIN, CRUOUT, CRUCLK). The select lines (S0-S4) are connected to the five least significant bits of the address bus; for a TMS 9900 system S0-S4 are connected to A10-A14, respectively. Chip enable ($\overline{CE}$) is generated by decoding the most significant bits of the address bus on CRU cycles; for a 9900 based system address bits 0-9 would be decoded. When $\overline{CE}$ goes active (LOW), the five select lines point to the CRU bit being accessed. When $\overline{CE}$ is inactive (HIGH), the PSI's CRU interface is disabled.

NOTE

When $\overline{CE}$ is inactive (HIGH) the 9901 sets its CRUIN pin to high impedance and disables CRUCLK from coming on chip. This means that CRUIN can be used as an OR tied bus. When $\overline{CE}$ is high the 9901 will still see the select lines, but no command action is taken.

In the case of a write operation, the TMS 9901 strobes data off the CRUOUT line with CRUCLK. For a read operation, the data is sent to the CPU on the CRUIN line.

8◀

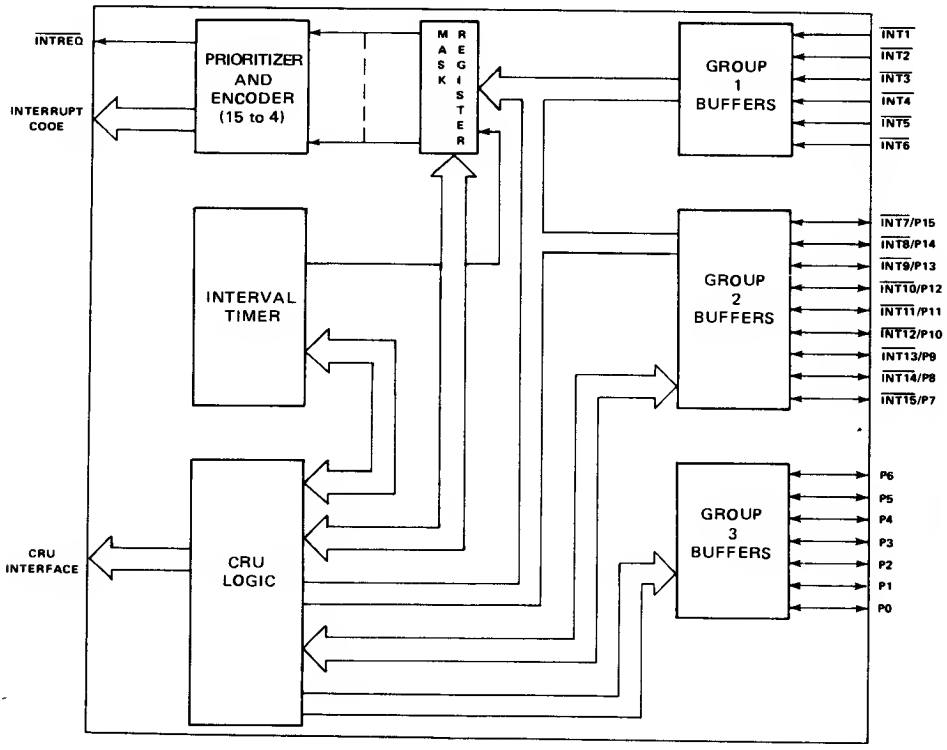


FIGURE 2—TMS 9901 PSI BLOCK DIAGRAM

Several TMS 9901 devices may be cascaded to expand I/O and interrupt handling capability simply by connecting all CRU and address select lines in parallel and providing each device with a unique chip enable signal: the chip enable ($\overline{CE}$) is generated by decoding the high-order address bits (A0-A9) on CRU cycles.

▸8

For those unfamiliar with the CRU concept, the following is a discussion of how to build a CRU interface. The CRU is a bit addressable (4096 bits), synchronous, serial interface over which a single instruction can transfer between one and 16 bits serially. Each one of the 4096 bits of the CRU space has a unique address and can be read and written to. During multi-bit CRU transfers, the CRU address is incremented at the beginning of each CRU cycle to point to the next consecutive CRU bit.

TABLE 1

SELECT BIT ASSIGNMENTS

SELECT BIT	S_0 S_1 S_2 S_3 S_4	CRU Read Data	CRU Write Data
0	0 0 0 0 0	CONTROL BIT(1)	CONTROL BIT(1)
1	0 0 0 0 1	$\overline{INT}$1/CLK1(2)	Mask 1/CLK1(3)
2	0 0 0 1 0	$\overline{INT}$2/CLK2	Mask 2/CLK2
3	0 0 0 1 1	$\overline{INT}$3/CLK3	Mask 3/CLK3
4	0 0 1 0 0	$\overline{INT}$4/CLK4	Mask 4/CLK4
5	0 0 1 0 1	$\overline{INT}$5/CLK5	Mask 5/CLK5
6	0 0 1 1 0	$\overline{INT}$6/CLK6	Mask 6/CLK6
7	0 0 1 1 1	$\overline{INT}$7/CLK7	Mask 7/CLK7
8	0 1 0 0 0	$\overline{INT}$8/CLK8	Mask 8/CLK8
9	0 1 0 0 1	$\overline{INT}$9/CLK9	Mask 9/CLK9
10	0 1 0 1 0	$\overline{INT}$10/CLK10	Mask 10/CLK10
11	0 1 0 1 1	$\overline{INT}$11/CLK11	Mask 11/CLK11
12	0 1 1 0 0	$\overline{INT}$12/CLK12	Mask 12/CLK12
13	0 1 1 0 1	$\overline{INT}$13/CLK13	Mask 13/CLK13
14	0 1 1 1 0	$\overline{INT}$14/CLK14	Mask 14/CLK14
15	0 1 1 1 1	$\overline{INT}$15/INTREQ(7)	Mask 15/$\overline{RST}$2(4)
16	1 0 0 0 0	PO Input(5)	PO Output(6)
17	1 0 0 0 1	P1 Input	P1 Output
18	1 0 0 1 0	P2 Input	P2 Output
19	1 0 0 1 1	P3 Input	P3 Output
20	1 0 1 0 0	P4 Input	P4 Output
21	1 0 1 0 1	P5 Input	P5 Output
22	1 0 1 1 0	P6 Input	P6 Output
23	1 0 1 1 1	P7 Input	P7 Output
24	1 1 0 0 0	P8 Input	P8 Output
25	1 1 0 0 1	P9 Input	P9 Output
26	1 1 0 1 0	P10 Input	P10 Output
27	1 1 0 1 1	P11 Input	P11 Output
28	1 1 1 0 0	P12 Input	P12 Output
29	1 1 1 0 1	P13 Input	P13 Output
30	1 1 1 1 0	P14 Input	P14 Output
31	1 1 1 1 1	P15 Input	P15 Output

NOTES:
(1) 0 Interrupt Mode 1 = Clock Mode
(2) Data present on $\overline{INT}$ input pin (or clock value) will be read regardless of mask value.
(3) While in the Interrupt Mode (Control Bit = 0) writing a "1" into mask will enable interrupt; a "0" will disable.
(4) Writing a zero to bit 15 while in the clock mode (Control Bit = 1) executes a software reset of the I/O pins.
(5) Data present on the pin will be read. Output data can be read without affecting the data.
(6) Writing data to the port will program the port to the output mode and output the data.
(7) INTREQ is the inverted status of the $\overline{INTREQ}$ pin.

When a 99XX CPU executes a CRU Instruction, the processor uses the contents of workspace register 12 as a base address. (Refer to the *9900 Microprocessor Data Manual* for a complete discussion on how CRU addresses are derived.) The CRU address is brought out on the 15-bit address bus; this means that the least significant bit of R12 is not brought out of the CPU. During CRU cycles, the memory control lines ($\overline{MEMEN}$, $\overline{WE}$, and DBIN) are all inactive; $\overline{MEMEN}$ being inactive (HIGH) indicates the address is not a memory address and therefore is a CRU address or external instruction code. Also, when $\overline{MEMEN}$ is inactive (HIGH) and a valid address is present, address bits A0-A2 must all be zero to constitute a valid CRU address; if address bits A0-A2 are other than all zeros, they are indicating an external instruction code. In summary, address bits A3-A14 contain the CRU address to be decoded, address bits A0-A2 must be zero and $\overline{MEMEN}$ must be inactive (HIGH) to indicate a CRU cycle.

2.2 Interrupt Interface

A block diagram of the interrupt control section is shown in Figure 3. The interrupt inputs (six dedicated, $\overline{INT1}$-$\overline{INT6}$, and nine programmable) are sampled on the falling edge of $\overline{\phi}$ and latched onto the chip for one $\overline{\phi}$ time by the SYNC LATCH, each $\overline{\phi}$ time. The output of the sync latch is inverted (interrupts are LOW active) and ANDed with its respective mask bit (MASK = 1, INTERRUPT ENABLED). On the rising edge of $\overline{\phi}$, the prioritizer and encoder senses the masked interrupts and produces a four-bit encoding of the highest priority interrupt present (see Tables 2 and 3). The four-bit prioritized code and $\overline{INTREQ}$ are latched off-chip with a sync latch on the falling edge of the next $\overline{\phi}$, which ensures proper synchronization to the processor.

Once an interrupt goes active (LOW), it should stay active until the appropriate interrupt service routine explicitly turns off the interrupt. If an interrupt is allowed to go inactive before the interrupt service routine is entered, an erroneous interrupt code could be sent to the processor. A total of five clock cycles occur between the time the CPU samples the $\overline{INTREQ}$ line and the time it samples the IC0-IC3 lines. For example, if an interrupt is active and the CPU recognizes that an interrupt is pending, *but before the CPU can sample the interrupt control lines the interrupt goes inactive*, the interrupt control lines will contain an incorrect code.

The interrupt mask bits on the TMS 9901 PSI are individually set or reset under software control. Any unused interrupt line should have its associated mask disabled to avoid false interrupts: To do this, the control bit (CRU bit zero), is first set to a zero for interrupt mode operation. Writing to TMS 9901 CRU bits 1-15 will enable or disable interrupts 1-15, respectively. Writing a one to an interrupt mask will *enable* that interrupt; writing a zero will *disable* that interrupt. Upon application of $\overline{RST1}$ (power-up reset), all mask bits are reset (LOW), the interrupt code is forced to all zeros, and $\overline{INTREQ}$ is held HIGH. Reading TMS 9901 CRU bits 1-15 indicates the status of the respective interrupt inputs; thus, the designer can employ the unused (disabled) interrupt input lines as data inputs (true data in).

2.3 Input/Output Interface

A block diagram of the TMS 9901 I/O interface is shown in Figure 4. Up to 16 individually controlled, I/O ports are available (seven dedicated, P0-P6, and nine programmable) and, as discussed above, the unused dedicated interrupt lines also can be used as input lines (true data in). Thus the 9901 can be configured to have more than 16 inputs. $\overline{RST1}$ (power-up reset) will program all I/O ports to input mode. Writing data to a port will automatically switch that port to the output mode. Once programmed as an output, a port will remain in output mode until $\overline{RST1}$ or $\overline{RST2}$ (command bit) is executed. An output port can be read and indicates the present state of the pin. A pin programmed to the output mode *cannot* be used as an input pin: *Applying an input current to an output pin may cause damage to the TMS 9901.* The TMS 9901 outputs are latched and buffered off-chip, and inputs are buffered onto the chip. The output buffers are MOS-to-TTL buffers and can drive two standard TTL loads.

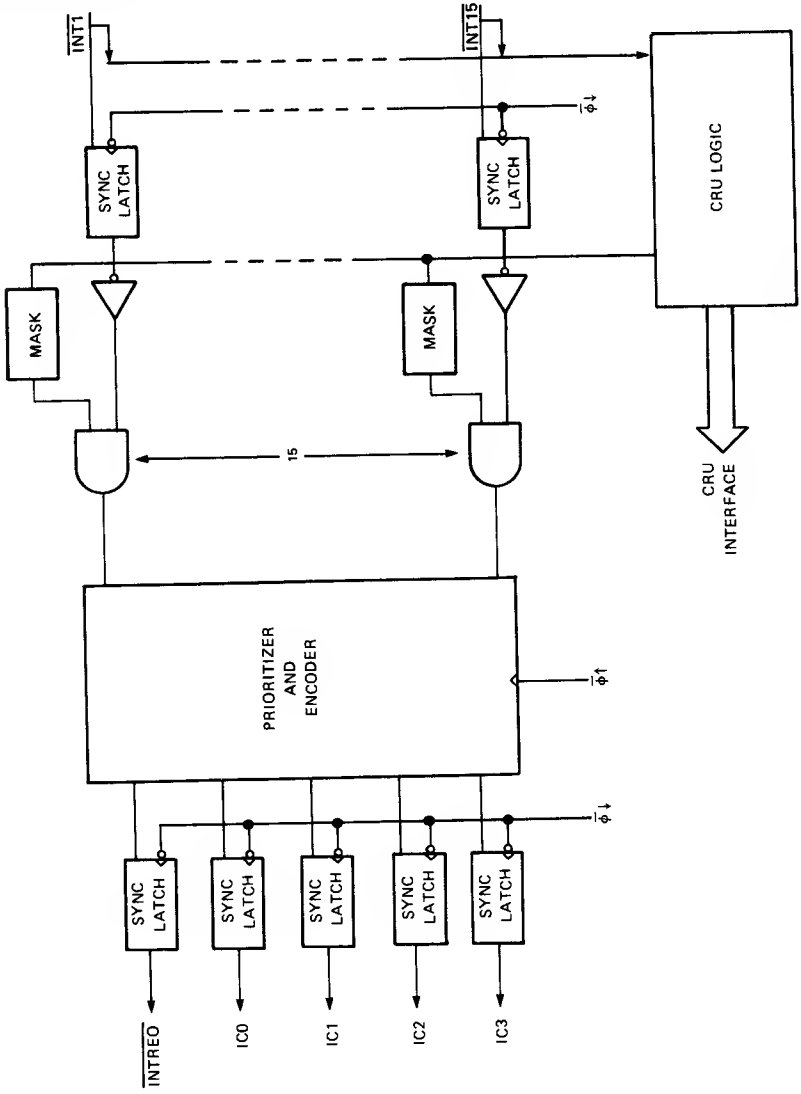


FIGURE 3— TMS 9901 PSI INTERRUPT CONTROL SECTION BLOCK DIAGRAM

TABLE 2
INTERRUPT CODE GENERATION

INTERRUPT/STATE	PRIORITY	I_{C0}	I_{C1}	I_{C2}	I_{C3}	$\overline{INTREQ}$
$\overline{RST}$ 1	–	0	0	0	0	1
$\overline{INT}$ 1	1 (HIGHEST)	0	0	0	1	0
$\overline{INT}$ 2	2	0	0	1	0	0
$\overline{INT}$ 3/CLOCK	3	0	0	1	1	0
$\overline{INT}$ 4	4	0	1	0	0	0
$\overline{INT}$ 5	5	0	1	0	1	0
$\overline{INT}$ 6	6	0	1	1	0	0
$\overline{INT}$ 7	7	0	1	1	1	0
$\overline{INT}$ 8	8	1	0	0	0	0
$\overline{INT}$ 9	9	1	0	0	1	0
$\overline{INT}$ 10	10	1	0	1	0	0
$\overline{INT}$ 11	11	1	0	1	1	0
$\overline{INT}$ 12	12	1	1	0	0	0
$\overline{INT}$ 13	13	1	1	0	1	0
$\overline{INT}$ 14	14	1	1	1	0	0
$\overline{INT}$ 15	15 (LOWEST)	1	1	1	1	0
NO INTERRUPT	–	1	1	1	1	1

TABLE 3
TMS 9980A OR TMS 9981 INTERRUPT LEVEL DATA

INTERRUPT CODE (IC0-IC2)	FUNCTION	VECTOR LOCATION (MEMORY ADDRESS IN HEX)	DEVICE ASSIGNMENT	INTERRUPT MASK VALUES TO ENABLE (ST12 THROUGH ST15)
1 1 0	Level 4	0 0 1 0	External Device	4 Through F
1 0 1	Level 3	0 0 0 C	External Device	3 Through F
1 0 0	Level 2	0 0 0 8	External Device	2 Through F
0 1 1	Level 1	0 0 0 4	External Device	1 Through F
0 0 1	Reset	0 0 0 0	Reset Stimulus	Don't Care
0 1 0	Load	3 F F C	Load Stimulus	Don't Care
0 0 0	Reset	0 0 0 0	Reset Stimulus	Don't Care
1 1 1	No-Op	------	------	

8

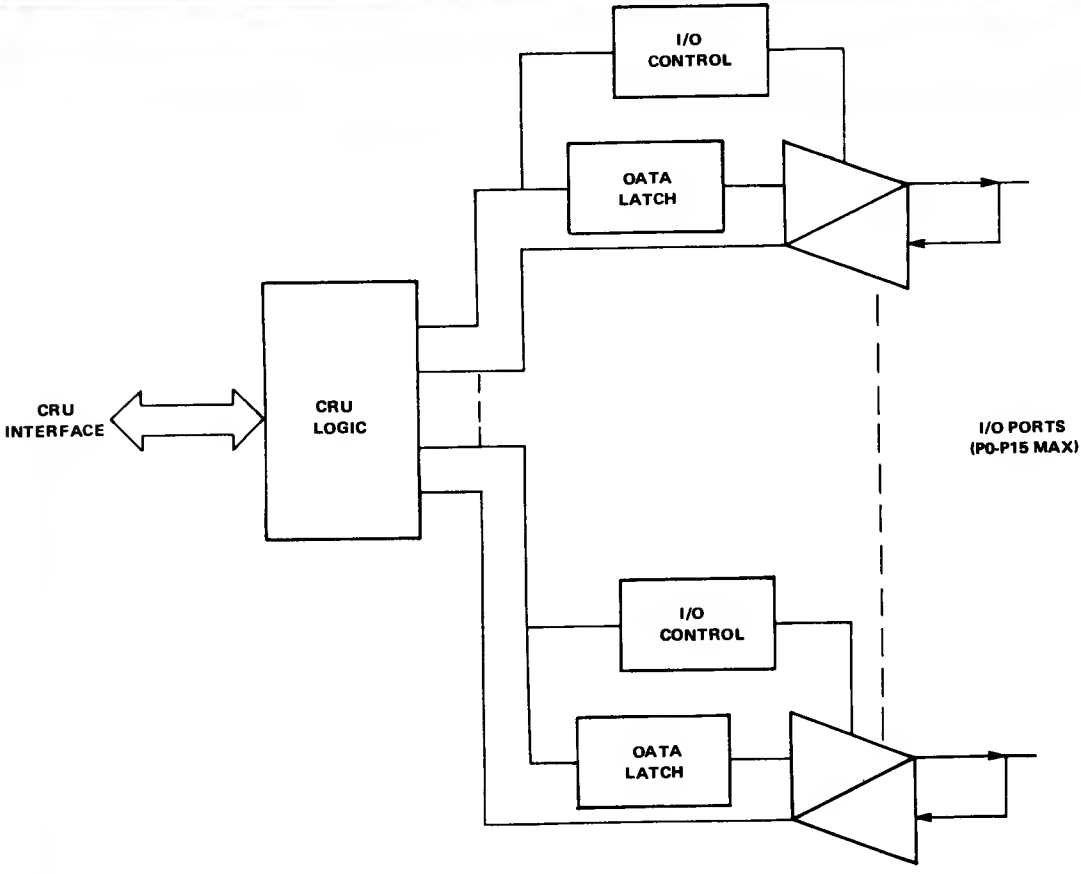


FIGURE 4—TMS 9901 I/O INTERFACE SECTION

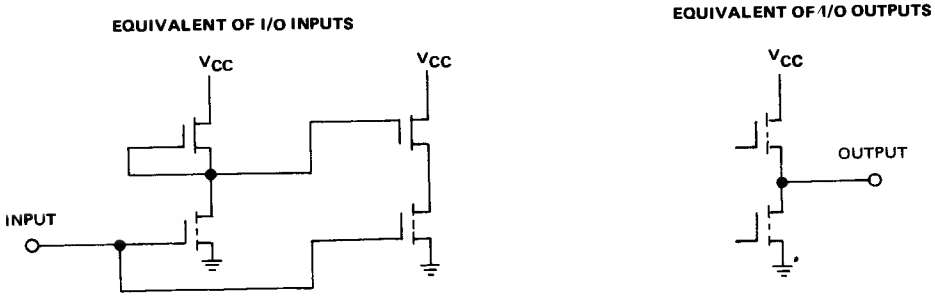


FIGURE 5 – INPUT AND OUTPUT EQUIVALENTS

2.4 Programmable Ports

A total of nine pins ($\overline{INT}$7/P15-$\overline{INT}$15/P7) on the TMS 9901 are user-programmable as either I/O ports or interrupts. These pins will assume all characteristics of the type pin they are programmed to be (as described in Sections 2.2 and 2.3). Any pin which is not being used for interrupt should have the appropriate interrupt mask disabled (mask = 0) to avoid erroneous interrupts to the CPU. To program one of the pins as an interrupt, its interrupt mask simply is enabled and the line may be used as if it were one of the dedicated interrupt lines. To program a pin as an I/O port, disable the interrupt mask and use that pin as if it were one of the dedicated I/O ports.

2.5 Interval Timer

Figure 6 is a block diagram of the TMS 9901 interval timer section. The clock consists of a 14-bit counter that decrements at a rate of $f(\overline{\phi})/64$ (at 3 MHz this results in a maximum interval of 349 milliseconds with a resolution of 21.3 microseconds). The clock can be used as either an interval timer or an event timer. To access the clock, select bit zero (control bit) must be set to a one. The clock is enabled to cause interrupts by writing a nonzero value to it and is then disabled from interrupting by writing zero to it or by a $\overline{RST}$1. The clock starts operating at no more than two ϕ times after it is loaded. When the clock decrementer is running, it will decrement down to zero and issue a level-3 interrupt. The decrementer, when it becomes zero, will also be reloaded from the clock register and decrementing will start again. (The zero state is counted as any other decrementer state.) The decrementer always runs, but it will not issue interrupts unless enabled; of course, the contents of the unenabled clock read register are meaningless.

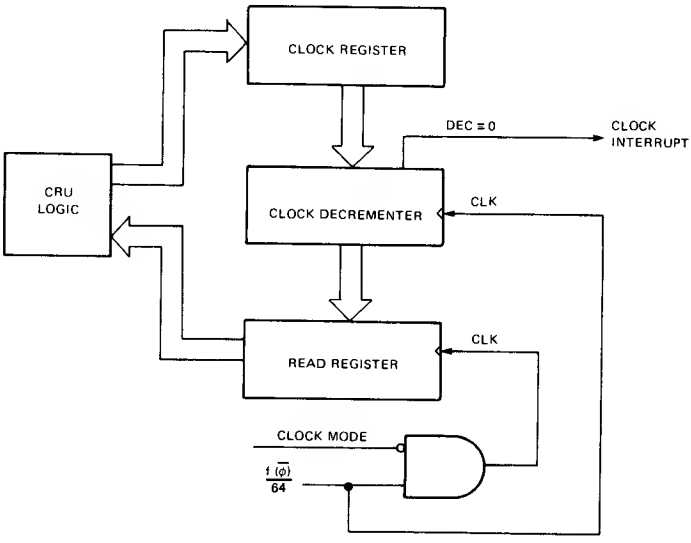


FIGURE 6—TMS 9901 INTERVAL TIMER SECTION

The clock is accessed by writing a one into the control bit (TMS 9901 CRU bit zero) to force CRU bits 1-15 to clock mode. Writing a nonzero value into the clock register then enables the clock and sets its time period. When the clock is enabled, it interrupts on level 3 and external level-3 interrupts are disabled. The mask for level 3 in the PSI must be set to a one so that the processor will see the clock interrupt. When the clock interrupt is active, the clock mask (mask bit 3) must be written into with either a one or zero to clear the interrupt; writing a zero also disables further interrupts.

If a new clock value is required, a new 14-bit clock start value can be programmed by executing a CRU write operation to the clock register. During programming, the decrementer is restarted with the current start value after each start value bit is written. A timer restart is easily implemented by writing a single bit to any of the clock bits. The clock is disabled by $\overline{RST}$1 (power up reset) or by writing a zero value into the clock register; $\overline{RST}$2 does not affect the clock.

The clock read register is updated every time the decrementer decrements when the TMS 9901 is not in clock mode. There are two methods to leave the clock mode : first, a zero is written to the control bit; or second, a TMS 9901 select bit greater than 15 is accessed. Note that when $\overline{CE}$ is inactive(HIGH), the PSI is not disabled from seeing the select lines. As the CPU is addressing memory, A10-A14 could very easily have a value of 15 or greater — A10-A14 are connected to the select lines; therefore, the TMS 9901 interval timer section can "think" it is out of clock mode and update the clock read register. Very simply, this means that a value cannot be locked into the clock read register by writing a one to CRU select bit zero (the control bit). The 9901 must be out of clock mode for at least one timer period to ensure that the contents of the clock read register has been updated. This means that to read the most recent contents of the decrementer, just before reading, the TMS 9901 *must* not be in the clock mode. The only sure way to manipulate clock mode is to use the control bit (select bit zero). When clock mode is reentered to access the clock read register, updating of the read register will cease. This is done so that the contents of the clock read register will not change while it is being accessed.

2.6 Power-Up Considerations

During hardware reset, $\overline{RST}$1 must be active (LOW) for a minimum of two clock cycles to force the TMS 9901 into a known state. $\overline{RST}$1 will disable all interrupts, disable the clock, program all I/O ports to the input mode, and force IC0-IC3 to all zeros with $\overline{INTREQ}$ held HIGH. The system software must enable the appropriate interrupts, program the clock, and configure the I/O ports as required. After initial power-up the TMS 9901 is accessed only as needed to service the clock, enable (disable) interrupts, or read (write) data to the I/O ports. The I/O ports can be reconfigured by use of the $\overline{RST}$2 software reset command bit.

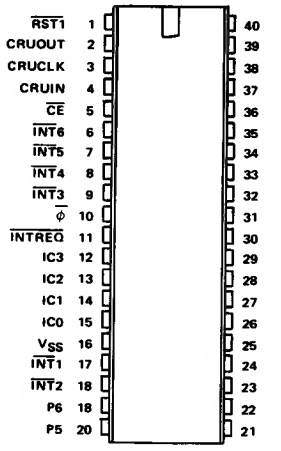
8◂

2.7 Pin Descriptions

Table 4 defines the TMS 9901 pin assignments and describes the function of each pin.

TABLE 4
TMS 9901 PIN ASSIGNMENTS AND FUNCTIONS

SIGNATURE	PIN	I/O	DESCRIPTION
$\overline{\text{INTREQ}}$	11	OUT	INTERRUPT Request. When active (low) $\overline{\text{INTREQ}}$ indicates that an enabled interrupt has been received. $\overline{\text{INTREQ}}$ will stay active until all enabled interrupt inputs are removed.
IC0 (MSB)	15	OUT	Interrupt Code lines. IC0-IC3 output the binary code corresponding to the highest priority enabled interrupt. If no enabled interrupts are active IC0—IC3 = (1,1,1,1).
IC1	14	OUT	
IC2	13	OUT	
IC3 (LSB)	12	OUT	
$\overline{\text{CE}}$	5	IN	Chip Enable. When active (low) data may be transferred through the CRU interface to the CPU. $\overline{\text{CE}}$ has no effect on the interrupt control section.
S0	39	IN	Address select lines. The data bit being accessed by the CRU interface is specified by the 5-bit code appearing on S0-S4.
S1	36	IN	
S2	35	IN	
S3	25	IN	
S4	24	IN	
CRUIN	4	OUT	CRU data in (to CPU). Data specified by S0-S4 is transmitted to the CPU by CRUIN. When $\overline{\text{CE}}$ is not active CRUIN is in a high-impedance state.
CRUOUT	2	IN	CRU data out (from CPU). When $\overline{\text{CE}}$ is active, data present on the CRUOUT input will be sampled during CRUCLK and written into the command bit specified by S0-S4.
CRUCLK	3	IN	CRU Clock (from CPU). CRUCLK specifies that valid data is present on the CRUOUT line.
$\overline{\text{RST1}}$	1	IN	Power Up Reset. When active (low) $\overline{\text{RST1}}$ resets all interrupt masks to "0", resets IC0 – IC3 = (0, 0, 0, 0), $\overline{\text{INTERQ}}$ = 1 disables the clock, and programs all I/O ports to inputs. $\overline{\text{RST1}}$ has a Schmitt-triger input to allow implementation with an RC circuit as shown in Figure 7.
V_{CC}	40		Supply Voltage. +5 V nominal.
V_{SS}	16		Ground Reference
$\overline{\phi}$	10	IN	System clock ($\overline{\phi}3$ in TMS 9900 system, $\overline{\text{CKOUT}}$ in TMS 9980 system).
$\overline{\text{INT1}}$	17	IN	Group 1, interrupt inputs. When active (Low) the signal is ANDed with its corresponding mask bit and if enabled sent to the interrupt control section. $\overline{\text{INT1}}$ has highest priority.
$\overline{\text{INT2}}$	18	IN	
$\overline{\text{INT3}}$	9	IN	
$\overline{\text{INT4}}$	8	IN	
$\overline{\text{INT5}}$	7	IN	
$\overline{\text{INT6}}$	6	IN	
$\overline{\text{INT7}}$/ P15	34	I/O	Group 2, programmable interrupt (active low) or I/O pins (true logic). Each pin is individually programmable as an interrupt, an input port, or an output port.
$\overline{\text{INT8}}$/ P14	33	I/O	
$\overline{\text{INT9}}$/ P13	32	I/O	
$\overline{\text{INT10}}$/P12	31	I/O	
$\overline{\text{INT11}}$/P11	30	I/O	
$\overline{\text{INT12}}$/P10	29	I/O	
$\overline{\text{INT13}}$/P9	28	I/O	
$\overline{\text{INT14}}$/P8	27	I/O	
$\overline{\text{INT15}}$/P7	23	I/O	
P0	38	I/O	Group 3, I/O ports (true logic). Each pin is individually programmable as an input port or an output port.
P1	37	I/O	
P2	26	I/O	
P3	22	I/O	
P4	21	I/O	
P5	20	I/O	
P6	19	I/O	



3. APPLICATIONS

3.1 Hardware Interface

Figure 7 illustrates the use of a TMS 9901 PSI in a TMS 9900 system. The TIM 9904 clock generator/driver syncs the $\overline{\text{RESET}}$ for both the TMS 9901 and the CPU. The RC circuit on the TIM 9904 provides the power-up and pushbutton $\overline{\text{RESET}}$ input to the clock chip. Address lines A0-A9 are decoded on CRU cycles to select the TMS 9901. Address lines A10-A14 are sent directly to PSI select lines S0-S4, respectively, to select which TMS 9901 CRU bit is to be accessed.

Figure 8 illustrates the use of a TMS 9901 with a TMS 9985 CPU. No TIM 9904 is needed with the TMS 9985, so the reset circuitry is connected directly to the system reset line. The clock ($\overline{\phi}$) then comes from the TMS 9985. All other circuitry is identical to the TMS 9900 system.

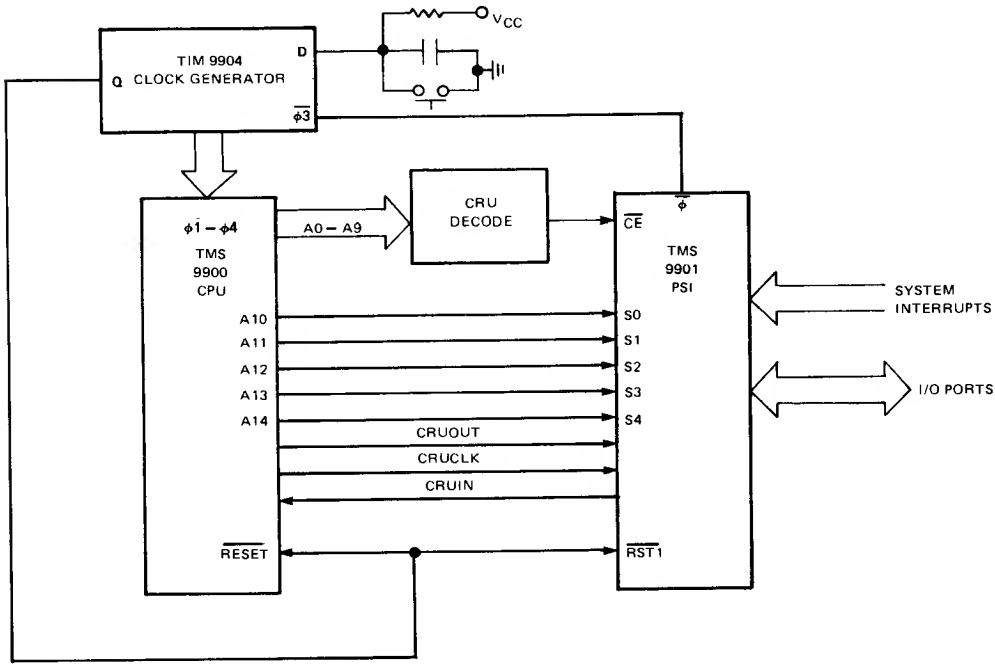


FIGURE 7—TMS 9900/TMS 9901 INTERFACE

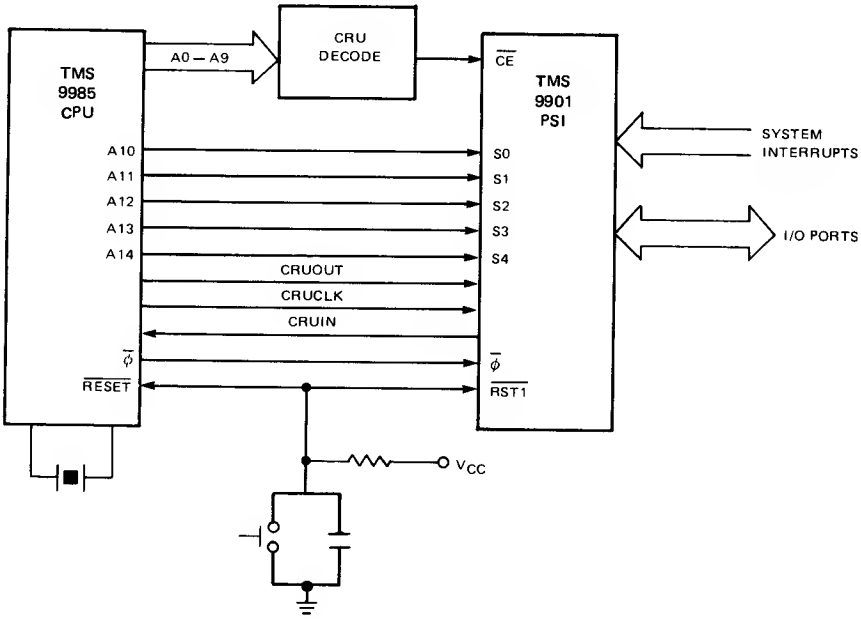


FIGURE 8–TMS 9985/TMS 9901 INTERFACE

3.2 Software Interface

Figure 9 lists the TMS 9900 code needed to control the TMS 9901 PSI. The code initializes the PSI to an eight-bit input port, an eight-bit output port, and enables interrupt levels 1-6. The six dedicated interrupt pins are all used for interrupts; their mask bits are set ON. The nine programmable pins are all used as I/O ports; mask bits 7-15 remain reset. P0-P7 are programmed as an eight-bit output port, and P8-P15 are programmed as an eight-bit input port.

8 Some code is added to read the contents of the clock read-register. The SBZ instruction takes the TMS 9901 out of clock mode long enough for the clock read register to be updated with the most recent decrementer value. When clock mode is reentered, the decrementer will cease updating the clock read-register so that the contents of the register will not be changing during a read operation.

The second section of code is typical code found in a clock interrupt service routine. All interrupts initially are disabled by the routine. These functions are not necessary, but are usually done to ensure system integrity. The interrupt mask should be restored as soon as the sensitive processing is complete. The interrupt is counted in the variable COUNT and is then cleared by writing a one to mask bit 3. If a zero is written to mask bit 3 to clear the interrupt, clock interrupt will be disabled from that point onward, but the clock will continue to run.

ASSUMPTION:

- System uses clock at maximum interval (349 msec @ 3MHz)
- Interrupts 1-6 are used
- Eight bits are used as an output port , P0 –P7
- Eight bits are used as an input port , P8 – P15
- $\overline{\text{RST}}$1 (power-up reset) has been applied
- The most significant byte of R1 contains data to be output.

```
        LI      R12, PSIBAS      Set up CRU base to point to 9901
        LDCR    @CLKSET, 0       16-bit transfer, set clock to max interval

        LDCR    @INTSET, 7       Enter interrupt mode and enable interrupts 1 – 6

        LI      R12, PSIBAS+32   Set CRU base to I/O ports – output
        LDCR    R1, 8            Output byte from R1, program ports 0 – 7 as output

        LI      R12, PSIBAS+48   Set CRU base to I/O ports – input
        STCR    R2, 8            Store a byte from input port into MS8T of R2

        LI      R12, PSIBAS      Set CRU base to 9901
        S8Z     0                Leave clock mode so decremented contents can be latched
        INCT    R12              Set CRU base to clock read register
        SBO     -1               Enter clock mode
        STCR    R3, 14           Read 14-bit clock read register contents into R3

CLKSET  DATA    >FFFF
INTSET  BYTE    >7E

CLKINT  $                        Clock interrupt service routine – level 3
        LIMI    0                Disable interrupts at CPU
        INC     @COUNT           Count the clock interrupt
        LI      R12, PSIBAS      Set CRU base to point to 9901
        SBZ     0                Enter interrupt mode
        SBO     3                Clear clock interrupt
```

FIGURE 9 – TMS 9900 SAMPLE SOFTWARE TO CONTROL THE TMS 9901

3.3 Interval Timer Application

A TM 990/100M microcomputer board application in which every 10 seconds a specific task must be performed is described below. The TMS 9901 clock is set to interrupt every 333.33 milliseconds. This is accomplished by programming the 14-bit clock register to $3D09_{16}$ ($15{,}625_{10}$). The TM 990/100M microcomputer board system clock runs at 3 MHz, giving a clock resolution of 21.33 microseconds. A decrementer period of 21.33 microseconds multiplied by 15,625 periods until interrupt gives 333.33 milliseconds between interrupts. The interrupt service routine must count 30 interrupts before 10 seconds elapses:

$$f_{(DEC)} = \frac{f(\phi)}{64}, \qquad T_{(DEC)} = \frac{1}{f_{(DEC)}} = \frac{64}{3{,}000{,}000} = 21.3333\ \mu s$$

Figure 10 is a flowchart of the software required to perform the above application, and Figure 11 is a listing of the code. Following the flowchart, the main routine sets up all initial conditions for the 9901 and clock service routine. The interrupt service routine decrements a counter in R2 which was initialized to 30. When the counter in R2 decrements to zero, 10 seconds have elapsed, and the work portion of the service routine is entered. Note carefully that the work portion of the service routine takes longer than 333.33 ms which is the time between clock interrupts from the 9901. Therefore, recursive interrupts are going to occur and some facility must be provided to handle them. Loading a new workspace pointer and transferring the saved WP, PC, and ST (R13-R15) from the interrupt workspace to the new workspace allows one level of recursion.

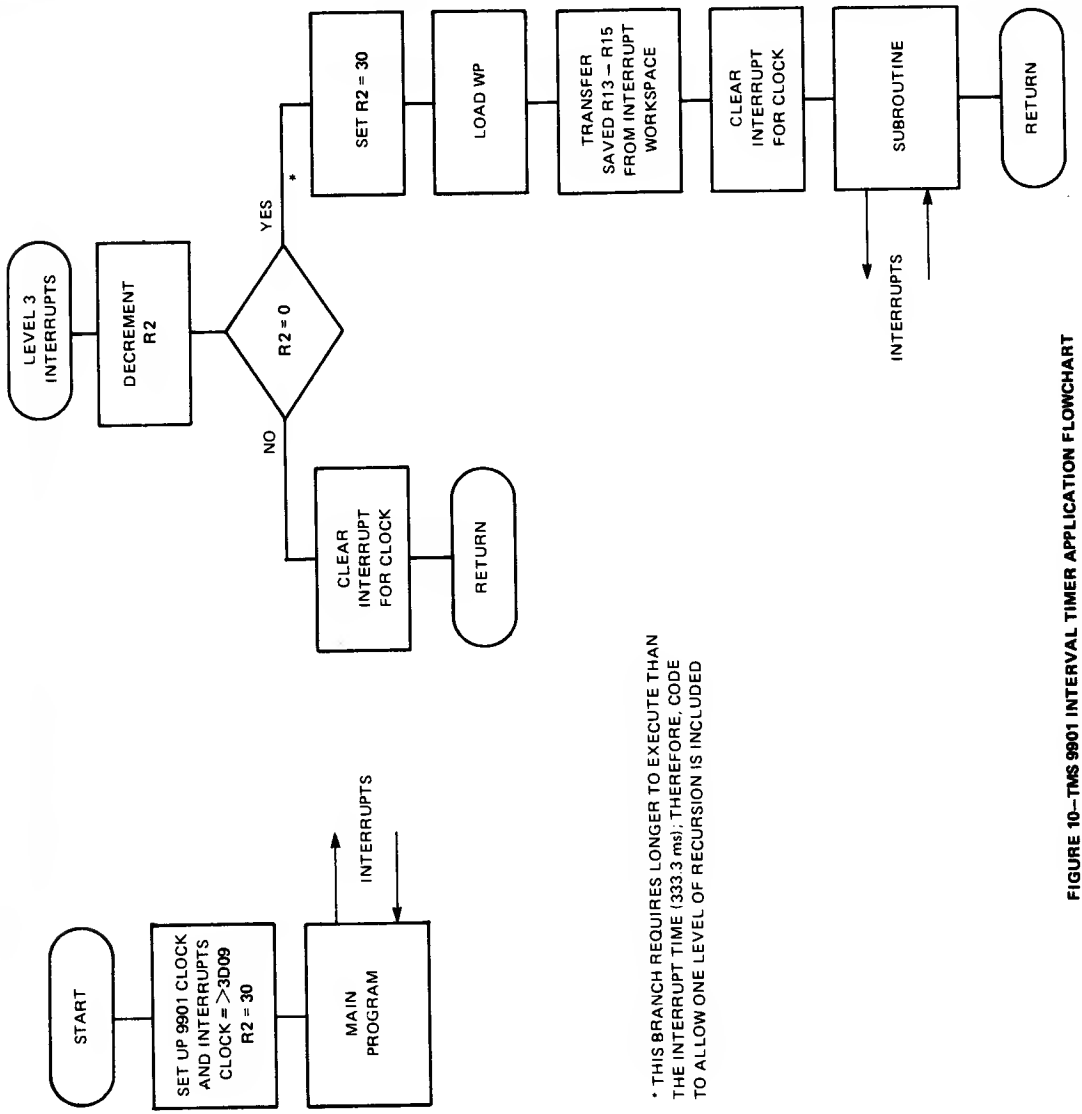


FIGURE 10–TMS 9901 INTERVAL TIMER APPLICATION FLOWCHART

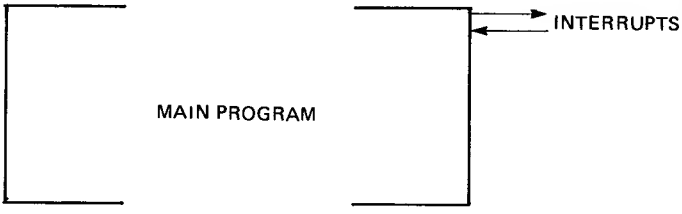
8◀

DEVICE INITIALIZATION

```
FE00 02E0 LWPI >FF20
FE02 FF20
FE04 020C LI R12,>100     9901 CRU BASE ADDRESS
FE06 0100
FE08 02E0 LWPI >FF68      INTERRUPT 3 WORKSPACE
FE0A FF68
FE0C 0201 LI R1,>7A13     DATA FOR 333.33MS CLOCK
FE0E 7A13
FE10 0202 LI R2,30        30 X 333.33MS = 10SEC
FE12 001E
FE14 020C LI R12,>100     9901 CRU BASE ADDRESS
FE16 0100
FE18 33C1 LDCR R1,15      LOAD 9901 CLOCK
FE1A 1E00 SBZ 0           SET 9901 TO INTERRUPT MODE
FE1C 1D03 SBO 3           UNMASK INTERRUPT 3
```

MAIN PROGRAM

```
FD00 02E0 LWPI >FF00      MAIN PROGRAM WORKSPACE
FD02 FF00
FD04 0300 LIMI 3          ENABLE INT 0-3
FD06 0003
```



▸8

NOTE: This code was assembled using the TM 990/402 line-by-line essembler.

FIGURE 11—INTERVAL TIMER

INTERRUPT 3 SERVICE ROUTINE
(WP = FF68)

```
FD80 0602 DEC R2          COUNT DOWN 30 IN R2
FD82 1302 JEQ >FD88       IF ZERO THEN JUMP
FD84 1D03 SBO 3           CLEAR 9901 CLOCK INTERRUPT
FD86 0380 RTWP            RETURN TO INTERRUPTED ROUTINE
FD88 0202 LI R2,30        RELOAD R2 FOR 10 SEC COUNT DOWN
FD8A 001E
FD8C 0460 B @>FC80        BRANCH TO SUBROUTINE
FD8E FC80
```

ROUTINE TO BE PERFORMED EVERY 10 SECONDS, IT TAKES
LONGER THAN 333.33 MS WHICH IS 9901 CLOCK PERIOD'

```
FC80 02E0 LWPI >FF20      WORKSPACE FOR SUBROUTINE
FC82 FF20
FC84 C360 MOV @>FF82,R13  TRANSFER SAVED WP,PC,ST FROM
FC86 FF82
FC88 C3A0 MOV @>FF84,R14  INT 3 WORKSPACE
FC8A FF84
FC8C C3E0 MOV @>FF86,R15
FC8E FF86
FC90 1D03 SBO 3           CLEAR 9901 CLOCK INTERRUPT
FC92 0300 LIMI 3          ENABLE INT 0-3
FC94 0003
```

```
FSC0        RTWP
```

FIGURE 11—(CONCLUDED)

4. TMS 9901 ELECTRICAL SPECIFICATIONS

4.1 Absolute Maximum Ratings Over Operating Free Air Temperature Range (Unless Otherwise Noted)*

Supply voltage, V_{CC} ..... −0.3 V to 10 V
All inputs and output voltages ..... −0.3 V to 10 V
Continuous power dissipation ..... 0.85 W
Operating free-air temperature range ..... 0°C to 70°C
Storage temperature range ..... −65°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to tha device. This is a stress rating only and functional operation of tha device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for axtended periods may affect device reliability.

4.2 Recommended Operating Conditions*

PARAMETER	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}	4.75	5.0	5.25	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH}	2.0		V_{CC}	V
Low-level input voltage, V_{IL}	V_{SS}−.3		0.8	V
Operating free-air temperature, T_A	0		70	°C

4.3 Electrical Characteristics Over Full Range of Recommended Operating Conditions (Unless Otherwise Noted)*

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
V_{OH}	High level output voltage	$I_{OH} = -100\ \mu A$	2.4		V_{CC}	V
		$I_{OH} = -200\ \mu A$	2.2		V_{CC}	V
V_{OL}	Low level output voltage	$I_{OL} = 3.2$ mA	V_{SS}		0.4	V
I_I	Input current (any input)	$V_I = 0$ V to V_{CC}			±100	μA
$I_{CC(av)}$	Average supply current from V_{CC}	$t_{c(\phi)} = 330$ ns, $T_A = 70°C$			150	mA
C_I	Small signal input capacitance, any input	f = 1 MHz			15	pF

4.4 Timing Requirements Over Full Range of Operating Conditions

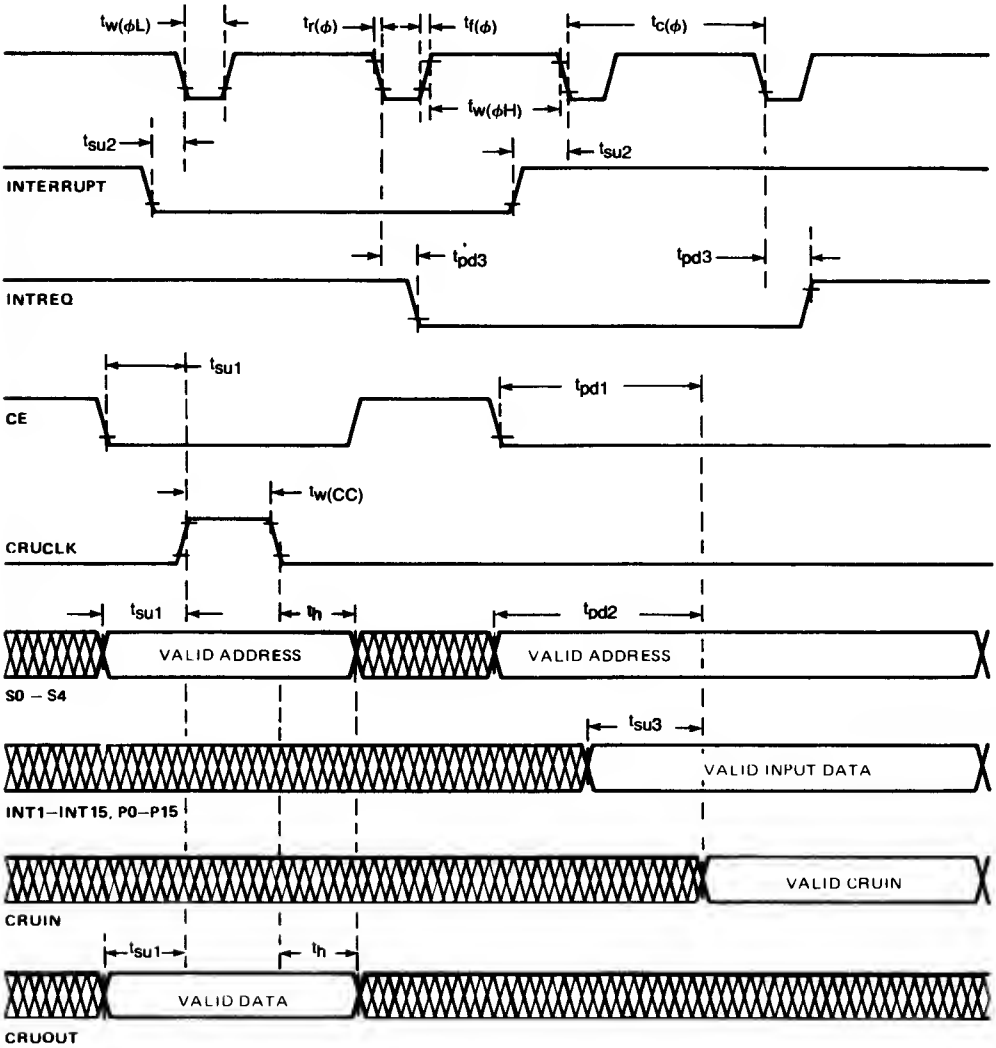
PARAMETER		MIN	TYP	MAX	UNIT
$t_{c(\phi)}$	Clock cycle time	300	333	2000	ns
$t_{r(\phi)}$	Clock rise time	5		40	ns
$t_{f(\phi)}$	Clock fall time	10		40	ns
$t_{w(\phi H)}$	Clock pulse width (high level)	225			ns
$t_{w(\phi L)}$	Clock pulse width (low level)	45		300	ns
$t_{w(CC)}$	CRUCLK pulse width	100	185		ns
t_{su1}	Setup time for $\overline{CE}$, S0-S4, or CRUOUT before CRUCLK	100			ns
t_{su2}	Setup time for interrupt before $\overline{\phi}$ low	60			ns
t_{su3}	Setup time for inputs before valid CRUIN	200			ns
t_h	Hold time for $\overline{CE}$, S0-S4, or CRUOUT after CRUCLK	60			ns

*NOTE: All voltage values are referenced to V_{SS}.

▸8

4.5 Switching Characteristics Over Full Range of Recommended Operating Conditions

PARAMETER		TEST CONDITION	MIN	TYP	MAX	UNIT
t_{pd1}	Propagation delay, $\overline{CE}$ to valid CRUIN	C_L = 100 pF			300	ns
t_{pd2}	Propagation delay, S0-S4 to valid CRUIN	C_L = 100 pF			320	ns
t_{pd3}	Propagation delay, $\overline{\phi}$ low to valid $\overline{INTREQ}$, IC0-IC3	C_L = 100 pF			110	ns
t_{pd}	Propagation delay, $\overline{CRUCLK}$ to valid data out (P0-P15)	C_L = 100 pF			300	ns



NOTE 1: ALL TIMING MEASUREMENTS ARE FROM 10% and 90% POINTS.

FIGURE 12–SWITCHING CHARACTERISTICS

5. TMS 9901-40 ELECTRICAL SPECIFICATIONS

5.1 ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Supply voltages, V_{CC} . . . . . −0.3 V to 10 V
All input and output voltages . . . . . −0.3 V to 10 V
Continuous power dissipation . . . . . 0.90 W
Operating free-air temperature range . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended period may affect device reliability.

5.2 RECOMMENDED OPERATING CONDITIONS

	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply voltage, V_{SS}		0		V
High-input voltage, V_{IH}	2.0	2.4	V_{CC}	V
Low-level input voltage, V_{IL}	V_{SS}-.3	0.4	0.8	V
Operating free-air temperature, T_A	0		70	°C

5.3 ELECTRICAL CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (UNLESS OTHERWISE NOTED)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
V_{OH}	High level output voltage	$I_{OH} = -100\mu A$	2.4		V_{CC}	V
		$I_{OH} = -200\mu A$	2.2		V_{CC}	V
V_{OL}	Low level output voltage	$I_{OL} = 3.2$ mA	V_{SS}		0.4	μA
I_I	Input current (any input)	$V_I = 0$ V to V_{CC}			± 100	V
$I_{CC(av)}$	Average supply current from V_{CC}	$t_c(\phi) = 330$ ns, $T_A = 25°C$			150	mA
C_I	Small Signal Input Capacitance, any input	f = 1 MHz			15	mF

5.4 TIMING REQUIREMENTS OVER FULL RANGE OF OPERATING CONDITIONS

PARAMETER		MIN	NOM	MAX	UNIT
$t_c(\phi)$	Clock cycle time	240	250	667	ns
$t_r(\phi)$	Clock rise time	5		40	ns
$t_f(\phi)$	Clock fall time	10		40	ns
$t_w(\phi L)$	Clock pulse width (low level)	40		300	ns
$t_w(\phi H)$	Clock pulse width (high level)	180			ns
$t_w(CC)$	CRUCLK pulse width	80	125		ns
t_{su1}	Setup time for S0-S4, $\overline{CE}$, or CRUOUT before CRUCLK	80	80		ns
t_{su2}	Setup time, interrupt before $\overline{\phi}$ low	50	50		ns
t_{su3}	Setup time for inputs before valid CRUIN	180	180		ns
t_h	Hold time for $\overline{CE}$, S0-S4, or CRUOUT after CRUCLK	50	50		ns

▶8

DESIGN GOAL

This document describes the design specifications for a product under development. Texas Instruments reserves the right to change these specifications in any manner, without notice.

5.5 SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PD1}	propagation delay, $\overline{CE}$ to Valid CRUIN	$C_L = 100pF$		220	220	ns
t_{PD2}	propagation delay, S0-S4 to Valid CRUIN			240	240	ns
t_{PD3}	propagation delay, ϕ low to Valid $\overline{INTREQ}$, IC0-IC3			80	80	ns
t_{PD}	propagation delay, CRUCLK to Valid Data Out (P0-P15)			200	200	ns

8◀

DESIGN GOAL

This document describes the design specifications for a product under development. Texas Instruments reserves the right to change these specifications in any manner, without notice.

1. INTRODUCTION

1.1 DESCRIPTION

The TMS 9902 Asynchronous Communications Controller (ACC) is a peripheral device designed for use with the Texas Instruments 9900 family of microprocessors. The TMS 9902 is fabricated using N-channel, silicon gate, MOS technology. The TMS 9902 is TTL-compatible on all inputs and outputs, including the power supply (+5 V) and single-phase clock. The TMS 9902 ACC provides an interface between a microprocessor and a serial, asynchronous, communications channel. The ACC performs the timing and data serialization and deserialization functions, facilitating microprocessor control of the asynchronous channel. The TMS 9902 ACC accepts *EIA Standard RS-232-C* protocol.

1.2 KEY FEATURES

- Low Cost, Serial, Asynchronous Interface
- Programmable, Five- to Eight-Bit, I/O Character Length
- Programmable 1, 1½, and 2 Stop Bits
- Even, Odd, or No Parity
- Fully Programmable, Data Rate Generation
- Interval Timer with Resolution from 64 to 16,320 Microseconds
- TTL-Compatibility, Including Power Supply
- Standard 18-Pin Plastic or Ceramic Package
- N-Channel, Silicon Gate Technology

1.3 TYPICAL APPLICATION

Figure 1 shows a general block diagram of a system incorporating a TMS 9902 ACC. Following is a tutorial discussion of this application. Subsequent sections of this Data Manual detail most aspects of TMS 9902 use.

The TMS 9902 interfaces with the CPU through the *communications register unit* (CRU). The CRU interface consists of five address select lines (S0-S4), chip enable ($\overline{CE}$), and three CRU lines (CRUIN, CRUOUT, CRUCLK). An additional input to the CPU is the ACC interrupt line ($\overline{INT}$). The TMS 9902 occupies 32 bits of CRU space; each of the 32 bits are selected individually by processor address lines A10-A14 which are connected to the ACC select lines S0-S4, respectively. Chip enable ($\overline{CE}$) is generated by decoding address lines A0-A9 for CRU cycles. Under certain conditions the TMS 9902 causes interrupts. The interrupt logic shown in Figure 1 can be a TMS 9901.

The ACC interfaces to the asynchronous communications channel on five lines: request to send ($\overline{RTS}$), data set ready ($\overline{DSR}$), clear to send ($\overline{CTS}$), serial transmit data (XOUT), and serial receive data (RIN). The request to send ($\overline{RTS}$) goes active (LOW) whenever the transmitter is activated. However, before data transmission begins, the clear to send ($\overline{CTS}$) input must be active. The data set ready ($\overline{DSR}$) input does not affect the receiver or transmitter. When $\overline{DSR}$ or $\overline{CTS}$ changes level, an interrupt is generated.

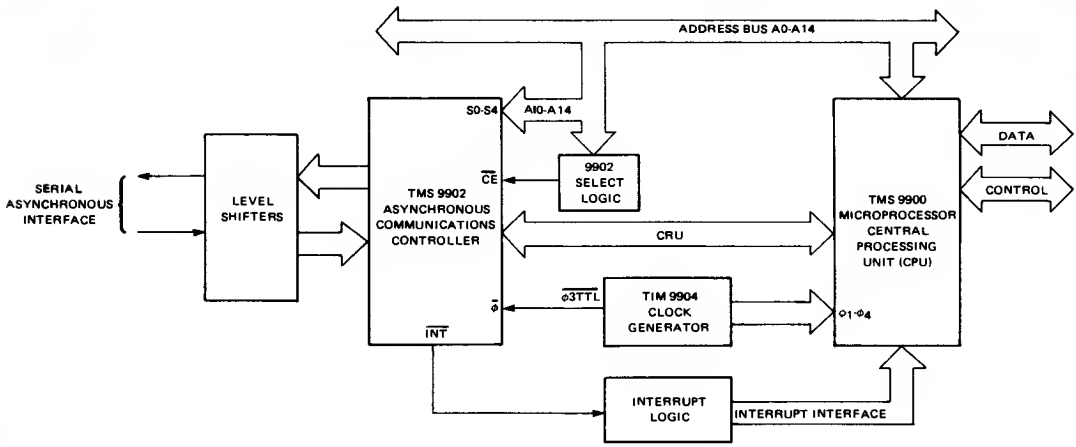


FIGURE 1. TYPICAL APPLICATION, TMS 9902 ASYNCHRONOUS COMMUNICATION CONTROLLER (ACC)

2. ARCHITECTURE

The TMS 9902 asynchronous communications controller (ACC) is designed to provide a low cost, serial, asynchronous interface to the 9900 family of microprocessors. The TMS 9902 ACC is diagrammed in Figure 2. The ACC has five main subsections: CRU interface, transmitter section, receiver section, interval timer, and interrupt section.

2.1 CRU INTERFACE

The communications register unit (CRU) is the means by which the CPU communicates with the TMS 9902 ACC. The ACC occupies 32 bits of CRU read and write space. Figure 3 illustrates the CRU interface between a TMS 9902 and a TMS 9900 CPU; Figure 4 illustrates the CRU Interface for a TMS 9980A or 9981 CPU. The CRU lines are tied directly to each other as shown in Figures 3 and 4. The least significant bits of the address bus are connected to the select lines. In a TMS 9900 CPU system A14-A10 are connected to S4-S0 respectively. The most significant address bits are decoded to select the TMS 9902 via the chip enable ($\overline{CE}$) signal. When $\overline{CE}$ is inactive (HIGH), the CRU interface of the 9901 is disabled.

NOTE

When $\overline{CE}$ is inactive (HIGH) the 9902 sets its CRUIN pin to high impedance and disables CRUCLK from coming on chip. This means the CRUIN line can be used as an OR-tied bus. The 9902 is still able to see the select lines even when $\overline{CE}$ is high.

For those unfamiliar with the CRU concept, the following is a discussion of how to build a CRU interface. The CRU is a bit addressable (4096 bits), synchronous, serial interface over which a single instruction can transfer between one and 16 bits serially. Each one of the 4096 bits of the CRU space has a unique address and can be read and written to. During multi-bit CRU transfers, the CRU address is incremented at the beginning of each CRU cycle to point to the next consecutive CRU bit.

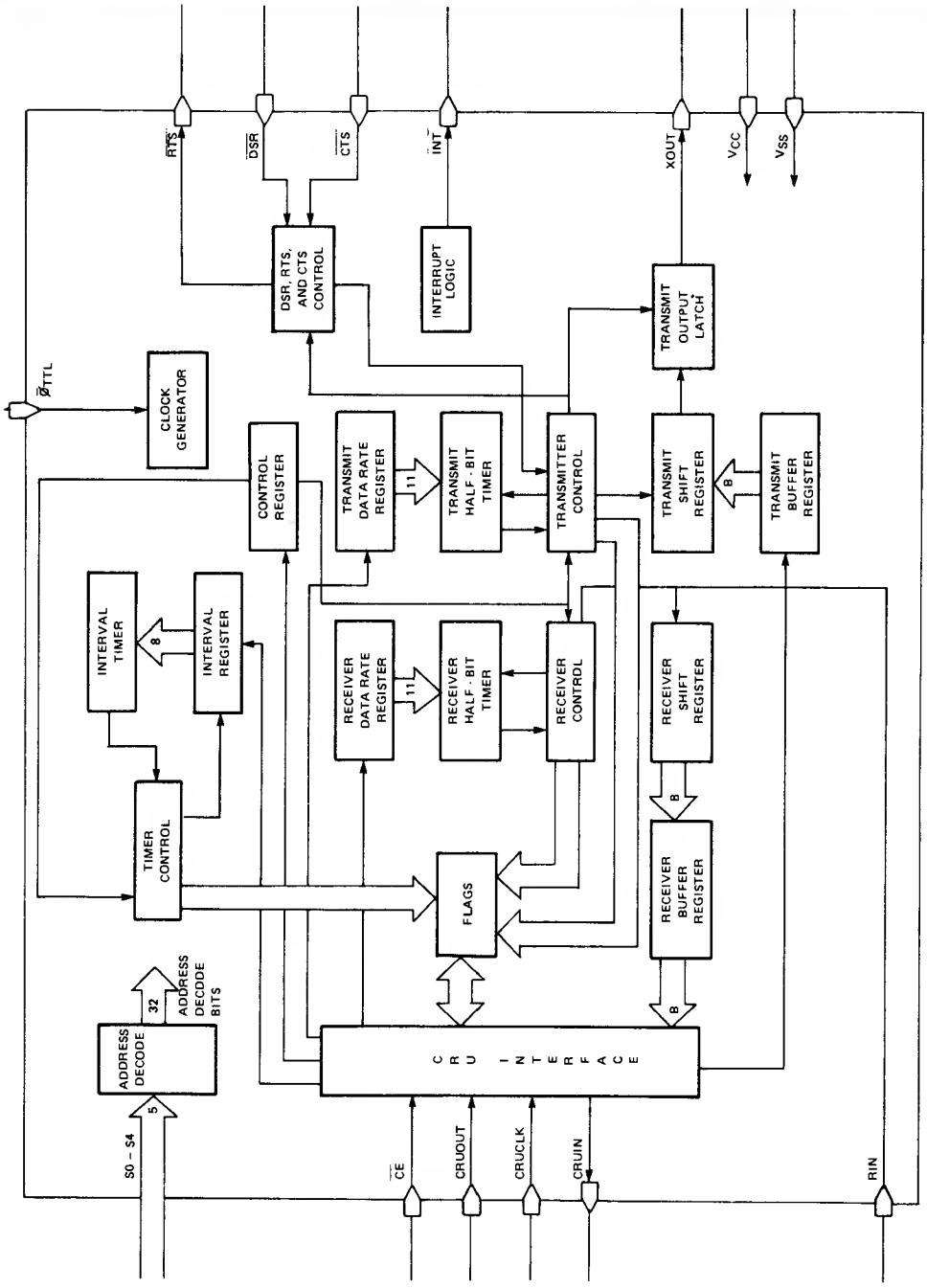


FIGURE 2. TMS 9902 ASYNCHRONOUS COMMUNICATIONS CONTROLLER (ACC) BLOCK DIAGRAM

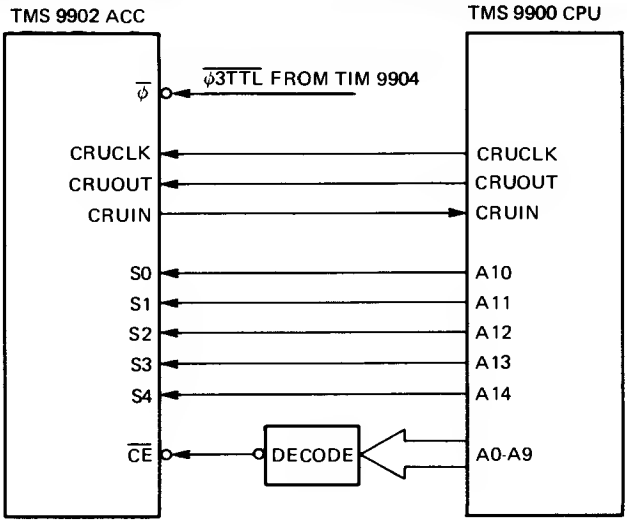


FIGURE 3. TMS 9902 – TMS 9900 CRU INTERFACE

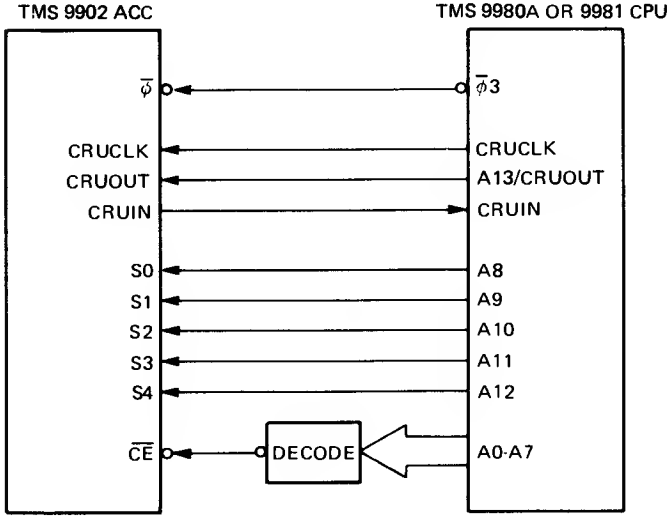


FIGURE 4. TMS 9902 – TMS 9980A OR 9981 CRU INTERFACE

When a 9900 CPU executes a CRU Instruction, the processor uses the contents of workspace register 12 as a base address. (Refer to the 9900 Microprocessor Data Manual for a complete discussion on how CRU addresses are derived.) The CRU address is brought out on the 15-bit address bus; this means that the least significant bit of R12 is not brought out of the CPU. During CRU cycles, the memory control lines ($\overline{MEMEN}$, $\overline{WE}$, and DBIN) are all inactive; $\overline{MEMEN}$ being inactive (HIGH) indicates the address is not a memory address and therefore is a CRU address or external instruction code. Also, when $\overline{MEMEN}$ is inactive (HIGH) and a valid address is present, address bits A0-A2 must all be zero to constitute a valid CRU address; if address bits A0-A2 are other than all zeros, they are indicating an external instruction code. In summary, address bits A3-A14 contain the CRU address to be decoded, address bits A0-A2 must be zero and $\overline{MEMEN}$ must be inactive (HIGH) to indicate a CRU cycle.

2.1.1 CPU OUTPUT FOR CRU

The TMS 9902 ACC occupies 32 bits of output CRU space, of which 23 bits are used: 31 and 21-0. These 23 bits are employed by the CPU to communicate command and control information to the TMS 9902. Table 1 shows the mapping between CRU address select (S lines) and ACC functions. Each CRU addressable output bit on the TMS 9902 is described in detail following Table 1.

TABLE 1
TMS 9902 ACC OUTPUT BIT ADDRESS ASSIGNMENTS

ADDRESS$_2$ S0	S1	S2	S3	S4	ADORESS$_{10}$	NAME	DESCRIPTION
1	1	1	1	1	31	RESET	Reset device.
					30-22		Not used.
1	0	1	0	1	21	DSCENB	Data Set Status Change Interrupt Enable.
1	0	1	0	0	20	TIMENB	Timer Interrupt Enable
1	0	0	1	1	19	XBIENB	Transmitter Interrupt Enable
1	0	0	1	0	18	RIENB	Receiver Interrupt Enable
1	0	0	0	1	17	BRKON	Break On
1	0	0	0	0	16	RTSON	Request to Send On
0	1	1	1	1	15	TSTMD	Test Mode
0	1	1	1	0	14	LDCTRL	Load Control Register
0	1	1	0	1	13	LDIR	Load Interval Register
0	1	1	0	0	12	LRDR	Load Receiver Data Rate Register
0	1	0	1	1	11	LXDR	Load Transmit Data Rate Register
					10-0		Control, Interval, Receive Data Rate, Transmit Data Rate, and Transmit Buffer Registers

▸8

Bit 31 (RESET) — **Reset.** Writing a one or zero to bit 31 causes the device to reset, consequently disabling all interrupts, initializing the transmitter and receiver, setting $\overline{RTS}$ inactive (HIGH), setting all register load control flags (LDCTRL, LDIR, LRDR, and LXDR) to a logic one level, and resetting the BREAK flag. No other input or output operations should be performed for 11 $\overline{\phi}$ clock cycles after issuing the RESET command.

Bit 30-Bit 22 — Not used.

INTERRUPT ENABLE	SELECT BIT	INTERRUPT FLAG	INTERRUPT ENABLED
DSCENB	21	DSCH	DSCINT
TIMENB	20	TIMELP	TIMINT
XIENB	19	XBRE	XINT
RIENB	18	RBRL	RINT

Bit 21 (DSCENB) — **Data Set Change Interrupt Enable.** Writing a one to bit 21 causes the $\overline{INT}$ output to be active (LOW) whenever DSCH (Data Set Status Change) is a logic one. Writing a zero to bit 21 causes DSCH interrupts to be disabled. Writing either a one or zero to bit 21 causes DSCH to reset. (Refer also to Section 2.5).

Bit 20 (TIMENB) — **Timer Interrupt Enable.** Writing a one to bit 20 causes the $\overline{INT}$ output to be active whenever TIMELP (Timer Elapsed) is a logic one. Writing a zero to bit 20 causes TIMELP interrupts to be disabled. Writing either a one or zero to bit 20 causes TIMELP and TIMERR (Timer Error) to reset. (Refer also to Sections 2.4 and 2.5.)

Bit 19 (XBIENB) — **Transmit Buffer Interrupt Enable.** Writing a one to bit 19 causes the $\overline{INT}$ output to be active whenever XBRE (Transmit Buffer Register Empty) is a logic one. Writing a zero to bit 19 causes XBRE interrupts to be disabled. The state of XBRE is not affected by writing to bit 19. (Refer also to Sections 2.2 and 2.5.)

Bit 18 (RIENB) — **Receiver Interrupt Enable.** Writing a one to bit 18 causes the $\overline{INT}$ output to be active whenever RBRL (Receiver Buffer Register Loaded) is a logic one. Writing a zero to bit 18 disables RBRL interrupts. Writing either a one or zero to bit 18 causes RBRL to reset. (Refer also to Sections 2.3 and 2.5.)

Bit 17 (BRKON) — **Break On.** Writing a one to bit 17 causes the XOUT (Transmitter Serial Data Output) to go to a logic zero whenever the transmitter is active and the Transmit Buffer Register (XBR) and the Transmit Shift Register (XSR) are empty. While BRKON is set, loading of characters into the XBR is inhibited. Writing a zero to bit 17 causes BRKON to reset and the transmitter to resume normal operation.

Bit 16 (RTSON) — **Request To Send On.** Writing a one to bit 16 causes the $\overline{RTS}$ output to be active (LOW). Writing a zero to bit 16 causes $\overline{RTS}$ to go to a logic one after the XSR (Transmit Shift Register) and XBR (Transmit Buffer Register) are empty, and BRKON is reset. Thus, the $\overline{RTS}$ output does not become inactive (HIGH) until *after* character transmission is completed.

Bit 15 (TSTMD) — **Test Mode.** Writing a one to bit 15 causes $\overline{RTS}$ to be internally connected to $\overline{CTS}$, XOUT to be internally connected to RIN, $\overline{DSR}$ to be internally held LOW, and the Interval Timer to operate 32 times its normal rate. Writing a zero to bit 15 re-enables normal device operation. There seldom is reason to enter the test mode under normal circumstances, but this function is useful for diagnostic and inspection purposes.

Bits 14-11 — **Register Load Control Flags.** Output bits 14-11 control which of the five registers are loaded when writing to bits 10-0. The flags are prioritized as shown in Table 2.

TABLE 2
TMS 9902 ACC REGISTER LOAD SELECTION

REGISTER LOAD CONTROL FLAG STATUS				REGISTER ENABLED
LDCTRL	LDIR	LRDR	LXDR	
1	X	X	X	Control Register
0	1	X	X	Interval Register
0	0	1	X	Receive Data Rate Register *
0	0	X	1	Transmit Data Rate Register *
0	0	0	0	Transmit Buffer Register

*If both LRDR and LXDR bits are set, both registers are loaded, assuming LDCTRL and LDIR are disabled; if only one of these registers is to be loaded, only that register bit is set, and the other register bit reset.

Bit 14 (LDCTRL) — **Load Control Register.** Writing a one to bit 14 causes LDCTRL to be set to a logic one. When LDCTRL = 1, any data written to bits 0-7 is directed to the Control Register. Note that LDCTRL is also set to a logic one when a one or zero is written to bit 31 (RESET). Writing a zero to bit 14 causes LDCTRL to reset to a logic zero, disabling loading of the Control Register. LDCTRL is also automatically reset to logic zero when a datum is written to bit 7 of the Control Register, reset normally occurs as the last bit is written when loading the Control Register with a LDCR instruction.

Bit 13 (LDIR) — **Load Interval Register.** Writing a one to bit 13 causes LDIR to set to a logic one. When LDIR = 1 and LDCTRL = 0, any data written to bits 0-7 is directed to the Interval Register. Note that LDIR is also set to a logic one when a datum is written to bit 31 (RESET); however, Interval Register loading is not enabled until LDCTRL is set to a logic zero. Writing a zero to bit 13 causes LDIR to be reset to logic zero, disabling loading of the Interval Register. LDIR is also automatically reset to logic zero when a datum is written to bit 7 of the Interval Register; reset normally occurs as the last bit is written when loading the Interval Register with a LDCR instruction.

Bit 12 (LRDR) — **Load Receive Data Rate Register.** Writing a one to bit 12 causes LRDR to set to a logic one. When LRDR = 1, LDIR = 0, and LDCTRL = 0, any data written to bits 0-10 is directed to the Receive Data Rate Register. Note that LRDR is also set to a logic one when a datum is written to bit 31 (RESET); however, Receive Data Rate Register loading is not enabled until LDCTRL and LDIR are set to a logic zero. Writing a zero bit to 12 causes LRDR to reset to a logic zero, disabling loading of the Receive Data Rate Register. LRDR is also automatically reset to logic zero when a datum is written to bit 10 of the Receive Data Rate Register; reset normally occurs as the last bit is written when loading the Receive Data Rate Register with a LDCR instruction.

Bit 11 (LXDR) — **Load Transmit Data Rate Register.** Writing a one to bit 11 causes LXDR to set to a logic one. When LXDR = 1, LDIR = 0, and LDCTRL = 0, any data written to bits 0-10 is directed to the Transmit Data Rate Register. Note that loading of both the Receive and Transmit Data Rate Registers is enabled when LDCTRL = 0, LDIR = 0, LRDR = 1, and LXDR = 1; thus these two registers may be loaded simultaneously when data is received and transmitted at the same rate. LXDR is also set to a logic one when a datum is written to bit 31 (RESET); however, Transmit Data Rate Register loading is not enabled until LDCTRL and LDIR are to logic zero. Writing a zero to bit 11 causes LXDR to reset to logic zero, consequently disabling loading of the Transmit Data Rate Register. Since bit 11 is the next bit addressed after loading the Transmit Data Rate Register, the register may be loaded and the LXDR flag reset with a single LDCR instruction where 12 bits (Bits 0-11) are written and a zero is written to Bit 11.

Bits 14-11 (All Zeros) — **Load Transmit Buffer Register.** See Section 2.1.2.5.

Bits 10-0 (Data) — **Data.** Information written to bits 10-0 is loaded into the controlling registers as indicated by LDCTRL, LDIR, LRDR, and LXDR (see Table 2). The different register bits are described in Section 2.1.2 below.

2.1.2 REGISTERS

2.1.2.1 Control Register

The Control Register is loaded to select character length, device clock operation, parity, and the number of stop bits for the transmitter; control register loading occurs when LDCTRL is active (see Table 2). Table 3 shows the bit address assignments for the Control Register.

TABLE 3
CONTROL REGISTER BIT ADDRESS ASSIGNMENTS

$ADDRESS_{10}$	NAME	DESCRIPTION
7	SBS1	← Stop Bit Select
6	SBS2	
5	PENB	Parity Enable
4	PODD	Odd Parity Select
3	CLK4M	$\overline{\phi}$ Input Divide Select
2	—	Not Used
1	RCL1	← Character Length Select
0	RCL0	

7	6	5	4	3	2	1	0
SBSI	SBS2	PENB	PODD	CLK4M	NOT USED	RCL1	RCL0
MSB							LSB

Bits 7 and 6
(SBS1 and SBS2) — **Stop Bit Selection.** The number of stop bits to be appended to each transmitter character is selected by bits 7 and 6 of the Control Register as shown below. The receiver only tests for a single stop bit, regardless of the status of bits 7 and 6.

STOP BIT SELECTION

SBS1 BIT 7	SBS2 BIT 6	NUMBER OF TRANSMITTED STOP BITS
0	0	1½
0	1	2
1	0	1
1	1	1

8◀

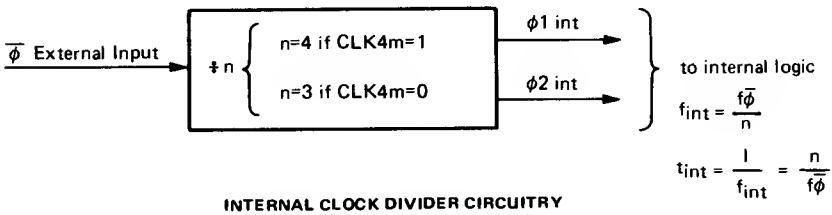
Bits 5 and 4
(PENB and PODD) — **Parity Selection.** The type of parity generated for transmission and detected for reception is selected by bits 5 and 4 of the Control Register as shown below. When parity is enabled (PENB = 1), the parity bit is transmitted and received in addition to the number of bits selected for the character length. Odd parity is such that the total number of ones in the character and parity bit, exclusive of stop bit(s), will be odd. For even parity, the total number of ones will be even.

PARITY SELECTION

PENB BIT 5	PODD BIT 4	PARITY
0	0	None
0	1	None
1	0	Even
1	1	Odd

Bit 3 (CLK4M) —

$\overline{\phi}$ Input Divide Select. The $\overline{\phi}$ input to the TMS 9902 ACC is used to generate internal dynamic logic clocking and to establish the time base for the Interval Timer, Transmitter, and Receiver. The $\overline{\phi}$ input is internally divided by either 3 or 4 to generate the two-phase internal clocks required for MOS logic, and to establish the basic internal operating frequency (f_{int}) and internal clock period (t_{int}). When bit 3 of the Control Register is set to a logic one (CLK4M = 1), $\overline{\phi}$ is internally divided by 4, and when CLK4M = 0, $\overline{\phi}$ is divided by 3. For example, when $f\overline{\phi}$ = 3 MHz, as in a standard 3 MHz TMS 9900 system, and CLK4M = 0, $\overline{\phi}$ is internally divided by 3 to generate an internal clock period t_{int} of 1 μs. The figure below shows the operation of the internal clock divider circuitry. The internal clock frequency should be no greater than 1.1 MHz; thus, when $f\overline{\phi}$ > 3.3 MHz, CLK4M should be set to a logic one.



INTERNAL CLOCK DIVIDER CIRCUITRY

Bits 1 and 0
(RCL1 and RCL0) —

Character Length Select. The number of data bits in each transmitted and received character is determined by bits 1 and 0 of the Control Register as shown below:

CHARACTER LENGTH SELECTION

RCL1 BIT 1	RCL0 BIT 0	CHARACTER LENGTH
0	0	5 Bits
0	1	6 Bits
1	0	7 Bits
1	1	8 Bits

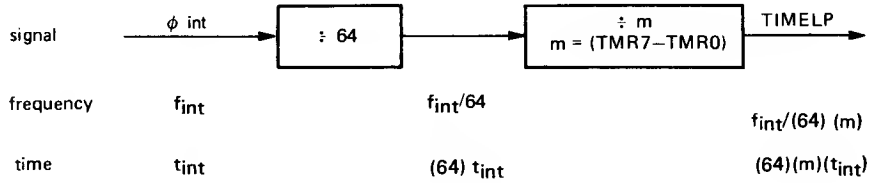
2.1.2.2 Interval Register

The Interval Register is enabled for loading when LDCTRL = 0 and LDIR = 1 (see Table 2). The Interval Register is used to select the rate at which interrupts are generated by the TMS 9902 Interval Timer. The figure below shows the bit assignments for the Interval Register when enabling for loading.

7	6	5	4	3	2	1	0
TMR7	TMR6	TMR5	TMR4	TMR3	TMR2	TMR1	TMR0
MS8							LS8

INTERVAL REGISTER BIT ADDRESS ASSIGNMENTS

The figure below illustrates the establishment of the interval for the Interval Timer. For example, if the Interval Register is loaded with a value of 80_{16} (128_{10}) the interval at which Timer Interrupts are generated is $t_{itvl} = t_{int} \cdot 64 \cdot M = (1\ \mu s)(64)(128) = 8.192$ ms when $t_{int} = 1\ \mu s$. $t_{int} = n/f\overline{\phi}$ where n = 4 if CLK4M = 1, 3 if CLK4M = 0.



TIME INTERVAL SELECTION

2.1.2.3 Receive Data Rate Register

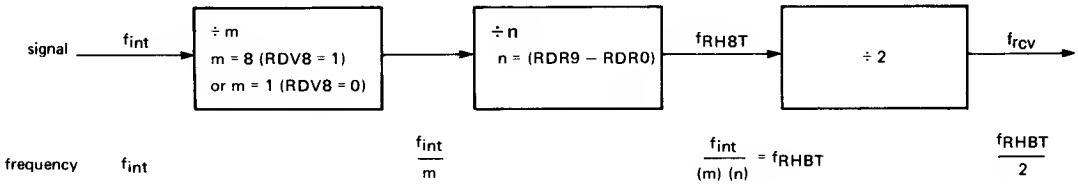
The Receive Data Rate Register (RDR) is enabled for loading when LDCTRL = 0, LDIR = 0, and LRDR = 1 (see Table 2). The Receive Data Rate Register is used to select the bit rate at which data is received. The diagram shows the bit address assignments for the Receive Data Rate Register when enabled for loading.

10	9	8	7	6	5	4	3	2	1	0
RDV8	RDR9	RDR8	RDR7	RDR6	RDR5	RDR4	RDR3	RDR2	RDR1	RDR0
MSB										LSB

RECEIVE DATA RATE REGISTER BIT ADDRESS ASSIGNMENTS

8◀

The diagram below illustrates the manner in which the receive data rate is established. Basically, two programmable counters are used to determine the interval for half the bit period of receive data. The first counter divides the internal system clock frequency (f_{int}) by either 8 (RDV8 = 1) or 1 (RDV8 = 0). The second counter has ten stages and may be programmed to divide its input signal by any value from 1 (RDR9 – RDR0 = 0000000001) to 1023 (RDR9 – RDR0 = 1111111111). The frequency of the output of the second counter (f_{rhbt}) is double the receive-data rate. For example, assume the Receive Data Rate Register is loaded with a value of 11000111000; RDV8 = 1, and RDR9 – RDR0 = 1000111000 = 238_{16} = 568_{10}. Thus, for f_{int} = 1 MHz, (see Control Register, bit 3) the receive data rate = $f_{rcv} = [(1 \times 10^6 \div 8) \div 568] \div 2 = 110.04$ bits per second.



RECEIVE DATA RATE SELECTION

Quantitatively, the receive-data rate f_{RCV} is described by the following algebraic expression:

$$f_{rcv} = \frac{f_{RHBT}}{2} = \frac{f_{int}}{(2)\ (m)\ (n)} = \frac{f_{int}}{(2)\ (8^{RDV8})\ (RDR9 - RDR0)}$$

2.1.2.4 Transmit Data Rate Register

The Transmit Data Rate Register (XDR) is enabled for loading when LDCTRL = 0, LDIR = 0, and LXDR = 1 (see Table 2). The Transmit Data Rate Register is used to select the data for the transmitter. The figure below shows the bit address assignments for the Transmit Data Rate Register when enabled for loading.

10	9	8	7	6	5	4	3	2	1	0
XDV8	XDR9	XDR8	XDR7	XDR6	XDR5	XDR4	XDR3	XDR2	XDR1	XDR0
MSB										LSB

The transmit data rate is selected with the Transmit Data Rate Register in the same manner the receive data rate is selected with the Receive Data Rate Register. The algebraic Expression for the Transmit Data Rate f_{xmt} is

$$f_{xmt} = \frac{f_{XHBT}}{2} = \frac{f_{int}}{(2)\ (8^{XDV8})\ (XDR9\text{-}XDR0)}$$

For example, if the Transmit Data Rate Register is loaded with a value of 00110100001; XDV8 = 0, and XDR9 – XDR0 = $1A1_{16} = 417_{10}$, if f_{int} = 1 MHz the transmit data rate = f_{xmt} = [(1 × 10^6 ÷ 1) ÷ 417] ÷ 2 = 1199.0 bits per second.

▶8 **2.1.2.5 Transmit Buffer Register**

The Transmit Buffer Register (XBR) is enabled for loading when LDCTRL = 0, LDIR = 0, LRDR = 0, LXDR = 0, and BRKON = 0 (see Table 2). The Transmit Buffer Register is used to store the next character to be transmitted. When the transmitter is active, the contents of the Transmit Buffer Register are transferred to the Transmit Shift Register (XSR) each time the previous character has been completely transmitted (XSR becomes empty). The bit address assignments for the Transmit Buffer Register are shown below:

7	6	5	4	3	2	1	0
XBR7	XBR6	XBR5	XBR4	XBR3	XBR2	XBR1	XBR0
MSB							LSB

TRANSMIT BUFFER REGISTER BIT ADDRESS ASSIGNMENTS

TABLE 4. CRU OUTPUT BIT ADDRESS ASSIGNMENTS

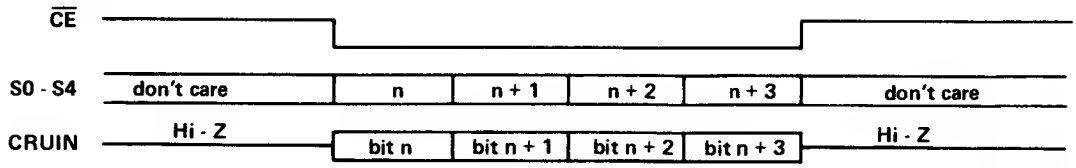
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	RESET	NOT USED									OSCENB	TIMENB	XBIENB	RIENB	BRKON	RTSON
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

All eight bits should be transferred into the register, regardless of the selected character length. The extraneous high order bits will be ignored for transmission purposes; however, loading of bit 7 is internally detected which causes the Transmit Buffer Register Empty (XBRE) status flag to reset.

2.1.3 INPUT TO CPU FOR CRU

The TMS 9902 ACC occupies 32 bits of input CRU space. The CPU reads the 32 bits from the ACC to sense the status of the device. Table 5 shows the mapping between CRU bit address and TMS 9902 read data. Each CRU addressable read bit is described following Table 5.

Status and data information is read from the ACC using $\overline{CE}$, S0-S4, and CRUIN. The following figure illustrates the relationship of the signals used to access four bits of data from the ACC.



ACC DATA ACCESS SIGNAL TIMING

TABLE 5
TMS 9902 ACC INPUT BIT ADDRESS ASSIGNMENTS

ADDRESS$_2$ S0	S1	S2	S3	S4	ADDRESS$_{10}$	NAME	DESCRIPTION
1	1	1	1	1	31	INT	Interrupt
1	1	1	1	0	30	FLAG	Ragister Load Control Flag Set
1	1	1	0	1	29	DSCH	Data Set Status Change
1	1	1	0	0	2B	CTS	Clear to Send
1	1	0	1	1	27	DSR	Data Set Ready
1	1	0	1	0	26	RTS	Requast to Send
1	1	0	0	1	25	TIMELP	Timer Elapsed
1	1	0	0	0	24	TIMERR	Timer Error
1	0	1	1	1	23	XSRE	Transmit Shift Register Empty
1	0	1	1	0	22	XBRE	Transmit Buffar Register Empty
1	0	1	0	1	21	RBRL	Receive Buffer Register Loaded
1	0	1	0	0	20	DSCINT	Data Set Status Change Interrupt (DSCH • DSCENB)
1	0	0	1	1	19	TIMINT	Timer Interrupt (TIMELP • TIMENB)
1	0	0	1	0	1B	–	Not Used (always = 0)
1	0	0	0	1	17	XBINT	Transmittar Interrupt (XBRE • XBIENB)
1	0	0	0	0	16	RBINT	Receiver Interrupt (RBRL • RIENB)
0	1	1	1	1	15	RIN	Receive Input
0	1	1	1	0	14	RSBD	Receive Start Bit Detect
0	1	1	0	1	13	RFBD	Receive Full Bit Detect
0	1	1	0	0	12	RFER	Receive Framing Error
0	1	0	1	1	11	ROVER	Receive Overrun Error
0	1	0	1	0	10	RPER	Receive Parity Error
0	1	0	0	1	9	RCVERR	Receive Error
0	1	0	0	0	8	–	Not Used (always = 0)
					7-0	RBR7 - RBR0	Raceive Buffar Registar (Received Data)

Bit 31 (INT) — INT = DSCINT (Data Set Status Change Interrupt) + TIMINT (Timer Interrupt) + XBINT (Transmitter Interrupt) + RBINT (Receiver Interrupt). The interrupt output ($\overline{INT}$) is active (LOW) when this status signal is a logic one. (Refer also to Section 2.6.)

Bit 30 (FLAG) — FLAG = LDCTRL + LDIR + LRDR + LXDR + BRKON. When any of the register load control flags or BRKON is set, FLAG = 1 (see Section 2.1.1).

Bit 29 (DSCH) — **Data Set Status Change.** DSCH is set when the $\overline{DSR}$ or $\overline{CTS}$ input changes state. To ensure recognition of the state change, $\overline{DSR}$ or $\overline{CTS}$ must remain stable in its new state for a minimum of two internal clock cycles. DSCH is reset by an output to bit 21 (DSCENB).

Bit 28 (CTS) — **Clear To Send.** The CTS signal indicates the inverted status of the $\overline{CTS}$ device input.

Bit 27 (DSR) — **Data Set Ready.** The DSR signal indicates the inverted status of the $\overline{DSR}$ device input.

Bit 26 (RTS) — **Request To Send.** The RTS signal indicates the inverted status of the $\overline{RTS}$ device output.

Bit 25 (TIMELP) — **Timer Elapsed.** TIMELP is set each time the Interval Timer decrements to 0. TIMELP is reset by an output to bit 20 (TIMENB).

Bit 24 (TIMERR) — **Timer Error.** TIMERR is set whenever the Interval Timer decrements to 0 and TIMELP (Timer Elapsed) is already set, indicating that the occurrence of TIMELP was not recognized and cleared by the CPU before subsequent intervals elapsed. TIMERR is reset by an output to bit 20 (TIMENB, Timer Interrupt Enable).

Bit 23 (XSRE) — **Transmit Shift Register Empty.** When XSRE = 1, no data is currently being transmitted and the XOUT output is at logic one unless BRKON (see Section 2.1.1) is set. When XSRE = 0, transmission of data is in progress.

Bit 22 (XBRE) — **Transmit Buffer Register Empty.** When XBRE = 1, the transmit buffer register does not contain the next character to be transmitted. XBRE is set each time the contents of the transmit buffer register are transferred to the transmit shift register, XBRE is reset by an output to bit 7 of the transmit buffer register (XBR7), indicating that a character has been loaded.

Bit 21 (RBRL) — **Receive Buffer Register Loaded.** RBRL is set when a complete character has been assembled in the receive shift register, and the character is transferred to the receive buffer register. RBRL is reset by an output to bit 18 (RIENB, Receiver Interrupt Enable).

Bit 20 (DSCINT) — **Data Set Status Change Interrupt.** DSCINT = DSCH (Data Set Status Change) AND DSCENB (Data Set Status Change Interrupt Enable). DSCINT indicates the presence of an enabled interrupt caused by the changing of state of DSR or CTS.

Bit 19 (TIMINT) — **Timer Interrupt.** TIMINT = TIMELP (Timer Elapsed) AND TIMENB (Timer Interrupt Enable). TIMINT indicates the presence of an enabled interrupt caused by the interval timer.

Bit 17 (XBINT) — **Transmitter Interrupt.** XBINT = XBRE (Transmit Buffer Register Empty) AND XBIENB (Transmit Buffer Interrupt Enable). XBINT indicates the presence of an enabled interrupt caused by the transmitter.

Bit 16 (RBINT) — **Receiver Interrupt.** RBINT = RBRL (Receive Buffer Register Loaded) AND RIENB (Receiver Interrupt Enable). RBINT indicates the presence of an enabled interrupt caused by the receiver.

Bit 15 (RIN) — **Receive Input.** RIN indicates the status of the RIN input to the device.

Bit 14 (RSBD) — **Receive Start Bit Detect.** RSBD is set a half bit time after the 1-to-0 transition of RIN, indicating the start bit of a character. If RIN is not still 0 at such time, RSBD is reset. Otherwise, RSBD remains true until the complete character has been received. This bit is normally used only for testing purposes.

Bit 13 (RFBD) — **Receive Full Bit Detect.** RFBD is set one bit time after RSBD is set to indicate the sample point for the first data bit of the received character. RSBD is reset when the character has been completely received. This bit is normally used only for testing purposes.

Bit 12 (RFER) — **Receive Framing Error.** RFER is set when a character is received in which the stop bit, which should be a logic one, is a logic zero. RFER should only be read when RBRL (Receive Buffer Register Loaded) is a one. RFER is reset when a character with the correct stop bit is received.

Bit 11 (ROVER) — **Receive Overrun Error.** ROVER is set when a new character is received before the RBRL (Receive Buffer Register Loaded) flag is reset, indicating that the CPU failed to read the previous character and reset RBRL before the present character is completely received. ROVER is reset when a character is received and RBRL is 0 when the character is transferred to the receive buffer register.

Bit 10 (RPER) — **Receive Parity Error.** RPER is set when a character is received in which the parity is incorrect. RPER is reset when a character with correct parity is received.

Bit 9 (RCVERR) — **Receive Error.** RCVERR = RFER (Receive Framing Error) + ROVER (Receiver Overrun Error) + RPER (Receive Parity Error). The RCVERR signal indicates the presence of an error in the most recently received character.

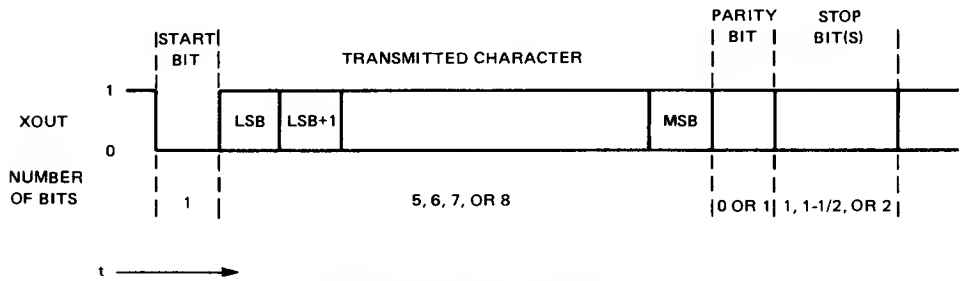
Bit 7-Bit 0
(RBR7-RBR0) — **Receive Buffer Register.** The Receive Buffer Register contains the most recently received character. For character lengths of fewer than eight bits, the character is right-justified, with unused most significant bit(s) all zero(es). The presence of valid data in the Receive Buffer Register is indicated when RBRL (Receive Buffer Register Loaded) is a logic one.

2.2 TRANSMITTER OPERATION

The operation of the transmitter is diagrammed in Figure 5. The transmitter is initialized by issuing the RESET command (output to bit 31), which causes the internal signals XSRE (Transmit Shift Register Empty) and XBRE (Transmit Buffer Register Empty) to set, and BRKON to reset. Device outputs $\overline{RTS}$ and XOUT are set, placing the transmitter in its idle state. When RTSON (Request-to-Send On) is set by the CPU, the $\overline{RTS}$ output becomes active (LOW) and the transmitter becomes active when the $\overline{CTS}$ input goes LOW.

2.2.1 Data Transmission

If the Transmit Buffer Register contains a character, transmission begins. The contents of the Transmit Buffer Register are transferred to the Transmit Shift Register, causing XSRE to reset and XBRE to set. The first bit transmitted (start bit) is always a logic zero. Subsequently, the character is shifted out, LSB first. Only the number of bits specified by RCL1 and RCL0 (character length select) of the Control Register are shifted. If parity is enabled, the correct parity bit is next transmitted. Finally the stop bit(s) selected by SBS1 and SBS0 of the Control Register are transmitted. Stop bits are always logic one. XSRE is set to indicate that no transmission is in progress, and the transmitter again tests XBRE to determine if the CPU has yet loaded the next character. The timing for a transmitted character is shown below.



TRANSMITTED CHARACTER TIMING

2.2.2 BREAK Transmission

The BREAK message is transmitted only if XBRE = 1, $\overline{CTS}$ = 0, and BRKON = 1. After transmission of the BREAK message begins, loading of the Transmit Buffer Register is inhibited and XOUT is reset. When BRKON is reset by the CPU, XOUT is set and normal operation continues. It is important to note that characters loaded into the Transmit Buffer Register are transmitted prior to the BREAK message, regardless of whether or not the character has been loaded into the Transmit Shift Register before BRKON is set. Any character to be transmitted subsequent to transmission of the BREAK message may not be loaded into the Transmit Buffer Register until after BRKON is reset.

8◀

2.2.3 Transmission Termination

Whenever XSRE = 1 and BRKON = 0 the transmitter is idle, with XOUT set to one. If RTSON is reset at this time, the $\overline{RTS}$ device output will go inactive (HIGH), disabling further data transmission until RTSON is again set. $\overline{RTS}$ will not go inactive, however, until any characters loaded into the Transmit Buffer Register prior to resetting RTSON are transmitted and BRKON = 0.

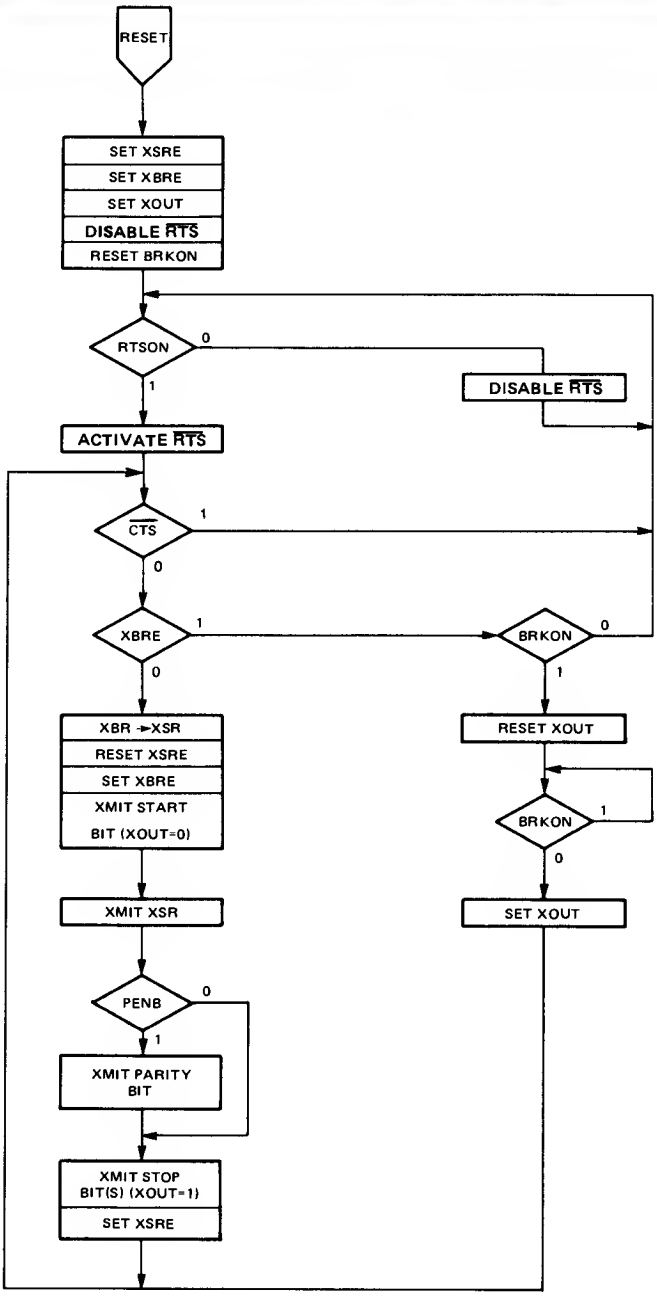


FIGURE 5. TMS 9902 TRANSMITTER OPERATION

2.3 RECEIVER OPERATION

2.3.1 Receiver Initialization

Operation of the TMS 9902 receiver is diagrammed in Figure 6. The receiver is initialized whenever the CPU issues the RESET command. The RBRL (Receive Buffer Register Loaded) flag is reset to indicate that no character is currently in the Receive Buffer Register, and the RSBD (Receive Start Bit Detect) and RFBD (Receive Full Bit Detect) flags are reset. The receiver remains in the inactive state until a one to zero transition is detected on the RIN device input.

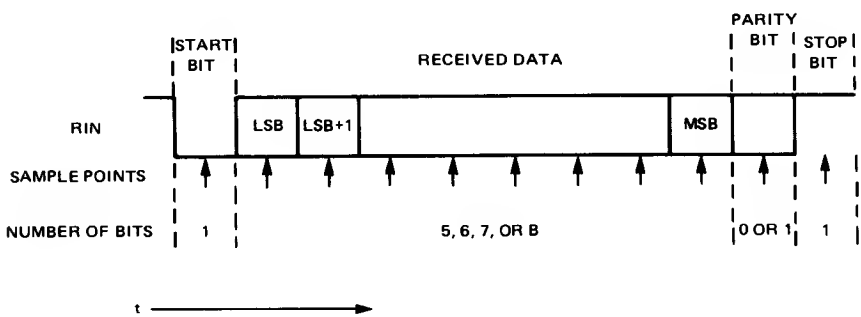
2.3.2 Start Bit Detection

The receiver delays a half bit time and again samples RIN to ensure that a valid start bit has been detected. If RIN = 0 after the half-bit delay, RSBD is set and data reception begins. If RIN = 1, no data reception occurs.

2.3.3 Data Reception

In addition to verifying the valid start bit, the half-bit delay after the one-to-zero transition also establishes the sample point for all subsequent data bits in a valid received character. Theoretically, the sample point is in the center of each bit cell, thus maximizing the limits of acceptable distortion of data cells. After the first full bit delay the least significant data bit is received and RFBD is set. The receiver continues to delay one-bit intervals and sample RIN until the selected number of bits are received. If parity is enabled, one additional bit is read for parity. After an additional bit delay, the received character is transferred to the Receive Buffer Register, RBRL is set, ROVER (Receive Overrun Error) and RPER (Receive Parity Error) are loaded with appropriate values, and RIN is tested for a valid stop bit. If RIN = 1, the stop bit is valid. RFER (Receive Framing Error), RSBD, and RFBD are reset, and the receiver waits for the next start bit to begin reception of the next character.

If RIN = 0 when the stop bit is sampled, RFER is set to indicate the occurrence of a framing error. RSBD and RFBD are reset, but sampling for the start bit of the next character does not begin until RIN = 1. The timing for a received character is depicted below.



RECEIVED CHARACTER TIMING

8◂

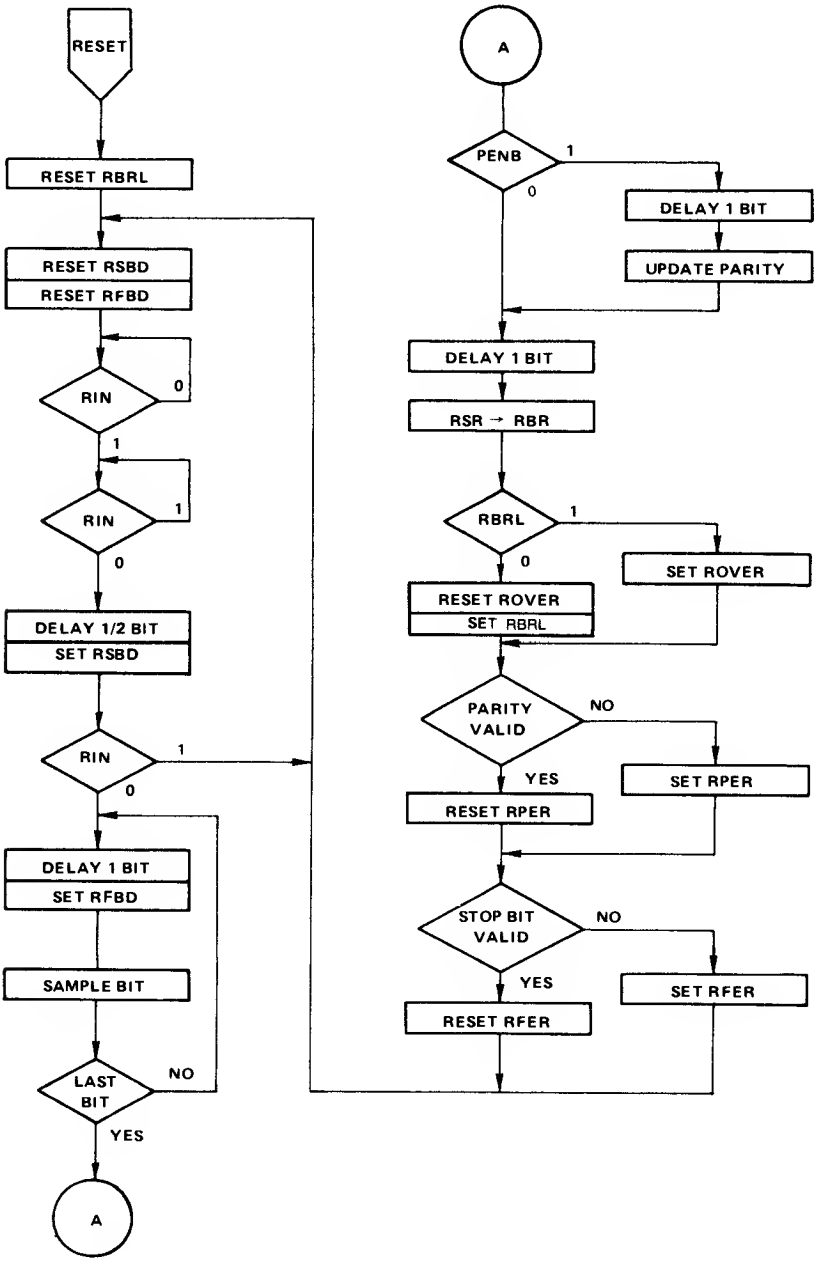


FIGURE 6. TMS 9902 RECEIVER OPERATION

2.4 INTERVAL TIMER OPERATION

A flowchart of the operation of the Interval Timer is shown in Figure 7. Execution of the RESET command by the CPU causes TIMELP (Timer Elapsed) and TIMERR (Timer Error) to reset and LDIR (Load Interval Register) to set. Resetting LDIR causes the contents of the Interval Register to be loaded into the Interval Timer, thus beginning the selected time interval. The timer is decremented every 64 internal clock cycles (every two internal clock cycles when in Test Mode) until it reaches zero, at which time the Interval Timer is reloaded by the Interval Register and TIMELP is set. If TIMELP was already set, TIMERR is set to indicate that TIMELP was not cleared by the CPU before the next time period elapsed. Each time LDIR is reset, the contents of the Interval Register are loaded into the Interval Timer, thus restarting the timer (refer also to Section 2.1.2.2).

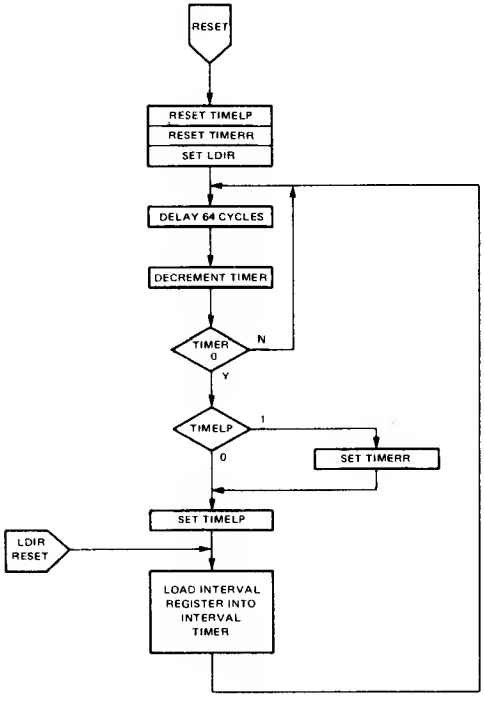


FIGURE 7. TMS 9902 INTERVAL TIMER OPERATION

INTERVAL TIMER SELECTION

2.5 INTERRUPTS

The interrupt output ($\overline{INT}$) is active (LOW) when any of the following conditions occurs and the corresponding interrupt has been enabled on the TMS 9902 by the CPU:

(1) $\overline{DSR}$ or $\overline{CTS}$ changes levels (DSCH = 1);

(2) a character has been received and stored in the Receive Buffer Register (RBRL = 1);

(3) the Transmit Buffer Register is empty (XBRE = 1); or

(4) the selected time interval has elapsed (TIMELP = 1).

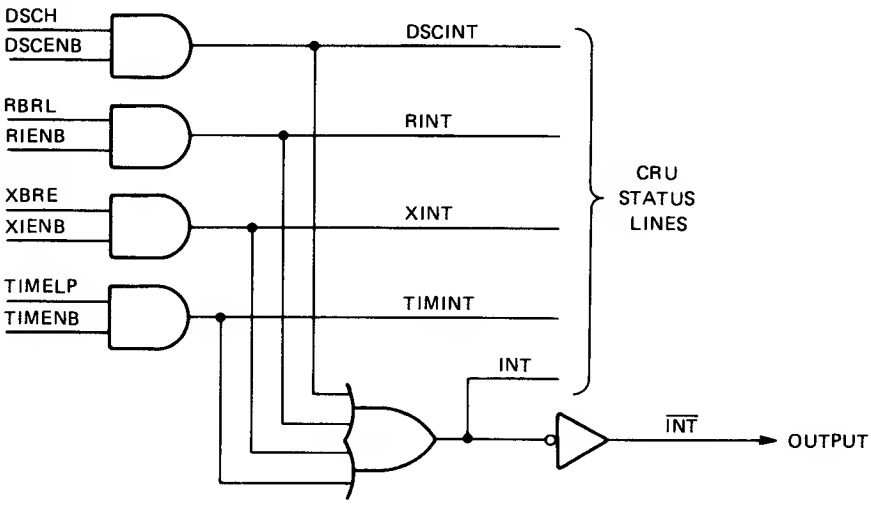


FIGURE 8. $\overline{INT}$ OUTPUT GENERATION

Figure 8 illustrates the logical equivalent of the ACC interrupt section. Table 6 lists the actions necessary to clear those conditions of the TMS 9902 that cause interrupts.

▶8

TABLE 6
TMS 9902 INTERRUPT CLEARING

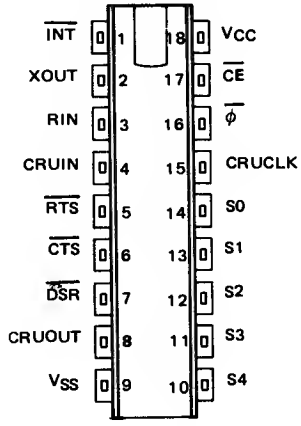
MNEMONIC	CAUSE	ACTION TO RESET
DSCINT	$\overline{CTS}$ or $\overline{DSR}$ change state	Write a bit to DSCENB (bit 21)*
RINT	Recieve Buffer Full	Write a bit to RIENB (bit 18)*
XINT	Transmit Buffer Register Empty	Load Transmit Buffer
TIMINT	Timer Elapsed	Write a bit to TIMENB (bit 20)*

*Writing a zero to clear the interrupt will clear the interrupt and disable further interrupts.

2.6 TMS 9902 TERMINAL ASSIGNMENTS AND FUNCTIONS

SIGNATURE	PIN	I/O	DESCRIPTION
$\overline{INT}$	1	O	Interrupt — when active (LOW), the $\overline{INT}$ output indicates that at least one of the interrupt conditions has occurred.
XOUT	2	O	Transmitter Serial Data Output line — XOUT, remains inactive (HIGH) when TMS 9902 is not transmitting.
RIN	3	I	Receiver Serial Data Input Line — RCV must be held in the inactive (HIGH) state when not receiving data. A transition from HIGH to LOW activates the receiver circuitry.
CRUIN	4	O	Serial data output pin from TMS 9902 to CRUIN input pin of the CPU.
$\overline{RTS}$	5	O	Request-to-Send output from TMS 9902 to modem. $\overline{RTS}$ is enabled by the CPU and remains active (LOW) during transmission from the TMS 9902.
$\overline{CTS}$	6	I	Clear-to-Send input from modem to TMS 9902. When active (LOW), it enables the transmitter section of TMS 9902.
$\overline{DSR}$	7	I	Data Set Ready input from modem to TMS 9902. $\overline{DSR}$ generates an interrupt when it changes state.
CRUOUT	8	I	Serial data input line to TMS 9902 from CRUOUT line of the CPU.
V_{SS}	9	I	Ground reference voltage.
S4 (LSB)	10	I	Address Select Lines. The data bit being accessed by the CPU interface is specified by the 5-bit code appearing on S0-S4.
S3	11	I	
S2	12	I	
S1	13	I	
S0	14	I	
CRUCLK	15	I	CRU Clock. When active (HIGH), indicates valid data on the CRUOUT line for the 9902.
$\overline{\phi}$	16	I	TTL Clock.
$\overline{CE}$	17	I	Chip Enable — when CE is inactive (HIGH), TMS 9902 CRU interface is disabled. CRUIN remains at high-impedance when CE is inactive (HIGH).
V_{CC}	18	I	Supply voltage (+5 V nominal).

TMS 9902
18-PIN PACKAGE



8

3. DEVICE APPLICATION

This section describes the software interface between the CPU and the TMS 9902 ACC and discusses some of the design considerations in the use of this device for asynchronous communications applications.

3.1 DEVICE INITIALIZATION

The ACC is initialized by the RESET command from the CPU (output bit 31), followed by loading the Control, Interval, Receive Data Rate, and Transmit Data Rate registers. Assume that the value to be loaded into the CRU Base Register (register 12) in order to point to bit 0 is 0040_{16}. In this application characters have seven bits of data plus even parity and one stop bit. The $\overline{\phi}$ input to the ACC is a 3 MHz signal. The ACC divides this signal frequency by three to generate an internal clock frequency of 1 MHz. An interrupt is generated by the Interval Timer every 1.6 milliseconds when timer interrupts are enabled. The transmitter operates at a data rate of 300 bits per second, and the receiver operates at 1200 bits per second.

NOTE

To operate both the transmitter and receiver at 300 bits per second, delete the "LDCR @RDR,11" instruction (see below), and the "LDCR @XDR,12" instruction will cause both data rate registers to be loaded and LRDR and LXDR to reset.

3.1.1 Initialization Program

The initialization program for the configuration described above is shown below. The RESET command disables all interrupts, initializes all controllers, and sets the four register load control flags (LDCTRL, LDIR, LRDR, and LXDR). Loading the last bit of each of the registers causes the load control flag to reset automatically.

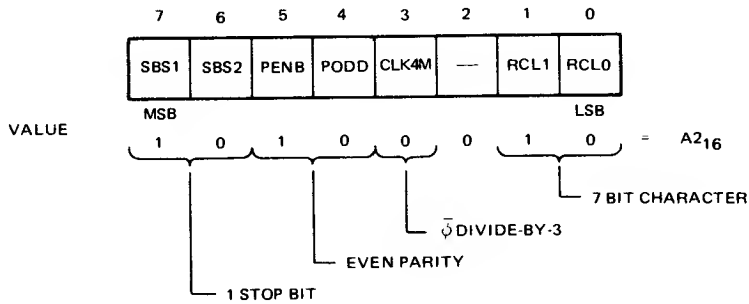
```
        LI      R12,>40         INITIALIZE CRU BASE
        SBO     31              RESET COMMAND
        LDCR    @ CNTRL,8       LOAD CONTROL AND RESET LDCTRL
        LDCR    @ INTVL,8       LOAD INTERVAL AND RESET LDIR
        LDCR    @ RDR,11        LOAD RDR AND RESET LRDR
        LDCR    @ XDR,12        LOAD XDR AND RESET LXDR
          .
          .
          .
CNTRL   BYTE    >A2
INTVL   BYTE    1600/64
RDR     DATA    >1A1
XDR     DATA    >4D0
```

▶8

The RESET command initializes all subcontrollers, disables interrupts, and sets LDCTRL, LDIR, LRDR, and LXDR, enabling loading of the control register.

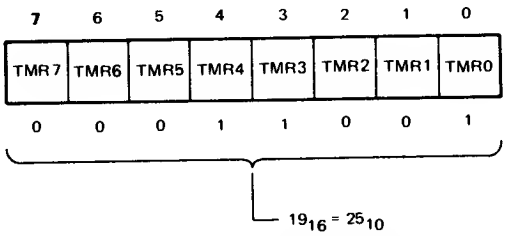
3.1.2 Control Register

The options listed in Table 3 in Section 2.1.2.1 are selected by loading the value shown below.



3.1.3 Interval Register

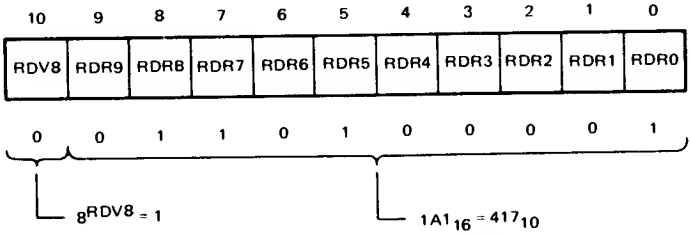
To set up the interval register to generate an interrupt every 1.6 milliseconds, load the value into the interval register to specify the number of 64-microsecond increments in the total interval desired.



25 X 64 MICROSECONDS = 1.6 MILLISECONDS

3.1.4 Receive Data Rate Register

To set the data rate for the receiver to 1200 bits per second, load the value into the Receive Data Rate register as shown below:



$10^6 \div 1 \div 417 \div 2 = 1199.04$ BITS PER SECOND

3.1.5 Transmit Data Rate Register

To program the data rate for the transmitter for 300 bits per second, load the following value into the Transmit Data Rate register:

10	9	6	7	6	5	4	3	2	1	0
XDV8	XDR9	XDR6	XDR7	XDR6	XDR5	XDR4	XDR3	XDR2	XDR1	XDR0
1	0	0	1	1	0	1	0	0	0	0

$_6$XDV6 = 6

$0D0_{16}$ = 208

$1 \times 10^6 \div 6 \div 206 \div 2 = 300.48$ BITS PER SECOND

3.2 DATA TRANSMISSION

The subroutine shown below demonstrates a simple loop for transmitting a block of data.

```
        LI      R0, LISTAD      INITIALIZE LIST POINTER
        LI      R1, COUNT       INITIALIZE BLOCK COUNT
        LI      R12, CRUBAS     INITIALIZE CRU BASE
        SBO     16              TURN ON TRANSMITTER
XMTLP   TB      22              WAIT FOR XBRE = 1
        JNE     XMTLP
        LDCR    *R0+,8          LOAD CHARACTER INCREMENT POINTER
                                    RESET XBRE
        DEC     R1              DECREMENT COUNT
        JNE     XMTLP           LOOP IF NOT COMPLETE
        SBZ     16              TURN OFF TRANSMITTER
```

After initializing the list pointer, block count, and CRU base address, RTSON is set to cause the transmitter and the $\overline{RTS}$ output to become active. Data transmission does not begin, however, until the $\overline{CTS}$ input becomes active. After the final character is loaded into the Transmit Buffer register, RTSON is reset. The transmitter and the $\overline{RTS}$ output do not become inactive until the final character is transmitted.

3.3 DATA RECEPTION

The following software will cause a block of data to be received and stored in memory.

```
CARRET   BYTE   >0D
RCVBLK   LI     R2, RCVLST      INITIALIZE LIST COUNT
         LI     R3, MXRCNT      INITIALIZE MAX COUNT
         LI     R4, CARRET      SET UP END OF BLOCK CHARACTER
RCVLP    TB     21              WAIT FOR RBRL = 1
         JNE    RCVLP
         STCR   *R2,8           STORE CHARACTER
         SBZ    18              RESET RBRL
         DEC    R3              DECREMENT COUNT
         JEQ    RCVEND          END IF COUNT = 0
         CB     *R2+,R4         COMPARE TO EOB CHARACTER, INCREMENT
                                    POINTER
         JNE    RCVLP           LOOP IF NOT COMPLETE
RCVEND   RT                      END OF SUBROUTINE
```

3.4 REGISTER LOADING AFTER INITIALIZATION

The Control, Interval, and Data Rate registers may be reloaded after initialization. For example, it may be desirable to change the interval of the timer. Assume that the interval is to be changed to 10.24 milliseconds; the instruction sequence is:

```
         SBO    13              SET LOAD CONTROL FLAG
         LDCR   @ INTVL2,8      LOAD REGISTER, RESET FLAG
           .
           .
           .
INTVL2   BYTE   10240/64
```

When transmitter interrupts are enabled, caution should be exercised to ensure that a transmitter interrupt does not occur while the load control flag is set. For example, if a transmitter interrupt occurs between execution of the "SBO 13" and the next instruction, the transmit buffer is not enabled for loading when the Transmitter Interrupt service routine is entered because the LDIR flag is set. This situation may be avoided by the following sequence:

```
         BLWP   @ITVCHG         CALL SUBROUTINE
           .
           .
           .
ITVCPC   LIMI   0               MASK ALL INTERRUPTS
         MOV    @ 24(R13),R12   LOAD CRU BASE ADDRESS
         SBO    13              SET FLAG
         LDCR   @ INTVL2,8      LOAD REGISTER AND RESET FLAG
         RTWP                   RESTORE MASK AND RETURN
           .
           .
           .
           .
ITVCHG   DATA   ACCWP, ITVCPC
INTVL2   BYTE   10240/64
```

In this case all interrupts are masked, ensuring that all interrupts are disabled while the load control flag is set.

3.5 INTERFACE TO A DATA TERMINAL

Following is a discussion of the TMS 9902 interface to a TI Model 733 data terminal as implemented on the TM 990/100M microcomputer module. Figure 9 diagrams the hardware interface, and Table 7 lists the software interface. The 733 data terminal is an ASCII-code, serial, asynchronous, EIA device equipped with a keyboard, thermal printer, and digital cassette tape.

3.5.1 Hardware Interface

The hardware interface between the TMS 9902 and the 733 data terminal is shown in Figure 9. The asynchronous communication conforms to *EIA Standard RS-232-C*. The 75188 and 75189 performs the necessary level shifting between TTL levels and RS-232-C levels. The ACC chip enable ($\overline{9902SEL}$) signal comes from decode circuitry which looks at A0-A9 on CRU cycles. The interrupt output ($\overline{INT}$) of the TMS 9902 is sent to the TMS 9901 for prioritization and encoding. When the 9902 is communicating with a terminal, the $\overline{RTS}$ pin can be connected to the $\overline{CTS}$ pin because the terminal will always be in the clear-to-send ($\overline{CTS}$) condition.

3.5.2 Software

The software required to initialize, read from, and write to the TMS 9902 ACC is listed in Table 7. These routines are taken directly from TIBUG (TM 990/402-1) which is the monitor that runs on the TM 990/100M boards. The coding shown is part of a routine entered because of a power-up reset. Before this section of code was entered, not shown, R12 is set to the correct value of the TMS 9902 CRU base address. The baud rate is detected by measuring the start bit length when an "A" is entered via the keyboard. The variable COUNT is incremented every time the SPLOOP loop is executed. When a zero is seen at 9902 bit 15 (RIN) the start bits are finished being received. The value of COUNT is then compared against a table of known values in TABLE to determine the baud rate.

TIBUG assumes that all 1200-baud data terminals are TI Model 733 data terminals. The TI Model 733 communicates at 1200 baud, but prints at 300 baud; this means that bits travel the communications line at 1200 baud, but the spacing between characters is 300 baud. A wait loop is included in the write character routine to handle this spacing requirement. The TIBUG T command is used to indicate that a 1200 baud terminal is true 1200 baud; i.e., not a TI 733.

This code is taken from the middle of TIBUG; thus constructs and symbols are used which are not defined here. Lines 261 and 262 of the code contain XOP calls. The READ OPCODE is really a call to XOP 13 and the MESG opcode is a call to XOP 14, which in turn calls XOP 12. This can be figured out if the assembled code for these opcodes is examined. Following is a list of EQU statements that appear at the beginning of TIBUG, but are not shown here:

```
COUNT     EQU     3
POINT     EQU     7
LINK      EQU    11
CRUBAS    EQU    12
```

Once again, these values could easily be obtained by looking at the assembled code for the statement in which the symbol is used.

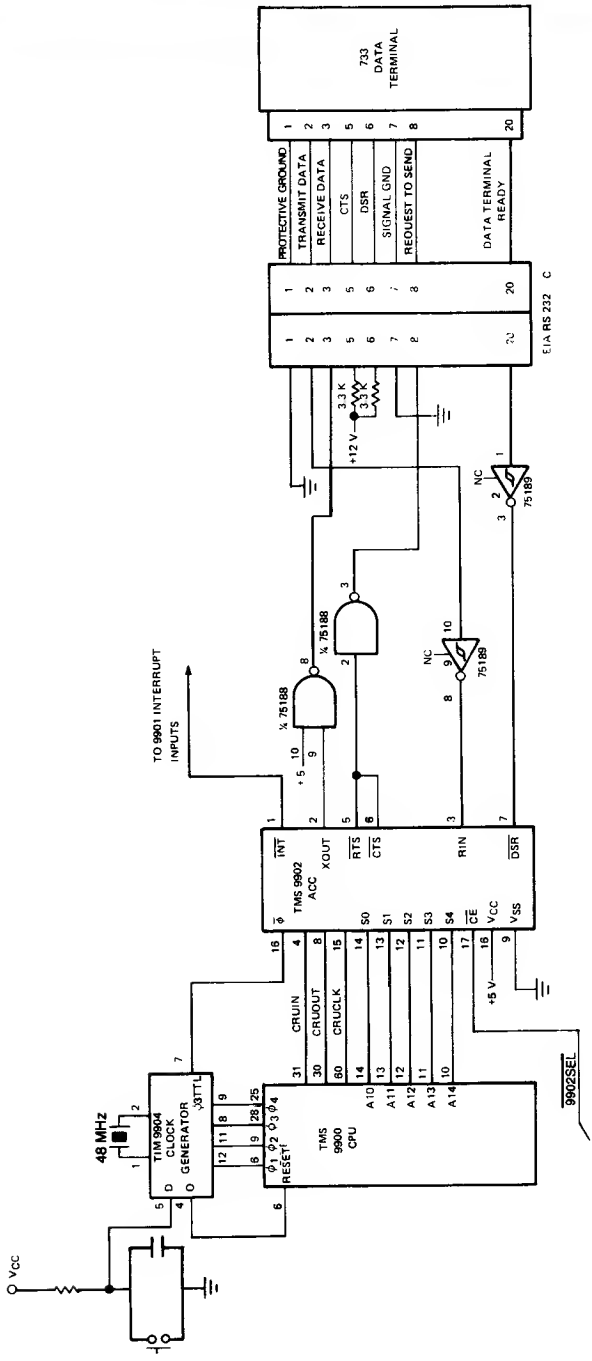


FIGURE 9. INTERFACE TO A 733 DATA TERMINAL

TABLE 7
TMS 9902 SOFTWARE

```
TIBUG
***COMMAND SEARCH AND SYSTEM INZ***

STATEMENT NO.  ADDRESS (HEX)  ASSEMBLED CODE

0231                    *
0232                    * INITIALIZE TMS9902 FOR: *BAUD RATE
0233                    *                         *7 BITS/CHARACTER
0234                    *                         *EVEN PARITY
0235                    *                         *2 STOP BITS
0236                    *                         *POLLED OPERATION
0237                    *
0238 015E 1D1F                SBO  31             RESET TMS9902 UART

0239 0160 3220                LDCR @CR,8          INITIALIZE TMS9902 CONTROL REG
     0162 01A4'
0240 0164 1E0D                SBZ  13             DO NOT INT INTERVAL REG
0241 0166 04C3                CLR  COUNT          RESET LOOP COUNT
0242 0168 1F0F        TSTSP   TB   15             SPACE?
0243 016A 13FE                JEQ  TSTSP          NO, JUMP BACK
0244 016C 0583        SPLOOP  INC  COUNT          TIME THE START BIT
0245 016E 1F0F                TB   15             FALL OUT ON A MARK
0246 0170 16FD                JNE  SPLOOP
0247                    *
0248                    * TABLE SEARCH FOR BAUD RATE
0249                    *
0250 0172 0207                LI   POINT,TABLE    SET POINTER TO TABLE
     0174 0194'
0251 0176 8DC3        BDLOOP  C    COUNT,*POINT+  MATCH?
0252 0178 1202                JLE  MATCH          YES, SET BAUD RATE
0253 017A 05C7                INCT POINT          NO, UPDATE POINTER
0254 017C 10FC                JMP  BDLOOP
0255      017E'       MATCH   EQU  $
0256 017E 3317                LDCR *POINT,12      INT. REC./XMT. DATA RATE
0257 0180 C1D7                MOV  *POINT,POINT
0258 0182 0287                CI   POINT,>1A0     1200 BAUD ?
     0184 01A0
0259 0186 1602                JNE  BANNER         LEAVE ASR FLAG ALONE
0260 0188 0720                SETO @ASR           SET 733ASR FLAG
     018A FFF4
0261 018C 2F45        BANNER  READ CHAR
0262 018E 2FA0                MESG @LOGON         PRINT LOG ON MESSAGE
     0190 022B'
0263 0192 10DC                JMP  JMMONT         TO TOP OF MONITOR
0264 0194 0040        TABLE   DATA >40,>D0        2400 BAUD
     0196 00D0
0265 0198 0070                DATA >70,>1A0       1200 BAUD
     019A 01A0
0266 019C 0200                DATA >200,>4D0      300 BAUD
     019E 04D0
0267 01A0 0400                DATA >400,>638      110 BAUD
     01A2 0638
0268 01A4   62        CR      BYTE >62
```

▸8

TABLE 7 (Continued)

```
TIBUG
*** WRITE CHARACTER ***

                    ********************************************
                    * WRITE CHARACTER -- XOP  R,12
                    *                          ---      NORMAL RETURN
                    *
                    * TRANSMIT THE CHARACTER IN THE LEFT BYTE OF
                    * USER REGISTER R. IF THE CHARACTER IS A
                    * CARRIAGE RETURN, THE ROUTINE WAITS 200 MSEC FOR
                    * THE CARRIAGE TO RETURN. IF THE TERMINAL IS
                    * A 733ASR AS DENOTED IN THE T COMMAND, EACH
                    * CHARACTER IS PADDED WITH 25 MSEC TO REDUCE
                    * THE TRANSFER RATE TO 300 BAUD.
                    ************************************************
01B6 020A           WENTRY LI   R10,3750
01B8 0EA6
01BA 020C                  LI   CRUBAS,>80    SET CRU BASE REG.
01BC 0080
01BE 1D10                  SBO  16            SET RTSON
01C0 1F16                  TB   22            TRANSMIT BUFFER REG. EMPTY?
01C2 16F9                  JNE  WENTRY        NO, WAIT UNTIL IT IS
01C4 321B                  LDCR *LINK,8       CHARACTER TO UART
01C6 D2DB                  MOVB *LINK,LINK
01C8 1E10                  SBZ  16            RESET RTSON
01CA 098B                  SRL  LINK,8
01CC 028B                  CI   LINK,>000D    CARRIAGE RETURN
01CE 000D
01D0 1608                  JNE  ASR733        NO, SKIP
01D2 0A3A                  SLA  R10,3
01D4 1F16           WLOOP1 TB   22            WAIT FOR XMISSION TO END
01D6 16FE                  JNE  WLOOP1
01D8 1F17                  TB   23
01DA 16FC                  JNE  WLOOP1
01DC 060A           WLOOP2 DEC  R10           WAIT LOOP
01DE 16FE                  JNE  WLOOP2
01E0 0380                  RTWP
01E2 C2E0           ASR733 MOV  @DUMPFG,LINK IN DUMP ROUTINE ?
01E4 FFF6
01E6 1303                  JEQ  WEXIT         YES,IGNORE ASR FLAG
01E8 C2E0                  MOV  @ASR,LINK     ASR733 ?
01EA FFF4
01EC 16F3                  JNE  WLOOP1        YES, WAIT 3 NULLS
01EE 0380           WEXIT  RTWP
```

```
TIBUG
*** READ CHARACTER ***

                    ************************************************
                    * READ CHARACTER -- XOP  R,13
                    *                          --       NORMAL RETURN
                    *
                    * READ WAITS FOR A CHARACTER TO BE ASSEMBLED IN
                    * THE UART. THE CHARACTER IS PLACED IN THE LEFT
                    * BYTE OF USER REGISTER R. THE RIGHT BYTE IS
                    * ZEROED. ALL ERRORS ARE IGNORED.
                    ************************************************
                    *
01A6 020C           RENTRY LI   CRUBAS,>80    SET CRU BASE REG.
01A8 0080
01AA 1F15                  TB   21            RECEIVE BUFFER REG. FULL?
01AC 16FC                  JNE  RENTRY        NO, LOOP
01AE 04DB                  CLR  *LINK
01B0 361B                  STCR *LINK,8
01B2 1E12                  SBZ  18
01B4 0380                  RTWP
```

8

4. TMS 9902 ELECTRICAL SPECIFICATIONS

4.1 Absolute Maximum Ratings Over Operating Free Air Temperature Range (Unless Otherwise Noted) *

Supply voltage, V_{CC} ..... −0.3 V to 10 V
All inputs and output voltages ..... −0.3 V to 10 V
Continuous power dissipation ..... 0.55 W
Operating free-air temperature range ..... 0°C to 70°C
Storage temperature range ..... −65°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

4.2 Recommended Operating Conditions *

PARAMETER	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}	4.75	5.0	5.25	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH}	2.0		V_{CC}	V
Low-level input voltage, V_{IL}	V_{SS}−.3		0.8	V
Operating free-air temperature, T_A	0		70	°C

4.3 Electrical Characteristics Over Full Range of Recommended Operating Conditions (Unless Otherwise Noted) *

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
V_{OH}	High level output voltage	$I_{OH} = -100\ \mu A$	2.4		V_{CC}	V
		$I_{OH} = -200\ \mu A$	2.2		V_{CC}	V
V_{OL}	Low level output voltage	$I_{OL} = 3.2$ mA	V_{SS}		0.4	V
I_I	Input current (any input)	$V_I = 0$ V to V_{CC}			±10	μA
$I_{CC(av)}$	Average supply current from V_{CC}	$t_{c(\phi)} = 330$ ns, $T_A = 70°C$			100	mA
C_i	Small signal input capacitance, any input	f = 1 MHz			15	pF

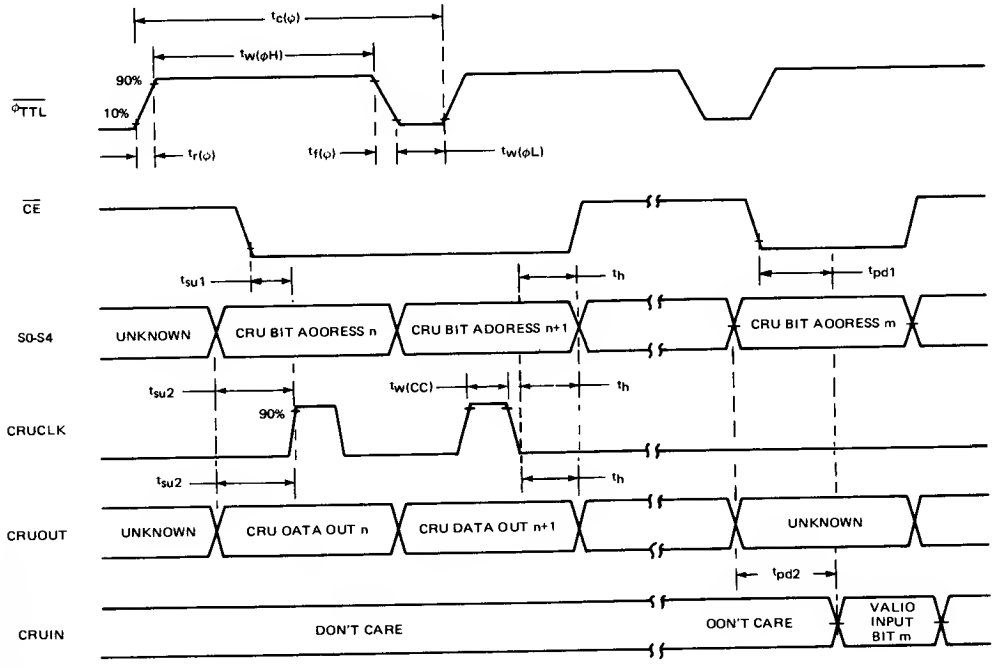
4.4 Timing Requirements Over Full Range of Operating Conditions

PARAMETER		MIN	TYP	MAX	UNIT
$t_{c(\phi)}$	Clock cycle time	300	333	667	ns
$t_{r(\phi)}$	Clock rise time	5		40	ns
$t_{f(\phi)}$	Clock fall time	10		40	ns
$t_{w(\phi H)}$	Clock pulse width (high level)	225			ns
$t_{w(\phi L)}$	Clock pulse width (low level)	45			ns
$t_{w(CC)}$	CRUCLK pulse width	100	185		ns
t_{su1}	Setup time for $\overline{CE}$ before CRUCLK	150			ns
t_{su2}	Setup time for S0-S4, or CRUOUT before CRUCLK	180			ns
t_h	Hold time for $\overline{CE}$, S0-S4, or CRUOUT after CRUCLK	60			ns

*NOTE: All voltage values are referenced to V_{SS}.

4.5 Switching Characteristics Over Full Range of Recommended Operating Conditions

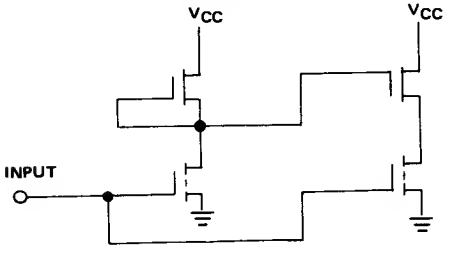
PARAMETER		TEST CONDITION	MIN	TYP	MAX	UNIT
t_{pd1}	Propagation delay, $\overline{CE}$ to valid CRUIN	CL = 100pF			300	ns
t_{pd2}	Propagation delay, S0-S4 to valid CRUIN	CL = 100pF			320	ns



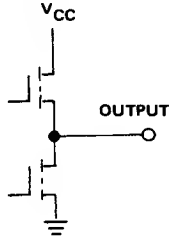
SWITCHING CHARACTERISTICS

NOTE: ALL SWITCHING TIMES ARE ASSUMED TO BE AT 10% OR 90% VALUES.

EQUIVALENT OF I/O INPUTS



EQUIVALENT OF I/O OUTPUTS



INPUT AND OUTPUT EQUIVALENTS

8

5. TMS 9902-40 ELECTRICAL SPECIFICATIONS

5.1 ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Supply voltage, V_{CC} . . . . . −0.3 V to 10 V
All Inputs and Output Voltages . . . . . −0.3 V to 10 V
Continuous Power Dissipation . . . . . 0.6 W
Operating Free-Air Temperature Range . . . . . 0°C to 70°C
Storage Temperature Range . . . . . −65°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to Absolute Maximum Rated conditions for extended periods may affect device reliability.

5.2 RECOMMENDED OPERATING CONDITIONS

	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH}	2.0	2.4	V_{CC}	V
Low-level input voltage, V_{IL}	V_{SS}-.3	0.4	0.8	V
Operating free-air temperature, T_A	0		70	°C

5.3 ELECTRICAL CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (UNLESS OTHERWISE NOTED)

	PARAMETERS	TEST CONDITIONS	MIN	TYP	MAX	UNIT
V_{OH}	High-level output voltage	$I_{OH} = -100\mu A$	2.4		V_{CC}	V
		$I_{OH} = -200\mu A$	2.2		V_{CC}	
V_{OL}	Low-level output voltage	$I_{OL} = 3.2$ mA			0.4	V
I_I	Input current (any input)	$V_I = 0$ V to V_{CC}			±10	μA
$I_{CC}(AV)$	Average supply current from V_{CC}	$t_c(\phi) = 330$ ns, $T_A = 25°C$			100	mA
C_i	Small Signal Input Capacitance, any input	f = 1 MHz			15	pF

5.4 TIMING REQUIREMENTS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS

	PARAMETER	MIN	TYP	MAX	UNIT
$t_c(\phi)$	Clock cycle time	240	250	667	ns
$t_r(\phi)$	Clock rise time	8		40	ns
$t_f(\phi)$	Clock fall time	10		40	ns
$t_w(\phi H)$	Clock pulse width (high level)	180			ns
$t_w(\phi L)$	Clock pulse width (low level)	40			ns
$t_w(CC)$	CRUCLK pulse width	80			ns
t_{su1}	Setup time for $\overline{CE}$ before CRUCLK	110	110		ns
t_{su2}	Setup time for S0-S4 or CRUOUT before CRUCLK	150	150		ns
t_h	Hold time for CE, S0-S4, or CRUOUT after CRUCLK	50	50		ns

DESIGN GOAL
This document describes the design specifications for a product under development. Texas Instruments reserves the right to change these specifications in any manner, without notice.

5.5 SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS

PARAMETERS		TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PD1}	Propagation delay, CE to valid CRUIN	$C_L = 100pF$		220	220	ns
t_{PD2}	Propagation delay, S0-S4 to valid CRUIN			240	240	ns

8-

DESIGN GOAL

This document describes the design specifications for a product under development. Texas Instruments reserves the right to change these specifications in any manner, without notice.

1. INTRODUCTION

1.1 DESCRIPTION

The TMS 9903 Synchronous Communications Controller (SCC) is a 20 pin peripheral device for the Texas Instruments TMS 9900 family of microprocessors. The TMS 9903 is TTL compatible on all inputs and outputs, including the power supply (+5V) and single phase clock. The SCC provides an interface between the microprocessor and a serial synchronous or asynchronous channel, performing data serialization and deserialization, facilitating microprocessor control of the communications channel. The TMS 9903 is fabricated using N–channel, silicon gate, MOS technology.

1.2 KEY FEATURES

- DC to 250 kilobits per second (kb/s) data rate, half or full duplex
- Dynamic character length selection
- Multiple line protocol capabilities: SDLC, Bi–Sync, HDLC, ADCCP
- Programmable CYCLIC–redundancy–check (CRC) generation and detection
- Interface to unclocked or NRZI data
- Programmable sync registers
- Interval timer with resolution from 64 – 16,320 microseconds (μs)
- Automatic zero insert and delete for SDLC, HDLC
- Fully TTL–compatible, including single +5 V power supply and clock
- Standard 20–pin plastic or ceramic package

1.3 TYPICAL APPLICATION

Figure 1 shows a general block diagram of a TMS 9900 based system incorporating a TMS 9903 SCC; Figure 2 is a similar diagram depicting a TMS 9980A or TMS 9981 based system. Following is an introductory discussion of the 9900 based application. Subsequent sections of this Data Manual detail most aspects of TMS 9903 usage.

The TMS 9903 interfaces with the CPU through the *communications register unit* (CRU). The CRU interface consists of five address select lines (S0–S4), chip enable ($\overline{CE}$), and three CRU lines (CRUIN, CRUOUT, CRUCLK). An additional input to the CPU is the SCC interrupt line ($\overline{INT}$). The TMS 9903 occupies 32 bits of CRU space; each of the 32 bits are selected individually by processor address lines A10–A14 which are connected to SCC select lines S0–S4, respectively. Chip enable ($\overline{CE}$) is generated by decoding address lines A0–A9 for CRU cycles. Under certain conditions the TMS 9903 causes interrupts, the SCC $\overline{INT}$ line is sent to the TMS 9901 for prioritization and encoding.

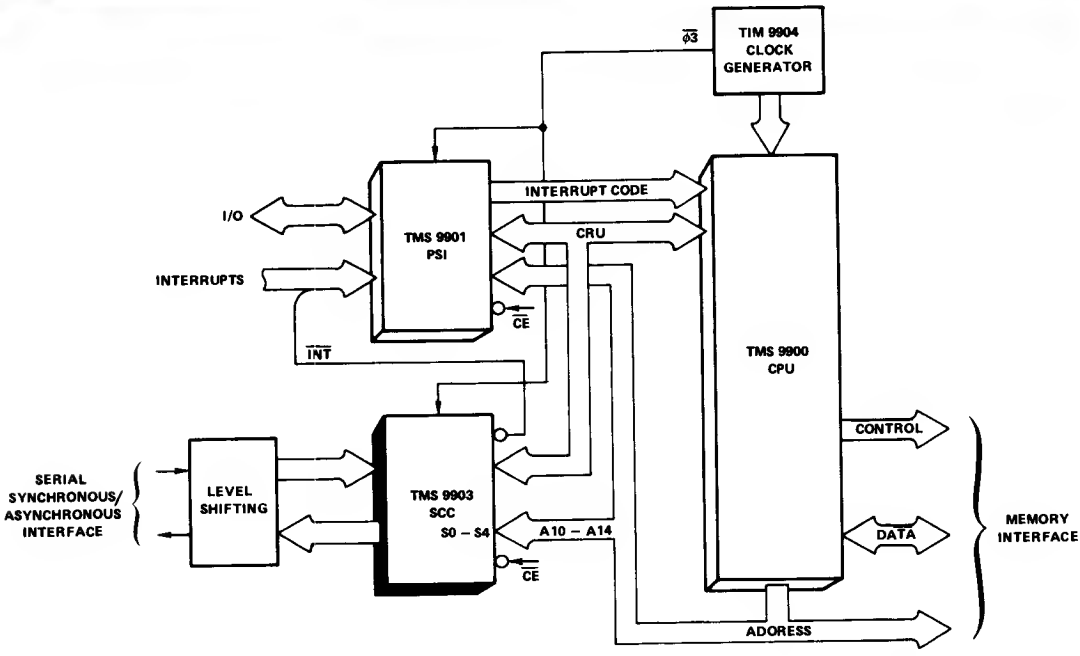


FIGURE 1. TMS 9903 SYNCHRONOUS COMMUNICATION CONTROLLER IN A TMS 9900 SYSTEM

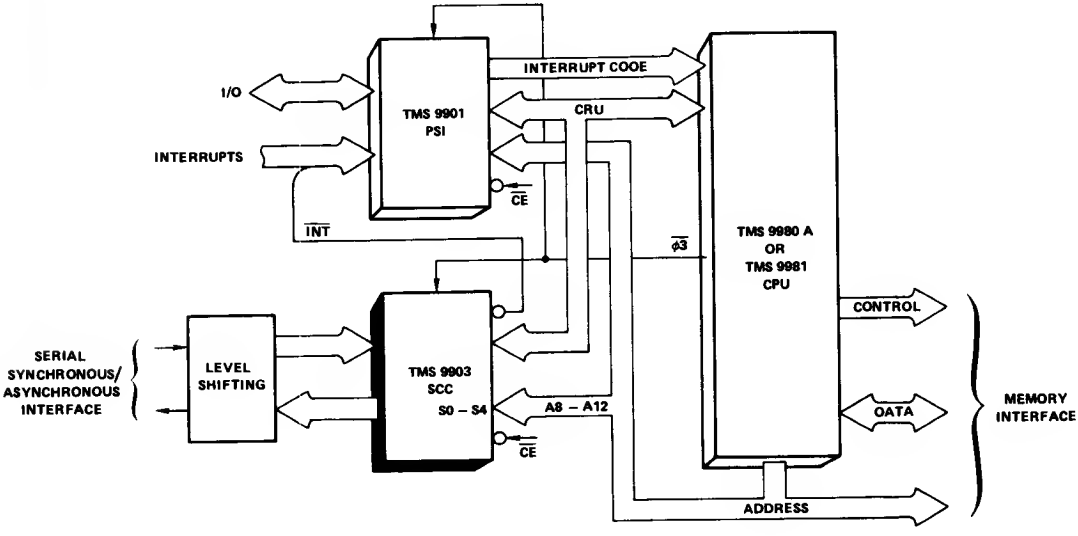


FIGURE 2. TMS 9903 SYNCHRONOUS COMMUNICATION CONTROLLER IN A TMS 9980 A, 9981, SYSTEM

8◀

The SCC interfaces to the synchronous communications channel on seven lines: request to send ($\overline{RTS}$), data set ready ($\overline{DSR}$), clear to send ($\overline{CTS}$), serial transmit data (XOUT), serial receive data (RIN), receiver clock (SCR), and transmitter clock ($\overline{SCT}$). The request to send ($\overline{RTS}$) goes active (LOW) whenever the transmitter is activated. However, before data transmission begins, the clear to send ($\overline{CTS}$) input must be active. The data set ready ($\overline{DSR}$) input does not affect the receiver or transmitter. When $\overline{DSR}$, $\overline{CTS}$, or automatic request-to-send (RTSAUT) changes level, an interrupt is generated, if enabled.

The TMS 9903 is capable of six different modes of operation, including two asynchronous modes. Standard synchronous protocols such as SDLC, HDLC, Bi-Sync, and ADCCP can be directly implemented on the SCC.

2. ARCHITECTURE

The TMS 9903 synchronous communications controller (SCC) is designed to provide a low cost, serial, synchronous or asynchronous interface to the 9900 family of microprocessors. A block diagram for the TMS 9903 is shown in Figure 3. The SCC has five main subsections: CRU interface, transmitter section, receiver section, interval timer, and interrupt section.

2.1 CRU INTERFACE

The communications register unit (CRU) is the means by which the CPU communicates with the TMS 9903 SCC. The SCC occupies 32 bits of CRU read and write space. Figure 4 illustrates the CRU interface between a TMS 9903 and a TMS 9900 CPU; Figure 5 illustrates the CRU interface for a TMS 9980A or TMS 9981 CPU. The CRU lines are tied directly to each other as shown in Figures 4 and 5. The least significant bits of the address bus are connected to the select lines. In a TMS 9900 CPU system A14-A10 are connected to S4-S0 respectively. The most significant address bits are decoded to select the TMS 9903 via the chip enable ($\overline{CE}$) signal. When $\overline{CE}$ is inactive (HIGH), the SCC CRU interface is disabled.

NOTE

When $\overline{CE}$ is inactive (high) the 9903 places the CRUIN line in its high impedence state and disables CRUCLK from coming on chip. Thus CRUIN can be used as an OR tied bus. $\overline{CE}$ being inactive will not disable the select lines from coming on chip, although no device action is taken.

For those unfamiliar with the CRU concept, the following is a discussion of how to build a CRU interface. The CRU is a bit addressable (4096 bits), synchronous, serial interface over which a single instruction can transfer between one and 16 bits serially. Each one of the 4096 bits of the CRU space has a unique address and can be read and written to. During multi-bit CRU transfers, the CRU address is incremented at the beginning of each CRU cycle to point to the next consecutive CRU bit.

When a 99XX CPU executes a CRU Instruction, the processor uses the contents of workspace register 12 as a base address. (Refer to the 9900 Microprocessor Data Manual for a complete discussion on how CRU addresses are derived.) The CRU address is brought out on the 15-bit address bus; this means that the least significant bit of R12 is not brought out of the CPU. During CRU cycles,the memory control lines ($\overline{MEMEN}$,$\overline{WE}$, and DBIN) are all inactive; $\overline{MEMEN}$ being inactive (HIGH) indicates the address is not a memory address and therefore is a CRU address or external instruction code. Also, when $\overline{MEMEN}$ is inactive (HIGH) and a valid address is present, address bits A0-A2 must all be zero to constitute a valid CRU address; if address bits A0-A2 are other than all zeros, they are indicating an external instruction code. In summary, address bits A3-A14 contain the CRU address to be decoded, address bits A0-A2 must be zero and $\overline{MEMEN}$ must be inactive (HIGH) to indicate a CRU cycle.

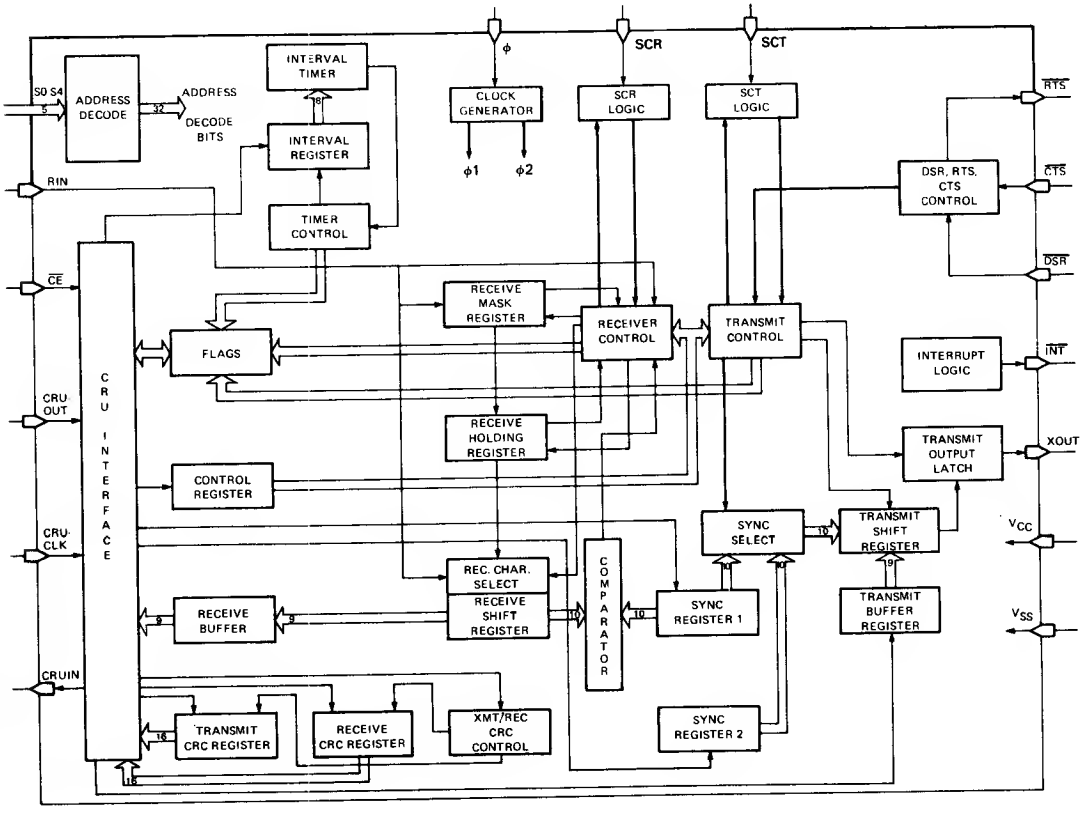


FIGURE 3. TMS 9903 SYNCHRONOUS COMMUNICATION CONTROLLER BLOCK DIAGRAM

8◂

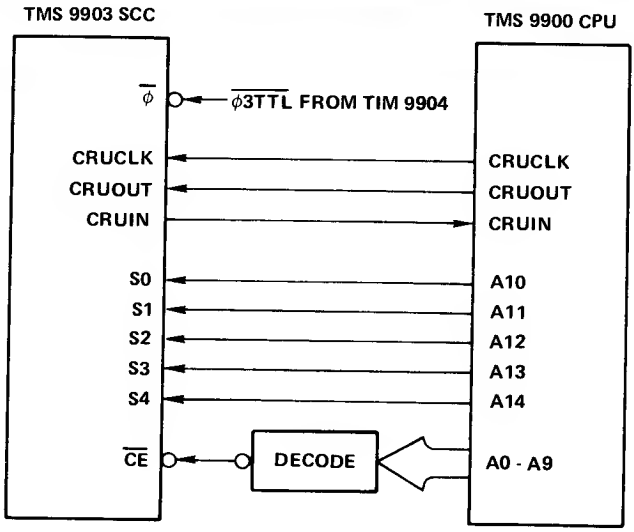


FIGURE 4. TMS 9903 CONTROL SIGNALS (TMS 9900 SYSTEM)

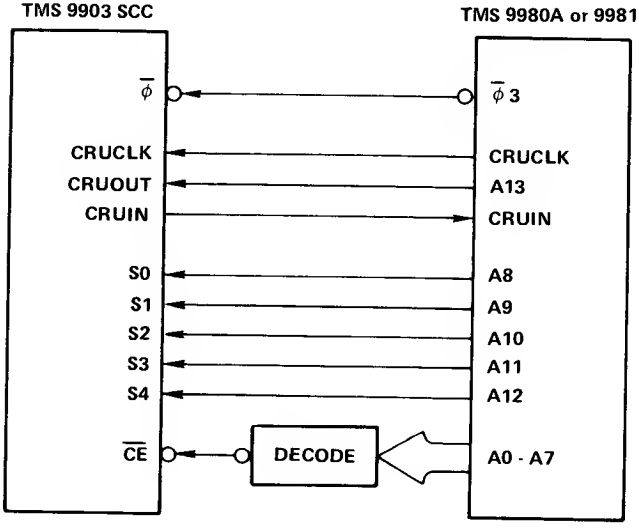


FIGURE 5. TMS 9903 CONTROL SIGNALS (TMS 9980A or 9981 SYSTEM)

TABLE 1. TMS 9903 OUTPUT SELECT BIT ASSIGNMENTS

ADDRESS	NAME	MODE 0	1	2	3	5	6	DESCRIPTION
31	RESET	X	X	X	X	X	X	Reset Device
30	CLRXMT (1)	X	X	X	X	X	X	Clear Transmitter
	CLRRCV (0)	X	X	X	X	X	X	Clear Receiver
29	CLXCRC (1)	X	X	X	X	X	X	Clear Transmit CRC Register
	CLRCRC(0)	X	X	X	X	X	X	Clear Receive CRC Register
28	–	X		X		X	X	Not used
	XZINH		X					Transmit Zero Insertion Inhibit
	RSYNDL				X			Receive Sync Character Delete
27	LDSYN2	X	X	X	X			Load Sync Character Register 2
	–					X	X	Not Used
26	–	X				X	X	Not Used
	RHRRD		X					Receive Holding Register Read
	LDSYN1			X	X			Load Sync Character Register 1
25	LXBC	X	X	X	X	X	X	Load Transmit Buffer and Transmit CRC Register
24	LXCRC	X	X	X	X	X	X	Load Transmit CRC Register
23	XPRNT	X	X		X			Transparent
	–			X				Not Used
	BRKON					X	X	Break On
22	XAINB	X	X					Transmitter Abort Interrupt Enable
	–			X	X	X	X	Not Used
21	DSCENB	X	X	X	X	X	X	Data Set Status Change Interrupt Enable
20	TIMENB	X	X	X	X	X	X	Timer Interrupt Enable
19	XBIENB	X	X	X	X	X	X	Transmitter Buffer Register Empty Interrupt Enable
18	RIENB	X	X	X	X	X	X	Receiver Interrupt Enable
17	RTS	X	X	X	X	X	X	Request To Send
16	XMTON	X	X	X	X	X	X	Transmitter On
15	TSTMD	X	X	X	X	X	X	Test Mode
14	LDCTRL	X	X	X	X	X	X	Load Control Register
13	LDIR	X	X	X	X	X	X	Load Interval Register
12	LRCRC	X	X	X	X	X	X	Load Receive CRC Register
11-0	DATA							Data To Selected Register

8◀

2.1.1 CPU Output for CRU

The TMS 9903 SCC occupies 32 bits of output CRU space, of which all are used. These bits are employed by the CPU to communicate command and control information to the TMS 9903. Table 1 shows the mapping between CRU address select (S lines) and SCC functions by operational mode; modes 4 and 7 are not implemented. Each CRU addressable output bit on the TMS 9903 is described in detail following Table 1.

Bit 31

All modes (RESET)— **Reset.** Writing a one or zero to bit 31 causes the device to reset, disabling all interrupts, initializing all controllers, and resetting all flags except LDCTRL and XBRE which are set.

Bit 30

All modes (CLRXMT)— **Clear Transmitter.** Writing a one to bit 30 initializes the transmitter and disables transmit interrupts.

(CLRRCV)— **Clear Receiver.** Writing a zero to bit 30 initializes the receiver and clears all receive interrupts.

Bit 29

All modes (CLXCRC)— **Clear Transmit CRC Register (XCRC).** Writing a one to bit 29 in all modes clears the XCRC register to all zeros.

(CLRCRC)— **Clear Receive CRC Register (RCRC).** Writing a zero to bit 29 in all modes clears the RCRC register to all zeros.

Bit 28

Modes 0, 2, 5, 6 — Not Used.

Modes 1 (XZINH)— **Transmit Zero Insertion Inhibit.** Writing a one to bit 28 in mode 1 causes the contents of the transmit buffer register (XBR) to be transmitted without the insertion of a zero after five consecutive ones. Writing a zero to bit 28 in mode 1 causes the transmitter to insert a zero after five consecutive ones are transmitted.

Mode 3 (RSYNDL)— **Received Sync Character Delete.** Writing a one to bit 28 in mode 3 causes received characters which are identical to the contents of sync character register 1 (SYNC1) to be ignored. This function is disabled when XPRNT (bit 23) is set. Writing a zero to bit 28 in mode 3 causes RSYNDL to be reset.

Bit 27

Modes 0, 1, 2, 3 (LDSYN2)— **Load Sync Character Register 2.** Writing a one to bit 27 in mode 0, 1, 2, or 3 enables loading of sync character register 2 (SYNC2) from output bit addresses 0–9. Writing a zero to bit 27 in mode 0, 1, 2, 3 resets LDSYN2.

Modes 5, 6 — Not Used.

Bit 26

Modes 0, 5, 6 — Not Used.

Mode 1 (RHRRD)— **Receive Holding Register Read.** Writing a one to bit 26 in mode 1 enables reading of the receive–holding register (RHR) contents at input bit addresses 0–15. Writing a zero to bit 26 in mode 1 resets RHRRD, RHRL (receive holding register loaded), RHROV (receive holding register overrun), and RZER (receive zero error).

Modes 2, 3 (LDSYN1)— **Load Sync Character Register 1.** Writing a one to bit 26 in mode 2 or 3 enables loading of sync character register 1 (SYNC1) from output bit addresses 0–9. Writing a zero to bit 26 in mode 2 or 3 resets LDSYN1.

Bit 25

All modes (LXBC)— **Load Transmit Buffer and CRC Register.** Writing a one to bit 25 in all modes enables loading of XBR (transmit buffer register) and XCRC (transmit CRC register) from output bit addresses 0–8, and enables reading of XCRC at input bit addresses 0–15. Writing a zero to bit 25 in all modes resets LXBC an XBRE (transmit buffer register empty).

Bit 24

All modes (LXCRC)— **Load Transmit CRC Register.** Writing a one to bit 24 in all modes enables loading the XCRC register from output bit addresses 0–9, and enables reading XCRC at input bit addresses 0–15. Writing a zero to bit 24 in all modes resets LXCRC.

Bit 23

Mode 0 (XPRNT)— **Transparent.** Writing a one to bit 23 in mode 0 causes the contents of SYNC2 to be transmitted whenever no data is available and the transmitter is active. Writing a zero to bit 23 in mode 0 causes the transmitter abort signal (XABRT) to set and transmitter operation to be suspended when no data is available and the transmitter is active.

Mode 1 (XPRNT)— **Transparent.** Writing a one to bit 23 in mode 1 causes the contents of SYNC2 to be transmitted without zero insertion when no data is available and the transmitter is active. Writing a zero to bit 23 in mode 1 causes XABRT to be set and transmit operations to be suspended when no data is available.

Mode 2 — Not Used.

Mode 3 (XPRNT)— **Transparent.** Writing a one to bit 23 in mode 3 causes fill sequence of (contents of SYNC2) followed by (contents of SYNC1) to be transmitted when no data is available. Writing a zero to bit 23 in mode 3 causes the fill sequence of (contents of SYNC1) followed by (contents of SYNC1) to be transmitted when no data is available.

Modes 5 and 6 (BRKON)— **Break ON.** Writing a one to bit 23 in mode 5 or 6 causes the output to go to a constant zero level when no data is available and the transmitter is active. Writing a zero to bit 23 in mode 5 and 6 causes BRKON to be reset. The transmit buffer register should not be loaded during transmission of a break.

INTERRUPT ENABLE FLAGS

INTERRUPT ENABLE	SELECT BIT	INTERRUPT FLAG	INTERRUPT NAME	DESCRIPTION
XAIENB	22	XABRT	XAINT	Transmitter Abort
DSCENB	21	DSCH	DSCINT	Data Set Status Change (CTS, RTS, RTSAUT)
TIMENB	20	TIMELP	TIMINT	Timer Elapsed
XBIENB	19	XBRE	XBINT	Transmit Buffer Register Empty
RIENB	18	RBRL	RINT	Receiver Buffer Register Loaded
RIENB	18	RHRL	RINT	Receiver Holding Register Loaded
RIENB	18	RABRT	RINT	Receiver Abort

Refer to Section 2.6

Bit 22
Modes 0 and 1 (XAIENB)—

Transmitter Abort Interrupt Enable. Writing a one to bit 22 in mode 0 or 1 resets XABRT (transmitter abort) and enables XABRT interrupts. Writing a zero to bit 22 in mode 0 or 1 resets XABRT and disables XABRT interrupts.

Modes 2, 3, 5 and 6

Not Used.

Bit 21
All modes (DSCENB)—

Data Set Status Change Interrupt Enable. Writing a one to bit 21 in all modes resets DSCH (data set status change) and enables DSCH interrupts. Writing a zero to bit 21 in all modes resets DSCH and disables DSCH interrupts.

Bit 20
All modes (TIMENB)—

Timer Interrupt Enable. Writing a one to bit 20 in all modes resets TIMELP (timer elapsed) and TIMERR (timer error) and enables TIMELP interrupts. Writing a zero to bit 20 in all modes resets TIMELP and TIMERR and disables TIMELP interrupts.

Bit 19
All modes (XBIENB)—

Transmit Buffer Register Empty Interrupt Enable. Writing a one to bit 19 in all modes enables XBRE interrupts. Writing a zero to bit 19 in all modes disables XBRE interrupts.

Bit 18
Modes 0, 2, 3, 5, 6 (RIENB)—

Receiver Interrupt Enable. Writing a one to bit 18 in mode 0, 2, 3, 5, or 6 resets RBRL (receive buffer register loaded) and ROVER (receiver overrun), and enables RBRL interrupts. Writing a zero to bit 18 in mode 0, 2, 3, 5, 6 resets RBRL and ROVER, and disables RBRL interrupts.

▶8

Mode 1 (RIENB)—

Receiver Interrupt Enable. Writing a one to bit 18 in mode 1 resets RBRL, RFLDT, ROVER, and RABRT (receiver abort), and enables RBRL, RABRT, and RHRL (receive holding register loaded) interrupts. Writing a zero to bit 18 in mode 1 resets RBRL, RFLDT, ROVER, and RABRT, and disables RBRL, RABRT, and RHRL interrupts.

Bit 17
All modes (RTS)—

Request to Send. Writing a one to bit 17 in all modes resets the $\overline{RTS}$ output and disables automatic control of $\overline{RTS}$ by the internal RTSAUT (automatic RTS control) signal. Writing a zero to bit 17 in all modes sets the $\overline{RTS}$ output HIGH and disables automatic control by RTSAUT.

Bit 16
All modes (XMTON)—

Transmitter On. Writing a one to bit 16 in all modes enables data transmission. Writing a zero to bit 16 in all modes disables data transmission when no data is available.

Bit 15
All modes (TSTMD)—

Test Mode. Writing a one to bit 15 in all modes causes the timer to decrement at 32 times the normal rate, and internally connects XOUT to RIN, RTSAUT to CTS, and SCR to $\overline{SCT}$. $\overline{SCT}$ is internally generated at the frequency to which TIMELP is set. Writing a zero to bit 15 in all modes resets TSTMD and enables normal device operation. The test mode should not be used in a loop configuration of mode 1; test mode is useful for testing and inspection purposes.

TABLE 2. REGISTER LOAD CONTROL FLAGS

FLAG*	CRUOUT BIT ADDRESS	REGISTER LOADED	BITS/REGISTER
LDSYN2	27	Sync Register 2 (SYNC2)	10
LDSYN1	26	Sync Register 1 (SYNC1)	10
LXBC	25	Xmt CRC Register (XCRC) and Xmt Buffer Reg. (XBR)	9
LXCRC	24	XCRC	10
LDCTRL	14	Control Register (CTRL)	12
LDIR	13	Interval Register	8
LRCRC	12	Receive CRC Register (RCRC)	10
None	—	XBR	9

*It is recommended that no more than one register load control flag be set at any one time.

Bit 14
All modes (LDCTRL)—

Load Control Register. Writing a one to bit 14 in all modes enables the loading of the control register from output bit addresses 0–11. Writing a zero to bit 14 in all modes resets LDCTRL. When a bit is written to select bit 11 (when loading the control register), the LDCTRL flag is automatically reset.

Bit 13

All modes (LDIR)—

Load Interval Register. Writing a one to bit 13 in all modes enables the loading of the interval register from output bit addresses 0–7. Writing a zero to bit 13 in all modes resets LDIR and causes the contents of the interval register to be loaded into the interval timer.

8◀

Bit 12
All modes (LRCRC)—

Load Receive CRC Register. Writing a one to bit 12 in all modes enables the loading of the receive CRC register from output bit addresses 0–9, and enables reading the RCRC (receive CRC register) on input bit addresses 0–15. Writing a zero to bit 12 in all modes resets LRCRC.

2.1.2 Control and Data Registers

Loading of the internal control and data registers is controlled by one of the single bit control function flags described in Section 2.1.1 and summarized in Table 2. The registers must be carefully loaded to ensure that no more than one flag is set at a time. Unlike the TMS 9902, when the MSB of a register is loaded, the load flag is not automatically reset except for the control register which is the only register which will automatically reset the load flag when the MSB of the register is written to.

The TMS 9903 SCC is capable of performing dynamic character length operations. The receive character length is set by bits 2–0 of the control register. Transmitted character and sync character registers are maintained internally to determine the character length. The length of the character to be transmitted is determined by the number of bits loaded into the transmit buffer register before the transmit buffer register empty flag is reset. Similarly, the character length of the two sync registers is determined by the number of bits loaded into the most recently loaded SYNC character. Thus, for transmission purposes the length of the two SYNC characters is the same. NOTE: When the receiver is comparing received data to SYNC1, only the number of bits selected as the receive character length are compared [i.e., RSCL (2–0) plus parity, if enabled].

2.1.2.1 Control Register

The control register is loaded to select the mode, configuration, CRC polynomial, received character length, data rate clock, and internal device clock rates of the TMS 9903. Table 3 shows the bit address assignments for the control register.

TABLE 3. CONTROL REGISTER BIT ADDRESS ASSIGNMENTS

ADDRESS (S0-S4)	NAME	DESCRIPTION
11	DRCK32	32X Data Rate Clock
10	CRC1	CRC Polynomial Select
9	CRC0	
8	MDSL2	Mode Select
7	MDSL1	
6	MDSL0	
5	CSL1	Configuration Select
4	CSL0	
3	CLK4M	4X System Clock Select
2	RSCL2	Receive Character Length Select
1	RSCL1	
0	RSCL0	

11	10	9	8	7	6	5	4	3	2	1	0
DRCK32	CRC1	CRC0	MDSL2	MDSL1	MDSL0	CLS1	CLS0	CLK4M	RSCL2	RSCL1	RSCL0

MSB LSB

CONTROL REGISTER BIT ADDRESS

Bit 11

Modes 0, 1, 2, 3 (DRCK32)— **32X Data Rate Clock.** Setting control bit 11 to one in mode 0, 1, 2, or 3 sets the SCT frequency at 32 times the transmit–data rate and the SCR frequency at 32 times the receive–data rate. SCR is set to resync on every transition of RIN. Also, if bit 11 is a one, zero–complementing NRZI data encoding is used (to send a one, the signal remains in the same state; to send a zero, the signal changes state). Setting bit 11 to zero in mode 0, 1, 2, or 3 causes the receive data to be sampled on every zero–to–one transition of SCR, and the transmit data to be shifted out on the one–to–zero transition of SCT. DRCK32 should always be reset when in a loop configuration of mode 1.

Modes 5, 6 (DRCK32)— **32X Data Rate Clock.** Setting control bit 11 to one in mode 5 or 6 sets the SCT frequency to 32 times the transmit data rate, and the SCR frequency to 32 times the receive data rate. SCR is resynched on every start bit received. Setting control bit 11 to zero in mode 5 or 6 causes receive data to be sampled on the zero–to–one transition of SCR, and transmit data to be shifted out on the one–to–zero transition of $\overline{SCT}$.

All modes — Setting control bit 11 to a one or zero resets LDCTRL (load control register). The control register is the only register that resets its load flag in this fashion.

Bits 10 and 9

All modes (CRC1 and CRC0)— **CRC Polynomial Select.** The polynomial used in the generation of the transmit and receive CRC's is selected by bits 10 and 9 of the control register, as shown below.

CRC POLYNOMIAL BIT SELECT

CRC	CRC1	CRC0	NAME	POLYNOMIAL
0	0	0	CRC-16	$X^{16}+X^{15}+X^{2}+1$
1	0	1	CRCC-12*	$X^{12}+X^{11}+X^{3}+X^{2}+X+1$
2	1	0	REV. CRCC-16	$X^{16}+X^{14}+X+1$
3	1	1	CRC-CCITT	$X^{16}+X^{12}+X^{5}+1$

*NOTE When using CRCC-12, the four most-significant bits of the CRC register will contain data that must be masked to assure validity of CRC comparisons.

8

Bits 8, 7 and 6
All modes (MDSL2, MDSL1, MDSL0)

Mode Select. The mode of operation for the transmitter and receiver is selected by bits 8, 7, and 6 of the control register as shown below.

MODE BIT SELECT

MODE	MDSL2	MDSL1	MDSL0	EXAMPLE PROTOCOL	SYNC CHARACTER	FILL-CHARACTER
0	0	0	0	GENERAL	NONE	(SYNC2) or NONE
1	0	0	1	SDLC	$7E_{16}$	(SYNC2) or NONE
2	0	1	0	GENERAL	(SYNC1)	(SYNC2)
3	0	1	1	BI-SYNC	(SYNC1-SYNC1)	(SYNC1-SYNC1) or (SYNC2-SYNC1)
4	1	0	0	NOT USED		
5	1	0	1	ASYNCHRONOUS OPERATION WITH TWO STOP BITS		
6	1	1	0	ASYNCHRONOUS OPERATION WITH ONE STOP BIT		
7	1	1	1	NOT USED		

Bits 5 and 4
All modes (CSL1, CSL0)—

Configuration Select. The configuration of the transmitter and receiver within each mode is set by bits 5 and 4 of the control register, as shown below. CSL1 is forced to zero on RESET.

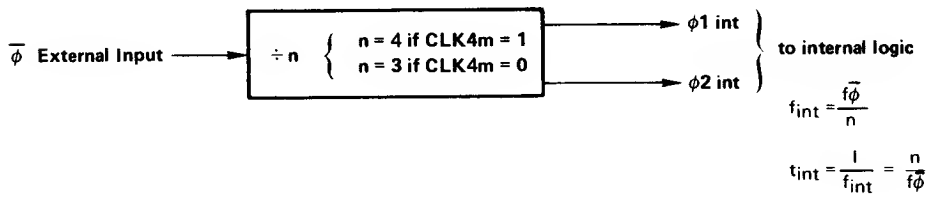
TRANSMIT/RECEIVE CONFIGURATION BIT SELECT

CONFIGURATION	CSL1	CSL0	MODE 0	1	2	3	5	6	DESCRIPTION
0	0	0	X	X	X	X	X	X	No Parity Generation or Detection SDLC Normal (Non-Loop)
1	0	1	X	X	X	X	X	X	No Parity Generation or Detection SDLC Loop Master
2	1	0	X	X	X	X	X	X	Even Parity Generated on Transmission and Detected on Reception SDLC Loop Slave — Pending Synchronization
3	1	1	X	X	X	X	X	X	Odd Parity Generated on Transmission and Detected on Reception SDLC Loop Slave — Active

▸8

Bit 3
All modes (CKL4M)—

Input Divide Select. The $\overline{\phi}$ input to the TMS 9903 SCC is used to generate internal dynamic logic clocking and to establish the time base for the interval timer. The $\overline{\phi}$ input is internally divided by either 3 or 4 to generate the two phase internal clocks required for MOS logic, and to establish the basic internal operating frequency (f_{int}) and internal clock period (t_{int}). When bit 3 of the control register is set to a logic one (CLK4M = 1), $\overline{\phi}$ is internally divided by 4, and when CLK4M = 0, $\overline{\phi}$ is divided by 3. For example, when $f\overline{\phi}$ = 3 MHz, (as in a standard 3 MHz TMS 9900 system) and CLK4M = 0, $\overline{\phi}$ is internally divided by 3 to generate an internal clock period t_{int} of 1 μs. The figure below shows the operation of the internal clock divider circuitry. The internal clock frequency should be no greater than 1.1 MHz; thus, when $f\overline{\phi}$ > 3.3 MHz, CLK4M should be set to a logic one.



INTERNAL CLOCK DIVIDER CIRCUITRY

Bits 2, 1, and 0
All modes (RSCL2, RSCL1, and RSCL0)

Received Character Length Select. The number of data bits in each received character is determined by bits 2, 1, and 0 of the control register, as shown below.

RECEIVE CHARACTER LENGTH SELECTION

RSCL2	RSCL1	RSCL0	BITS/CHAR.
0	0	0	5
0	0	1	6
0	1	0	7
0	1	1	8
1	0	0	9

8

2.1.2.2 Interval Register

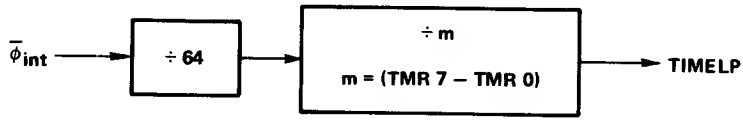
The interval register is enabled for loading whenever LDIR = 1. The interval register is used to select the rate at which timer interrupts are generated by the SCC interval timer. The figure below shows the bit–address assignments for the interval register when enabled for loading.

7	6	5	4	3	2	1	0
TMR 7	TMR 6	TMR 5	TMR 4	TMR 3	TMR 2	TMR 1	TMR 0
MSB							LSB

INTERNAL-REGISTER BIT ADDRESSES

The figure below illustrates the establishment of the interval for the timer. For example, if the interval register is loaded with a value of $40_{16}(64_{10})$ with t_{int} = 1 μs, the interval at which the timer decrements to zero and interrupts the CPU is

$$t_{interval} = t_{int} \times 64 \times M$$
$$= (1\ \mu s) \times 64 \times 64$$
$$= 4.096\ ms$$



A0001472

TIME INTERVAL SELECTION

2.1.2.3 Receive CRC Register

The receive CRC register is enabled for loading when LRCRC = 1. The receive CRC register is used to verify data integrity in the synchronous communication channel. When LRCRC is set, output to bit address 0–9 updates the contents of the receive CRC register according to the CRC polynomial selected by the control register. Also, when LRCRC is set, the receive CRC register can be read on CRU input addresses 0–15. When read, the MSB of the register is read first, and the LSB is read last. The receive CRC register block diagram is shown in Figure 6.

NOTE

Single bits of the CRC registers cannot be accessed. As individual bits are sent to or read from the CRC registers, they are shifted in or out, respectively, of the register from CRC bit 16.

2.1.2.4 Transmit CRC Register

The transmit CRC register is enabled for loading when either LXCRC = 1 (load transmit CRC register) or LXBC = 1 (load transmit buffer register and transmit CRC register). When either LXBC or LXCRC is set, output to bit addresses 0–9 updates the contents of the transmit CRC register according to the CRC polynomial selected by the control register. When set, the LXBC or LXCRC flag selects the transmit CRC register contents to be read by the CPU at input addresses 0–15. LXBC and LXCRC flags are reset by a command from the CPU.

Operation of the transmit CRC register is analogous to that of the receive CRC register shown in Figure 6.

▶8 *2.1.2.5 Sync Character Register 1*

Sync character register 1 is enabled for loading when LDSYN1 = 1. The sync character register 1 is used for synchronization and as a fill sequence for the transmitter. When LDSYN1 is set, output to bit addresses 0–9 is loaded into sync character register 1. The LDSYN1 flag is reset by a command from the CPU.

2.1.2.6 Sync Character Register 2

Sync character register 2 is enabled for loading whenever LDSYN2 = 1. The contents of sync character register 2 are used for a fill sequence for the transmitter. When LDSYN2 is set, output to bit addresses 0–9 is loaded into sync character register 2. The LDSYN2 flag is reset by a command from the CPU.

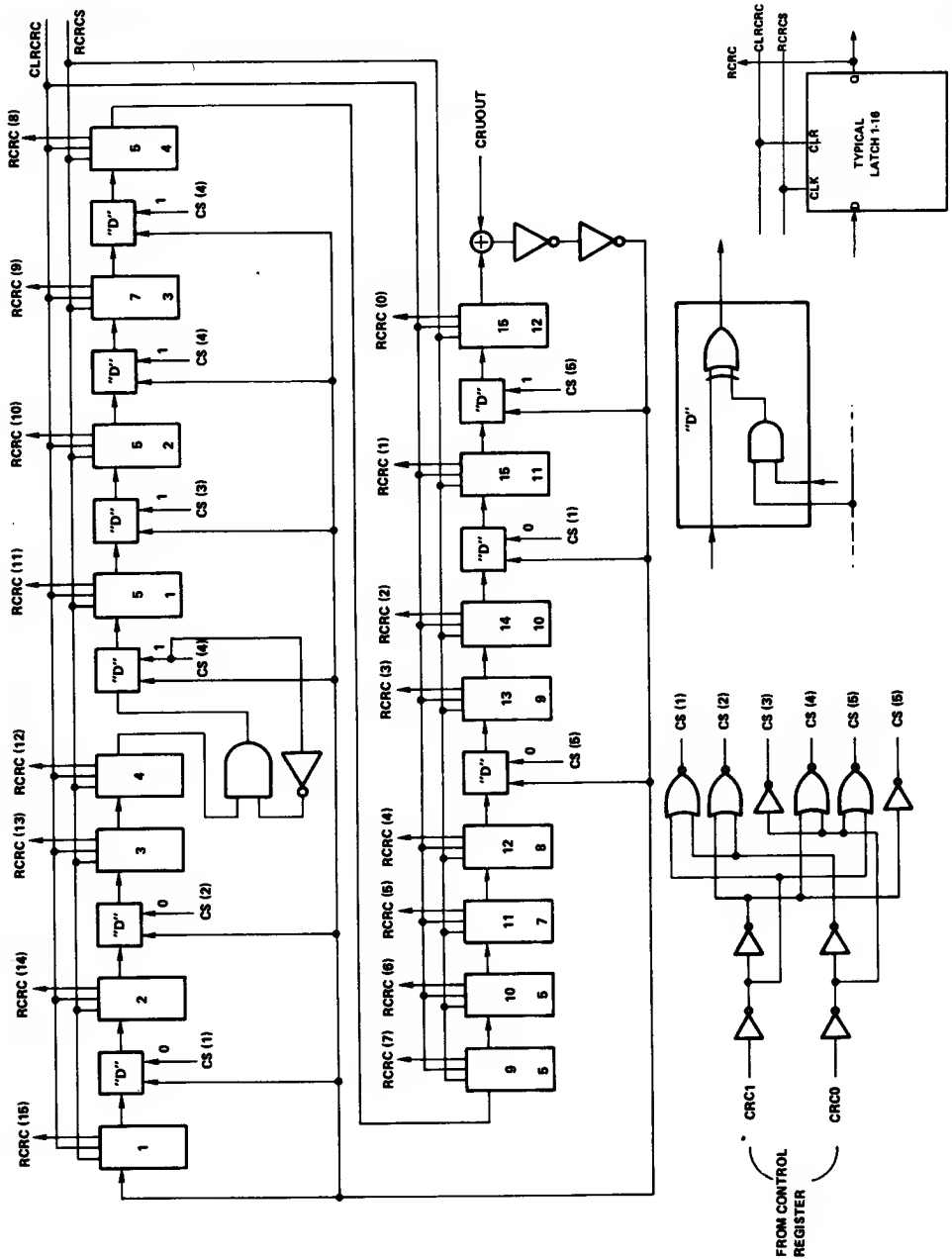


FIGURE 8. RECEIVE CRC REGISTER DIAGRAM

8

2.1.2.7 *Transmit Buffer Register*

Two conditions enable the transmit buffer register for loading. If all flags are zero or if LXBC = 1, the transmit buffer register is enabled for loading. The transmit buffer is used for the storage of the next character to be transmitted. When the transmitter is active, the contents of the transmit buffer register are transferred to the transmit shift register when the previous character has been completely transmitted. When LXBC is set, the output to bit addresses 0–8 loaded into the transmit buffer register simultaneously updates the contents of the transmit CRC register, according to the CRC polynomial selected by the control register. Also, when LXBC is set, the transmit CRC register contents are enabled for reading on input–bit addresses 0–15. The LXBC flag is reset by a command from the CPU.

8	7	6	5	4	3	2	1	0
XBR 8	XBR 7	XBR 6	XBR 5	XBR 4	XBR 3	XBR 2	XBR 1	XBR 0
MSB								LSB

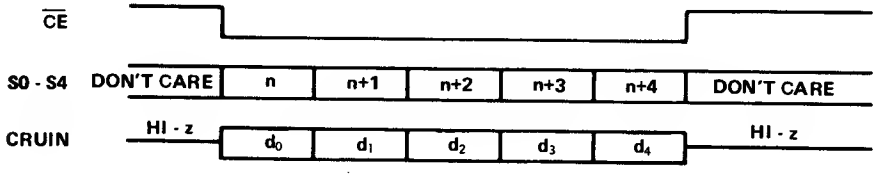
TRANSMIT BUFFER REGISTER BIT ADDRESSES

TABLE 4. CRU OUTPUT SELECT BIT ASSIGNMENTS

ADDR	LDCTRL = 1	LDSYN1 = 1	LDSYN2 = 1	LDIR = 1	LRCRC = 1	LXCRC = 1	LXBC = 1		ADDR FLAGS=0
11	DRCK32								
10	CRC(1)								
9	CRC(0)	S1OD(9)	S2OD(9)		RCRC(9)	XCRC(9)			
8	MDSL(2)						XCRC(8)	XBR(8)	XBR(8)
7	MDSL(1)			IROD(7)					
6	MDSL(0)								
5	CSL(1)								
4	CSL(0)								
3	CLK4M								
2	RSCL(2)								
1	RSCL(1)								
0	RSCL(0)	S1OD(0)	S2OD(0)	IROD(0)	RCRC(0)	XCRC(0)	XCRC(0)	XBR(0)	XBR(0)

2.1.3 Status and Data Input to CPU

Status and data information is read from the SCC by the CPU on the CRUIN line whenever $\overline{CE}$ is active (LOW). The input bit is selected from one of 32 input bit addresses using the five address lines S0–S4. When $\overline{CE}$ is high, CRUIN is in its high–impedance state, permitting the CRUIN control line to be wire–ORed with other input devices. The following figure illustrates the relationships of the signals used to access data from the SCC. Table 5 describes the input select bit address assignments of the SCC. Following Table 5 is a detailed discussion of each bit. Addresses 0–15 can be read as shown only when all load flags are reset.



TMS 9903 DATA ACCESS SIGNALS

TABLE 5. TMS 9903 INPUT BIT ADDRESS ASSIGNMENTS

ADDRESS	NAME	MODE 0	1	2	3	5	6	DESCRIPTION
31	INT	X	X	X	X	X	X	Interrupt
30	FLAG	X	X	X	X	X	X	Any Register Load Control Flag Set
29	DSCH	X	X	X	X	X	X	Data Set Status Change
28	CTS	X	X	X	X	X	X	Clear to Send
27	DSR	X	X	X	X	X	X	Data Set Ready
26	RTSAUT	X	X	X	X	X	X	Automatic Request to Send
25	TIMELP	X	X	X	X	X	X	Timer Elapsed
24	TIMERR	X	X	X	X	X	X	Timer Error
23	XABRT	X	X					Transmitter Abort
	—			X	X	X	X	Not Used
22	XBRE	X	X	X	X	X	X	Transmit Buffer Register Empty
21	RBRL	X	X	X	X	X	X	Receive Buffer Regiater Loaded
20	DSCINT	X	X	X	X	X	X	Data Set Status Change Interrupt
19	TIMINT	X	X	X	X	X	X	Timer Interrupt
18	XAINT	X	X					Transmit Abort Interrupt
	—			X	X	X	X	Not Used (Always = 0)
17	XBINT	X	X	X	X	X	X	Transmit Buffer Interrupt
16	RINT	X	X	X	X	X	X	Receiver Interrupt
15	RIN	X	X	X	X	X	X	Receiver Input
14	RABRT		X					Receive Abort
	—	X		X	X			Not Used (Always = 0)
	RSBD					X	X	Receive Start Bit Detect
13	RHRL		X					Receive Holding Register Loaded
	—	X		X	X			Not Used (Always = 0)
	RFBD					X	X	Receive Full Bit Detect
12	RHROV		X					Receive Holding Register Overrun
	—	X		X	X			Not Used (Always = 0)
	RFER					X	X	Receive Framing Error
11	ROVER	X	X	X	X	X	X	Receive Overrun
10	RPER	X		X	X	X	X	Receive Parity Error
	RZER		X					Receive Zero Error
9	RCVERR	X		X	X	X	X	Receive Error
	RFLDT		X					Receive Flag Detect
8-0	RBR	X	X	X	X	X	X	Receive Buffer Register (Received Data)

8

Bit 31
All modes (INT)—

Interrupt. All modes INT = DSCINT + TIMINT + RBINT + XAINT + XBINT. The interrupt output control line (INT) is active when this status signal is a logic one.

Bit 30
All modes (FLAG)—

Register Load Control Flag Set. In all modes FLAG = LDCTRL + LDSYN1 + LDSYN2 + LDIR + LRCRC + LXBC + LXCRC. Flag = 1 when any of the register load control flags is set.

Bit 29
All modes (DSCH)—

Data Set Status Change. In all modes DSCH is set when the $\overline{DSR}$ or $\overline{CTS}$ inputs, or RTSAUT changes state. To ensure recognition of the state change, $\overline{DSR}$ or $\overline{CTS}$ must remain stable in its new state for a minimum of two internal clock cycles. DSCH is reset by an output to output bit 21 (DSCENB).

Bit 28
All modes (CTS)—

Clear To Send. The CTS signal indicates the inverted status of the $\overline{CTS}$ device input in all modes.

Bit 27
All modes (DSR)—

Data Set Ready. The DSR signal indicates the inverted status of the $\overline{DSR}$ device input in all modes.

Bit 26
All modes (RTSAUT)—

Automatic Request to Send. The RTSAUT signal indicates the output status of RTSAUT, the automatic RTS controller in all modes.

Bit 25
All modes (TIMELP)—

Timer Elapsed. The TIMELP signal is set in all modes each time the interval timer decrements to 0. TIMELP is reset by an output to output bit 20 (TIMENB).

Bit 24
All modes (TIMERR)—

Timer Error. The TIMERR signal is set in all modes when the selected time interval elapses and TIMELP is already set. TIMELP is reset by an output to output bit address 20 (TIMENB).

Bit 23
Modes 0, 1 (XABRT)—

Transmitter Abort. The XABRT signal is set by the transmitter in modes 0 and 1 when no data is available for transmission and no provisions have been made to identify a fill sequence (i.e., XPRNT is not set). XABRT is reset by an output to output bit address 22 (XAIENB).

▶8

Modes 2, 3, 5, 6

Not Used.

Bit 22
All modes (XBRE)—

Transmit Buffer Register Empty. The XBRE signal is set in all modes when the transmit buffer register (XBR) contents are transmitted to the transmit shift register (XSR) and when the transmitter is initialized. XBRE is reset by a zero output to output bit 25 (LXBC).

Bit 21
All modes (RBRL)— **Receive Buffer Register Loaded.** The RBRL signal is set in all modes when a complete character has been transferred from the receive shift register (RSR) to the RBR. RBRL is reset by an output to output bit 18 (RIENB).

Bit 20
All modes (DSCINT)— **Data Set Status Change Interrupt.** In all modes DSCINT = DSCH (input bit 29) AND DSCENB (output bit 21). DSCINT indicates the presence of an enabled interrupt caused by the change in status of $\overline{DSR}$, RTSAUT, or $\overline{CTS}$.

Bit 19
All modes (TIMINT)— **Timer Interrupt.** In all modes TIMINT = TIMELP (input bit 25) AND TIMENB (output bit 20). TIMINT indicates the presence of an enabled interrupt caused by the interval timer.

Bit 18
Modes 0, 1 (XAINT)— **Transmit Abort Interrupt.** In modes 0 and 1 XAINT = XABRT (input bit 23) AND XAIENB (output bit 22). XAINT indicates the presence of an enabled interrupt caused by a transmitter abort.

Modes 2, 3, 5, 6 Not Used.

Bit 17
All modes (XBINT)— **Transmit Buffer Interrupt.** In all modes XBINT = XBRE (input bit 22) AND XMTON (output bit 16) AND XBIENB (output bit 19). XBINT indicates the presence of an enabled interrupt caused by an empty transmit–buffer.

Bit 16
All modes (RINT)— **Receiver Interrupt.** In all modes RINT = [RBRL (input bit 21) OR RHRL (input bit 13) OR RABRT (input bit 14)] AND RIENB (output bit 18). RINT indicates the presence of an enabled interrupt caused by a loaded receive buffer or a loaded receive holding register or a receiver abort (mode 1 only).

Bit 15
All modes (RIN)— **Receiver Input.** In all modes RIN indicates the status of the RIN input to the device.

Bit 14
Mode 1 (RABRT)— **Receiver Abort.** RABRT is set in mode 1 when a flag sequence (01111110) has been previously detected and seven consecutive ones are received. RABRT is reset by an output to output bit address 18 (RIENB).

Modes 5, 6 (RSBD)— **Receive Start Bit Detect.** In modes 5 and 6 RIN is sampled one half-bit time after the one-to-zero transition of RIN. If RIN is still zero at such time, RSBD is set, indicating the start of a character. RSBD remains true until the complete character has been received. If RIN is not zero at the half-bit time, RSBD remains reset and the receiver waits for the next one-to-zero transition of RIN. This bit is normally used for testing purposes.

Modes 0, 2, 3 Not Used, (always equals zero)

Bit 13

Mode 1 (RHRL)— **Receive Holding Register Loaded.** RHRL is set in mode 1 when the receiver has received a complete frame. RHRL is reset by the output of a zero to output bit address 26, RHRRD (receive holding register read).

Modes 5, 6 (RFBD)— **Receive Full Bit Detect.** RFBD is set in modes 5 and 6 one full bit time after RSBD is set to indicate the sampling point for the first data bit of the received character. RFBD is reset when the character has been completely received. This bit is normally used for testing purposes.

Modes 0, 2, 3— Not Used (always equals zero).

Bit 12

Mode 1 (RHROV)— **Receive Holding Register Overrun.** RHROV is set in mode 1 when the contents of the RHR are altered before RHRL is reset. RHROV is reset by the output of a zero to output bit address 26 (RHRRD).

Modes 5, 6 (RFER)— **Receive Framing Error.** RFER is set in modes 5 and 6 when a character is received in which the stop bit, which should be a logic one, is a logic zero. RFER should only be read when RBRL (input bit 21) is a logic one. RFER is reset when a character with the correct stop bit is received.

Modes 0, 2, 3 Not Used (always equals zero).

Bit 11

All modes (ROVER)— **Receiver Overrun.** ROVER is set in all modes when the RBR (receive buffer register) is loaded with a new character before RBRL is reset, indicating that the CPU failed to read the previous character and reset RBRL before the present character is completely received. ROVER is reset when a character is received and RBRL = 0 when the character is transferred to the RBR or an output to output bit address 18 (RIENB).

Bit 10

Modes 0, 2, 3, 5, 6 (RPER)— **Receive Parity Error.** RPER is set in mode 0, 2, 3, 5, and 6 when the character transferred to the RBR was received with incorrect parity. RPER is reset when a character with correct parity is transferred to the RBR.

Mode 1 (RZER)— **Receive Zero Error.** RZER is set in mode 1 when the last five bits received prior to the FLAG character ($7E_{16}$) are all ones without being followed by a zero. RZER is reset by resetting RHRRD (Receiver Hold Register Read).

Bit 9

▶8 Modes 0, 2, 3, 5, 6 (RCVERR)— **Receive Error.** In modes 0, 2, 3, 5, and 6 RCVERR = ROVER OR RPER OR RFER. RCVERR indicates the presence of an error in the most recently received character.

Mode 1 (RFLDT)— **Receive Flag Detect.** RFLDT is set in mode 1 when the FLAG character ($7E_{16}$) is detected in the input stream. RFLDT is reset by an output to output bit address 18 (RIENB). RFLDT is also set when RABRT is set.

Bit 8 - Bit 0

All modes (RBR8-RBR0)— **Receive Buffer Register.** The receive-buffer register contains the most recently received complete character. For received character lengths of fewer than nine bits, the character is right justified to the LSB position (RBR0), with unused most-significant bit(s) all zero(s). The presence of data in the RBR is indicated when RBRL is a logic one.

TABLE 6. CRU INPUT ADDRESS ASSIGNMENTS

ADDR	MODE 0	MODE 1	MODE 2	MODE 3	MODE 5/6	NAME
31	X	X	X	X	X	INT
30	X	X	X	X	X	FLAG
29	X	X	X	X	X	DSCH
28	X	X	X	X	X	CTS
27	X	X	X	X	X	DSR
26	X	X	X	X	X	RTSAUT
25	X	X	X	X	X	TIMELP
24	X	X	X	X	X	TIMERR
23	X	X				XABRT
			X	X	X	
22	X	X	X	X	X	XBRE
21	X	X	X	X	X	RBRL
20	X	X	X	X	X	DSCINT
19	X	X	X	X	X	TIMINT
18	X	X				XAINT
			X	X	X	
17	X	X	X	X	X	XBINT
16	X	X	X	X	X	RINT

ADDR	MODE 0	MODE 1	MODE 2	MODE 3	MODE 5/6	ALL FLAGS=0	LRCRC = 1	LXCRC = 1	LXBC = 1	RHRRD = 1
15	X	X	X	X	X	RIN	RCRC(15)	XCRC(15)	XCRC(15)	RHR(15)
14	X		X	X						
		X				RABRT				
					X	RSBD				
13	X		X	X						
		X				RHRL				
					X	RFBD				
12	X		X	X						
		X				RHROV				
					X	RFER				
11	X	X	X	X	X	ROVER				
10	X		X	X	X	RPER				
		X				RZER				
9	X		X	X	X	RCVERR				
		X				RFLDT				
8	X	X	X	X	X	RBR8				
7	X	X	X	X	X					
6	X	X	X	X	X					
5	X	X	X	X	X					
4	X	X	X	X	X					
3	X	X	X	X	X					
2	X	X	X	X	X					
1	X	X	X	X	X					
0	X	X	X	X	X	RBR0	RCRC(0)	XCRC(0)	XCRC(0)	RHR(0)

8

2.2 GENERAL TRANSMITTER DESCRIPTION

2.2.1 Transmitter Hardware Configuration

Figure 7 is a block diagram of the transmitter section of the TMS 9903 SCC. Either the XBR (transmitter buffer register), SYNC1, or SYNC2 may be loaded into the XSR (transmitter shift register). The LSB of the XSR (XSRLSB) is buffered and output as an external signal XOUT (in mode 1 loop slave configuration RIN is retransmitted prior to synchronization). Two internal registers — XSYNCL (transmitter sync character length) and XSCL (transmitter shift register character length) - are maintained to determine the number of bits per character in XBR, SYNC1, and SYNC2. Since the SYNC1 and SYNC2 registers are of the same length, but not necessarily the same length as the XBR register, the address of the last or highest order bit loaded into both registers is stored in the XSYNCL register, and XSCL contains the number of bits loaded into the XBR register. The XBR register may contain a different length character each time it is loaded. The XBCNT (transmitter bit count) register is loaded with the contents of either XSYNCL or XSCL each time a character is loaded into the XSR. The XPAR (transmitter parity) register serially accumulates the parity of each character and, when enabled, appends the correct parity bit to the transmitted character. The XOCNT (transmitter ones count) register is used in mode 1 operation to accumulate the number of consecutive ones transmitted. The SCTX signal is generated as a synchronous signal of one interval clock cycle each time a bit is to be shifted. If DRCK32 is reset, or $\overline{CTS}$ is inactive (HIGH), SCTX is generated on every one-to-zero transition of $\overline{SCT}$. In the divide-by-32 mode (DRCK32 = 1) if $\overline{CTS}$ goes from one to zero while $\overline{SCT}$ is high, transmission will begin on the second one-to-zero transition of $\overline{SCT}$. The transmitter output, XOUT, will then be updated on every 32nd one-to-zero transition of $\overline{SCT}$ thereafter. On every one-to-zero

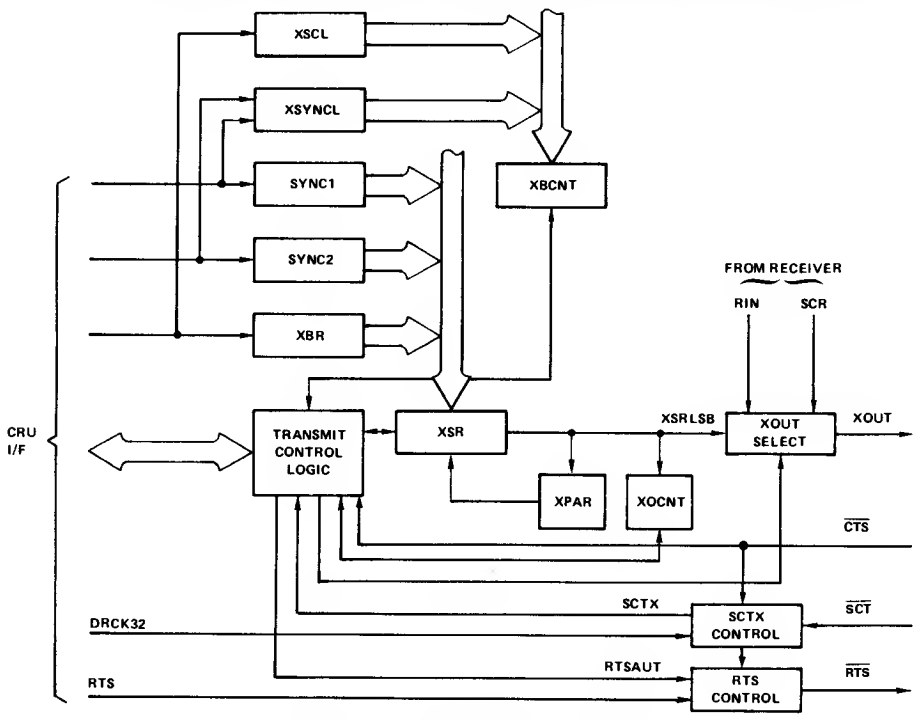


FIGURE 7. TMS 9903 SCC TRANSMITTER BLOCK DIAGRAM

transition of $\overline{SCT}$, the $\overline{RTS}$ signal is updated by the internal, automatic request-to-send signal, (RTSAUT) unless output bit address 17 (RTS) is addressed. If RTS is addressed the $\overline{RTS}$ signal is controlled by the level of output bit 17 until either the RESET or CLRXMT (clear transmitter) command is issued.

2.2.2 Transmitter Initialization

Figure 8 is the flowchart for transmitter initialization. The transmitter is reset to the inactive state when the RESET or CLRXMT commands are issued. To ensure that the control bits are properly loaded into the transmitter, issue CLRXMT after loading the control register the first time. The transmitter remains inactive until the XMTON command is set, enabling transmission and raising RTSAUT. When the CTS command is set to logic one, data transmission begins and continues until the final character is transmitted after XMTON is reset. (Refer also to Figure 13)

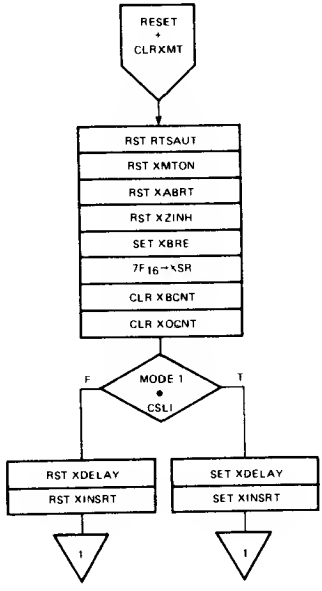


FIGURE 8. TRANSMITTER INITIALIZATION

2.3 GENERAL RECEIVER DESCRIPTION

2.3.1 Receiver Hardware Configuration

8◂

Figure 9 is a block diagram of the receiver section of the TMS 9903. The value of control register bit 11 — 32X data rate clock (DRCK32) — determines the sampling point for RIN. For DRCK32 = 0, RIN is sampled on every zero-to-one transition of SCR. For DRCK32 = 1, RIN is sampled every 32nd SCR beginning with the zero-to-one transition of the 16th SCR after synchronization. The received character is assembled in the receive shift register (RSR) according to the length specified in control register bits 2, 1,0 — receive character length select (RSCL). The value of RSCL is transferred to the RBCNT (receiver bit count) register when the contents of the RSR are transferred to the receive buffer register (RBR). This double buffering of the received character and the character length provide variable character length capability. The character length may be altered any time prior to the transfer of the next received character to the RBR. In all modes of operation except mode 1, the parity checker is updated with each bit shifted into the RSR. If parity is enabled, the receiver compares the assembled parity bit to the received parity bit, and then sets it to zero when the

character is transferred to the RBR. When the character is transferred to the RBR the receive buffer register loaded flag RBRL is set. If RBRL was set already, the receiver overrun flag ROVER is set. Incorrect received parity will set the parity error flag (RPER) in all but mode 1 operation. Note that parity generation and detection is not available in mode 1 operation. The comparator and sync character register SYNC1 are utilized in the several modes to provide flag and sync character detection. For a detailed discussion of each operation, see the discussion of the particular mode of operation desired.

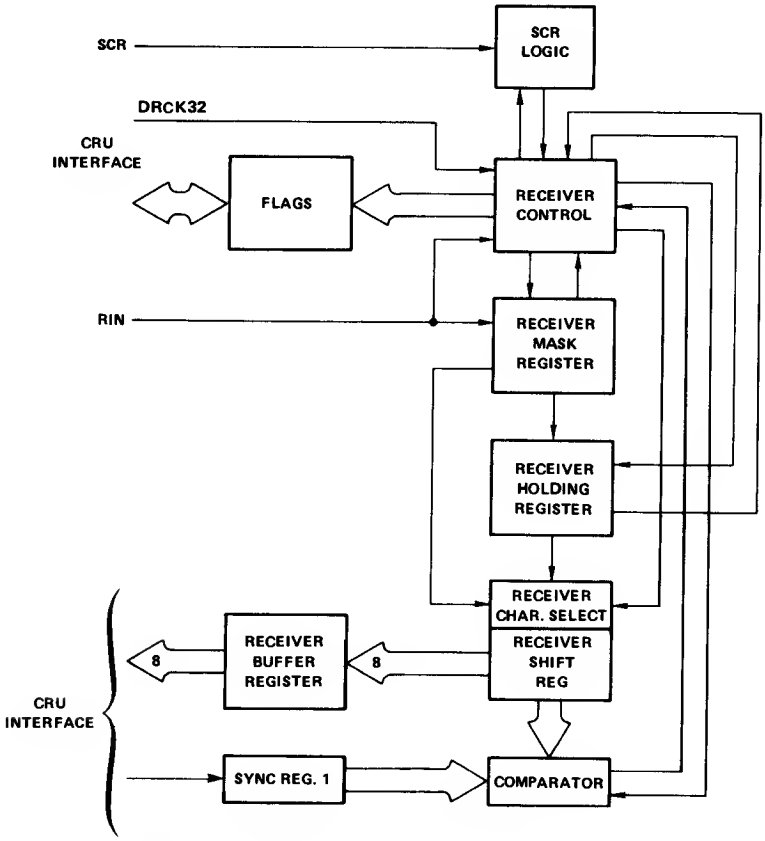


FIGURE 9. TMS 9903 RECEIVER BLOCK DIAGRAM

▶8

2.3.2 Receiver Initialization

The receiver is initialized by the RESET and CLRRCV (clear receiver) commands from the CPU. This causes the receive mask register (used in mode 1 operation only) to be initialized to all ones, the receive shift register and parity to be initialized, and all receiver-related flags to be reset.

Initializing the RSR sets the *N-1* least-significant bits to logic one and sets the MSB (bit N) to logic zero, where *N* is the number of bits per character. The detection of the zero shifted out of the RSR signals the assembly of a complete character. For this reason the CLRRCV command should be issued after loading the control register to assure the correct assembly of the first character received after loading.

2.4 TRANSMITTER AND RECEIVER OPERATION

The TMS 9903 has six different operational modes (0, 1, 2, 3, 5, and 6). Following is a detail discussion for each mode of the transmitter and receiver operations.

2.4.1 Mode 0 Operations

2.4.1.1 Transmitter Operation

Figure 10 is a flowchart for mode 0 transmitter operation. If parity is enabled, the parity bit is appended to the transmitted character. When the character has been shifted out and no data is available (XBRE = 1), the transmitter will either abort operation or transmit the contents of SYNC2, depending on the value of XPRNT (transparent). Note that parity is not generated when SYNC2 is transmitted; therefore, if parity is desired, the correct parity bit must be appended to the sync character when it is loaded into SYNC2.

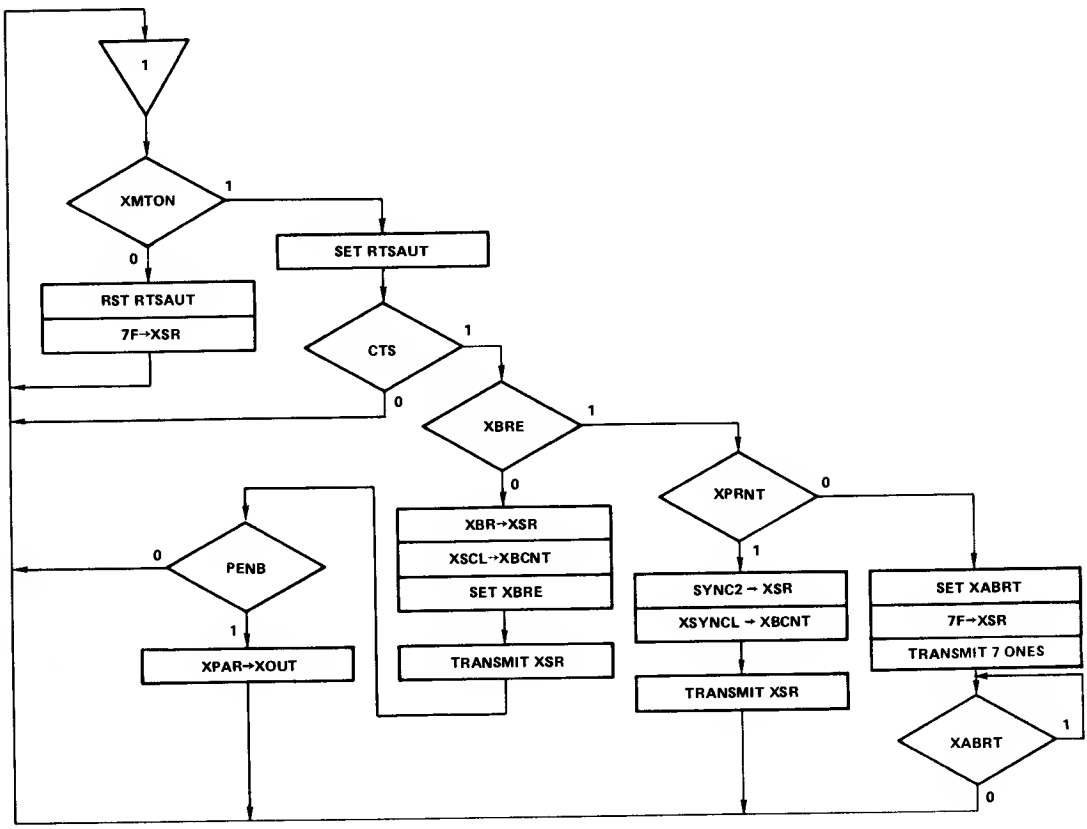


FIGURE 10. MODE O TRANSMITTER OPERATION

2.4.1.2 *Mode 0 Receiver Operation*

Figure 11 is a flowchart for mode 0 receiver operation. This mode is the basic subset of receiver operation for all modes. The general description of receiver operation described in Section 2.3. above applies to mode 0 operation.

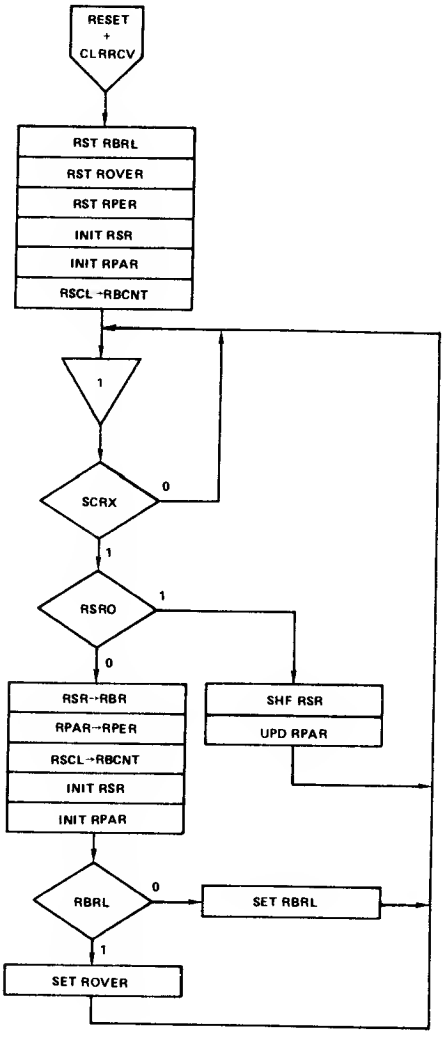


FIGURE 11. MODE 0 RECEIVER OPERATION

2.4.2 Mode 1 Operation

2.4.2.1 Mode 1 Transmitter Operation

Figure 12 is a flowchart of transmitter operation in mode 1. Beginning transmission varies slightly, depending on the configuration selected with control register bits 5 and 4, the configuration select (CSL1, CSL0).

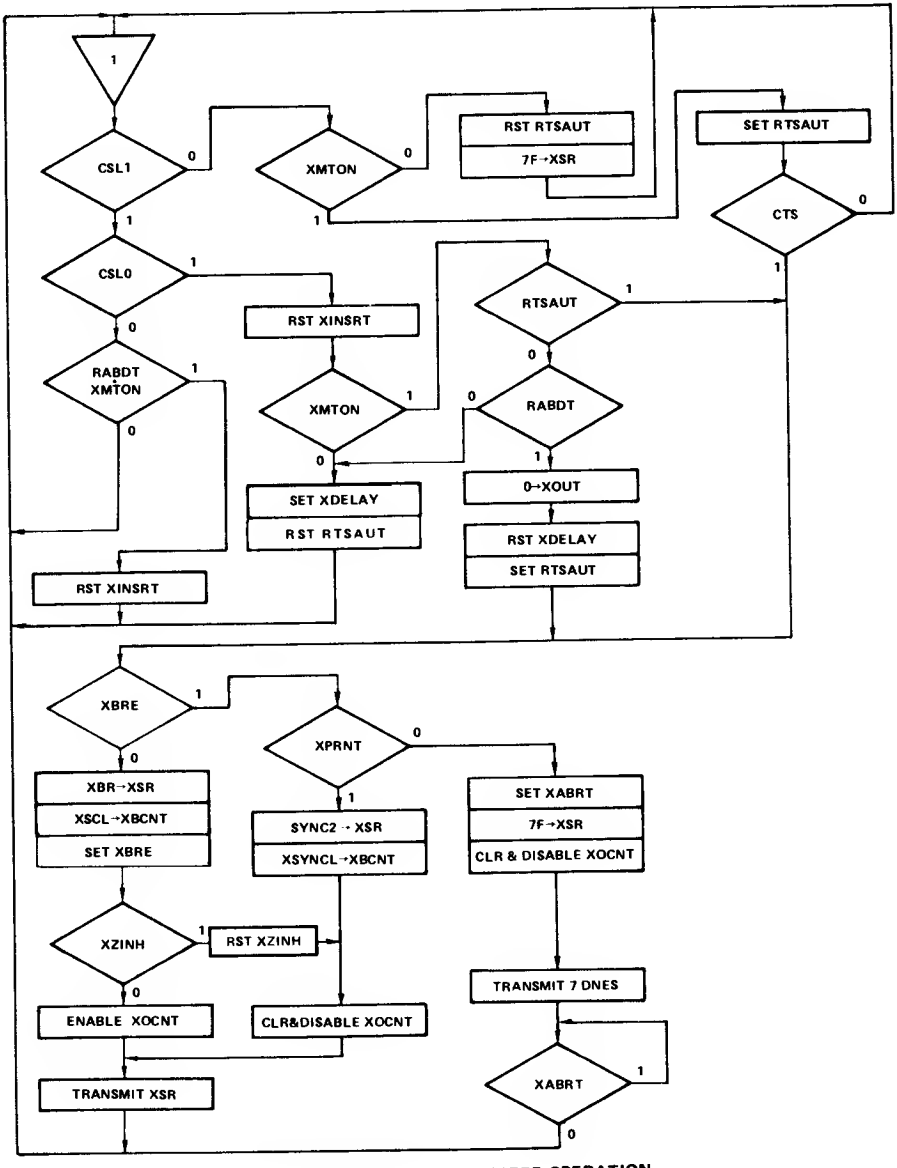


FIGURE 12. MODE 1 TRANSMITTER OPERATION

2.4.2.1.1 Normal and Loop Master (CSL1 = 0) Operation. The operation of the transmitter is the same when CSL1 = 0, regardless of the status of CSL0. When XMTON is set, RTSAUT becomes active and data transmission begins with CTS = 1. As each character is transferred from XBR to XSR, the XZINH flag is tested. If XZINH = 1, XOCNT is cleared and zero-insertion is disabled. If XZINH = 0, a zero bit will be inserted after each fifth consecutive transmitted one. If XBRE = 1 when a character is to be loaded into the XSR, the transmitter will either abort (when XPRNT = 0) or transmit the contents of SYNC2 (when XPRNT = 1). When SYNC2 is transmitted, XOCNT is cleared and disabled, prohibiting zero-insertion. If the transmitter aborts, the XABRT flag is set and a minimum of seven ones are transmitted. The transmitter will remain inactive until XABRT is cleared.

2.4.2.1.2 Loop Slave (Pending Synchronization) (CSL1 = 1, CSL0 = 0) Operation. As a loop slave the device must first synchronize itself to the communication line before actively transmitting data. Initially, the line is monitored to search for an *end-of-poll* (EOP = 11111110) character, which occurs when RABDT = 1. At this time, if XMTON = 1, the transmitter introduces a single-bit delay by retransmitting the final one, and subsequently retransmitting each received data bit. The logic associated with XOUT is shown in Figure 13. When XINSRT = 1 and XDELAY = 1, XOUT = RIN. When XINSRT is reset by detection of an EOP, RIN is delayed a single bit-time before being transmitted on XOUT.

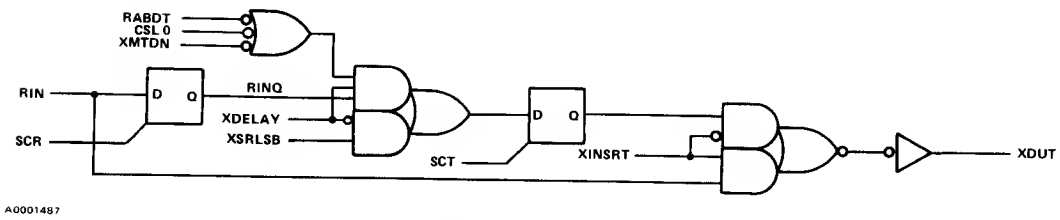


FIGURE 13. XOUT SELECT LOGIC

2.4.2.1.3 Loop Slave (Active) (CSL1 = 1, CSL0 = 1) Operation. After loop synchronization has been achieved, transmission may begin by first detecting an EOP (11111110). The last one is inverted to provide the beginning flag of the transmitted frame, and normal data transmission begins.

▶8

2.4.2.2 Mode 1 Receiver Operation

Figure 14 is a flowchart of the mode 1 receiver operation and Figure 15 shows the register circuitry used to perform these operations. As described in Section 2.3.2 above, executing the RESET or CLRRCV commands resets all flags, initializes the receiver registers, and loads all ones into the mask register.

2.4.2.2.1 Synchronization. Each bit time (SCRX = 1) data is shifted into RMSK. When a FLAG character bit pattern of $7E_{16}$ is detected (RFLG = 1), the receiver achieves synchronization and the bit pattern 001111111_2 is loaded into the nine-bit RSR.

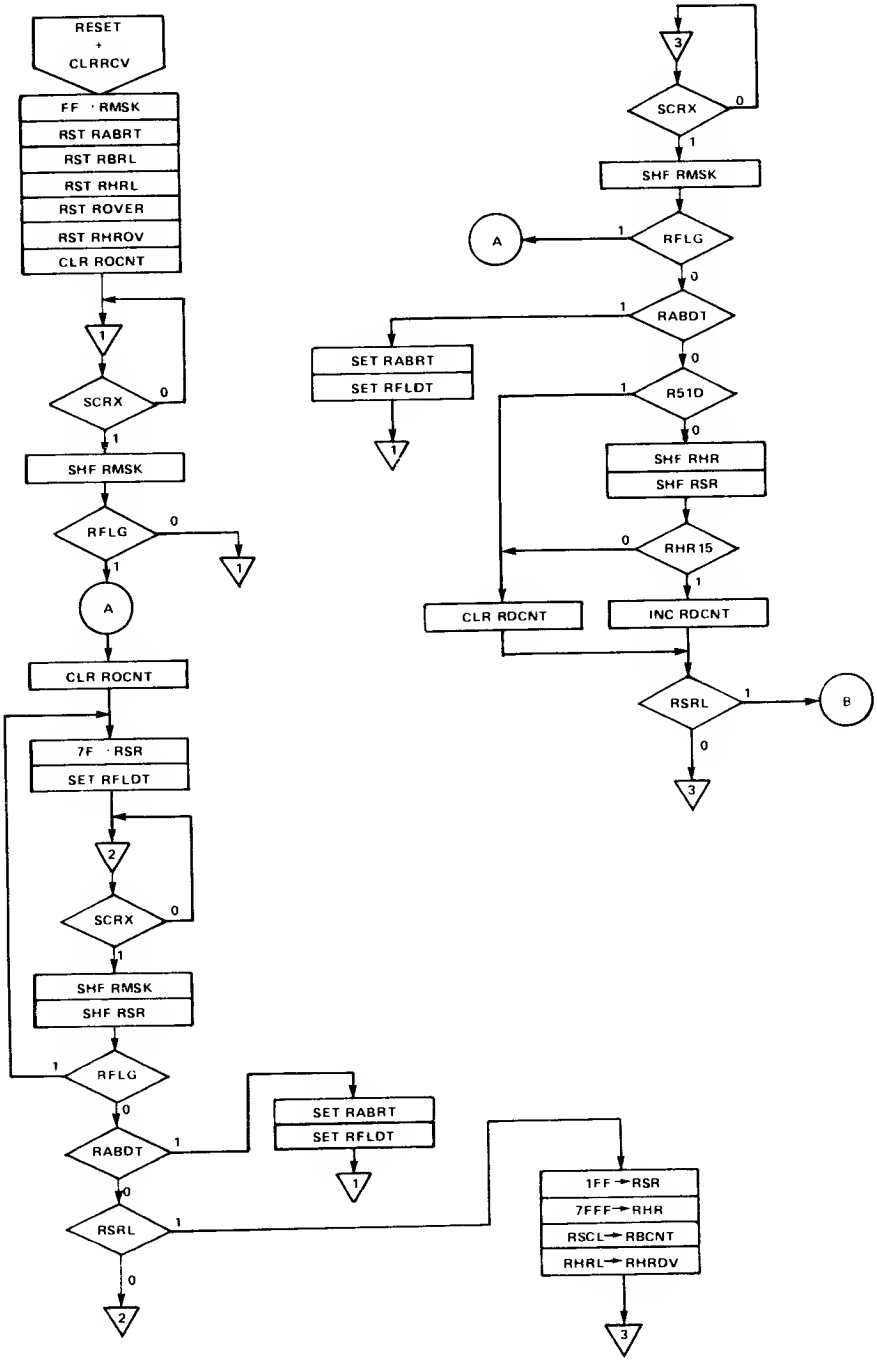


FIGURE 14. MODE 1 RECEIVER OPERATION (PAGE 1 OF 2)

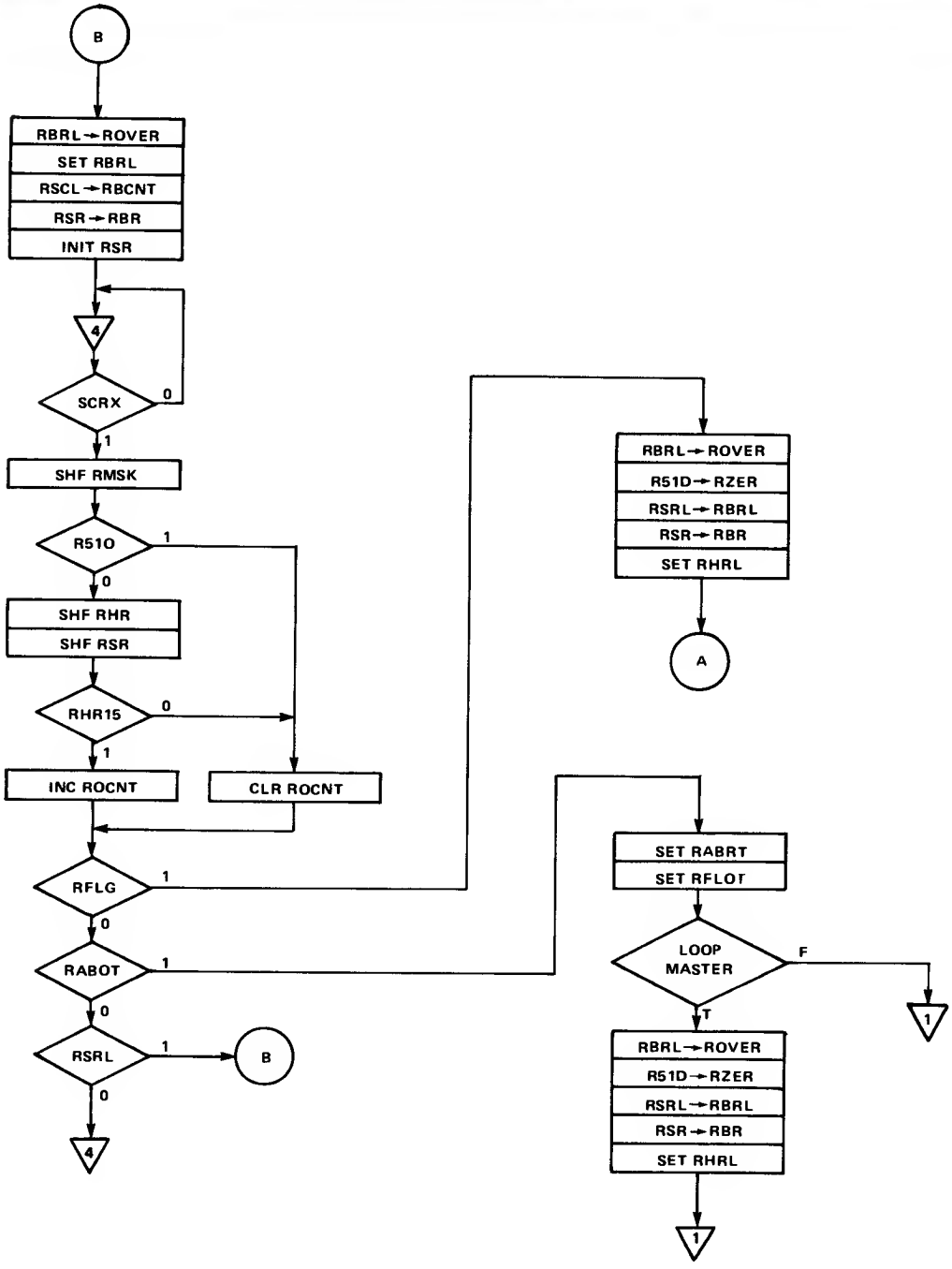


FIGURE 14. MOOE 1 RECEIVER OPERATION (PAGE 2 OF 2)

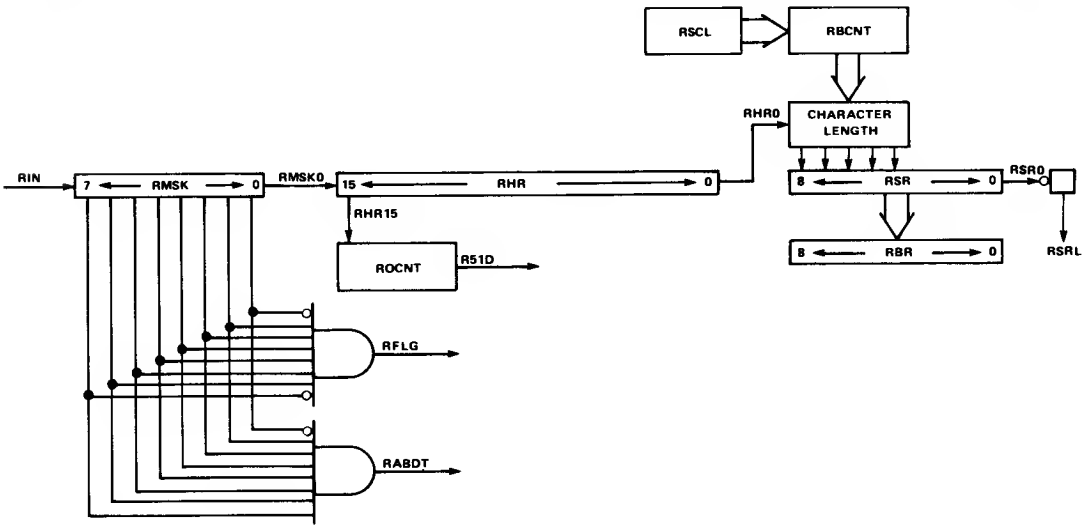


FIGURE 15. MODE 1 RECEIVER CIRCUITRY

2.4.2.2.2 **Eight-Bit Delay.** Each bit time, RIN is shifted into RMSK, and RSR is shifted right until RSR0 = 0. This sets RSRL, indicating eight bits have been shifted. If the FLAG pattern is detected again, the eight-bit delay is repeated. The FLAG pattern consists of six consecutive ones (01111110). If more than six consecutive ones are detected in RMSK, RABDT is set to a one and the receiver aborts. FLAG patterns, abort patterns, and zeroes generated by five-ones-zero insertion are all deleted from the serial bit stream before being shifted into RHR.

2.4.2.2.3 **16 + (RSCL) Bit Delay.** After the eight-bit delay, the RHR (receiver holding register) is loaded with 7FFF16 and the RSR is loaded with all ones. The contents of RSCL (receive character length select field of the control register) are loaded into RBCNT (receive bit count register), which selects which bit of the RSR is the MSB. Each bit time, RIN is shifted into RMSK. When R51D = 1 (five consecutive ones detected), the next bit — the inserted zero bit — is not shifted from RMSK0 to RHR15; otherwise, RMSK0 is shifted into RHR15, RHR is shifted right, RHR0 is shifted into the selected MSB of RSR, and RSR is shifted right. This operation continues until RSRL = 1, indicating that the delay has been completed, and RMSK, RHR, and RSR all contain valid data. The fully assembled character is then transferred from the RSR to the receive buffer register (RBR) and the receive buffer register loaded flag (RBRL) is set. 8◀

2.4.2.2.4 **Character Assembly.** Each time RSRL = 1, RSCL is transferred to RBCNT, RSR to RBR, RBRL is set; and RSR is initialized to all ones except for the MSB of the selected character length. That is, for a seven-bit character, RSR is loaded with 0001111111_2. Data is shifted through RMSK, RHR, and RSR each bit time, performing zero-deletion until a FLAG pattern or an abort sequence is detected.

2.4.2.2.5 **Receiver Abort.** When the receiver detects the abort pattern, RABRT is set and control returns to the initial state where the FLAG pattern is required for synchronization.

2.4.2.2.6 Flag Detection. After entry into character assembly, the receive operation continues until a flag is detected, indicating the end of a frame. When this occurs, several operations are performed:

(1) RSR is transferred to RBR.
(2) If RBRL is set, ROVER is set.
(3) If RSRL = 1, RBRL is set.
(4) RHRL is set.
(5) Control returns to the eight-bit delay described in paragraph 2.4.2.2.2 above.

Thus, RHRL is set whenever the end of a frame is detected. If a complete character is received, RBRL is set; otherwise RBRL is not set and the number of bits received can be determined by shifting the contents of RBR right until the first zero is shifted out. After the receive CRC register (RCRC) is updated with the most recently received data, RHR may be compared with RCRC to determine if the received CRC contained in RHR matches the expected CRC contained in RCRC. If RZER (receiver zero error) = 1, it indicates a zero was not appended to the last five consecutive ones received. This occurs only if the last 13 received bits are "0111111011111_2".

2.4.2.2.7 Variable Receive Character Length. Since the *advanced data communication control protocol* (ADCCP) permits variable length characters in the same frame, the receiver double-buffers the received character length. Each time RSR is transferred to RBR, RSCL is transferred to RBCNT. Thus, RSCL (bits 2-0 of the control register) may be altered any time before the next character is transferred into RBR as long as a minimum setup time of two internal clocks is met.

2.4.2.2.8 Loop Master Operation. When the TMS 9903 is configured to operate as a loop master (CSL1 = 0, CSL0 = 1), the EOP character (11111110_2, or RABDT = 1) is interpreted in the same manner as the FLAG character with respect to terminating frame reception. However, a FLAG must be received before synchronization occurs for the reception of the next frame.

2.4.3 Mode 2 Operation

2.4.3.1 Mode 2 Transmitter Operation

Figure 16 is a flowchart of mode 2 transmitter operation. If parity is enabled, the parity bit is appended to the transmitted character. When the character has been shifted out and no data is available (XBRE = 1), the contents of SYNC2 are transmitted without parity. If parity is required for the sync character, it must be appended to the character when it is loaded into SYNC2.

2.4.3.2 Mode 2 Receiver Operation

Figure 17 is a flowchart of mode 2 receiver operation. In mode 2 operation, after initialization, the receiver assembles a character in the RSR and compares it to the sync character contained in SYNC1. Once the RSR receives the sync character, receiver operation is similar to that of mode 0 receiver operation discussed in Section 2.4.1.2 above.

▶8

2.4.4 Mode 3 Operation

2.4.4.1 Mode 3 Transmitter Operation

Figure 18 is a flowchart of mode 3 transmitter operation. If parity generation is enabled, the correct parity bit is assembled as the character is shifted out of the XSR and appended as the last bit. When the character has been transmitted and no further data is available (XBRE = 1), and XPRNT = 0, the contents of SYNC1 are loaded and shifted out twice to give a fill sequence of SYNC1 — SYNC1. If XPRNT = 1 and XBRE = 1, the contents of SYNC2 are loaded and shifted, followed by the contents of SYNC1, giving a fill sequence of SYNC2 — SYNC1.

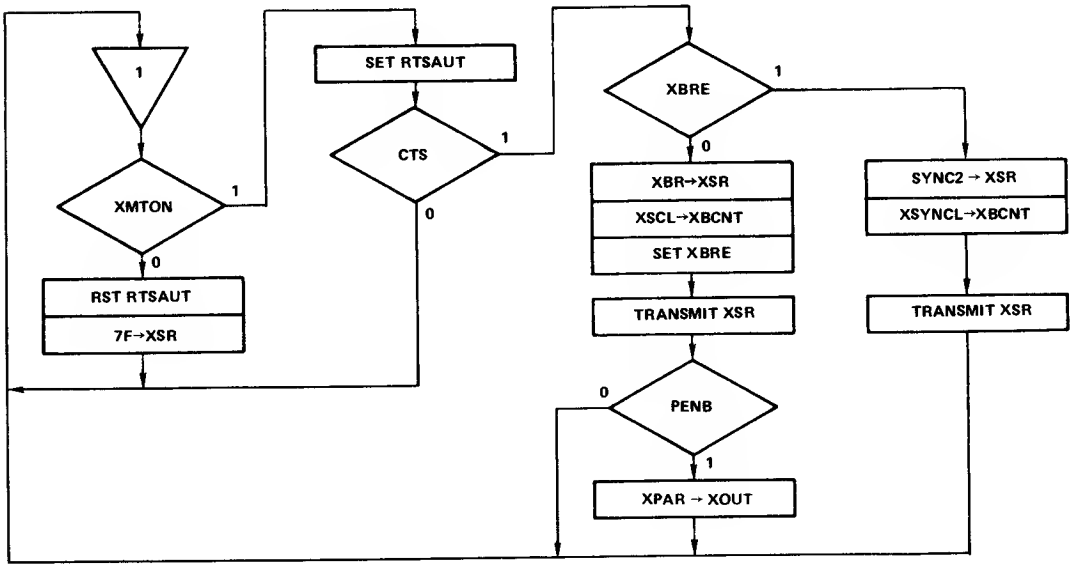


FIGURE 16. MODE 2 TRANSMITTER OPERATION

2.4.4.2 *Mode 3 Receiver Operation*

Figure 19 is a flowchart of mode 3 receiver operation. In mode 3 operation, after initialization, the receiver assembles two consecutive SYNC1 characters before returning to mode 0 operation.

8◀

2.4.5 Mode 5 and 6 Operation

Although the TMS 9903 is designed primarily for synchronous communication control, it can be used for asynchronous operation if it is set to operate in mode 5 or 6, and if external baud rate clocks are provided for both SCR and $\overline{SCT}$. Mode 5 is asynchronous operation with one start and two stop bits, while mode 6 is asynchronous operation with one start and one stop bit.

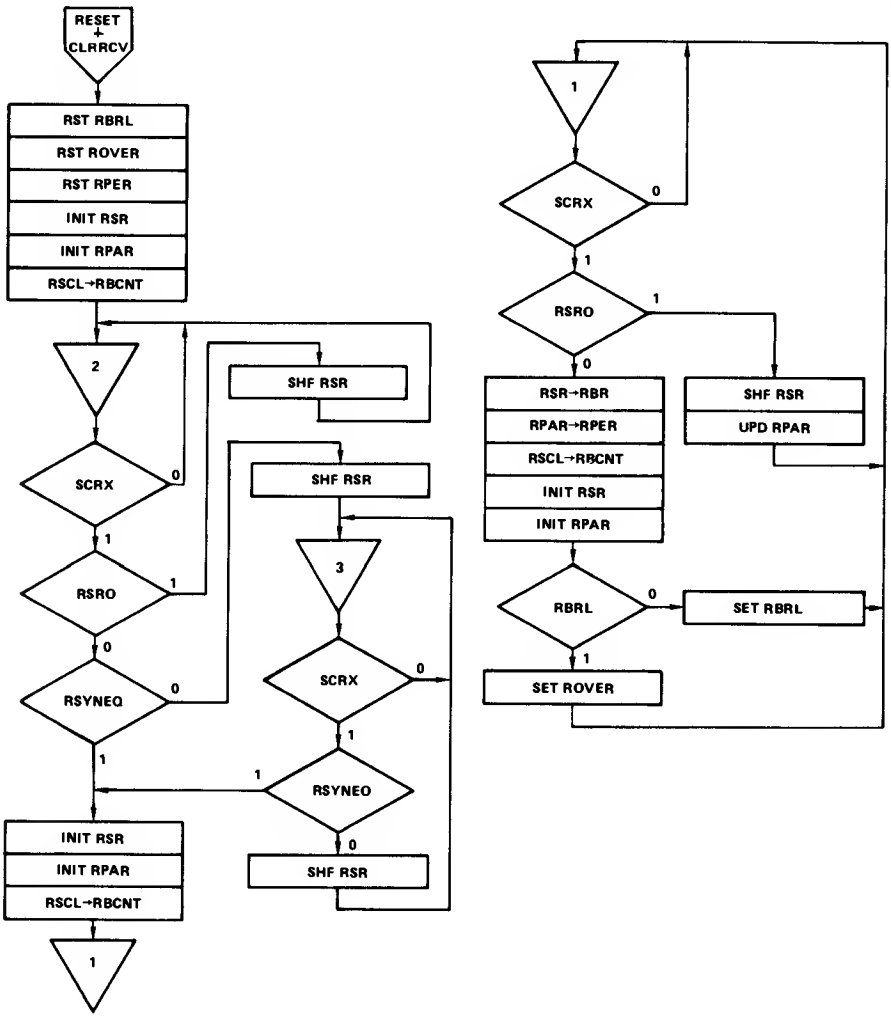


FIGURE 17. MODE 2 RECEIVER OPERATION

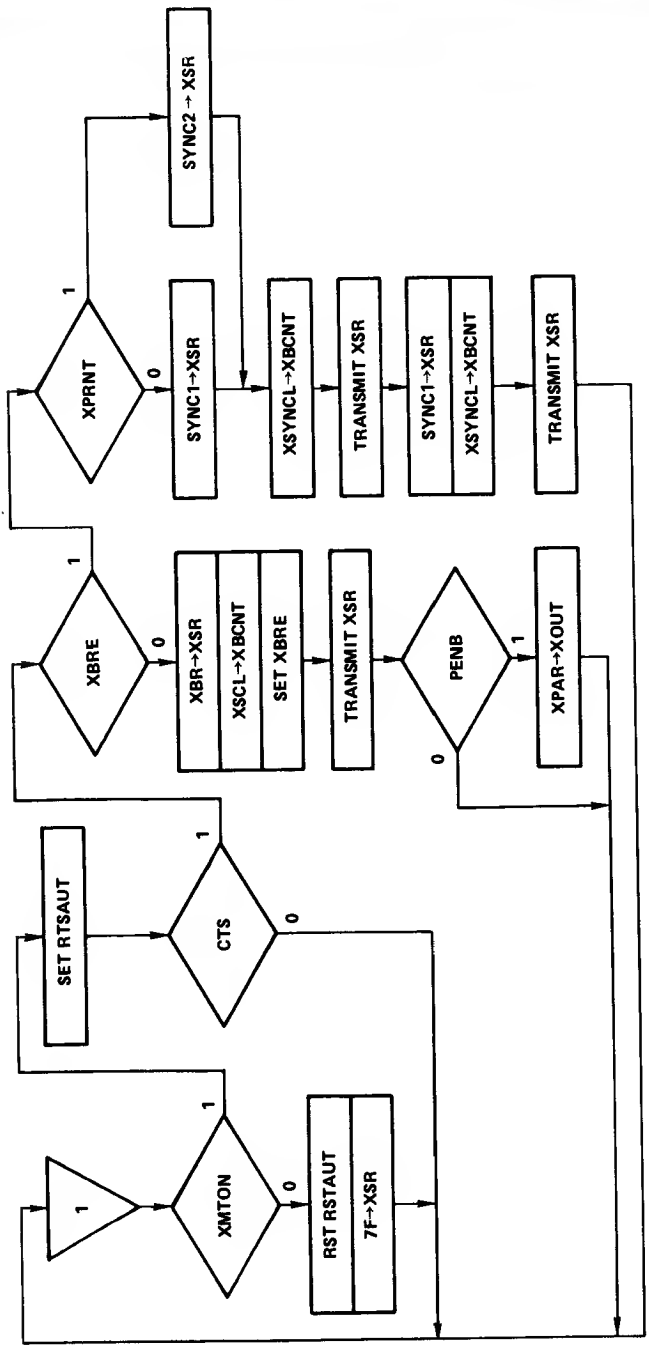


FIGURE 18. MOOE 3 TRANSMITTER OPERATION

8◀

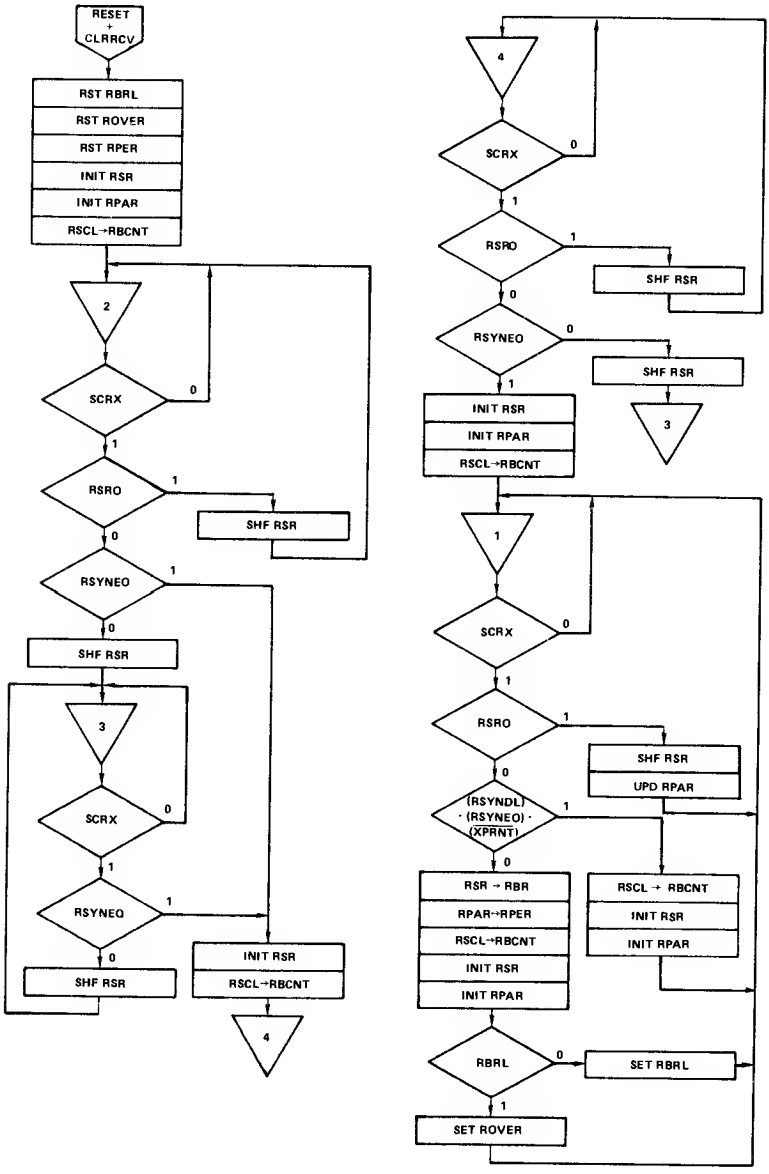


FIGURE 19. MODE 3 RECEIVER OPERATION

2.4.5.1 *Mode 5 and 6 Transmitter Operation*

Operation of the transmitter in modes 5 and 6 is described in the Figure 20 flowchart. The transmitter is initialized by issuing the RESET or CLRXMT commands, which cause the internal signals XBRE to be set and XMTON to be reset. Device outputs RTS and XOUT are set, placing the transmitter in the inactive state. When XMTON is set by the CPU, the RTS output becomes active. Transmission then begins when CTS becomes active.

If BRKON is set, the character in transmission is completed; any character in the XBR is loaded into the XSR and transmitted; and XOUT is set to zero. Further loading of XBR should be avoided until BRKON is reset. If BRKON = 0, XOUT is set to logic one when the transmitter completes the current transmission and no further data is loaded into XBR.

2.4.5.2 *Mode 5 and 6 Receiver Operation*

Figure 21 is a flowchart of mode 5 or 6 receiver operation. The receiver is initialized when the RESET or CLRRCV command is issued in mode 5 or 6. The RBRL flag is reset to indicate that no character is in the RBR, and the RSBD and RFBD flags are reset. The receiver remains inactive until a one-to-zero transition of the RIN device-input is detected which sets SBD.

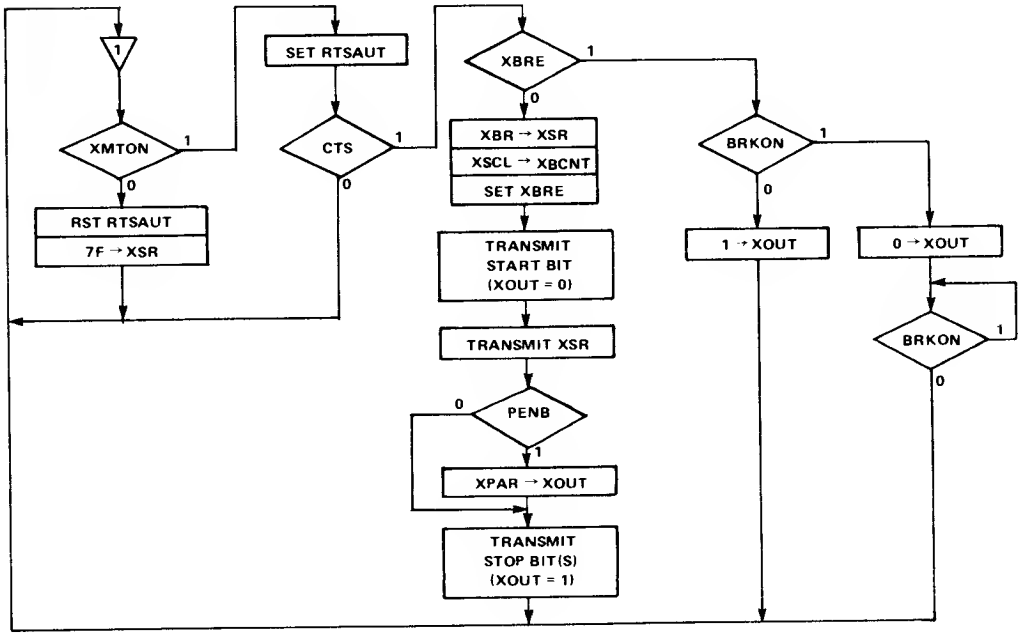


FIGURE 20. MODE 5 OR 6 TRANSMITTER OPERATION

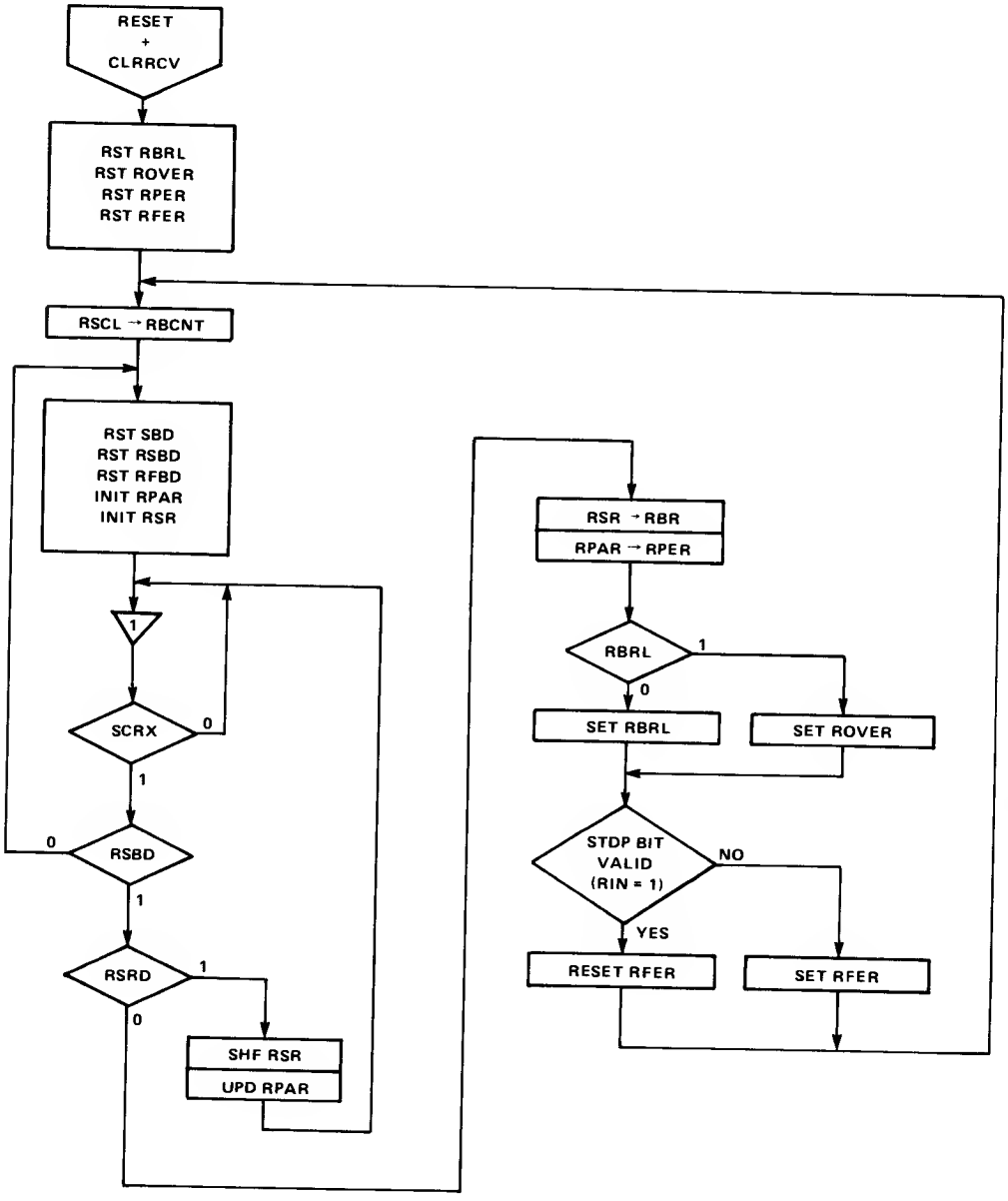


FIGURE 21. MODE 5 OR 6 RECEIVER OPERATION

2.4.5.2.1 Start Bit Detect. The receiver delays one-half bit time from SBD being set and again samples RIN to ensure that a valid start bit has been detected. If RIN = 0 after the half-bit delay, RSBD (receive start bit detect) is set and data reception begins. If RIN = 1, no data reception occurs. SBD and RSBD are reset and wait for the next one-to-zero transition of RIN.

2.4.5.2.2 Data Reception. In addition to verfying the valid start bit, the half-bit delay after the one-to-zero transition also establishes the sample point for all subsequent data bits in this character. Theoretically, the sample point is in the center of each bit cell, thus maximizing the limits of acceptable distortion of data cells. After the first full bit delay the least significant data bit is received and RFBD is set. The receiver continues to delay one-bit intervals and samples RIN until the selected number of bits are received. If parity is enabled, one additional bit is read for parity. After an additional bit delay, the received character is transferred to the receive buffer register, RBRL is set, ROVER and RPER are loaded with appropriate values, and RIN is tested for a valid stop bit. If RIN = 1, the stop bit is valid. RFER, RSBD, and RFBD are reset and the receiver waits for the next start bit to begin reception of the next character.

If RIN = 0 when the stop bit is sampled, RFER is set to indicate the occurrence of a framing error. RSBD and RFBD are reset, but sampling for the start bit of the next character does not begin until RIN = 1.

2.5 INTERVAL TIMER SECTION

A block diagram of the interval timer is shown in Figure 22. When the load interval register flag (LDIR) is set, output to CRU bit addresses 0-7 is loaded into the interval register. The LDIR flag is reset by a command from the CRU. After LDIR is reset, the contents of the interval register are loaded into the interval timer, and the interval timer is enabled. The interval timer is decremented at the rate of the prescaler output. When the interval timer decrements to 0, the TIMELP flag is set and the contents of the interval register are reloaded into the interval timer. If TIMELP has not been cleared by the CPU by the time that the interval timer decrements to zero again, the TIMERR flag is set (the zero state is counted the same as other counter states). A flowchart for interval timer operation is illustrated in Figure 23.

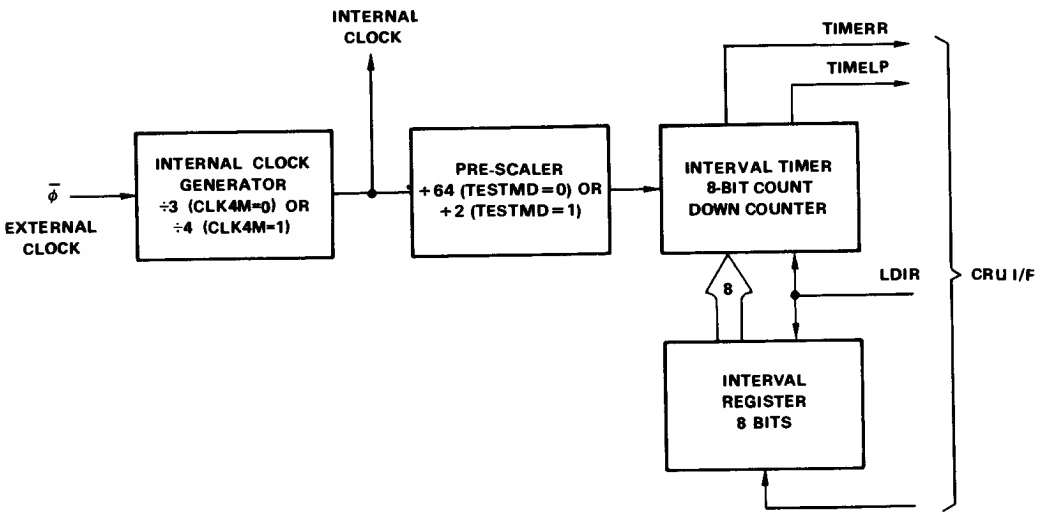


FIGURE 22. INTERVAL TIMER BLOCK DIAGRAM

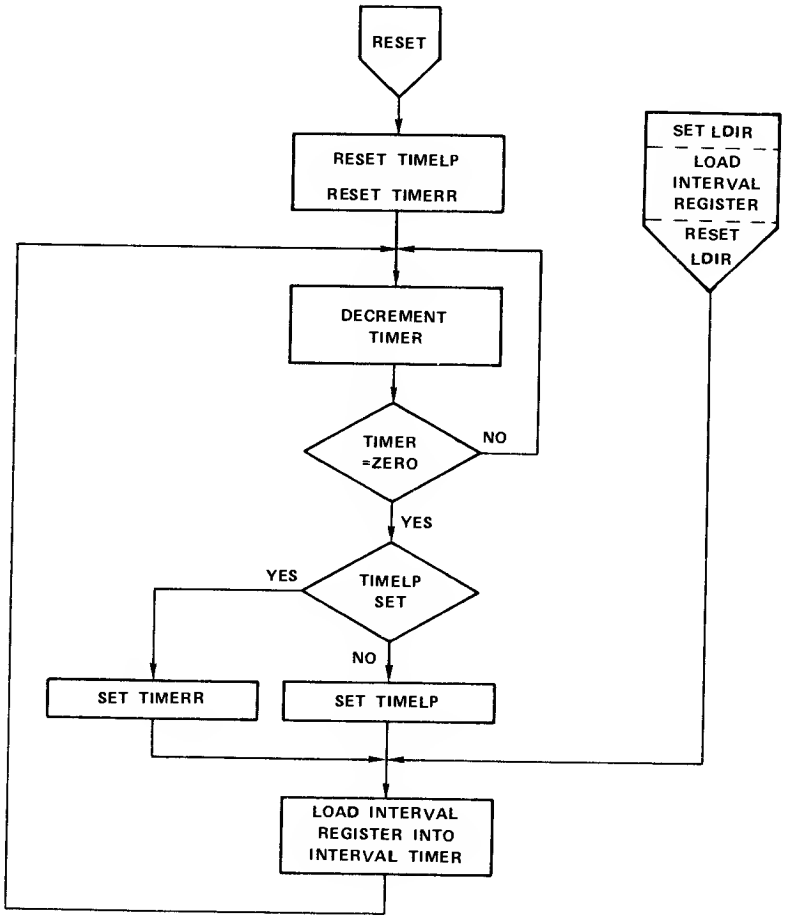


FIGURE 23. INTERVAL TIMER OPERATION

2.5.1 Time Interval Programming

8

The rate at which the interval timer sets TIMELP during normal operation is determined by the value loaded into the interval register. In normal operation (TSTMD = 0), the prescaler output has a frequency 1/64 of the internal system clock. Thus, when a standard 3- or 4-MHz external clock is used to generate a 1-MHz internal clock, the interval timer is decremented once every 64 microseconds. The interval register selects the number of 64-microsecond intervals contained in each interval timer period. Thus, the interval may range from 64 microseconds (interval register = 01_{16}) to 16,320 microseconds (interval register = FF_{16}) in 64-microsecond increments.

2.5.2 Test Mode Interval Timer Operation

When TSTMD = 1, the prescaler divides the internal system clock frequency by two rather than by 64, causing the interval timer to operate at 32 times the rate at which it operates when TSTMD = 0 for identical interval register contents.

2.6 INTERRUPTS

The interrupt-output control line (INT) is active (low) when any of the following conditions occur and the corresponding interrupt has been enabled by the CPU:

1) DSCH = 1. DSCH (data set status change) is set when DSR, CTS, or RTSAUT changes levels

2) TIMELP = 1. TIMELP (timer elapsed) is set when the selected time interval has elapsed.

3) XBRE = 1. XBRE (transmit buffer register empty) is set when the transmit buffer register is empty.

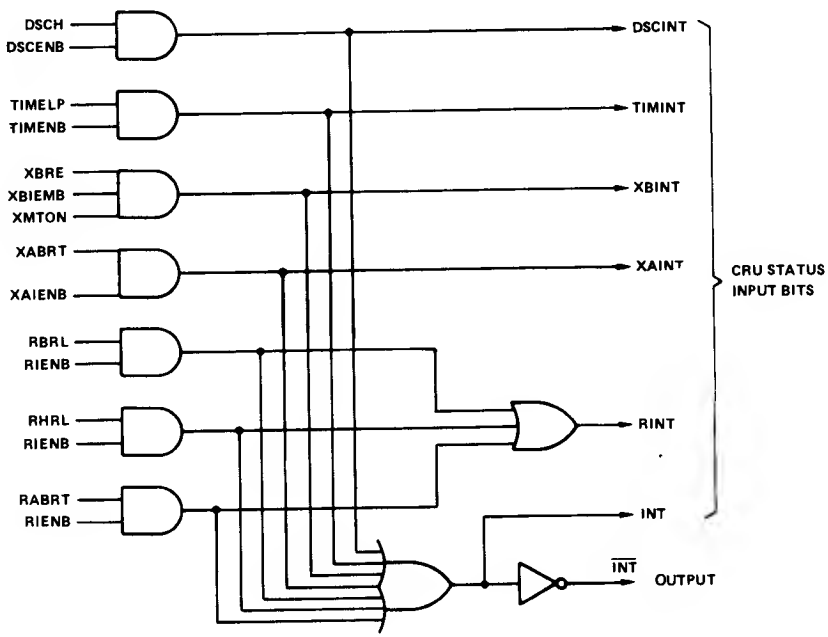
4) XABRT =1. XABRT (transmitter abort) is set in mode 0 and 1 when no data is available for transmission, no provision is made for a fill character, and XMTON is turned ON.

5) RBRL = 1. RBRL (receive buffer register loaded) is set when a complete character is transferred from the receive shift register to the receive buffer register.

6) RHRL = 1. RHRL (receive holding register loaded) is set in mode 1 when the receiver receives a complete frame.

7) RABRT = 1. RABRT (receive abort) is set in mode 1 when the FLAG character is detected and seven consecutive ones are received.

Interrupts are enabled in the SCC by writing a one to the associated enable bit (see Section 2.1.1). Figure 24 shows the logical equivalent of the TMS 9903 interrupt circuitry



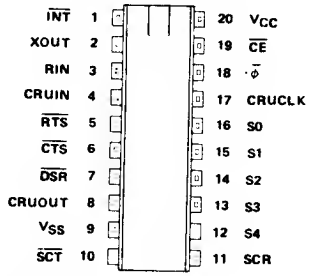
NOTE: See Tables 1 and 5 for input and output signal definitions.

FIGURE 24. INTERRUPT GENERATION LOGIC

2.7 TMS 9903 TERMINAL ASSIGNMENTS AND FUNCTIONS

SIGNATURE	PIN	I/O	DESCRIPTION
$\overline{INT}$	1	OUT	Interrupt. When active (low), the INT output indicates that at least one of the interrupt conditions has occurred.
XOUT	2	OUT	Transmitter serial data output line.
RIN	3	IN	Receiver serial data input line.
CRUIN	4	OUT	Serial Data Output line fromTMS 9903 to CRUIN input line of the CPU.
$\overline{RTS}$	5	OUT	Request to Send output from TMS 9903 to modem. This output is enabled by the CPU and remains active (low) during data transmission from TMS 9903.
$\overline{CTS}$	6	IN	Clear-to-Send input from modem to TMS 9903. When active (low), it enables the transmitter section of the TMS 9903.
$\overline{DSR}$	7	IN	Data Set Ready input from modem to TMS 9903. This input generates an interrupt when going On or Off.
CRUOUT	8	IN	Serial data input line to TMS 9903 from CRUOUT line of the CPU.
V_{SS}	9	IN	Ground Reference Voltage.
$\overline{SCT}$	10	IN	Transmit Clock — Transmitter data is shifted out on one-to-zero transition of $\overline{SCT}$.
SCR	11	IN	Receiver Clock — Receiver serial data (RIN) is sampled at zero-to-one transition of SCR.
S4(LSB)	12	IN	Address bus S0-S4 are the lines that are addressed by the CPU to select a particular TMS 9903 function.
S3	13	IN	
S2	14	IN	
S1	15	IN	
S0(MSB)	16	IN	
CRUCLK	17	IN	CRU Clock. When active (high), TMS 9903 samples the input data on CRUOUT line.
$\overline{\phi}$	18	IN	TTL Clock.
$\overline{CE}$	19	IN	Chip Enable — When $\overline{CE}$ is inactive (high), the TMS 9903 address decoding is inhibitied. CRUIN remains at high impedance when $\overline{CE}$ is inactive (high).
V_{CC}	20	IN	Supply voltage (+5 V nominal).

TMS 9903 PIN ASSIGNMENTS
20 PIN DUAL-IN-LINE PACKAGE
(TOP VIEW)



3. DEVICE APPLICATION

This section describes the software interface between the CPU and the TMS 9903 and discusses some of the design considerations in the use of this device in synchronous and asynchronous communications applications.

3.1 DEVICE INITIALIZATION

The following discussions assume that the value to be loaded into the CRU base register (register 12) in order to point to bit 0 is 0040_{16}, and the ϕ input to the SCC is a 4-MHz signal. The SCC divides this signal by four to generate an internal clock frequency of 1 MHz. An interrupt is generated by the interval timer every 1.6 milliseconds when timer interrupts are enabled.

When power is applied, the SCC must be initialized by the CPU with the instruction sequence shown below. The actual data (i.e., CTRL) loaded into the control register and specific initialization requirements are application-dependent and are further described in the following discussions of individual mode operations.

```
RESET     EQU     31
CLRRCV    EQU     30
CLRXMT    EQU     30
CTRL      DATA    >XXXX

LI        R12,>40          Initialize CRU Base.
SBO       RESET            Issue RESET command which resets the TMS 9903 and sets
                           the LDCTRL — Load Control Register — flag.
LDCR      CTRL, 12         Load the control register, automatically resetting LDCTRL.
SBZ       CLRRCV           Initialize Receiver.
SBO       CLRXMT           Initialize Transmitter.
```

The RESET command resets all flags (other than LDCTRL), resets all output bits, and disables all interrupts. The contents of the XBR, XCRC, RCRC, RHR, RBR, SYNC1, SYNC2, and the interval register are unaffected.

The receiver should be initialized with the CLRRCV command after the control register is loaded to ensure that the receiver logic will assemble the first received character at the proper length.

The transmitter should be initialized with the CLRXMT command after the control register is loaded to ensure that the transmitter logic will operate according to the flowchart for the selected mode.

3.1.1 Mode 0 Operation

Mode 0 operation is the most unstructured of the TMS 9903 operating modes, placing all synchronization and control requirements on the CPU. It functions as the basic subset of all other modes of operation and, as such, can be used in essentially any control protocol the user desires, limited only by the ability of the user software to provide the necessary control. The following instruction sequence will set the TMS 9903 to operate in mode 0.

```
RESET     EQU     31
CLRRCV    EQU     30
CLRXMT    EQU     30
LDSYN2    EQU     27
XPRNT     EQU     23
          .
          .
          .
          LI      R12,>40          Initialize CRU Base.
          SBO     RESET            Reset device and set LDCTRL.
          LDCR    @ CTLFLD,12      Load Control Register and Reset LDCTRL.
          SBZ     CLRRCV           Initialize Receiver.
          SBO     CLRXMT           Initialize Transmitter
          SBO     LDSYN2
          LDCR    @ SYNC2,8        Load Sync Character Register 2.
          SBZ     LDSYN2
          .
          .
          .
SYNC2     BYTE    >16              ASCII Sync Character
CTLFLD    DATA    >002A
```

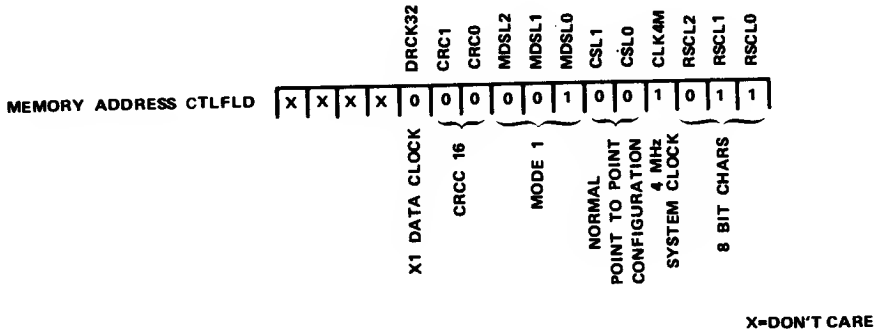
					DRCK32	CRC1	CRC0	MDSL2	MDSL1	MDSL0	CSL1	CSL0	CLK4M	RSCL2	RSCL1	RSCL0
MEMORY ADDRESS CTLFLD	X	X	X	X	0	0	0	0	0	0	1	0	1	0	1	0
					X1 DATA CLOCK	CRCC-16		MODE 0			EVEN PARITY		4MHZ CLOCK	7 BIT CHAR		

X=DON'T CARE

3.1.2 Mode 1 Operation

Mode 1 operation is selected to support the synchronous data link control (SDLC) protocol. SDLC supports full duplex communication links and places no constraints on the communications codes involved in information transfer. SDLC operation is initialized with the software shown below. This software sets the TMS 9903 to operate in mode 1 with eight-bit characters. The TMS 9903 further allows SDLC operation in several configurations — point-to-point, multipoint, loop master, loop slave, etc. In this case, operation is in the point-to-point configuration as set up by the configuration select bits shown. As in the case described for Bi-Sync operation, user software will then handle message preparation, transmission, reception, and accountability, while the TMS 9903 message link handles synchronization and control.

```
RESET     EQU     31
CLRRCV    EQU     30
CLRXMT    EQU     30
LDSYN2    EQU     27
CLXCRC    EQU     29
CLRCRC    EQU     29
LXCRC     EQU     24
LRCRC     EQU     12
          :
          LI      R12,>40
          SBO     RESET          Reset Device
          LDCR    @ CTLFLD,12    Load Control Register
          SBZ     CLRRCV         Initialize Receiver
          SBO     CLRXMT         Initialize Transmitter
          SBO     LDSYN2         Load Fill Character Into Sync
          LDCR    @ SYNC2,8      Character Register 2
          SBZ     LDSYN2
          SBO     CLXCRC         Clear XMT CRC Register to all zeroes
          SBZ     CLRCRC         Clear RCV CRC Register to all zeroes
          SBO     LXCRC
          LDCR    @ INIB1,8      Initialize Transmit
          LDCR    @ INIB2,8      CRC Registers to all Ones
          SBZ     LXCRC
          SBO     LRCRC
          LDCR    @ INIB1,8      Initialize Receive CRC
          LDCR    @ INIB2,8      Registers to all Ones
          SBZ     LRCRC
          :
SYNC2     BYTE    >11
CTLFLD    DATA    >004B          Sync Character
INIB1     BYTE    >57
INIB2     BYTE    >15
```



3.1.3 Mode 2 Operation

Mode 2 operation provides the framework for a general communication link control protocol using a character contained in SYNC1 for initial synchronization, and a character contained in SYNC2 for a fill sequence in the absence of data to be transmitted (XBRE = 1). The instruction sequence shown below will initialize the TMS 9903 to operate in mode 2.

```
RESET     EQU     31
CLRRCV    EQU     30
CLRXMT    EQU     30
LDSYN2    EQU     27
LDSYN1    EQU     26
          .
          .
          .
          LI      R12,>40       Initialize CRU Base
          SBO     RESET         Reset SCC and set LDCTRL
          LDCR    @ CTLFLD,12   Load Control Register and Reset LDCTRL
          SBZ     CLRRCV        Initialize Receiver
          SBO     CLRXMT        Initialize Transmitter
          SBO     LDSYN2   }
          LDCR    @ SYNC2,8  } Load Fill Character in SYNC2
          SBZ     LDSYN2   }
          SBO     LDSYN1   }
          LDCR    @ SYNC1,8  } Load Sync Character in SYNC1
          SBZ     LDSYN1   }
          .
          .
          .
SYNC1     BYTE    >16
SYNC2     BYTE    >11
CTLFLD    DATA    >00AA
```

					DRCK32	CRC1	CRC0	MDSL2	MDSL1	MDSL0	CSL1	CSL0	CLK4M	RSCL2	RSCL1	RSCL0
MEMORY ADDRESS CTLFLD	X	X	X	X	0	0	0	0	1	0	1	0	1	0	1	0
					X1 DATA CLOCK	CRCC 16		MDDE 2			EVEN PARITY		4 MHz SYSTEM CLOCK	7 BIT CHARS		

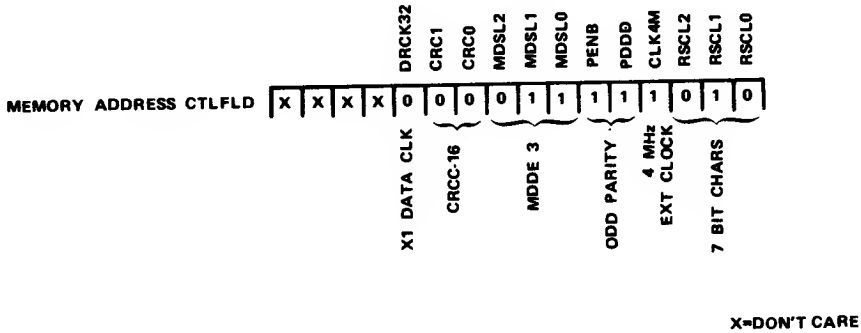
X=DON'T CARE

3.1.4 Mode 3 Operation

One of the most common synchronous data link control protocols now in use is Bi-Sync, which uses a fixed character length set of data and control characters and half-duplex operation. Bi-Sync operation is invoked with the software shown below. The software instructions shown load the control register with bits set to initialize the TMS 9903 to operate in mode 3 with received character length of seven bits and odd parity.

Note that transmitted character length is determined dynamically from the length of the character loaded into the transmit buffer. Hence, transmitting fixed seven-bit characters from the CPU to the TMS 9903, with odd parity generation selected and enabled, automatically generates the fixed length eight-bit characters required for Bi-Sync transmission. In normal operation the TMS 9903 will automatically insert SYN characters into the bit stream (from the SYNC1 register) whenever the transmitter buffer is empty and no character has been loaded by the CPU. In receive operation with RSYNDL set, the TMS 9903 will delete all SYN characters embedded in the received character stream.

```
RESET     EQU     31
CLRRCV    EQU     30
CLRXMT    EQU     30
RSYNDL    EQU     28
LDSYN2    EQU     27
LDSYN1    EQU     26
          :
          LI      R12,>40
          SBO     RESET        Issue Reset Command and Set Load
                               Control Flag LDCTRL
          LDCR    @ CTLFLD,12  Load Control Register with 12 Bits, the
                               Last of Which Resets LDCTRL
          SBZ     CLRRCV       Initialize the Receiver
          SBO     CLRXMT       Initialize the Transmitter
          SBO     LDSYN1  }
          LDCR    @ SYNC1,8 > Load SYNC1 Register
          SBZ     LDSYN1  }
          SBO     LDSYN2  }
          LDCR    @ SYNC2,8 > Load SYNC2 Register
          SBZ     LDSYN2  }
          SBO     RSYNDL       Set RCVR to Delete SYNC Characters
                               (XPRNT = 1 will Override RSYNDL)
          :
SYNC1     BYTE    >16          ASCII "SYN" Character
SYNC2     BYTE    >10          ASCII "DLE" Character
CTLFLD    DATA    >00FA        Sets TMS 9903 for Mode 3, Odd Parity, 7 Bit Characters
```



If the capability to utilize all bit combinations of the eight-bit data field is required, control bit XPRNT can be set for transparent operations. This will cause the SYNC1-SYNC1 fill sequence (i.e., normally SYN-SYN) to be replaced with SYNC2-SYNC1 (i.e., DLE SYN). Note that in transparent operation, more software is required to ensure that all data-link control commands are preceded by the DLE (data-link escape) character.

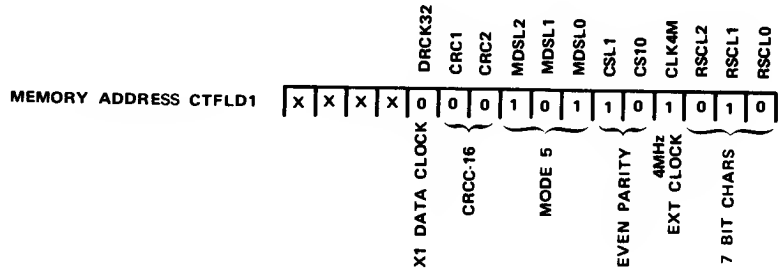
User software routines then will handle the preparation, transmission, reception, and accountability of individual messages, with link synchronization and control done by the TMS 9903.

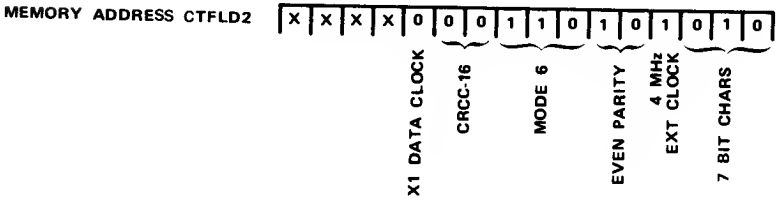
3.1.5 Mode 5 and 6 Operation

Modes 5 and 6 are the asynchronous operation modes of the TMS 9903. Mode 5 provides operation with one start and two stop bits, and mode 6 with one start and one stop bit. The software shown below will initialize the TMS 9903 into mode 5 or 6 asynchronous operation mode, depending upon the mode select bits. Loading the control register with the contents of memory address CTFLD1 selects mode 5 and CTFLD2 selects mode 6.

```
RESET     EQU     31
CLRRCV    EQU     30
CLRXMT    EQU     30
          .
          .
          .
          LI      R12,>40        Initialize CRU Base
          SBO     RESET          Reset SCC and Set LDCTRL
          LDCR    @ CTFLDX,12    Load Control Register and Reset LDCR
          SBZ     CLRRCV         Initialize Receiver
          SBO     CLRXMT         Initialize Transmitter
          .
          .
          .
CTFLD1    DATA    >016A
CTFLD2    DATA    >01AA
```

8◂





X=DON'T CARE

3.1.6 Interval Timer Operation

The software shown below will set up the interval timer to generate an interrupt every 1.6 milliseconds. The value loaded into the interval register specifies the number of 64-microsecond increments in the total interval.

```
TIMENB   EQU    20
LDIR     EQU    13
INTVL    BYTE   >19        19₁₆ = 25₁₀, 25 × 64 μs = 1.6 ms
         .
         .
         .
         SBO    LDIR       Set Load Interval Register Flag
         LDCR   @ INTVL,8  Load IR with 25 Increments
         SBZ    LDIR       Reset LDIR
         SBO    TIMENB     Enable Interval Timer Interrupts
         .
         .
         .
```

3.2 DATA TRANSMISSION

The software* shown below demonstrates a representative subroutine for transmitting a block of data.

```
        LI      R0,LISTAD     Initialize List Pointer
        LI      R1,COUNT      Initialize Block Count
        LI      R12,CRUBAS    Initialize CRU BASE
        SBO     XMTON         Turn on Transmitter (XMTON = 16)
XMTLP   TB      XBRE          Xmit Buffer Empty?
        JNE     XMTLP         No, Wait
        SBO     LXBC
                              Load Transmit Buffer, Transmit CRC
                              Register, and Increment Pointer
        LDCR    *R0+,8
        SBZ     LXBC          Reset XBRE (LXBC = 25)
        DEC     R1            Decrement Counter
        JNE     XMTLP         Loop If Not Complete
        SBO     LXCRC         Set LXCRC to
        STCR    R3,0          Read Transmit CRC
        SBZ     LXCRC         Reset LXCRC
        SWPB    R3
        TB      XBRE
        JNE     $-1
        LDCR    R3,8
        SBZ     LXBC
        SWPB    R3
        TB      XBRE
        JNE     $-1
        LDCR    R3,8
        SBZ     LXBC
        SBZ     XMTON         Turn Off Transmitter
```

After initializing the list pointer, block count, and CRU base address, XMTON is set, enabling data transmission. The internal automatic RTS signal (RTSAUT) becomes active and transmission begins when $\overline{CTS}$ becomes active. Each character to be transmitted is loaded with the LXBC flag set to load the transmit buffer and to update simultaneously the transmit CRC register. If the CRC register is not in use, the characters can be loaded with no flags set, which will then load only the transmit buffer. After the last character is transmitted, the accumulated CRC is read from the SCC and transmitted, and XMTON is reset. The transmitter and $\overline{RTS}$ become inactive upon completion of transmission of the last character. Note that $\overline{RTS}$ can be CPU-controlled by setting and resetting RTS (bit address 17). This disables the RTSAUT signal until the transmitter is reset by the RESET or CLRXMT command.

8

*The software in these examples represents generalized routines. Specific details will vary with the mode of operation selected

3.3 DATA RECEPTION

The software shown below will cause a block of data to be received and stored in memory.

```
        LI      R1,TEMPT    Initialize Working Storage
        LI      R2,RCLST    Initialize List Address
        LI      R3,MAXCNT   Initialize Max Count
        LI      R4,>0D00    Initialize End of Block Character (ASCII CR)
RCVLP   TB      21          Test for RBRL = 1
        JNE     RCVLP
        STCR    *R2,8       Store Character
        SBZ     18          Reset RBRL
        SBO     12          Set LRCRC to
        LDCR    *R2,8       Update Receive CRC Register
        SBZ     12          Reset LRCRC
        DEC     R3          Decrement Count
        JEQ     RCVEND      End if Count = 0
        CB      *R2+,R4     Compare to EOB Character and Increment Point
        JNE     RCVLP       Loop If Not Complete
RCVEND  TB      21          Test For RBRL = 1
        JNE     RCVEND
        STCR    R1,8        Store Transmitted CRC Value
        SBZ     18          Reset RBRL
        SWPB    R1          Swap CRC Bytes
        TB      21          Test for RBRL = 1
        JNE     $-1
        STCR    R1,8        Store Transmitted CRC Value
        SBZ     18          Reset RBRL
        SBO     12          Set LRCRC to
        STCR    R6,0        Read Receive CRC Register
        SBZ     12          Reset LRCRC
        C       R1,R6       If Received CRC Not Equal to
        JNE     ERR         Expected CRC, Jump to Error Routine
        RTWP                Else Return
        .
        .
        .
```

The above routine receives the block of data and compares the received CRC block check to the value accumulated in the receive CRC register. Note that in mode 1 operation the RCVEND instructions to read the received CRC could be replaced with:

```
RCVEND  SBO     26          Set RHRRD
        STCR    R1,0        Read the Receive Holding Register
        SBZ     26          Reset RHRRD
        SBO     12          Set LRCRC
        STCR    R6,0        Read Receive CRC Register
        SBZ     12          Reset LRCRC
        C       R1,R6       Compare
        JNE     ERR         Jump to Error Routine If Not Equal
        .
        .
        .
```

3.4 REGISTER LOADING AFTER INITIALIZATION

The interval register may be reloaded after initialization. For example, to change the interval of the timer to 10.24 milliseconds, the instruction sequence is

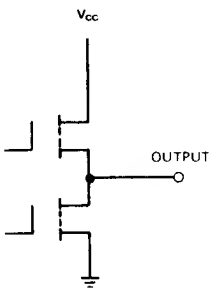
```
        SBO     13          Set Load Control Flag
        LDCR    @ INTVL2,8  Load Register
        SBZ     13          Reset Flag
        .
        .
        .
INTVL2  BYTE    10240/64
```

Caution should be exercised when transmitter interrupts are enabled to ensure that the transmitter interrupt does not occur while the load control flag is set. For example, if the transmitter interrupts between execution of the "SBO 13" and the next instruction, the transmit buffer is not enabled for loading when the transmitter interrupt service routine is entered because the LDIR flag is set. This situation may be avoided by the following sequence:

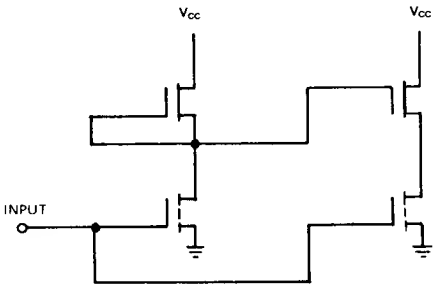
```
        BLWP    @ITVCHG     Call Subroutine
        .
        .
        .
ITVCPC  LIMI    0           Mask All Interrupts
        MOV     @ 24(R13),R12 Load CRU Base Address
        SBO     13          Set Flag
        LDCR    @INTVL2,8   Load Register
        SBZ     13          Reset Flag
        RTWP                Restore Mask and Return
        .
        .
        .
ITVCHG  DATA    ACCWP,ITVCPC
INTVL2  BYTE    10240/64
```

In this case all interrupts are masked, ensuring that all interrupts are disabled while the load control flag

EQUIVALENT OF OUTPUTS



EQUIVALENT OF INPUTS



8◂

4. TMS 9903 ELECTRICAL SPECIFICATIONS

4.1 ABSOLUTE MAXIMUM RATING OVER OPERATING FREE AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Supply voltage, V_{CC} ..... −0.3 V to 10 V
All inputs and output voltages ..... −0.3 V to 20 V
Continuous power dissipation 0.7W
Operating free-air temperature range ..... 0°C to 70°C
Storage temperature range ..... −65°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only end functional operetion of the device et these or eny other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods mey affect device reliability.

4.2 RECOMMENDED OPERATING CONDITIONS

PARAMETER	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH}	2.2	2.4	V_{CC}	V
Low-level input voltage, V_{IL}	V_{ss} −.3	0.4	0.8	V
Operating free-air temperature, T_A	0		70	°C

4.3 ELECTRICAL CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (UNLESS OTHERWISE NOTED)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_I	Input current (any input)	V_I = 0 V to V_{CC}	−10		10	μA
V_{OH}	High-level output voltage	I_{OH} = −100 μA I_{OH} = −400 μA	2.4 2	3		V
V_{OL}	Low-level output voltage	I_{OL} = 3.2 mA			0.4	V
$I_{CC(av)}$	Average supply current from V_{CC}	Operating at $t_{c(\phi)}$ = 250 ns, T_A = 25°C			100	mA
C_i	Capacitance, any input	f = 1 MHz, all other pins at 0 V			15	pF

8

4.4 TIMING REQUIREMENTS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS

PARAMETER		MIN	TYP	MAX	UNIT
$t_{c(\phi)}$	Clock cycle time		333	2000	ns
$t_{r(\phi)}$	Clock rise time		12		ns
$t_{f(\phi)}$	Clock fall time		12		ns
$t_{w(\phi H)}$	Pulse width, clock high		240		ns
$t_{w(\phi L)}$	Pulse width, clock low		55		ns
$t_{w(CC)}$	CRUCLK pulse width		100		ns
$t_{su(ad)}$	Address setup time, CRUOUT before CRUCLK		220		ns
$t_{su(CF)}$	Chip enable setup time before CRUCLK		180		ns
$t_{h(ad)}$	Address hold time, CE and CRUOUT after CRUCLK		80		ns
$t_{h(CI)}$	Hold time, CRUIN after address		20		ns
$t_{d(ad\ CI)}$	Delay time, address to CRUIN valid		400		ns

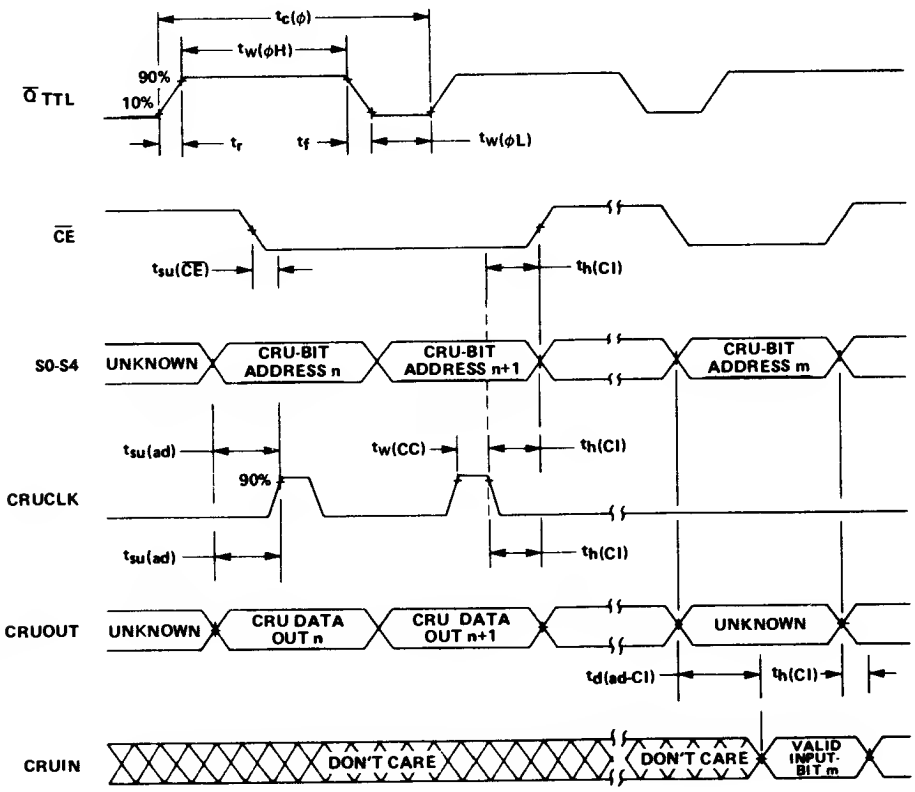
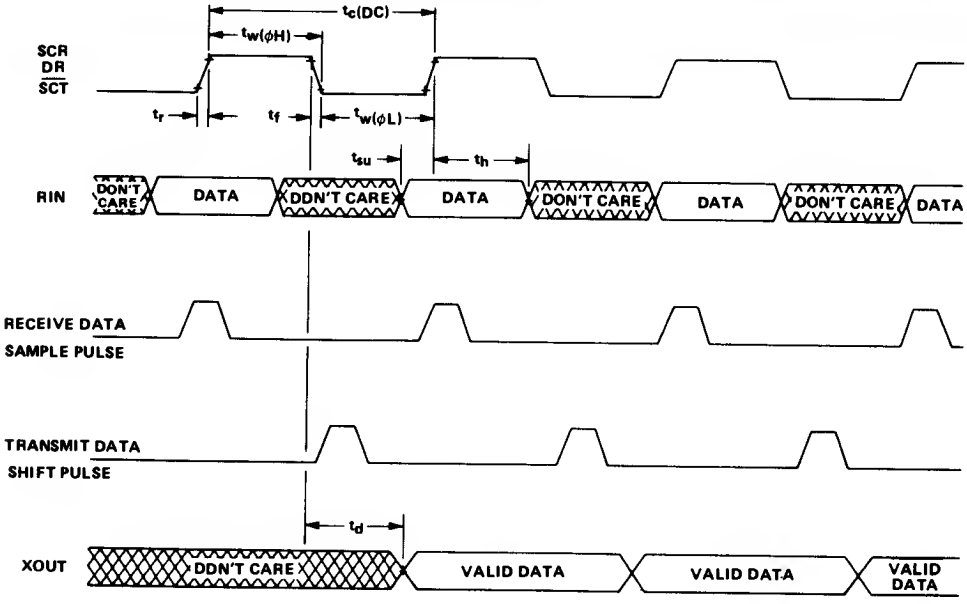


FIGURE 25. TIMING DIAGRAM



	PARAMETER	MIN	TYP	MAX	UNIT
$t_{c(DC)}$	Receiver/transmit data clock cycle time		4		ns
$t_{w(\phi H)}$	Clock pulse width (high level)		2		ns
$t_{w(\phi L)}$	Clock pulse width (low level)		2		ns
t_r	Rise time		12		ns
t_f	Fall time		12		ns
t_{su}	Setup time for RIN before SCR		250		ns
t_h	Hold time for RIN after SCR		50		ns
t_d	Delay time, SCT to valid XOUT		400		ns

▶8

FIGURE 26. RECEIVE/TRANSMIT DATA CLOCK TIMING DIAGRAM

TIM 9904
FOUR-PHASE CLOCK GENERATOR/DRIVER

1. INTRODUCTION

1.1 Description

The TIM 9904 four-phase clock generator/driver (SN74LS362) is a 20-pin dual-in-line package peripheral device designed for use with the Texas Instruments TMS 9900 microprocessor family and other microprocessors. The TIM 9904 internal oscillator can be controlled by a fundamental or overtone crystal, or capacitor and a tank circuit, or an external oscillator. The TIM 9904 is fabricated using low-power Schottky technology and is available in both plastic and ceramic packages.

1.2 Key Features

- Clock generator/driver for the TMS 9900 or other microprocessors
- MOS and TTL four-phase outputs
- Self-contained oscillator can be crystal- or tank-controlled
- External oscillator can be used
- Clocked D-type flip-flop with Schmitt-trigger input for reset signal synchronization.
- Standard 20 pin plastic and ceramic package

2. ARCHITECTURE

The TIM 9904 clock generator/driver (Figure 1) comprises an oscillator, a divide-by-four counter, a second divide-by-four-counter with gating to generate four clock phases, high-level (12-volt) output drivers, low-level (5-volt) complementary output drivers, and a D-type flip-flop controlled by an external signal and a $\phi 3$ clock. The four high-level clock phases provide clock inputs to a TMS 9900 (or other) microprocessor. The four complementary TTL-level clocks can be used to time memory or other logic functions in a TMS 9900 computer system. The D-type flip-flop can be used, for example, to provide a reset signal to a TMS 9900, timed by $\phi 3$, on receipt of an input to the FFD input from power turn-on or a manual switch closure. Other applications are possible. A safety feature incorporated in the ϕ outputs causes the ϕ outputs to go low if an open occurs in the V_{CC} supply common to TIM 9904 and TMS 9900, thus protecting the TMS 9900.

The frequency of the internal oscillator can be established by a quartz crystal or a capacitor and LC circuit. Either a fundamental or overtone crystal may be used. The LC circuit connected to the tank inputs selects the desired crystal overtone or establishes the internal oscillator frequency when a capacitor is used instead of a crystal. An LC circuit must always be used at the tank inputs when using the internal oscillator. An external oscillator may be used, if desired.

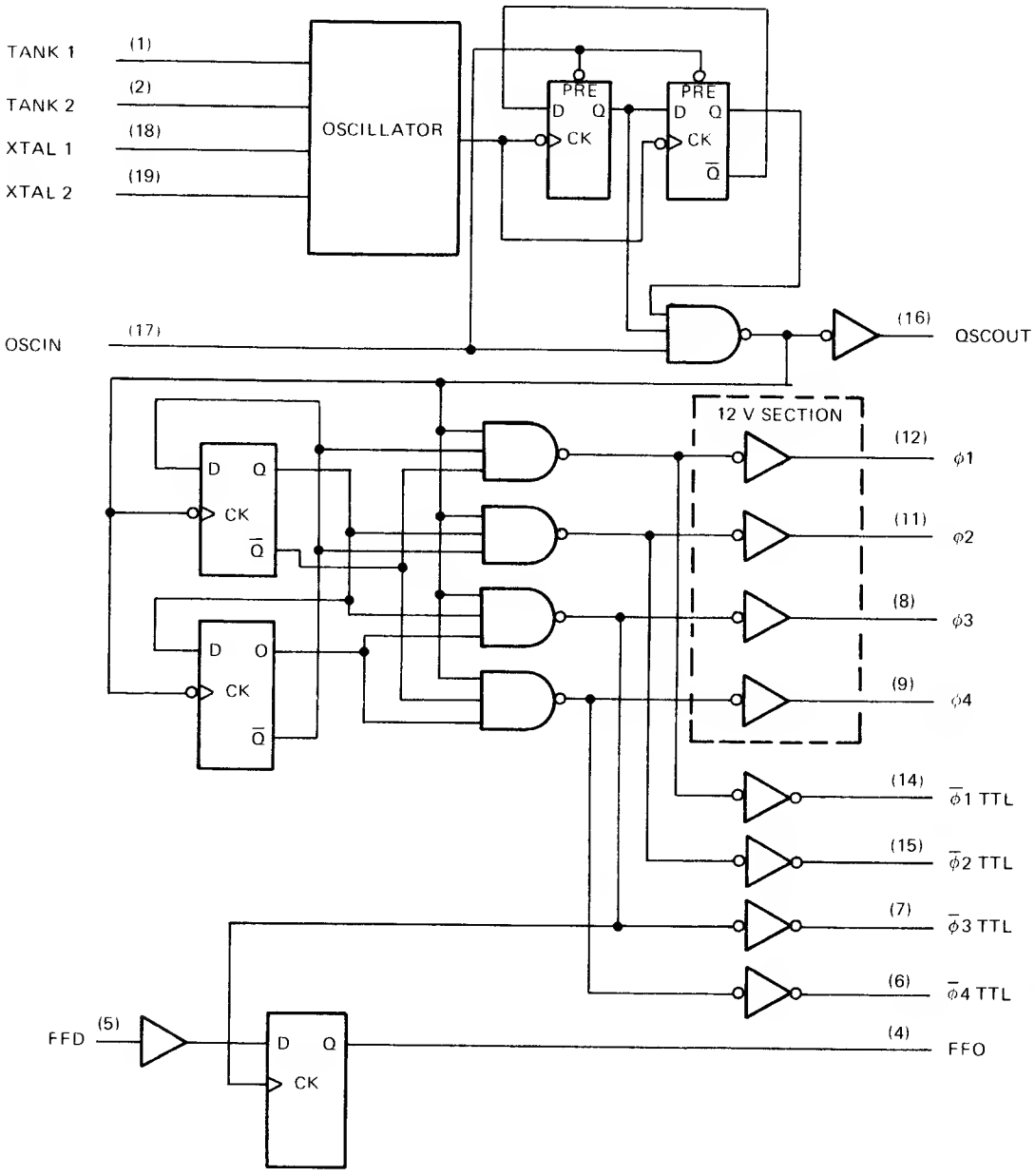


FIGURE 1—TIM 9904 CLOCK GENERATOR/DRIVER FUNCTION BLOCK DIAGRAM

3. DEVICE OPERATION

Connected to a TMS 9900 as shown in Figure 2, the TIM 9904 oscillator operates with a quartz crystal and an LC circuit connected to the tank terminals. For operation of the TMS 9900 microprocessor at 3 MHz, the frequency reference requires a resonant frequency of 48 MHz (16 x 3 MHz). The quartz crystal used as a frequency reference should be designed for series-mode operation with a resistance in the 20- to 75-ohm range, and be capable of a minimum 2-mW power dissipation. Typical frequency tolerance is ±0.005 percent. For 48-MHz operation a third-overtone crystal is used. The inductance L connected across the tank terminals should be 0.47 μH ± 10 percent, and the capacitance C (including board capacity) should be 22 pF ± 5 percent. The LC circuit should be tuned to the third-overtone crystal frequency for best results. The tank circuit should be physically located as close as possible to the TMS devices 9904.

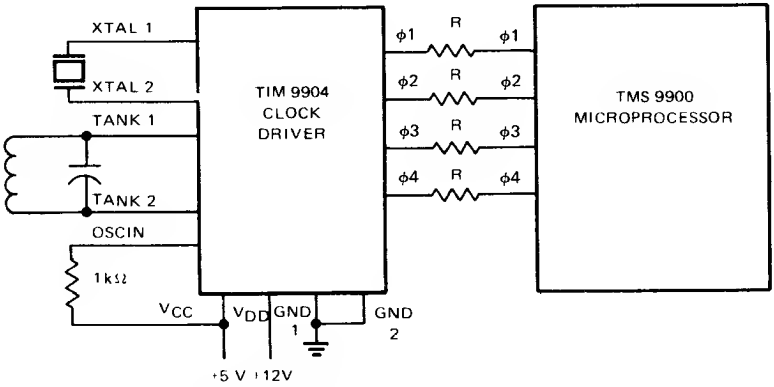


FIGURE 2—TIM 9904, CRYSTAL-CONTROLLED OPERATION

A 0.1-μF capacitor can be substituted for the quartz crystal. With a capacitor rather than a crystal, the LC tuned circuit establishes the operating frequencies. LC component values for operation at any frequency can be computed from $f_{osc} = 1/(2\pi\sqrt{LC})$ where f_{osc} is the oscillator frequency, L is the inductance value in henries, and C is the capacitance value in farads.

When the internal oscillator is used, OSCIN should be connected to V_{CC} through a resistor (1 kΩ nominal), and an LC tank circuit must be connected to the tank inputs except when a fundamental crystal is being used. An external oscillator can be used by connecting it to OSCIN and disabling the internal oscillator by connecting the crystal terminals to V_{CC} and leaving the tank inputs open. The external oscillator must have a frequency four times the desired output clock frequency and a 25 percent duty cycle. The first low-level external clock pulse will preset the divide-by-four counter, allowing the external oscillator signal to directly drive the phase generator. Figure 3 is a timing diagram illustrating operation with an external oscillator.

Resistors between the TIM 9904 $\phi1$, $\phi2$, $\phi3$, and $\phi4$ outputs and the corresponding clock input terminals of the TMS 9900 should be in the 10- 20-ohm range (see Figure 2). The purpose of the resistors is to minimize overshoot and undershoot. The required resistance value is dependent on circuit layout; clock signal interconnections should be as short as possible.

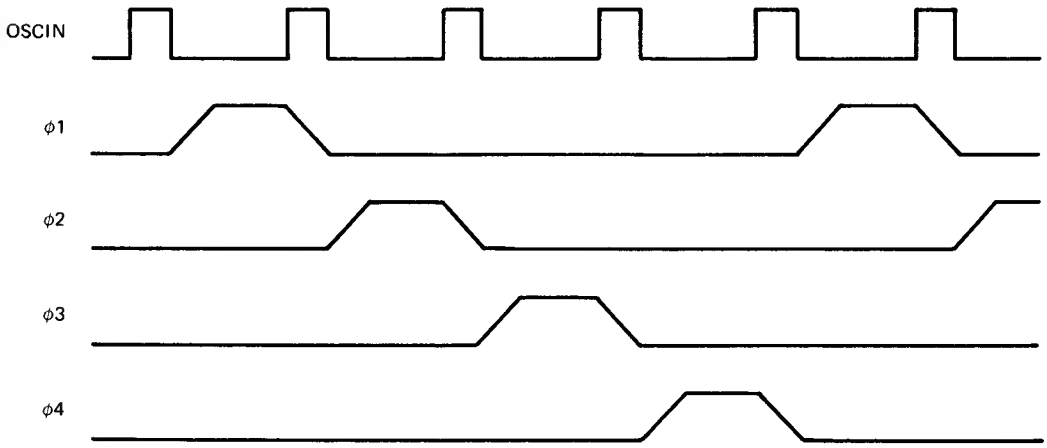


FIGURE 3—EXTERNAL OSCILLATOR TIMING FOR USE WITH TIM 9904

The D-type flip-flop associated with TIM 9904 pins FFD and FFQ can be used to provide a power-on reset and a manual reset to the TMS 9900 as shown in Figure 4. A Schmitt-trigger circuit driving the D input generates a fast-rising waveform when the input voltage rises to a specific value. At power turn-on, voltage across the 0.1 μF capacitor in Figure 4 will rise towards V_{CC}. This circuit provides a delay that resets the TMS 9900 after V_{CC} has stabilized. An optional manual reset switch can be connected to the delay circuit to reset the TMS 9900 at any time. The TMS 9900 HOLD signal could alternately be actuated by FFD.

The ground terminals GND1 and GND2 normally should be connected together and to system ground.

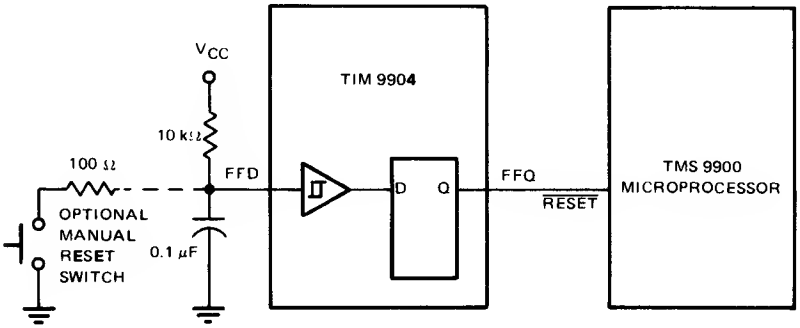


FIGURE 4—POWER-ON RESET

4. DEVICE APPLICATION

4.1 Modes of Operation

The TIM 9904 may be used in one of the following modes to provide clocking for the TMS 9900 or other microprocessor:

- *Overtone operation* — overtone crystal; tank-circuit bandpass filters the selected harmonic
- *Fundamental operation* — fundamental crystal; tank circuit not required
- *Tank-controlled operation* — no crystal; frequency determined as resonant frequency of the tank circuit
- *Externally-controlled operation* — internal oscillator disabled; TTL input signal determines frequency.

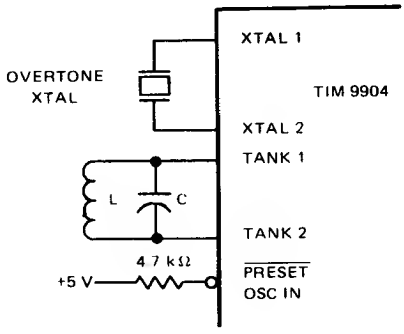
4.1.1 Overtone Operation

Overtone operation is used when crystal-stabilized, high-frequency ($f_{cy} > 1.5$ MHz) clocking is required. The crystal is operated at a harmonic of its fundamental frequency, and the tank-circuit bandpass filters the crystal frequency so as to select the desired harmonic. For example, if 3-MHz operation is required ($f_{cy} = 3$ MHz), f_{osc} must be 48 MHz. Since fundamental crystals are generally not available with frequencies above approximately 24 MHz, a 48-MHz, third-overtone crystal may be employed. The tank circuit should have a resonant frequency of 48 MHz to bandpass filter the third overtone.

The resonant frequency is determined by the equation:

$$f_{res} = \frac{1}{2\pi\sqrt{LC}}$$

The PRESET/OSCIN input is held at high. Figure 5 typifies the connection of components for overtone operation.



$$f_{OSC} = f_{XTAL} = 16\, f_{cy} = \frac{1}{2\pi\sqrt{LC}}$$

$$f_{OSC} = 4\, f_{OSC\ OUT}$$

$$0.5\ \text{MHz} < f_{cy} < 3\text{MHz}$$

FIGURE 5—OVERTONE OPERATION

4.1.2 Fundamental Operation

If a crystal is available with a fundamental frequency 16 times the required f_{cy}, a tank circuit is not required and the TANK1 and TANK2 inputs are connected to each other through a 100-ohm resistor, as shown in Figure 6. The PRESET/OSCIN input is held at high level.

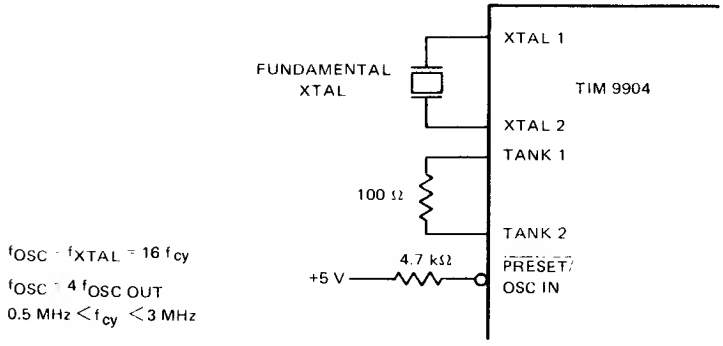


FIGURE 6—FUNDAMENTAL-FREQUENCY CRYSTAL OPERATION

4.1.3 Tank Controlled Operation

For applications in which crystal quality stability is not needed, the crystal may be replaced with a 0.1-μF capacitor and $f_{osc} = f_{res} = 1/(2\pi\sqrt{LC})$. Slight variations with changing V_{CC} and temperature can be expected. Tank-controlled operation interconnections are shown in Figure 7.

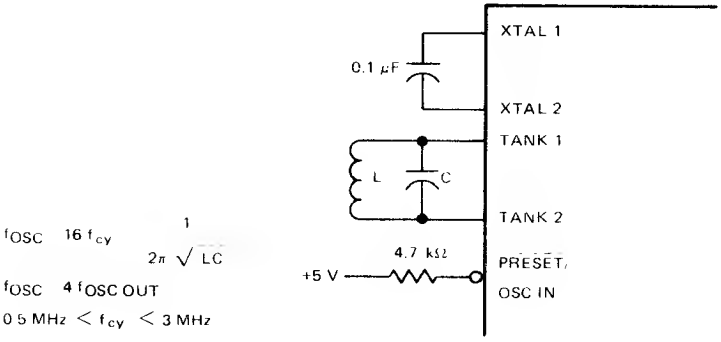
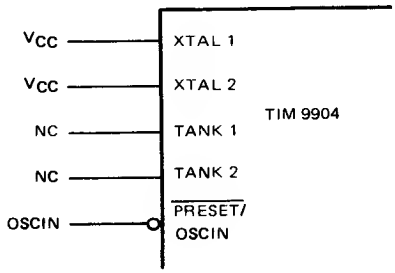


FIGURE 7—TANK-CONTROLLED FREQUENCY OPERATION

4.1.4 *Externally Controlled Operation*

If a TTL signal is available with the appropriate frequency and waveform, such signal may be connected to the $\overline{\text{PRESET}}$/OSCIN input of the TIM 9904 as shown in Figure 8. The internal oscillator is disabled by leaving tank inputs open and by connecting the crystal inputs to V_{CC}.



$f_{OSCIN} = f_{cy}$
$f_{OSCIN} = f_{OSCOUT}$, $OSCOUT = \overline{OSCIN}$
$0.5\ MHz < f_{cy} < 3\ MHz$

FIGURE 8—EXTERNALLY-CONTROLLED OPERATION

4.2 Component Selection

The criterion for selecting the values of the discrete components to be used with the TIM 9904 are recommended in this section.

4.2.1 *Crystal*

The following crystal specifications are suggested.

- Series resonant, 20- to 75- ohm series resistance, 2-mW maximum power dissipation.
- $f_{XTAL} = 16\ f_{cy}$
- For $f_{cy} = 3$ MHz, specify 48 MHz, third overtone
- Suggested stability: 0.05 percent from 0° to 70°C.

8◀

4.2.2 *Tank Circuit*

Because the value of the capacitance will be in the picofarad range, board capacity must be considered when selecting component values for the LC tank circuit. The board capacitance (C_B) will be additive to the device capacitance (C_D), as shown in Figure 9. Board capacitance may be computed in the following manner:

(1) Connect devices to the TIM 9904 as shown in Figure 5. To ensure that $f_{cy} \leq 3$ MHz, select values for L and C_D such that

$$\frac{1}{2\pi\sqrt{LC_D}} \leq 48\text{ MHz}$$

(2) Measure the frequency (f_{CY}) of one of the TTL clock outputs ($\overline{\phi 1}$, $\overline{\phi 2}$, $\overline{\phi 3}$, or $\overline{\phi 4}$).

(3) Since $f_{osc} = 16\, f_{cy} = \dfrac{1}{2\pi\sqrt{LC}} = \dfrac{1}{2\pi\sqrt{L(C_B + C_D)}}$

C_B can be determined from the equation:

$$C_B = \frac{1}{L \cdot (32\pi F_{cy})^2} - C_D$$

For example, assume that L = 0.5 μH and C_D = 22 pF:

$$\frac{1}{2\pi\sqrt{LC_D}} = 47.987\text{ MHz} < 48\text{ MHz}$$

F_{cy} is determined to be 2.413 MHz. Therefore:

$$C_B = \frac{1}{L \cdot (32\pi F_{cy})^2} - C_D = \frac{1}{0.5\text{x}10^{-6} \cdot (32\pi \cdot 2.413\text{x}10^6)^2} - 22 \text{ x } 10^{-12}\text{ farad}$$

C_B = 11.98 pF ≈ 12 pF.

In order to obtain $f_{res} \approx 48$ MHz using 0.5 μH, $C = C_B + C_D$ = 22 pF; thus, C_D must be 10 pF with a board capacitance of 12 pF. When using the tank circuit in overtone operation, the f_{res} should be within 5 percent of f_{osc}, requiring that the product of LC should be within 10 percent of the ideal values for $f_{res} = f_{osc}$. This may be accomplished by using devices with nominal values so that $f_{osc} = 1/(2\pi\sqrt{LC})$, and with 5 percent tolerances.

For the above example with C_B = 12 pF:

L = 0.5 μH ± 5 percent

C = 10 pF ± 5 percent

thus providing a comfortable margin for deviations of component value on a production basis.

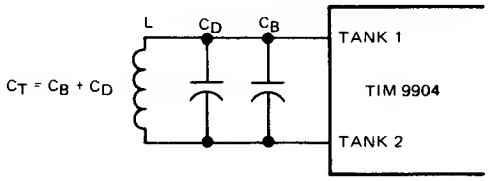


FIGURE 9—EFFECT OF BOARD ON TANK CIRCUIT RESONANT FREQUENCY

4.2.3 Series Resistors

Resistors with values on the order of 10 to 22-ohms should be installed between the $\phi 1$-$\phi 4$ outputs of the TIM 9904 and the corresponding inputs of TMS 9900. These serve two purposes:

- Reduce overshoot and ringing
- Protect the drivers from overvoltage and undervoltage signals.

Connect the resistors as illustrated in Figure 9.

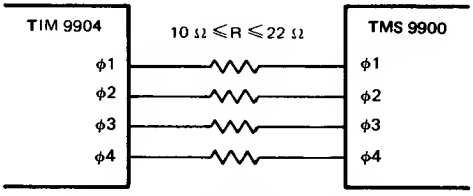
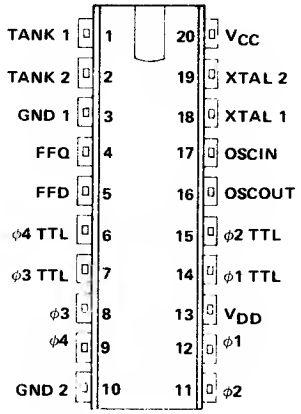


FIGURE 10—SERIES MOS CLOCK RESISTORS

8

4.3 TIM 9904 Terminal Assignments

SIGNATURE	PIN	I/O	DESCRIPTION
TANK 1	1	I	Tank circuit connection
TANK 2	2	I	Tank circuit connection
GND 1	3		Ground reference
FFQ	4	O	Output of D flip-flop
FFD	5		D Input to Schmitt triggered flip-flop
$\overline{\phi}$4 TTL	6	O	TTL Phase 4 inverted
$\overline{\phi}$3 TTL	7	O	TTL Phase 3 inverted
ϕ3	8	O	MOS Phase 3
ϕ4	9	O	MOS Phase 4
GND 2	10		Ground reference
ϕ2	11	O	MOS Phase 2
ϕ1	12	O	MOS Phase 1
V_{DD}	13	I	Supply voltage 12 V nominal
$\overline{\phi}$1 TTL	14	O	TTL Phase 1 inverted
$\overline{\phi}$2 TTL	15	O	TTL Phase 2 inverted
OSCOUT	16	O	Oscillator output
OSCIN	17	I	TTL external oscillator input
XTAL1	18	I	Crystal
XTAL2	19	I	Crystal
V_{CC}	20	I	Supply voltage 5 V nominal



▶8

5. ELECTRICAL SPECIFICATIONS

5.1 Absolute Maximum Ratings Over Operating Free-Air Temperature Range (Unless Otherwise Noted)

Supply voltage: V_{CC} (see Note 1) ..... 7 V
V_{DD} (see Note 1) ..... 13 V
Input voltage: OSCIN ..... 5.5 V
FFD ..... −0.5 V to 7 V
Operating free-air temperature range ..... 0°C to 70°C
Storage temperature range ..... −65°C to 150°C

NOTE 1 Voltage values are with respect to the network ground terminals connected together

5.2 Recommended Operating Conditions

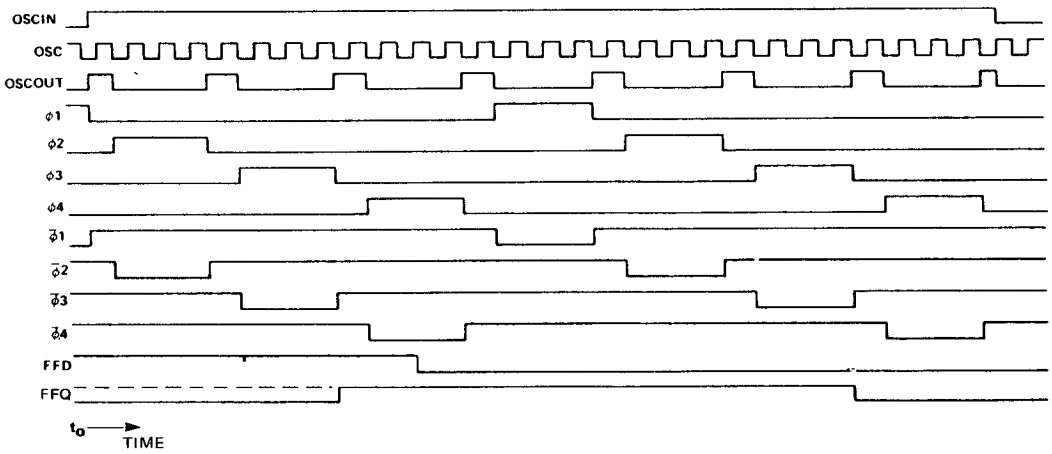
		MIN	NOM	MAX	UNIT
Supply voltages	V_{CC}	4.75	5	5.25	V
	V_{DD}	11.4	12	12.6	V
High-level output current, I_{OH}	$\phi1, \phi2, \phi3, \phi4$			−100	μA
	All others			−400	μA
Low-level output current, I_{OL}	$\phi1, \phi2, \phi3, \phi4$			4	mA
	All others			8	mA
Internal oscillator frequency, f_{osc}			48	54	MHz
External oscillator pulse width, $t_{w(osc)}$		25			ns
Setup time, FFD input (with respect to falling edge of $\phi3$), t_{su}		50			ns
Hold time, FFD input (with respect to falling edge of $\phi3$), t_h		−30			ns
Operating free-air temperature, T_A		0		70	°C

8◀

5.3 Electrical Characteristics Over Recommended Operating Free-Air Temperature Range (Unless Otherwise Noted)

PARAMETER			TEST CONDITIONS		MIN	TYP‡	MAX	UNIT
V_{IH}	High-level input voltage				2			V
V_{IL}	Low-level input voltage	FFD					0.5	V
		OSCIN					0.8	
$V_{T+} - V_{T-}$	Hysteresis	FFD			0.4	0.8		V
V_{IK}	Input clamp voltage		V_{CC} = 4.75 V, V_{DD} = 11.4 V, I_I = −18 mA				−1.5	V
V_{OH}	High-level output voltage	$\phi1, \phi2, \phi3, \phi4$	V_{CC} = 4.75 V, V_{DD} = 11.4 V to 12.6 V	I_{OH} = −100 μA	V_{DD}−2	V_{DD}−1.5	V_{DD}	V
		Other outputs		I_{OH} = −400 μA	2.7	3.4		
V_{OL}	Low-level output voltage	$\phi1, \phi2, \phi3, \phi4$	V_{CC} = 4.75 V, V_{DD} = 11.4 V	I_{OL} = 4 mA		0.25	0.4	mA
		Other outputs		I_{OL} = 4 mA		0.25	0.4	
				I_{OL} = 8 mA		0.35	0.5	
I_I	Input current at maximum input voltage	FFD	V_{CC} = 5.25 V, V_{DD} = 12.6 V	V_I = 7 V			0.1	mA
		OSCIN		V_I = 5.5 V			0.3	
I_{IH}	High-level input current	FFD	V_{CC} = 5.25 V, V_{DD} = 12.6 V, V_I = 2.7 V				20	μA
		OSCIN					60	
I_{IL}	Low-level input current	FFD	V_{CC} = 5.25 V, V_{DD} = 12.6 V, V_I = 0.4 V				−0.4	mA
		OSCIN					−3.2	
I_{OS}	Short-circuit output current‡	All except $\phi1, \phi2, \phi3, \phi4$	V_{CC} = 5.25 V		−20		−100	mA
I_{CC}	Supply current from V_{CC}		V_{CC} = 5.25 V, FFD and OSCIN at GND, Outputs open			105	175	mA
I_{DD}	Supply current from V_{DD}		V_{CC} = 5.25 V, V_{DD} = 12.6 V, FFD and OSCIN at GND, Outputs open			12	20	mA

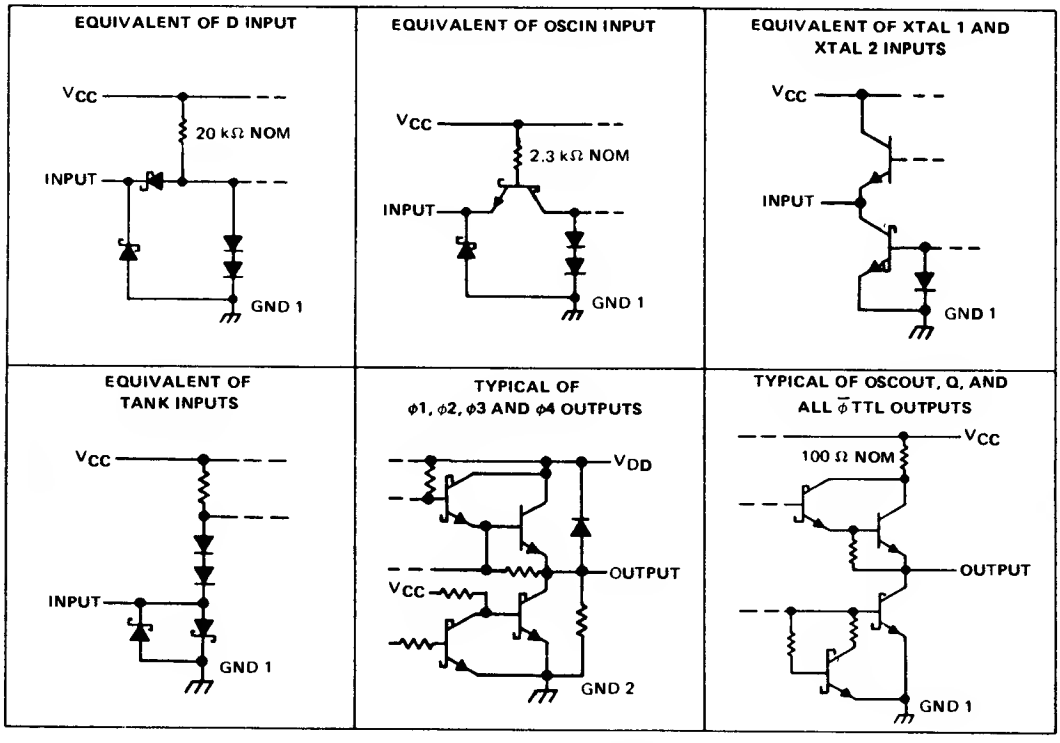
†All typical values are at V_{CC} = 5 V, V_{DD} = 12 V, T_A = 25°C.
‡Not more than one output should be shorted at a time, and duration of the short circuit should not exceed one second. Outputs $\phi1$, $\phi2$, $\phi3$, and $\phi4$ do not have short-circuit protection



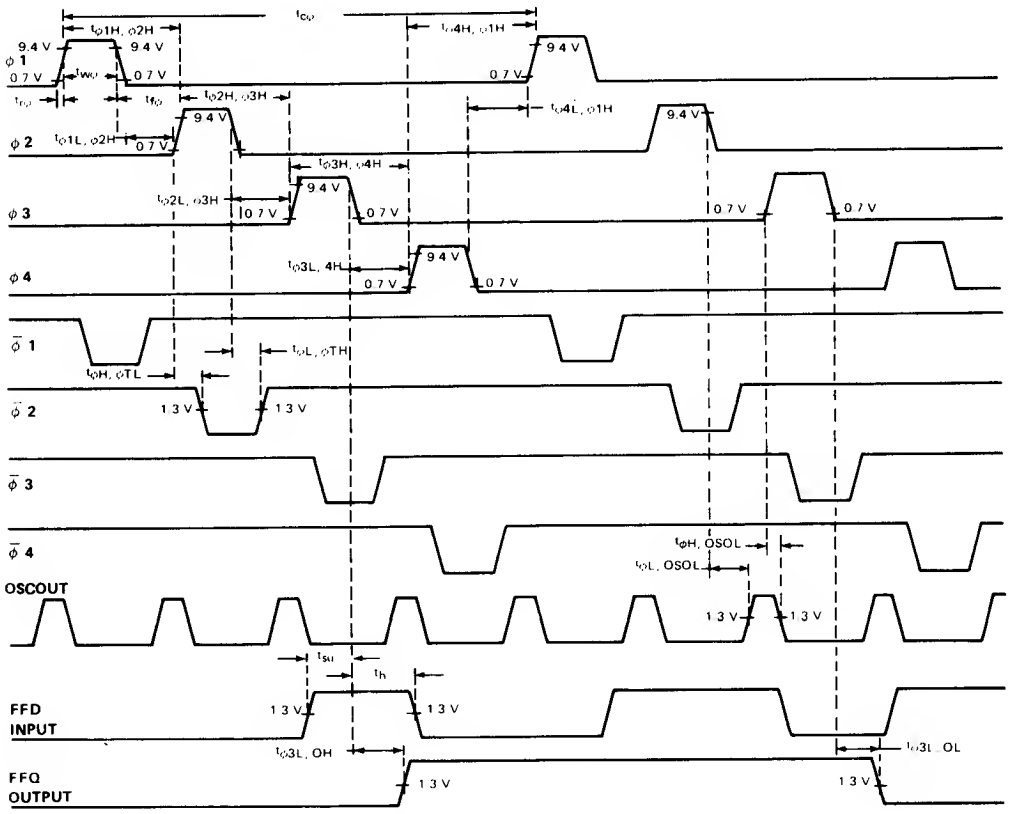
TYPICAL PHASE RELATIONSHIPS OF INPUTS AND OUTPUTS (INTERNAL OSC)

5.4 Switching Characteristics, $T_A = 25°C$, $V_{CC1} = 5$ V, $V_{CC2} = 12$ V, $f_{osc} = 48$ MHz

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
f_{out}	Output frequency, any ϕ or $\bar{\phi}$ TTL			3		MHz
f_{out}	Output frequency, OSCOUT			12		MHz
$t_{c(\phi)}$	Cycle time, any ϕ output		330	333	340	ns
$t_{r(\phi)}$	Rise time, any ϕ output		−5		20	ns
$t_{f(\phi)}$	Fall time, any ϕ output		10	14	20	ns
$t_{w(\phi)}$	Pulse width, any ϕ output high		40	55	70	ns
$t_{\phi 1L, \phi 2H}$	Delay time, $\phi 1$ low to $\phi 2$ high		0	5	15	ns
$t_{\phi 2L, \phi 3H}$	Delay time, $\phi 2$ low to $\phi 3$ high		0	5	15	ns
$t_{\phi 3L, \phi 4H}$	Delay time, $\phi 3$ low to $\phi 4$ high		0	5	15	ns
$t_{\phi 4L, \phi 1H}$	Delay time, $\phi 4$ low to $\phi 1$ high	Output loads.	0	5	15	ns
$t_{\phi 1H, \phi 2H}$	Delay time, $\phi 1$ high to $\phi 2$ high	$\phi 1, \phi 3, \phi 4$: 100 pF to GND	73	83	96	ns
$t_{\phi 2H, \phi 3H}$	Delay time, $\phi 2$ high to $\phi 3$ high	$\phi 2$: 200 pF to GND	73	83	96	ns
$t_{\phi 3H, \phi 4H}$	Delay time, $\phi 3$ high to $\phi 4$ high	Others: $R_L = 2$ kΩ,	73	83	96	ns
$t_{\phi 4H, \phi 1H}$	Delay time, $\phi 4$ high to $\phi 1$ high	$C_L = 15$ pF	73	83	96	ns
$t_{\phi H, \phi TL}$	Delay time, ϕ_n high to $\bar{\phi}_n$ TTL low		−14	−4	6	ns
$t_{\phi L, \phi TH}$	Delay time, ϕ_n low to $\bar{\phi}_n$ TTL high		−29	−19	−9	ns
$t_{\phi 3L, QH}$	Delay time, $\phi 3$ low to FFQ output high		−18	−8	2	ns
$t_{\phi 3L, OL}$	Delay time, $\phi 3$ low to FFO output low		−19	−9	1	ns
$t_{\phi L, DSOH}$	Delay time, ϕ low to OSCOUT high		−30	−20	−10	ns
$t_{\phi H, OSDL}$	Delay time, FFO high to OSCOUT low		−27	−17	−7	ns



SCHEMATICS OF INPUTS AND OUTPUTS



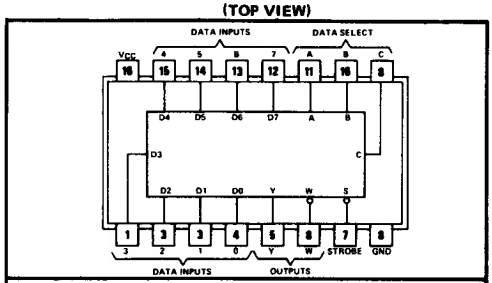
SWITCHING CHARACTERISTICS, VOLTAGE WAVEFORMS

▶8

- **Three-State Versions of '151, 'LS151, 'S151**
- **Three-State Outputs Interface Directly with System Bus**
- **Perform Parallel-to-Serial Conversion**
- **Permit Multiplexing from N-lines to One Line**
- **Complementary Outputs Provide True and Inverted Data**
- **Fully Compatible with Most TTL and DTL Circuits**

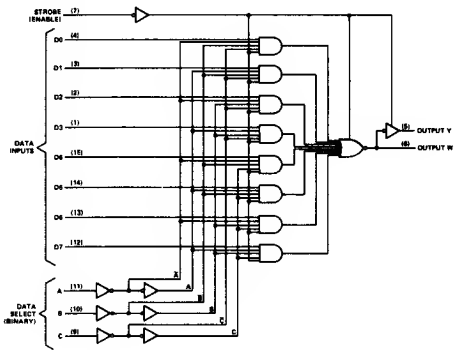
TYPE	MAX NO. OF COMMON OUTPUTS	TYPICAL AVG PROP DELAY TIME (D TO Y)	TYPICAL POWER DISSIPATION
SN54251	49	17 ns	250 mW
SN74251	129	17 ns	250 mW
SN54LS251	49	17 ns	35 mW
SN74LS251	129	17 ns	35 mW
SN54S251	39	8 ns	275 mW
SN74S251	129	8 ns	275 mW

SN54251, SN54LS251, SN54S251 . . . J OR W PACKAGE
SN74251, SN74LS251, SN54S251 . . . J OR N PACKAGE
(TOP VIEW)



positive logic: see function table

functional block diagram



description

These monolithic data selectors/multiplexers contain full on-chip binary decoding to salect one-of-eight data sources and feature a strobe-controlled three-state output. The stroba must be at a low logic level to enable these devicas. The three-state outputs permit a number of outputs to be connected to a common bus. Whan the strobe input is high, both outputs are in a high-impedance state in which both the upper and lower transistors of each totem-pole output are off, and the output neither drives nor loads the bus significantly. When the strobe is low, the outputs are activated and operate as standard TTL totem-pole outputs.

To minimize tha possibility that two outputs will attempt to take a common bus to opposite logic lavals, the output control circuitry is designed so that the average output disable time is shorter than the average output enable time. The SN54251 and SN74251 have output clamp diodes to attenuate raflections on the bus line.

FUNCTION TABLE

INPUTS				OUTPUTS	
SELECT			STROBE		
C	B	A	S	Y	W
X	X	X	H	Z	Z
L	L	L	L	D0	$\overline{D0}$
L	L	H	L	D1	$\overline{D1}$
L	H	L	L	D2	$\overline{D2}$
L	H	H	L	D3	$\overline{D3}$
H	L	L	L	D4	$\overline{D4}$
H	L	H	L	D5	$\overline{D5}$
H	H	L	L	D6	$\overline{D6}$
H	H	H	L	D7	$\overline{D7}$

H = high logic level, L = low logic level
X = irrelevant, Z = high impedence (off)
D0, D1 . . . D7 = the level of the respective D input

8

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage . . . . . 7 V
Off-state output voltage . . . . . 5.5 V
Operating free-air temperature range: SN54LS251 . . . . . −55°C to 125°C
SN74LS251 . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C

NOTE 1: Voltaga values ara with respect to network ground tarminal.

recommended operating conditions

	SN54LS251			SN74LS251			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}	4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}			−1			−2.6	mA
Low-level output current, I_{OL}			4			8	mA
Operating free-air temperature, T_A	−55		125	0		70	°C

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER		TEST CONDITIONS†		SN54LS251			SN74LS251			UNIT
				MIN	TYP‡	MAX	MIN	TYP‡	MAX	
V_{IH}	High-level input voltage			2			2			V
V_{IL}	Low-level input voltage					0.7			0.8	V
V_{IK}	Input clamp voltage	V_{CC} = MIN,	I_I = −18 mA			−1.5			−1.5	V
V_{OH}	High-level output voltage	V_{CC} = MIN, V_{IL} = MAX,	V_{IH} = 2 V, I_{OH} = MAX	2.4	3.4		2.4	3.1		V
V_{OL}	Low-level voltage	V_{CC} = MIN, V_{IH} = 2 V, V_{IL} = VIL max	I_{OL} = 4 mA		0.25	0.4		0.25	0.4	V
			I_{OL} = 8 mA					0.35	0.5	
I_{OZ}	Off-state (high-impedance-state) output current	V_{CC} = MAX, V_{IH} = 2 V	V_O = 2.7 V			20			20	μA
			V_O = 0.4 V			−20			−20	
I_I	Input current at maximum input voltage	V_{CC} = MAX,	V_I = 7 V			0.1			0.1	mA
I_{IH}	High-level input current	V_{CC} = MAX,	V_I = 2.7 V			20			20	μA
I_{IL}	Low-level input current	V_{CC} = MAX,	V_I = 0.4 V			−0.4			−0.4	mA
I_{OS}	Short-circuit output current§	V_{CC} = MAX		−30		−130	−30		−130	mA
I_{CC}	Supply current	V_{CC} = MAX, See Note 3	Condition A		6.1	10		6.1	10	mA
			Condition B		7.1	12		7.1	12	

†For conditions shown as MIN or MAX, use the appropriata velue specified under racommended oparating conditions for tha applicable type.
‡All typical values are at V_{CC} = 5 V, T_A = 25°C.
§Not more then one output should be shorted at a time, and duration of the short-circuit should not exceed one second.
NOTE 3: I_{CC} is meesured with the outputs open and all deta end select inputs at 4.5 V under the following conditions:
A. Strobe grounded.
B. Strobe at 4.5 V.

8

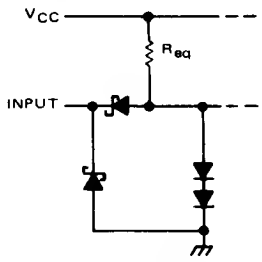
switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER¶	FROM (INPUT)	TO (OUTPUT)	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	A, B, or C (4 levels)	Y	C_L = 15 pF, R_L = 2 kΩ, See Note 4		29	45	ns
t_{PHL}					28	45	
t_{PLH}	A, B, or C (3 levels)	W			20	33	ns
t_{PHL}					21	33	
t_{PLH}	Any D	Y			17	28	ns
t_{PHL}					18	28	
t_{PLH}	Any D	W			10	15	ns
t_{PHL}					9	15	
t_{ZH}	Strobe	Y			30	45	ns
t_{ZL}					26	40	
t_{ZH}	Strobe	W			17	27	ns
t_{ZL}					24	40	
t_{HZ}	Strobe	Y	C_L = 5 pF, R_L = 2 kΩ, See Note 4		30	45	ns
t_{LZ}					15	25	
t_{HZ}	Strobe	W			37	55	ns
t_{LZ}					15	25	

¶ t_{PLH} ≡ Propagation delay time, low-to-high-level output
t_{PHL} ≡ Propagation delay time, high-to-low-level output
t_{ZH} ≡ Output enable time to high level
t_{ZL} ≡ Output enable time to low level
t_{HZ} ≡ Output disable time from high level
t_{LZ} ≡ Output disable time from low level
NOTE 4: See load circuits and waveforms on page 3-11.

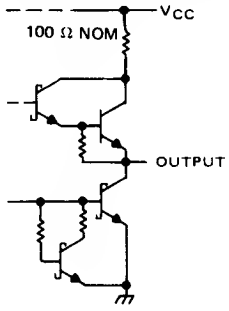
schematics of inputs and outputs

EQUIVALENT OF EACH INPUT



A, B, C, S: R_{eq} = 20 kΩ NOM
D0 thru D7: R_{eq} = 17 kΩ NOM

TYPICAL OF BOTH OUTPUTS



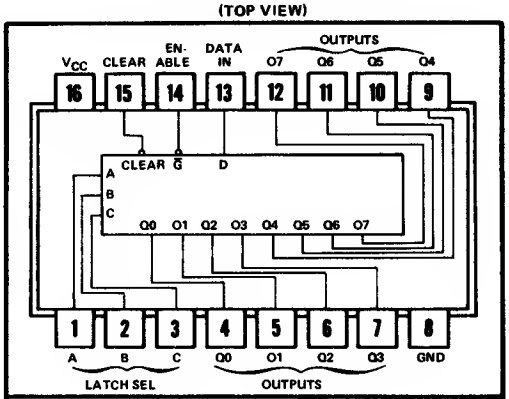
8

- 8-Bit Parallel-Out Storage Register Performs Serial-to-Parallel Conversion With Storage
- Asynchronous Parallel Clear
- Active High Decoder
- Enable/Disable Input Simplifies Expansion
- Direct Replacement for Fairchild 9334
- Expandable for N-Bit Applications
- Four Distinct Functional Modes
- Typical Propagation Delay Times:

	'259	'LS259
Enable-to-Output . . .	12	17
Data-to-Output	12	18
Address-to-Output . .	16	20
Clear-to-Output	16	20

- Fan-Out
  - I_{OL} (Sink Current)
    - '259 . . . . . 16 mA
    - SN54LS259 . . . . . 4 mA
    - SN74LS259 . . . . . 8 mA
  - I_{OH} (Source Current)
    - '259 . . . . . −0.8 mA
    - 'LS259 . . . . . −0.4 mA
- Typical I_{CC}
    - '259 . . . . . 60 mA
    - 'LS259 . . . . . 22 mA

SN54259, SN54LS259 . . . J OR W PACKAGE
SN74259, SN74LS259 . . . J OR N PACKAGE
(TOP VIEW)



logic: see function table

description

These 8-bit addressable latches are designed for general purpose storage applications in digital systems. Specific uses include working registers, serial-holding registers, and active-high decoders or demultiplexers. They are multifunctional devices capable of storing single-line data in eight addressable latches, and being a 1-of-8 decoder or demultiplexer with active-high outputs.

Four distinct modes of operation are selectable by controlling the clear and enable inputs as enumerated in the function table. In the addressable-latch mode, data at the data-in terminal is written into the addressed latch. The addressed latch will follow the data input with all unaddressed latches remaining in their previous states. In the memory mode, all latches remain in their previous states and are unaffected by the data or address inputs. To eliminate the possibility of entering erroneous data in the latches, the enable should be held high (inactive) while the address lines are changing. In the 1-of-8 decoding or demultiplexing mode, the addressed output will follow the level of the D input with all other outputs low. In the clear mode, all outputs are low and unaffected by the address and data inputs.

FUNCTION TABLE

INPUTS		OUTPUT OF ADDRESSED LATCH	EACH OTHER OUTPUT	FUNCTION
CLEAR	$\overline{G}$			
H	L	D	Q_{i0}	Addressable Latch
H	H	Q_{i0}	Q_{i0}	Memory
L	L	D	L	8-Line Demultiplexer
L	H	L	L	Clear

LATCH SELECTION TABLE

SELECT INPUTS			LATCH ADDRESSED
C	B	A	
L	L	L	0
L	L	H	1
L	H	L	2
L	H	H	3
H	L	L	4
H	L	H	5
H	H	L	6
H	H	H	7

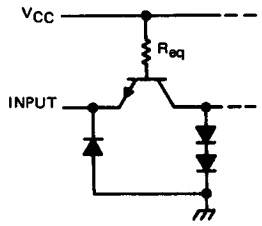
H ≡ high level, L ≡ low level
D ≡ the level at the data input
Q_{i0} ≡ the level of Q_i (i = 0, 1, . . . 7, as appropriate) before the indicated steady-state input conditions were established.

▶8

schematic of inputs and outputs

'259

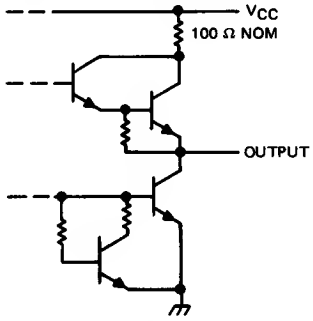
EQUIVALENT OF EACH INPUT



Latch select, data in, or clear: R_{eq} = 4 kΩ NOM
Enabla: R_{eq} = 2.2 kΩ NOM

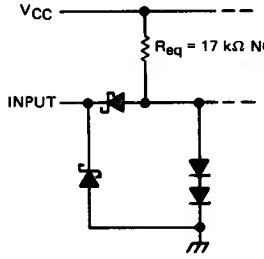
'259

TYPICAL OF ALL OUTPUTS



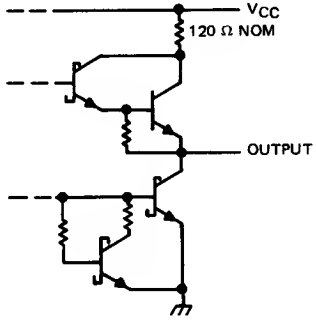
'LS259

EQUIVALENT OF EACH INPUT



'LS259

TYPICAL OF ALL OUTPUTS



absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage (see Note 1)		7 V
Input voltage:	SN54259, SN74259	5.5 V
	SN54LS259, SN74LS259	7 V
Operating free-air temperature range:	SN54259, SN54LS259	−55°C to 125°C
	SN74259, SN74LS259	0°C to 70°C
Storage temperature range		−65°C to 150°C

NOTE 1: Voltage values are with respect to network ground terminal.

8◀

recommended operating conditions

		SN54LS259			SN74LS259			UNIT
		MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}		4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}				−400			−400	μA
Low-level output current, I_{OL}				4			8	mA
Width of clear or enable pulse, t_w		15			15			ns
Setup time, t_{su}	Data	15↑			15↑			ns
	Address	15↑			15↑			
Hold time, t_h	Data	0↑			0↑			ns
	Address	0↑			0↑			
Operating free-air temperature, T_A		−55		125	0		70	°C

†The arrow indicates that the rising edge of the enable pulse is used for reference.

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER		TEST CONDITIONS†			SN54LS259			SN74LS259			UNIT
					MIN	TYP‡	MAX	MIN	TYP‡	MAX	
V_{IH}	High level input voltage				2			2			V
V_{IL}	Low level input voltage						0.7			0.8	V
V_{IK}	Input clamp voltage	V_{CC} = MIN,	I_I = −18 mA				−1.5			−1.5	V
V_{OH}	High-level output voltage	V_{CC} = MIN, V_{IL} = V_{IL} max,	V_{IH} = 2 V, I_{OH} = −0.4 mA		2.5	3.4		2.7	3.4		V
V_{OL}	Low-level output voltage	V_{CC} = MIN, V_{IL} = V_{IL} max,	V_{IH} = 2 V,	I_{OL} = 4 mA		0.25	0.4		0.25	0.4	V
				I_{OL} = 8 mA					0.35	0.5	
I_I	Input current at maximum input voltage	V_{CC} = MAX,	V_I = 7 V				0.1			0.1	mA
I_{IH}	High-level input current	V_{CC} = MAX,	V_I = 2.7 V				20			20	μA
I_{IL}	Low-level input current	V_{CC} = MAX,	V_I = 0.4 V				−0.4			−0.4	mA
I_{OS}	Short-circuit output current§	V_{CC} = MAX			−20		−100	−20		−100	mA
I_{CC}	Supply current	V_{CC} = MAX,	See Note 2			22	36		22	36	mA

†For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
‡All typical values are at V_{CC} = 5 V, T_A = 25°C.
§Not more than one output should be shorted at a time, and duration short-circuit should not exceed one second.
NOTE 2: I_{CC} is measured with the inputs grounded and the outputs open.

switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER	FROM (INPUT)	TO (OUTPUT)	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PHL}	Clear	Any Q	C_L = 15 pF, R_L = 2 kΩ, See Note 3		17	27	ns
t_{PLH}	Data	Any Q			20	32	ns
t_{PHL}					13	21	
t_{PLH}	Address	Any Q			24	38	ns
t_{PHL}					18	29	
t_{PLH}	Enable	Any Q			22	35	ns
t_{PHL}					15	24	

t_{PLH} ≡ propagation delay time, low-to-high-level output
t_{PHL} ≡ propagation delay time, high-to-low-level output
NOTE 3: Load circuit is shown on page 3-11.

'147, 'LS147

- **Encodes 10-Line Decimal to 4-Line BCD**
- **Applications Include:**

  Keyboard Encoding
  Range Selection

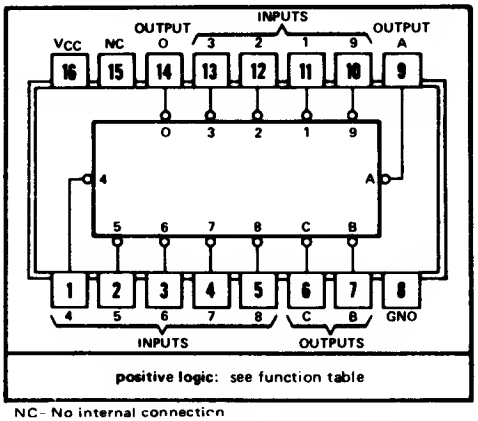
'148, 'LS148

- **Encodes 8 Data Lines to 3-Line Binary (Octal)**
- **Applications Include:**

  N-Bit Encoding
  Code Converters and Generators

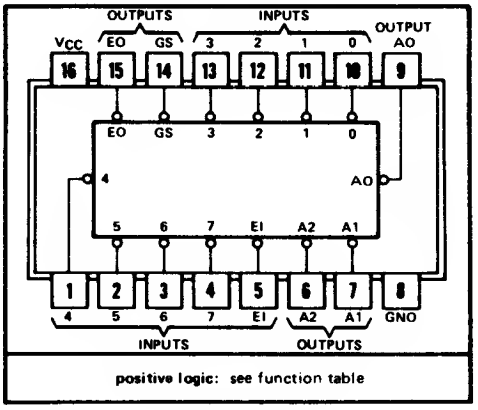
TYPE	TYPICAL DATA DELAY	TYPICAL POWER DISSIPATION
'147	10 ns	225 mW
'148	10 ns	190 mW
'LS147	15 ns	60 mW
'LS148	15 ns	60 mW

SN54147, SN54LS147 . . . J OR W PACKAGE
SN74147, SN74LS147 . . . J OR N PACKAGE
(TOP VIEW)



NC – No internal connection

SN54148, SN54LS148 . . . J OR W PACKAGE
SN74148, SN74LS148 . . . J OR N PACKAGE
(TOP VIEW)



description

These TTL encoders feature priority decoding of the inputs to ensure that only the highest-order data line is encoded. The '147 and 'LS147 encode nine data lines to four-line (8-4-2-1) 8CD. The implied decimal zero condition requires no input condition as zero is encoded when all nine data lines are at a high logic level. The '148 and 'LS148 encode eight data lines to three-line (4-2-1) binary (octal). Cascading circuitry (enable input EI and enable output EO) has been provided to allow octal expansion without the need for external circuitry. For all types, data inputs and outputs are active at the low logic level. All inputs are buffered to represent one normalized Series 54/74 or 54LS/74LS load, respectively.

'147, 'LS147
FUNCTION TABLE

INPUTS									OUTPUTS			
1	2	3	4	5	6	7	8	9	D	C	B	A
H	H	H	H	H	H	H	H	H	H	H	H	H
X	X	X	X	X	X	X	X	L	L	H	H	L
X	X	X	X	X	X	X	L	H	L	H	H	H
X	X	X	X	X	X	L	H	H	H	L	L	L
X	X	X	X	X	L	H	H	H	H	L	L	H
X	X	X	X	L	H	H	H	H	H	L	H	L
X	X	X	L	H	H	H	H	H	H	L	H	H
X	X	L	H	H	H	H	H	H	H	H	L	L
X	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

H = high logic level, L = low logic level, X = irrelevant

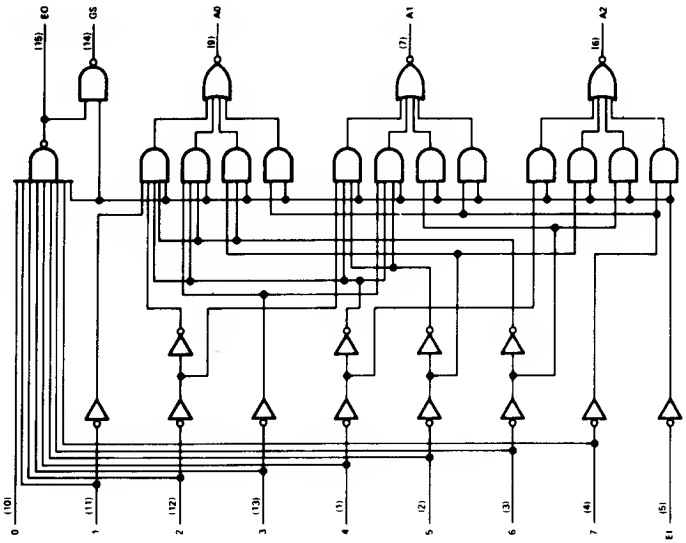
'148, 'LS148
FUNCTION TABLE

INPUTS								OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

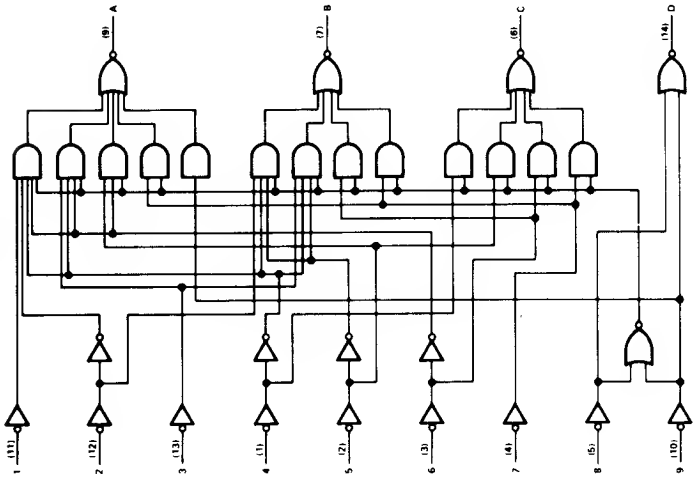
8

functional block diagrams

'148, 'LS148



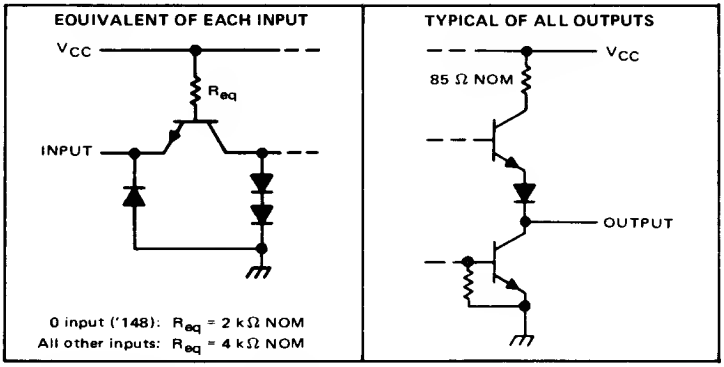
'147, 'LS147



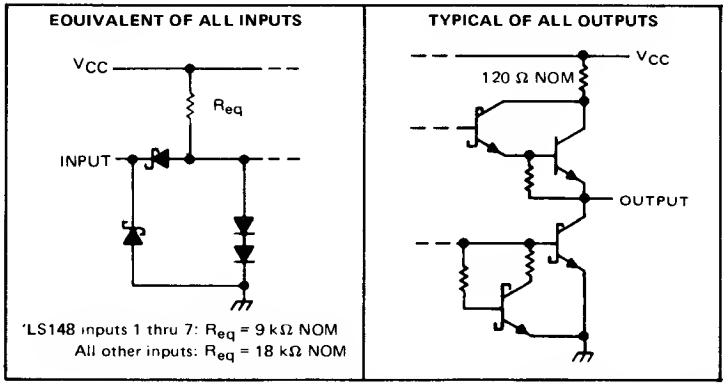
▶8

schematics of inputs and outputs

'147, '148



'LS147, 'LS148



absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage: '147, '148 . . . . . 5.5 V
'LS147, 'LS148 . . . . . 7 V
Interemitter voltage: '148 only (see Note 2) . . . . . 5.5 V
Operating free-air temperature range: SN54', SN54LS Circuits . . . . . −55°C to 125°C
SN74', SN74LS Circuits . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C

NOTES: 1. Voltage values, except intermitter voltage, are with respect to network ground terminal.
2. This is the voltage between two emitters of a multiple-emitter transistor. For '148 circuits, this rating applies between any two of the eight data lines, 0 through 7.

8

recommended operating conditions

	SN54'			SN74'			SN54LS'			SN74LS'			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}	4.5	5	5.5	4.75	5	5.25	4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}			−800			−800			−400			−400	μA
Low-level output current, I_{OL}			16			16			4			8	mA
Operating free-air temperature, T_A	−55		125	0		70	−55		125	0		70	C

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER			TEST CONDITIONS†		'147 MIN	'147 TYP‡	'147 MAX	'148 MIN	'148 TYP‡	'148 MAX	UNIT
V_{IH}	High-level input voltage				2			2			V
V_{IL}	Low-level input voltage						0.8			0.8	V
V_{IK}	Input clamp voltage		V_{CC} = MIN,	I_I = −12 mA			−1.5			−1.5	V
V_{OH}	High-level output voltage		V_{CC} = MIN, V_{IL} = 0.8 V,	V_{IH} = 2 V, I_{OH} = −800 μA	2.4	3.3		2.4	3.3		V
V_{OL}	Low-level output voltage		V_{CC} = MIN, V_{IL} = 0.8 V,	V_{IH} = 2 V, I_{OL} = 16 mA		0.2	0.4		0.2	0.4	V
I_I	Input current at maximum input voltage		V_{CC} = MAX,	V_I = 5.5 V			1			1	mA
I_{IH}	High-level input current	0 input	V_{CC} = MAX,	V_I = 2.4 V						40	μA
		Any input except 0					40			80	
I_{IL}	Low-level input current	0 input	V_{CC} = MAX,	V_I = 0.4 V						−1.6	mA
		Any input except 0					−1.6			−3.2	
I_{OS}	Short-circuit output current§		V_{CC} = MAX		−35		−85	−35		−85	mA
I_{CC}	Supply current		V_{CC} = MAX, See Note 3	Condition 1		50	70		40	60	mA
				Condition 2		42	62		35	55	mA

NOTE 3: For '147, I_{CC} (condition 1) is measured with input 7 grounded, other inputs and outputs open; I_{CC} (condition 2) is measured with all inputs and outputs open. For '148, I_{CC} (condition 1) is measured with inputs 7 and EI grounded, other inputs and outputs open; I_{CC} (condition 2) is measured with all inputs and outputs open.

†For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.

‡All typical values are at V_{CC} = 5 V, T_A = 25°C.

§Not more than one output should be shorted at a time.

SN54147, SN74147 switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER¶	FROM (INPUT)	TO (OUTPUT)	WAVEFORM	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	Any	Any	In-phase output	C_L = 15 pF, R_L = 400 Ω, See Note 4		9	14	ns
t_{PHL}						7	11	
t_{PLH}	Any	Any	Out-of-phase output			13	19	ns
t_{PHL}						12	19	

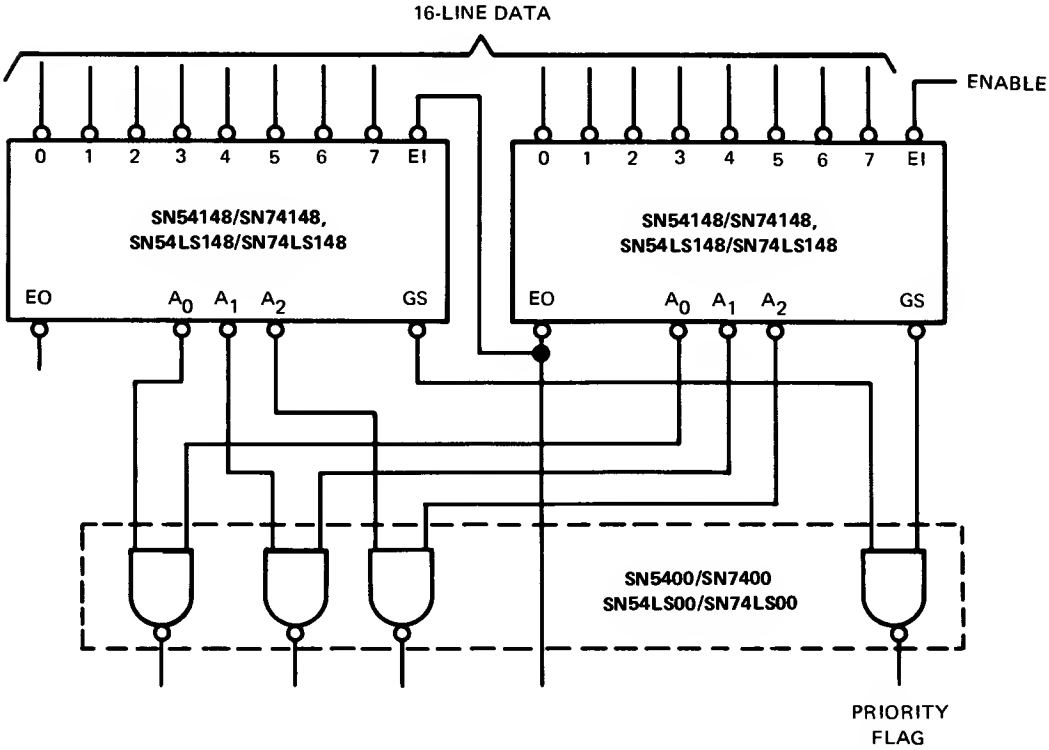
SN54148, SN74148 switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER¶	FROM (INPUT)	TO (OUTPUT)	WAVEFORM	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	0 thru 7	A0, A1, or A2	In-phase output	C_L = 15 pF, R_L = 400 Ω, See Note 4		10	15	ns
t_{PHL}						9	14	
t_{PLH}	0 thru 7	A0, A1, or A2	Out-of-phase output			13	19	ns
t_{PHL}						12	19	
t_{PLH}	0 thru 7	EO	Out-of-phase output			6	10	ns
t_{PHL}						14	25	
t_{PLH}	0 thru 7	GS	In-phase output			18	30	ns
t_{PHL}						14	25	
t_{PLH}	EI	A0, A1, or A2	In-phase output			10	15	ns
t_{PHL}						10	15	
t_{PLH}	EI	GS	In-phase output			8	12	ns
t_{PHL}						10	15	
t_{PLH}	EI	EO	In-phase output			10	15	ns
t_{PHL}						17	30	

¶t_{PLH} = propagation delay time, low to high level output
t_{PHL} = propagation delay time, high to low level output

NOTE 4: Load circuits and waveforms are shown on page 3-10.

TYPICAL APPLICATION DATA

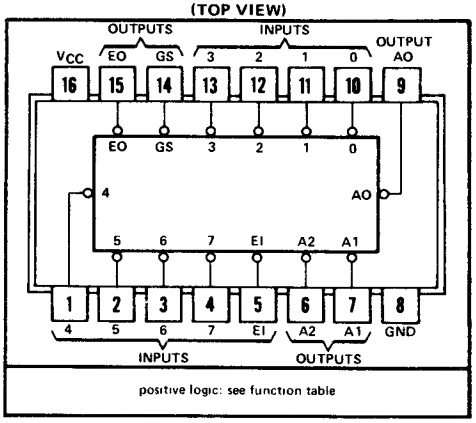


Full 4-bit binary 16-line-to-4-line encoding can be implemented as shown above. The enable input must be low to enable the function. Decoding with 2-input NAND gates produces true (active-high) data for the 4-line binary outputs. If active-low data is required, the SN5408/SN7408 or SN54LS08/SN74LS08 AND gate may be used, respectively.

8

- **3-State Outputs Drive Bus Lines Directly**
- **Encodes 8 Data Lines to 3-Line Binary (Octal)**
- **Applications Include:**
  N-Bit Encoding
  Code Converters and Generators
- **Typical Data Delay . . . 15 ns**
- **Typical Power Dissipation . . . 60 mW**

SN54LS348 . . . J OR W PACKAGE
SN74LS348 . . . J OR N PACKAGE
(TOP VIEW)



description

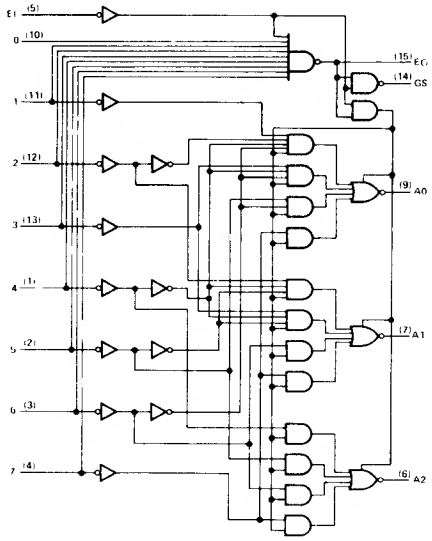
These TTL encoders feature priority decoding of the inputs to ensure that only the highest-order data line is encoded. The 'LS348 circuits encode eight data lines to three-line (4-2-1) binary (octal). Cascading circuitry (enable input EI and enable output EO) has been provided to allow octal expansion. Outputs A0, A1, and A2 are implemented in three-state logic for easy expansion up to 64 lines without the need for external circuitry. See Typical Application Data.

FUNCTION TABLE

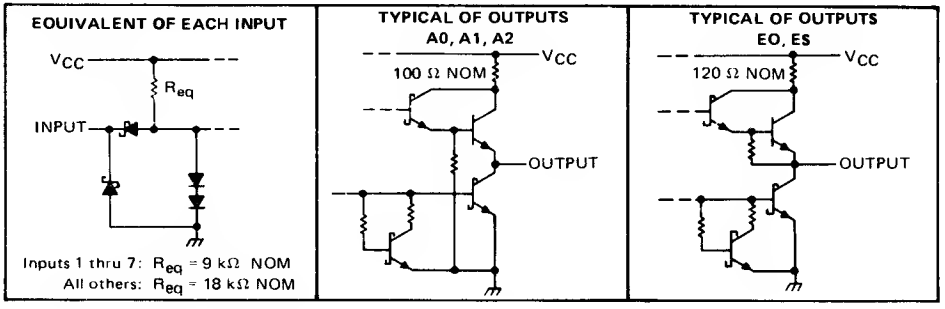
INPUTS									OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	Z	Z	Z	H	H
L	H	H	H	H	H	H	H	H	Z	Z	Z	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

H = high logic level, L = low logic level, X = irrelevant
Z = high-impedance state

functional block diagram



schematic of inputs and outputs



absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage . . . . . 7 V
Operating free-air temperature range: SN54LS348 . . . . . −55°C to 125°C
SN74LS348 . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C

NOTE 1: Voltage values are with respect to network ground terminal.

recommended operating conditions

		SN54LS348			SN74LS348			UNIT
		MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}		4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}	A0, A1, A2			−1			−2.6	mA
	EO, GS			−400			−400	μA
Low-level output current, I_{OL}	A0, A1, A2			12			24	mA
	EO, GS			4			8	mA
Operating free-air temperature, T_A		−55		125	0		70	°C

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER			TEST CONDITIONS†		SN54LS348			SN74LS348			UNIT
					MIN	TYP‡	MAX	MIN	TYP‡	MAX	
V_{IH}	High-level input voltage				2			2			V
V_{IL}	Low-level input voltage						0.7			0.8	V
V_{IK}	Input clamp voltage		V_{CC} = MIN,	I_I = −18 mA			−1.5			−1.5	V
V_{OH}	High-level output voltage	A0, A1, A2	V_{CC} = MIN, V_{IH} = 2 V, V_{IL} = V_{ILmax}	I_{OH} = −1 mA	2.4	3.1					V
				I_{OH} = −2.6 mA				2.4	3.1		
		EO, GS		I_{OH} = −400 μA	2.5	3.4		2.7	3.4		
V_{OL}	Low-level Output voltage	A0, A1, A2	V_{CC} = MIN, V_{IH} = 2 V, V_{IL} = V_{ILmax}	I_{OL} = 12 mA		0.25	0.4		0.25	0.4	V
				I_{OL} = 24 mA					0.35	0.5	
		EO, GS		I_{OL} = 4 mA		0.25	0.4		0.25	0.4	
				I_{OL} = 8 mA					0.35	0.5	
I_{OZ}	Off-State (high-impedance state) output current	A0, A1, A2	V_{CC} = MAX, V_{IH} = 2 V	V_O = 2.7 V			20			20	μA
				V_O = 0.4 V			−20			−20	
I_I	Input current at maximum input voltage	Inputs 1 thru 7	V_{CC} = MAX,	V_I = 7 V			0.2			0.2	mA
		All other inputs					0.1			0.1	
I_{IH}	High-level input current	Inputs 1 thru 7	V_{CC} = MAX,	V_I = 2.7 V			40			40	μA
		All other inputs					20			20	
I_{IL}	Low-level input current	Inputs 1 thru 7	V_{CC} = MAX,	V_I = 0.4 V			−0.8			−0.8	mA
		All other inputs					−0.4			−0.4	
I_{OS}	Short-circuit output current§	Outputs A0, A1, A2	V_{CC} = MAX		−30		−130	−30		−130	mA
		Outputs EO, GS			−20		−100	−20		−100	
I_{CC}	Supply current		V_{CC} = MAX, See Note 2	Condition 1		13	25		13	25	mA
				Condition 2		12	23		12	23	

NOTE 2: I_{CC} (condition 1) is measured with inputs 7 and EI grounded, other inputs and outputs open. I_{CC} (condition 2) is measured with all inputs and outputs open.

†For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions

‡All typical values are at V_{CC} = 5 V, T_A = 25°C.

§Not more than one output should be shorted at a time.

8

switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER¶	FROM (INPUT)	TO (OUTPUT)	WAVEFORM	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	0 thru 7	A0, A1, or A2	In-phase output	C_L = 45 pF, R_L = 667 Ω, See Note 3		11	17	ns
t_{PHL}						20	30	
t_{PLH}	0 thru 7	A0, A1, or A2	Out-of-phase output			23	35	ns
t_{PHL}						23	35	
t_{PZH}	EI	A0, A1, or A2				25	39	ns
t_{PZL}						24	41	
t_{PLH}	0 thru 7	EO	Out-of-phase output	C_L = 15 pF, R_L = 2 kΩ, See Note 3		11	18	ns
t_{PHL}						26	40	
t_{PLH}	0 thru 7	GS	In-phase output			38	55	ns
t_{PHL}						9	21	
t_{PLH}	EI	GS	In-phase output			11	17	ns
t_{PHL}						14	36	
t_{PLH}	EI	EO	In-phase output			17	21	ns
t_{PHL}						25	40	
t_{PHZ}	EI	A0, A1, or A2		C_L = 5 pF, R_L = 667 Ω		18	27	ns
t_{PLZ}						23	35	

¶ t_{PLH} = propagation delay time, low-to-high-level output
t_{PHL} = propagation delay time, high-to-low-level output
t_{PZH} = output enable time to high level
t_{PZL} = output enable time to low level
t_{PHZ} = output disable time from high level
t_{PLZ} = output disable time from low level
NOTE 3: Load circuits and waveforms are shown on page 3-11.

TYPICAL APPLICATION DATA

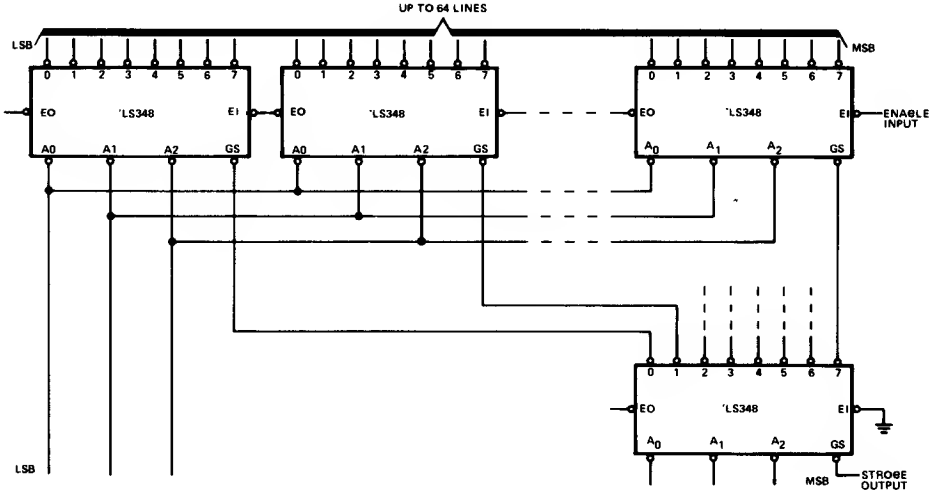


FIGURE 1—PRIORITY ENCODER WITH UP TO 64 INPUTS.

TMS 9909 NL, JL FLOPPY DISK CONTROLLER

990/9900 FAMILY MICROCOMPUTER COMPONENTS

- Supports up to 4 Double-Sided Drives
- Single Density (FM) or
- Double-Density (MFM or M²FM)
- IBM 3740- and 2D-compatible and custom formats
- Programmable Stepper-Motor and Data Transfer Rates
- Write Precompensation
- 5-Inch or 8-Inch Diskettes
- Soft- and Hard-Sector Compatible
- Internal Phase Acquisition and
- Address Mark Detection

DESCRIPTION

The TMS 9909 Floppy Disk Controller (FDC) is designed to provide complete subsystem integration of a floppy diskette mass storage capability. The FDC is a general purpose peripheral device for microprocessor systems and is programmable by the CPU for data encoding formats, number and type of diskette drives, etc. This FDC programmability offers control for the interface between most host systems and virtually any floppy disk drive produced.

The FDC performs the following functions:

- Step to any track on the diskette
- Format tracks (initialize)
- Read and write diskette data
- Send status to host system.

The TMS 9909 Floppy Disk Controller is designed to provide high-level processing features for data transfer using single- and double-density formats. Integration of the FDC system is state-of-the-art, producing maximum performance with minimized hardware complexity, low component count and reduced system cost.

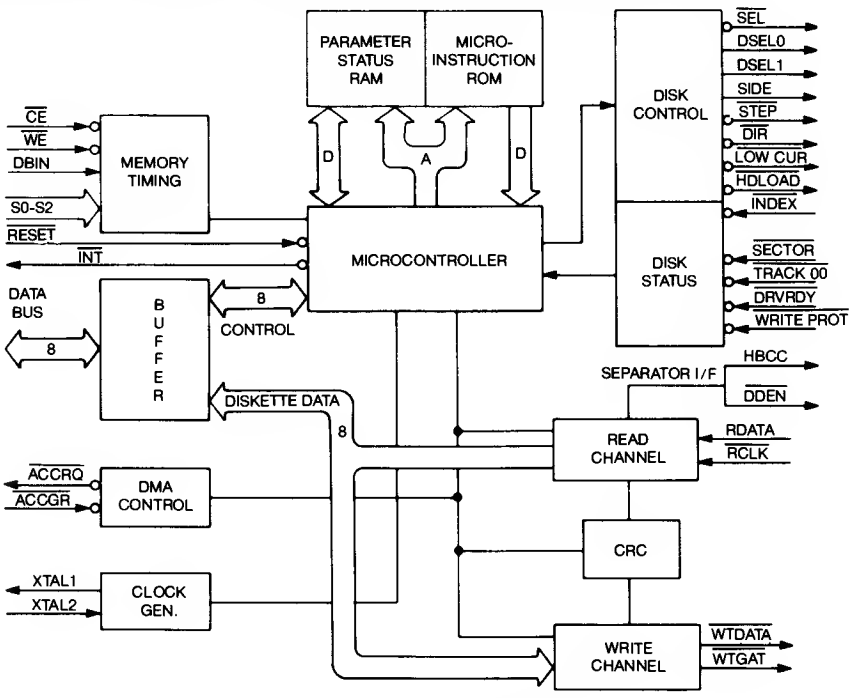


Figure 1. Functional Block Diagram

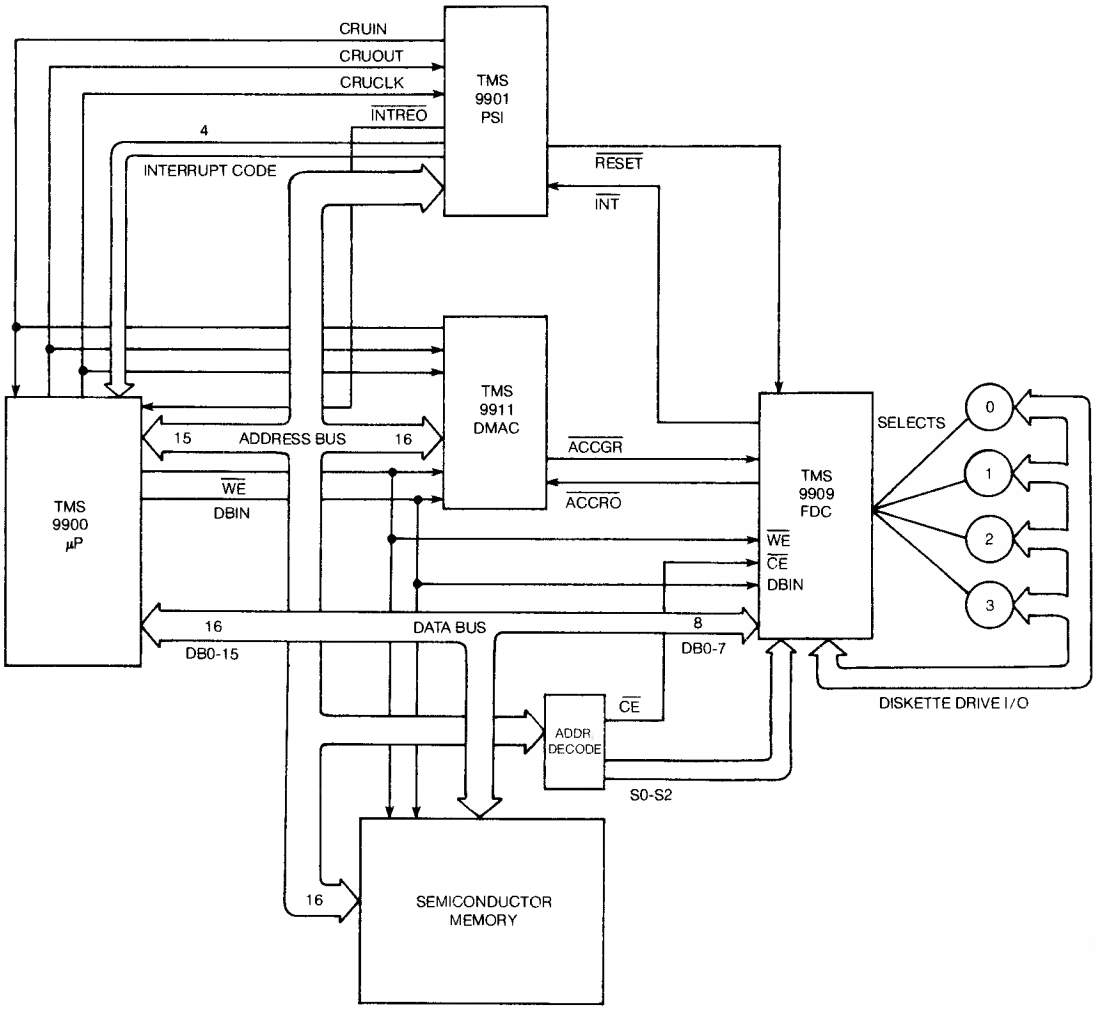
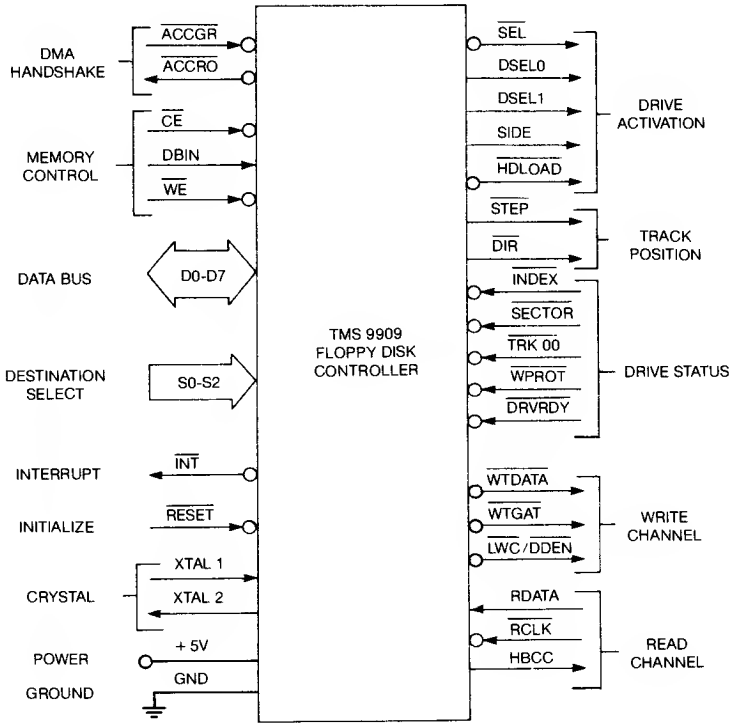


Figure 2—TMS 9900 Microcomputer System incorporating the TMS 9909 Floppy Disk Controller

PIN DESCRIPTION



TMS 9909 PIN FUNCTIONS

Signature	*I/O*	*Description*
ACCRQ	O	To activate the DMA channel, the FDC asserts ACCESS REQUEST.
ACCGR	I	The DMA channel responds with ACCESS GRANT when transfer is to begin.
CE	I	CHIP ENABLE serves to enable command and parameter input and status output to the host system.
DBIN	I	Data Bus In specifies the direction of data flow between the host system and FDC.
WE	I	WRITE ENABLE pulses provide a window for data input to the FDC.
Data Bus DB0-7	I/O	The data bus is used to transfer information between the host system's data bus and the FDC's command, parameter and status registers.
S0-S2	I	To access data the host system must select a desination register using S0-S2.
INT	O	Upon completion of an operation or detection of an error, the FDC issues an interrupt to the host.
RESET	I	When the system powers up, a pulse on the reset line puts the FDC in its initialized state.
XTAL1	O	A 6-MHz crystal is connected between XTAL1 and XTAL2 to generate timing for the TMS 9909.
XTAL2	I	Alternately, a 6-MHz reference can be connected to XTAL2 with the XTAL1 pin unconnected.
+5V	—	FDC power supply.
GND	—	System ground connection.

8◀

Signature	*I/O*	*Description*
$\overline{\text{LWC}}/\overline{\text{DDEN}}$	O	LOW WRITE CURRENT is active when the track written is greater than 43, (programmable) for drives which reduce current on the inner tracks.
		DOUBLE DENSITY is output to the data separator during read operations.
$\overline{\text{STEP}}$	O	STEP pulses are issued by the FDC to move the selected drive's read/write head in the specified
$\overline{\text{DIRECTION}}$	O	DIRECTION, which is a level to define STEP-IN or STEP-OUT.
$\overline{\text{SEL}}$	O	SELECT is activated when DSEL0 and 1 contain an active drive address.
DSEL0	O	DRIVE SELECT 0 and 1 are encoded lines for up to four system drives to activate their control
DSEL1	O	and I/O lines.
SIDE	O	SIDE select determines the side of a dual-sided diskette used.
		Drive Status Lines Go Active On the Following Conditions:
$\overline{\text{INDEX}}$	I	The INDEX hole in the diskette is aligned with the diskette's index hole sensor.
$\overline{\text{SECTOR}}$	I	For hard-sectored diskettes, the SECTOR hole in the diskette is aligned with the diskette drive's sector hole sensor.
$\overline{\text{TRK00}}$	I	Selected drive's head is at its maximum radius (TRACK ZERO).
$\overline{\text{WPROT}}$	I	WRITE PROTECT, when active (low) indicates a read-only diskette has been selected.
$\overline{\text{DRVRDY}}$	I	DRIVE READY, indicates drive is powered up, door is closed, and diskette is properly installed.
RDATA	I	Data from the disk comes into the FDC via the READ DATA line.
$\overline{\text{RCLK}}$	I	READ CLOCK is a pulse synchronized to diskette clock and data half bit-cells.
HBCC	O	The HALF BIT-CELL CLOCK at two times the transfer rate, is used to start the data separator near data synchronization. Also, HBBC can be used to change write precompensation timing.
$\overline{\text{WTDATA}}$	O	The data to be written on the floppy disk is transmitted on the WRITE DATA line.
$\overline{\text{WTGAT}}$	O	WRITE GATE is true during data output to the diskette drive.

PROGRAM DESCRIPTION

The TMS 9909 Floppy Disk Controller is designed for ease of use. To execute a given command, the user writes a command code and parameter list to the FDC's eight-bit data port. When the last parameter has been transferred, the FDC begins command execution as an independent processor.

Setup for FDC commands proceeds as follows. The TMS 9909 is allocated eight memory addresses by the host system, all of which decode into a common chip enable. Selection of the base address with no offset (i.e., S0, S1, S2 = 000) gives the host write-only access to the COMMAND REGISTER and read-only access to the STATUS REGISTER. To initiate a command, the host checks the FDC's status for an idle condition, then writes a COMMAND CODE to the COMMAND REGISTER. Next a list of parameters is written into FDC RAM at
▸8 S0-S2 address >0. After transfer of parameters is complete, command execution begins.

Track accessing is very flexible with the TMS 9909. Programmable stepper-motor control rates are loaded into the FDC during the RECALIBRATE and ASSIGN RATES command. RECALIBRATE moves the heads of selected drives to track 00. Read, write and write format commands include a parameter indicating the new physical track to seek.

During READ and WRITE sequences, the FDC transfers data between the selected diskette drive and system memory. Memory addressing and timing for data transfer are supervised by a Direct Memory Access Controller like the TMS 9911. The number of sectors of diskette data transferred is programmable from one up to an entire track. DMA handshake is activated for each byte of READ or WRITE data, every 16μs at a 500 k bits per second transfer rate.

Formatting diskettes is another capability of the TMS 9909. After the host CPU writes the command FORMAT TRACK and its associated parameters, the FDC activates its DMA handshake to acquire the contents and length of each field of the user's format. Since the host system provides the length and contents of each format field, specifying an address mark and a fill byte or all sector contents, the user has unlimited flexibility in selecting a format. Formatting IBM-compatible single- and double-density diskettes is straightforward, as is customizing formats to provide optimized interleaving for real-time applications.

An eight-bit status register can be read at the address of the command input register at any time by the CPU. The primary status register provides operating status including:

- Command execution in progress
- Diskette drive not ready
- Selected drive at track 00
- CRC error
- Data underflow/overflow
- ID not found, etc.

After the command in progress has completed its task, the contents of the FDC's internal RAM may be read (in the same way as parameters were written earlier) to provide detailed status information.

Completion of the command in progress, available to host system software in the status register, is signalled in hardware by the INTERRUPT pin of the TMS 9909. Any FDC command can be aborted by the host CPU by hardware activation of the RESET pin, or in software by writing an ABORT command to the command register.

The command macros of the TMS 9909 floppy disk controller are sufficiently powerful to fulfill the needs of most microcomputer systems on a stand-alone basis. For large micro/minicomputer systems, the modular commands can be combined for preprocessing by a dedicated microcomputer, such as the TMS 9985, to provide a sophisticated file management subsystem.

TMS 9909 FLOPPY DISK CONTROLLER COMMANDS

Type	*Command*	*Function*
Controller State Commands	Reset Controller	Initialize FDC internal state and output pins
	Abort Execution	Terminate active command. (At sector end for writes)
	Clear Interrupt	Deactivate $\overline{\text{INT}}$ pin, acknowledging command completion
Format Initialization Commands	Assign ID Attributes	Define address marks and contents of ID fields.
	Assign Fill and Sync	Define fill byte (one is written after end of sector) and sync byte (for phase acquisition) for current format.
Drive Control Commands	Recalibrate Drives, Assign Rates	Define stepper rates and restore selected drives to track zero.
	Seek, Check and Read Data	Seek physical track, locate desired ID (unless unformatted, i.e., Read ID) and transfer sectors (hard or soft) of diskette data to host system.
	Seek, Check and Write Data	Seek physical track and write sectors of data on a formatted diskette, hard or soft sector. Low write current and/or precompensation are selectable.
	Seek and Write Format	Seek physical track and write format fields on a whole track or single hard sector.

TMS9911 JL, NL
DIRECT MEMORY ACCESS CONTROLLER

- Generation of All Memory Control Signals
- Supports 2 Independent DMA Devices
- Cascadeable to Multiple DMA Channels
- Memory Address and Limit Registers for Each Channel
- Automatic Interrupt Generation
- Operates Under Program Control of the Microprocessor Unit.

DESCRIPTION

The TMS9911 DMAC is an LSI member of the 9900 family of microprocessors and support peripherals. The DMAC is used in 9900 microprocessor systems where devices other than a single CPU require direct access to memory. The DMAC generates memory control signals and sequential memory addresses for two DMA channels (i.e., two independent DMA devices), allowing these devices to access memory autonomously with respect to the CPU. Multiple DMAC's may be used to extend the number of DMA channels beyond two. The interfaces of the DMAC to the CPU, system memory, and DMA peripheral devices are defined in such a manner as to require a minimum amount of additional electronics.

By using the CRU for set up and status of the TMS9911, a DMA controller has been configured in a 40-pin package with no data bus at all. The TMS9911 provides an easy-to-use cost effective method for implementing Direct Memory Access for 9900-family peripherals.

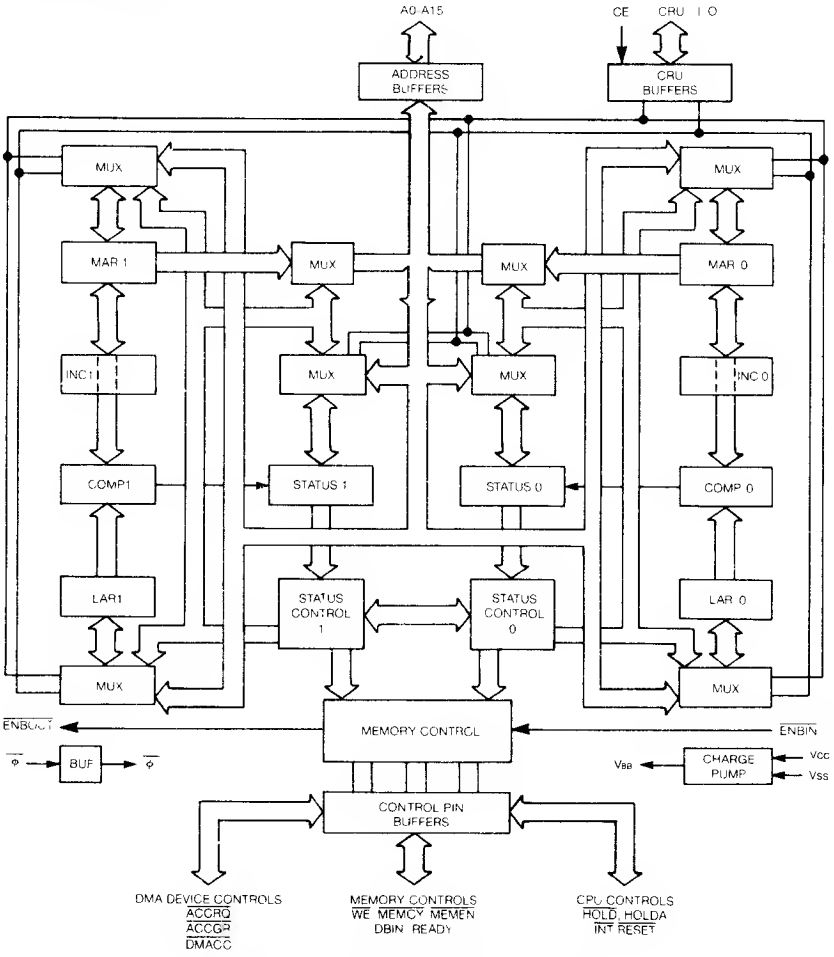


Figure 1. TMS 9911 Functional Block Diagram

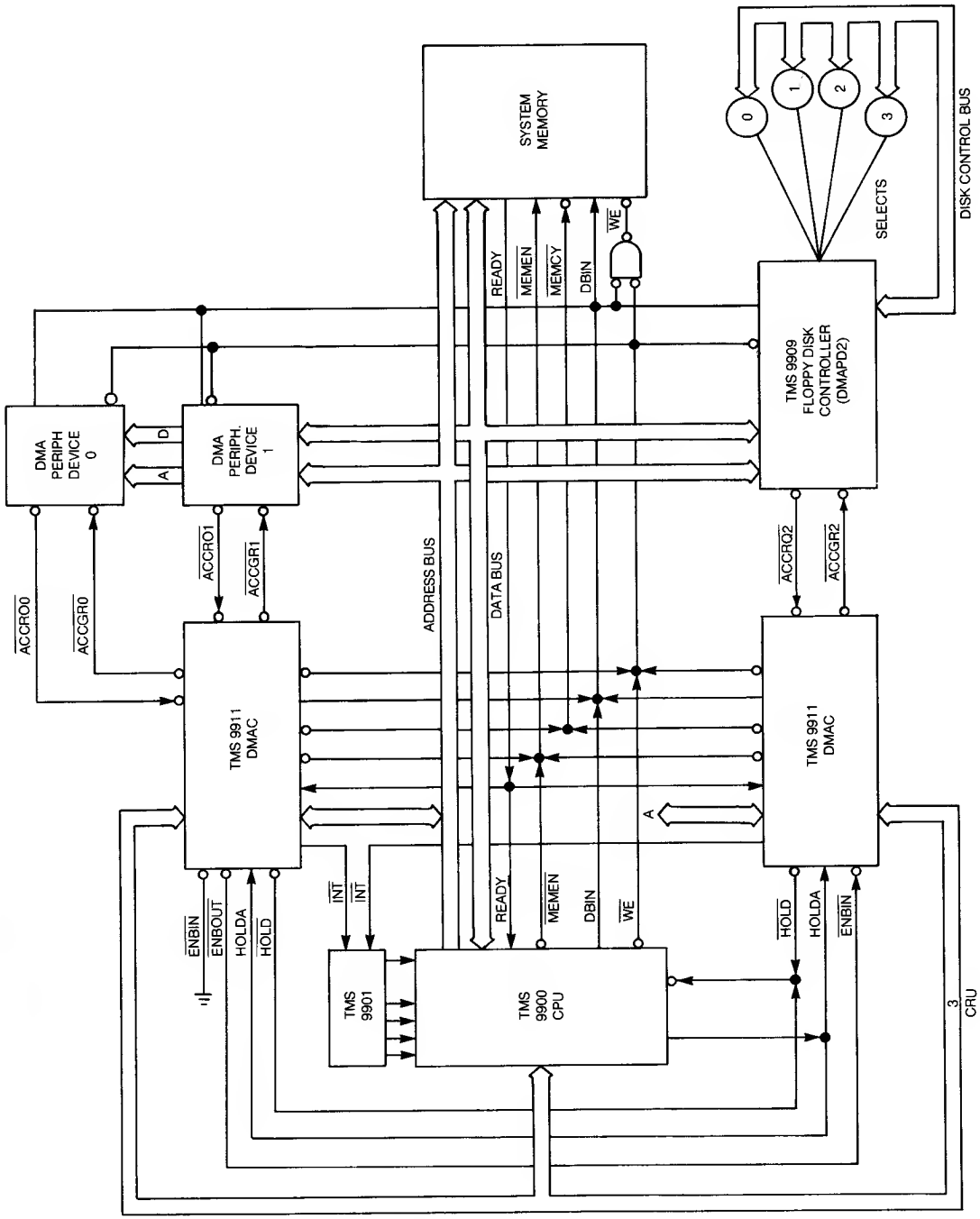
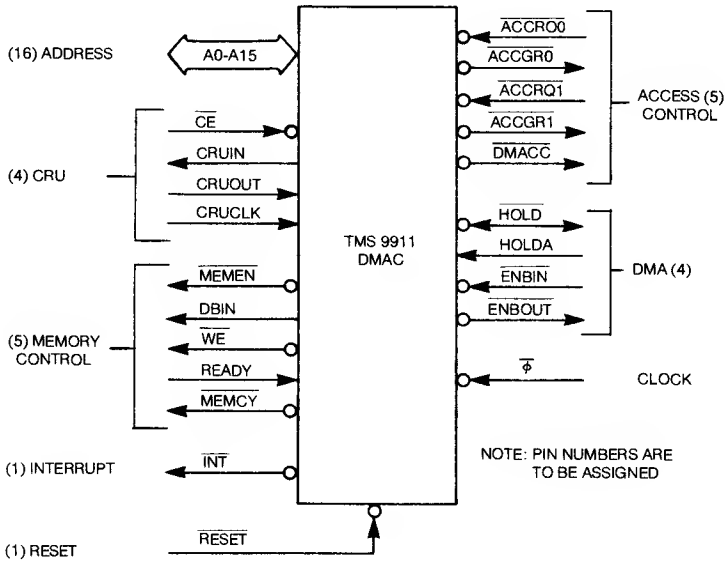


Figure 2. TMS 9900/TMS 9911 System Bus Interconnect



TMS 9911 DMAC Pin Description

TMS 9911 PIN FUNCTIONS

Signature	*I/O*	*Description*
		ADDRESS BUS
A0(MSB)	OUT	A0 through A15 comprise the Address Bus. The address bus outputs the memory address to or from which data is to be transferred while the DMAC accesses memory. A0-A15 outputs are at high-impedance while the DMAC is not accessing memory. A10-A14 are inputs to the DMAC, selecting the address of the bit to or from which the CPU is transferring data via the CRU. Although A15 is not normally implemented in TMS 9900 systems, this line may be used to select which half of the 16-bit data word is to be loaded when the DMA device is transferring a single byte.
A1	OUT	
A2	OUT	
A3	OUT	
A4		
A5	OUT	
A6	OUT	
A7	OUT	
A8	OUT	
A9	OUT	
A10	I/O	
A11	I/O	
A12	I/O	
A13	I/O	
A14	I/O	
A15(LSB)	OUT	
		CRU INTERFACE
$\overline{CE}$	IN	Chip Enable. $\overline{CE}$ is low when the CPU is transferring data to or from the DMAC via the CRU. The other CRU lines are ignored when $\overline{CE}$ is high. $\overline{CE}$ is normally generated by decoding a particular range of CRU addresses from the high order address lines A0-A9, and should not be low when memory accesses are being performed.

Signature	*I/O*	*Description*
CRUIN	OUT	CRU Input Data (to the CPU from the DMAC). CRUIN is at high impedance when $\overline{CE}$ is high. When $\overline{CE}$ is low, CRUIN contains the value of the bit addressed by A10-A14 to be read by the CPU.
CRUOUT	IN	CRU Output Data (from the CPU to the DMAC). CRUOUT contains the value of the datum to be transferred to the DMAC by the CPU during CRU operations.
CRUCLK	IN	CRU Output Data Clock. CRUCLK strobes the datum contained on CRUOUT to the bit addressed by A10-A14 when CE is low.
		MEMORY CONTROL
$\overline{MEMEN}$	OUT	Memory Enable. $\overline{MEMEN}$ is at high impedance except when the DMAC is accessing memory. When $\overline{MEMEN}$ is low, a memory cycle is in progress and no other device may access memory until the cycle is completed.
DBIN	OUT	Memory Data Read Enable. DBIN is at high-impedance except when the DMAC is accessing memory. During DMAC memory cycles DBIN indicates the direction of memory data transfer; i.e. DBIN = 1 for memory read and = 0 for memory write operations.
$\overline{WE}$	OUT	Write Enable. $\overline{WE}$ is at high impedance except when the DMAC is accessing memory. The timing for $\overline{WE}$ is identical to that of the $\overline{WE}$ output of the 9900 CPU's. $\overline{WE}$ performs the function of strobing data from a DMA device into memory during DMA.
READY	IN	Memory Transfer Ready. READY is sampled at the end of each clock cycle during each DMAC memory cycle. If READY = 0, the memory cycle is extended an additional clock cycle and READY is sampled again until READY = 1, at which time the memory cycle continues to completion.
$\overline{MEMCY}$	I/O	Memory Cycle. $\overline{MEMCY}$ is at high impedance except when the DMAC is accessing memory. $\overline{MEMCY}$ is low during all but the last clock cycle of each DMAC memory cycle. As an input, $\overline{MEMCY}$ is used to avoid bus conflicts during DMAC-to-DMAC control transfer.
		DMA DEVICE HANDSHAKE
$\overline{ACCRQ0}$	IN	DMA Device — Access Request. $\overline{ACCRQ0}$ is asserted by the DMA device connected to channel 0 when it wishes to access memory.
$\overline{ACCGR0}$	OUT	DMA Device 0 Access Granted. $\overline{ACCGR0}$ is active while the DMAC is performing a data transfer between memory and the DMA device connected to Channel 0.
$\overline{ACCRQ1}$	IN	DMA Device 1 Access Request. $\overline{ACCRQ1}$ provides the identical function for DMA Device 1 as does $\overline{ACCRQ0}$ for DMA Device 0.
$\overline{ACCGR1}$	OUT	DMA Device 1 Access Granted. ACCGR1 provides the identical function for DMA Device 1 as does ACCGR0 for DMA Device 0.
$\overline{DMACC}$	OUT	DMA accessing memory is active when the system buses are under DMA control. $\overline{DMACC}$ can be used to control the drive direction of bidirectional buffers (when required) such that A10-A14, $\overline{MEMCY}$ and $\overline{HOLD}$ are outputs when $\overline{DMACC}$ is active.

Signature	*I/O*	*Description*
		INTERRUPT OUTPUT
$\overline{INT}$	OUT	Interrupt Output. $\overline{INT}$ is low when either channel has transferred the specified number of bytes of data and the interrupt for that channel is enabled.
		DMA CONTROL
$\overline{HOLD}$	I/O	Hold Request. $\overline{HOLD}$ is an open-drain output which may be tied to other DMACs as an input to the CPU. $\overline{HOLD}$ becomes active when one of the Access Request signals is active and the channel corresponding to the access request is enabled. As an input, $\overline{HOLD}$ is used to set the internal $\overline{HOLD}$ condition by the equation: $\overline{HOLD}$ J(INT) = ACCRQn* (HOLD + $\overline{HOLDA}$).
HOLDA	IN	Hold Acknowledge. HOLDA is asserted by the CPU to indicate that it is entering the Hold state to allow a DMAC to access memory. It becomes active in response to the $\overline{HOLD}$ signal.
$\overline{ENBIN}$	IN	Enable Input. $\overline{ENBIN}$ must be low in order for the DMAC to access memory. When $\overline{ENBIN}$ = 1, a higher priority device is contending for access, thus inhibiting the DMAC from granting access.
$\overline{ENBOUT}$	OUT	Enable Output. $\overline{ENBOUT}$ is active only when $\overline{ENBIN}$ = 0 and neither channel is enabled. and requesting access. $\overline{ENBOUT}$ is connected to the $\overline{ENBIN}$ input of the DMAC of next lower priority.
		CLOCK SIGNALS
$\overline{\phi}$	IN	Clock Input. When the DMAC is used with a TMS 9900 Microprocessor this signal is provided by the $\overline{\phi 1}$ output of the TIM 9904 Clock Generator. In a 9980 system this signal is connected to the $\overline{CKOUT}$ output of the TMS 9980, and $\overline{HOLD}$ should be sychronized to avoid changing during $\overline{\phi 1}$.
		DEVICE RESET
$\overline{RESET}$	IN	Device Reset. When $\overline{RESET}$ is active, the DMAC is reset to a known state where both DMA channels are disabled. (MAR = LAR = 0, All status bits inactive, CHSEL = CnASEL = 0).
VCC	IN	5 Volts DC ± 5%
GND	IN	0 Volt reference. Pin 21 provides ground for the TMS 9911 logic. Pin 32 is the ground for signal buffers.

8

PROGRAM DESCRIPTION

Each of the 2 channels of the DMAC has two 16-bit registers: a Memory Address Register (MAR), and a Last Address Register (LAR). The Memory Address Register contains the memory address which the next memory cycle by that channel will access. After each memory access is completed, the Memory Address Register is automatically updated to the address for the next memory access. As the Memory Address Register is incremented, it is compared to the value contained in the Last Address Register. If the comparison is true, a status bit and (if enabled) an interrupt to the CPU are activated, indicating that the desired block of data has been transferred.

Each access requires that the DMAC gain control of the System Memory Bus. When the DMAC has control of the bus, no other DMAC or the CPU may perform a memory operation until the DMAC completes its memory cycle. Each memory access is performed by generating the necessary address and control signals to transfer a byte or word of data between system memory and the DMA peripheral device connected to the active channel.

The sequence of operations for using the TMS9911 DMAC is as follows: The host CPU sets up the control registers of the DMAC through the system's serial communication channel, the CRU.

At system power-up, the host must put the DMAC into an initialized state. This can be done with hardware by activating the DMAC's RESET pin, or through software addressing the CRU output bit SWRST (software reset).

The channel to be used for the current DMA transfer is selected by setting the CHSEL (channel select) CRU bit. Two independent channels are implemented on each TMS9911 chip with an automatic priority of CH0>CH1. When more than two DMA channels are required in a system, multiple TMS9911 circuits can be used.with priority established using the $\overline{\text{ENBIN}}$/$\overline{\text{ENBOUT}}$ chain, as shown in the system diagram.

The DMAC can control the transfer of sixteen-bit words or eight-bit bytes. The mode of operation is selected using the WRDSLn CRU bit. When byte mode is chosen, address bit A15 changes with each transfer. Memory systems which allow byte transfers must implement A15.

Having established the operational mode of this channel, the host CPU sets up the contents of the Memory Address Register and Last Address Register. Access to the MAR or LAR is gained by setting the sense of the CnASEL CRU bit. Thus the user executes a Set Bit One to CnASEL and LDCR for the sixteen bits of the MAR. Execution of another sixteen-bit LDCR sets up the contents of the LAR and the DMAC channel is configured to run.

If an interrupt should be issued by the DMAC when its operation is complete the IENB CRU bit for this channel must be set. Interrupts from TMS9911's are processed by the TMS9901 Programmable Systems Interface as shown in the system diagram. Alternately, interrupts of DMAC's can be wire-ORed and the CPU can poll DMAC channels to locate the active OPCOMP CRU bit.

The DMAC channel cannot begin to function until a final CRU bit, CHENBn, is set. Operation of the DMA channel is then under direct control of the $\overline{\text{ACCRQ}}$/$\overline{\text{ACCGR}}$ handshake.

Each $\overline{\text{ACCRQ}}$ by the DMA peripheral device causes the DMAC to gain memory bus control through the $\overline{\text{HOLD}}$/HOLDA handshake with the host CPU. $\overline{\text{ACCRQ}}$ may be held active continually by the DMA device, but is normally released after each transfer or block of transfers.

Termination of DMA channel activity can occur several ways. Though the host CPU can reset the DMAC or clear its CHENBn CRU bit, normal DMA termination occurs when the contents of the Memory Address Register equals that of the Last Address Register. Transfer does not occur to the LAR address. LAR is computed for the transfer of n bytes as follows:

$$LAR = MAR + n$$

Hence if MAR = 4 and 6 bytes are to be transferred, LAR = 10 and the last transfer is to address 9.

Status of the TMS9911 can be read through the CRU. CRU status input bits inform the host of the state of CRU output bits and provide a method for checking operation completeness by software.

TMS 9914 GENERAL PURPOSE INTERFACE BUS ADAPTER

990/9900 FAMILY MICROCOMPUTER COMPONENTS

- IEEE Std. 488-1975 Compatible
- Source and Acceptor Handshake
- Complete Talker and Listener Functions with Extended Addressing
- Controller and System Controller Capability
- Service Request
- Remote and Local with Lockout
- Serial and Parallel Polling
- Device Clear
- Device Trigger
- Compatible with TMS 9911 DMA Controller
- Single +5 V Power Supply
- Interfaces directly to SN75160/1/2 Transceivers

DESCRIPTION

The TMS 9914 General Purpose Interface Bus Adapter is a microprocessor controlled versatile device which enables the designer to implement all of the functions or a subset described in the IEEE Std. 488-1975. Using this standard, a variety of instruments can be interconnected and remotely or automatically programmed and controlled. The TMS 9914 is fabricated with N-channel silicon-gate technology and is completely TTL compatible on all inputs and outputs including the power supply (+5 V). It needs a single phase clock (nominally 5 MHz) which may be independent of the microprocessor system clock and, therefore, it can easily be interfaced with most microprocessors. The general purpose interface bus adapter (GPIBA) performs the majority of the functions contained in IEEE STd. 488-1975 and is versatile enough to allow software implementation of those sections not directly implemented in hardware.

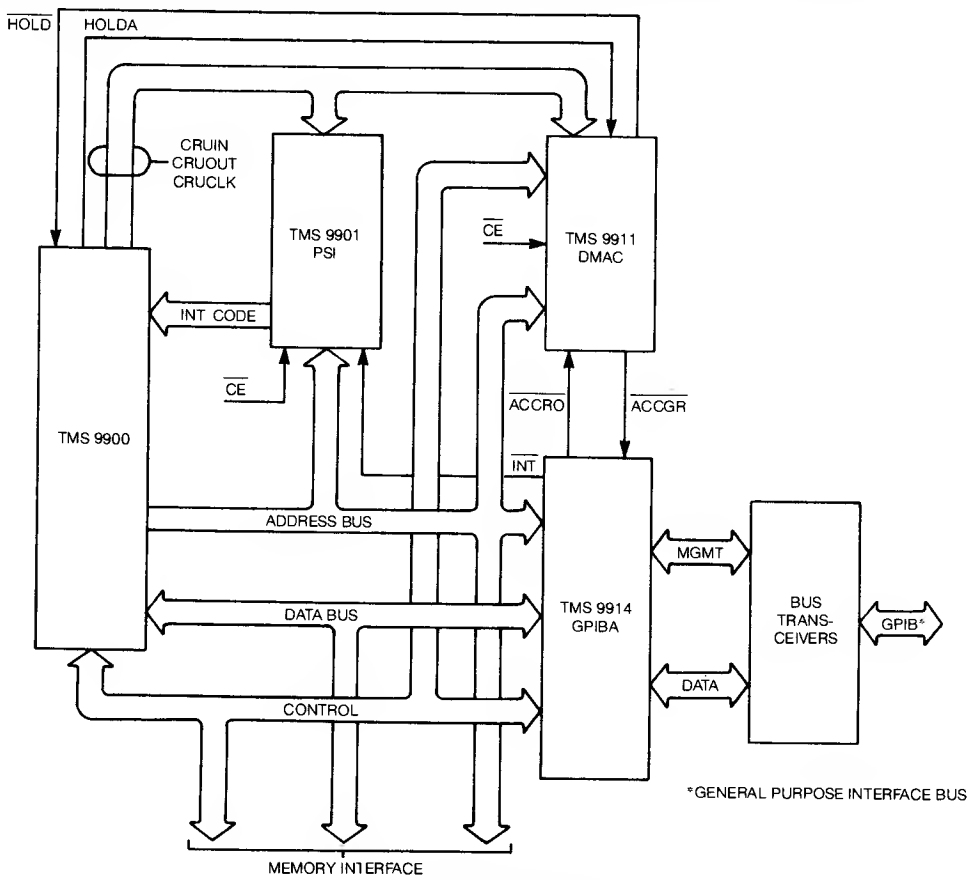


Figure 1. Typical System Interconnect

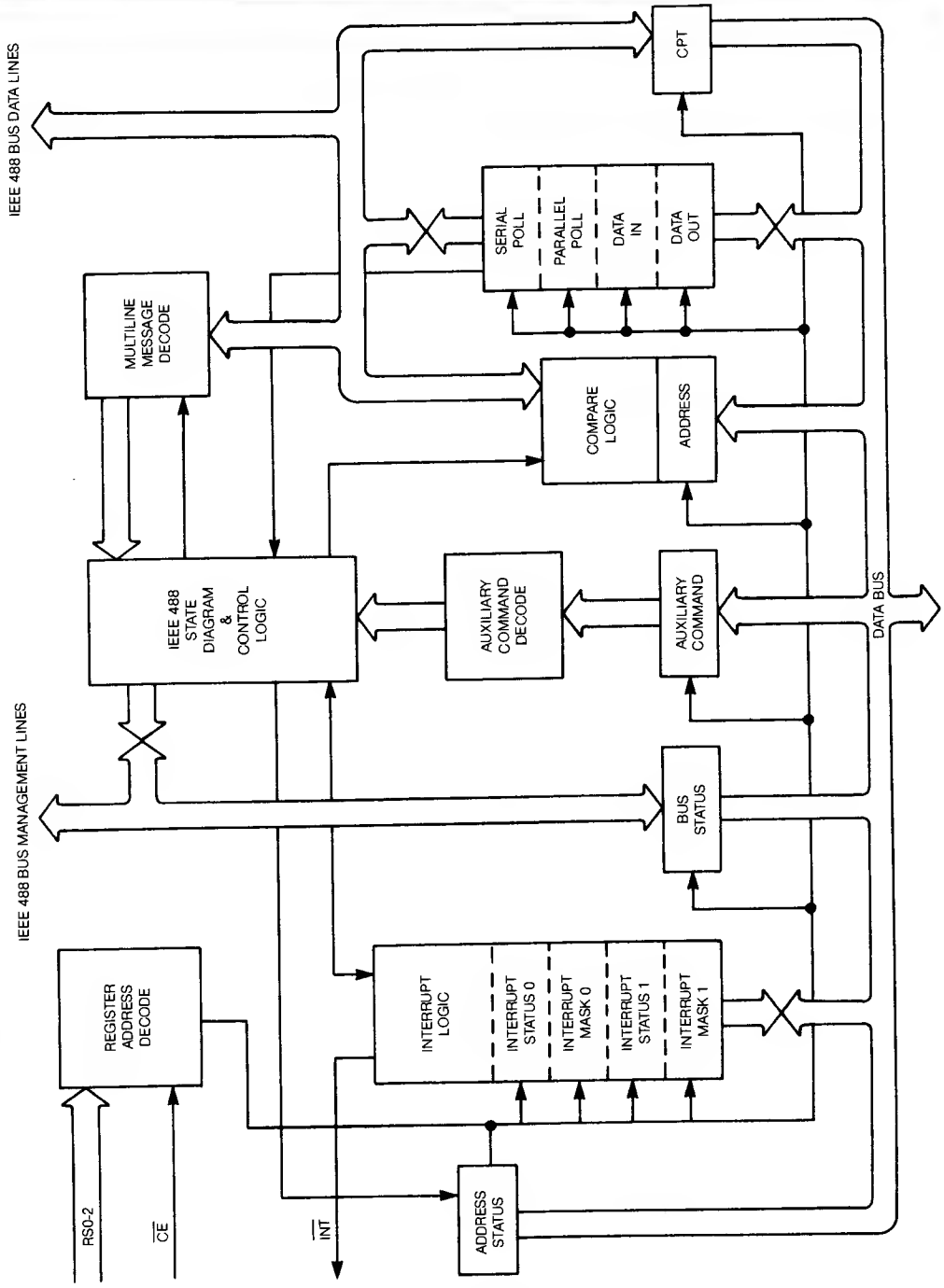


Figure 2. TMS 9914 Simplified Block Diagram

Table 1. Pin Description

Name	*I/O*	*Description*
DI01 through DI08	I/O	DATA I/O lines: allow data transfer between the TMS 9914 and the IEEE 488 data bus.
DAV	I/O	DATA VALID: Handshake Line. Sent by source device to indicate to acceptors that there is valid data on the IEEE bus data lines.
NRFD	I/O	NOT READY FOR DATA: Handshake Line. Sent by the acceptor to the source device to indicate when it is ready for a new byte of data.
NDAC	I/O	DATA NOT ACCEPTED: Handshake Line. Sent by acceptor to source device to indicate when it has accepted the current byte on the data bus.
ATN	I/O	ATTENTION: Management Line. Sent by the controller. When ATN is asserted, the information on the data lines is interpreted as commands, sent by the controller . . . When ATN is false, the data lines carry data.
IFC	I/O	INTERFACE CLEAR. Management Line. Sent by system controller to set the interface system, portions of which are contained in all interconnected devices in a known quiescent state. System controller assumes control. Open drain output with internal pullup.
REN	I/O	REMOTE ENABLE: Management Line. Sent by system controller and is used in conjunction with other messages to select between two alternate sources of programming data, e.g. via interface or front panel. Open drain output with internal pullup.
SRQ	I/O	SERVICE REQUEST: Management Line. Issued by a device on the bus to the controller to indicate a need for service.
EOI	I/O	END OR IDENTIFY: Management Line. If ATN is false, this signal is sent by the "talker" to indicate the end of a multiple byte transfer. If sent by the controller with ATN true, this will perform the parallel polling sequence.
$\overline{\text{CONTROLLER}}$	O	Bus transceiver control line. Indicates that the device is the controller.
TE	O	TALK ENABLE: Bus transceiver control line. Indicates the direction of data transfer on the data bus.
D0 through D7	I/O	Data I/O lines that allow transfer of data between TMS 9914 and the microprocessor.
RS0 through RS2	I	Address lines through which the TMS 9914 registers can be accessed by the microprocessor.
DBIN	I	When true (high) DBIN indicates to the TMS 9914 that the microprocessor is about to read from one of its registers. When false, that the microprocessor is about to write to one of its registers.
$\overline{\text{WE}}$	I	$\overline{\text{WRITE ENABLE}}$: indicates to the TMS 9914 that one of its registers is being written to.
$\overline{\text{CE}}$	I	$\overline{\text{CHIP ENABLE}}$: selects and enables the TMS 9914 for an microprocessor data transfer.
$\overline{\text{INT}}$	O	$\overline{\text{INT}}$: Open drain output. Sent to microprocessor to indicate the occurrence of an event on the bus requiring service.
$\overline{\text{ACCRQ}}$	O	$\overline{\text{ACCESS REQUEST}}$: Signal to TMS 9911 DMA controller requesting DMA.

PIN OUTS TO BE ASSIGNED

NOTE: The names of the IEEE bus lines have been maintained, and are therefore negative logic signals.

FUNCTIONAL DESCRIPTION

The TMS 9914 interfaces to the CPU with an eight-bit bidirectional data bus, three register select lines, two DMA control lines, reset and interrupt request lines, a DBIN and a $\overline{WE}$ line.

The internal architecture of the TMS 9914 is arranged into 13 registers, there being seven WRITE and six READ registers. Some are actually address ports through which current status can be obtained. Table 2 lists these registers and their addresses. The microporcessor accesses a TMS 9914 register by supplying the correct register address in conjunction with $\overline{WE}$ and DBIN. The $\overline{CE}$ is used to enable the address decode.

Table 2. TMS 9914 Registers and Addresses

NAME	TYPE	RS2	RS1	RS0	DBIN	$\overline{WE}$
INTERRUPT STATUS 0	R	0	0	0	1	1
INTERRUPT MASK 0	W	0	0	0	0	0
INTERRUPT STATUS 1	R	0	0	1	1	1
INTERRUPT MASK 1	W	0	0	1	0	0
ADDRESS STATUS	R	0	1	0	1	1
BUS STATUS	R	0	1	1	1	1
AUXILIARY COMMAND	W	0	1	1	0	0
ADDRESS SWITCH	R	1	0	0	1	1
ADDRESS	W	1	0	0	0	0
SERIAL POLL	W	1	0	1	0	0
COMMAND PASS THROUGH	R	1	1	0	1	1
PARALLEL POLL	W	1	1	0	0	0
DATA IN	R	1	1	1	1	1
DATA OUT	W	1	1	1	0	0

NOTE: The Address Switch register is external to the TMS 9914

In DMA operation the TMS 9911 supplies the memory address but not the peripheral device address (i.e., RS0-2, $\overline{CE}$) are not supplied). When the TMS 9914 sets $\overline{ACCRQ}$ low true, it is either because of a byte input or a byte output, and this will happen whether or not DMA transfer will take place. If in response to $\overline{ACCRQ}$ an $\overline{ACCGR}$ (access granted) is received, the $\overline{ACCRQ}$ will be reset and a DMA transfer will take place between the system memory and either the Data In or Data Out register. If the data transfer is with the microprocessor and if the microprocessor addresses either the Data In or Data Out register, the $\overline{ACCRQ}$ line will be reset. Note that in DMA mode the sense of DBIN is inverted.

Table 3 lists the commands which are directly handled by the TMS 9914, and those which require intervention by the microprocessor for their implementation.

Table 3. Remote Multiple Message Coding

		DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1		Note
Addressed Command Group	ACG	X	0	0	0	X	X	X	X	AC	
Device Clear	DCL	X	0	0	1	0	1	0	0	UC	
Group Execute Trigger	GET	X	0	0	0	1	0	0	0	AC	
Go To Local	GTL	X	0	0	0	0	0	0	1	AC	
Listen Address Group	LAG	X	0	1	X	X	X	X	X	AD	
Local Lock Out	LLO	X	0	0	1	0	0	0	1	UC	
My Listen Address	MLA	X	0	1	L	L	L	L	L	AD	1
My Talk Address	MTA	X	1	0	T	T	T	T	T	AD	2
My Secondary Address	MSA	X	1	1	S	S	S	S	S	SE	3, 4
Other Secondary Address	OSA									SE	4, 5
Other Talk Address	OTA				TAG • MTA					AD	
Primary Command Group	PCG									—	6
Parallel Poll Configure	PPC	X	0	0	0	0	1	0	1	AC	7
Parallel Poll Enable	PPE	X	1	1	0	S	P	P	P	SE	8, 9
Parallel Poll Disable	PPD	X	1	1	1	D	D	D	D	SE	8, 10
Parallel Poll Unconfigure	PPU	X	0	0	1	0	1	0	1	UC	11
Secondary Command Group	SCG	X	1	1	X	X	X	X	X	SE	
Selected Device Clear	SDC	X	0	0	0	0	1	0	0	AC	
Serial Poll Disable	SPD	X	0	0	1	1	0	0	1	UC	
Serial Poll Enable	SPE	X	0	0	1	1	0	0	0	UC	
Take Control	TCT	X	0	0	0	1	0	0	1	AC	12
Talk Address Group	TAG	X	1	0	X	X	X	X	X	AD	
Universal Command Group	UCG	X	0	0	1	X	X	X	X	UC	
Unlisten	UNL	X	0	1	1	1	1	1	1	AD	
Untalk	UNT	X	1	0	1	1	1	1	1	AD	

Symbols: AC — Addressed Command
AD — Address (Talk or Listen)
UC — Universal Command
SE — Secondary (Command or Address)
0 — Logical Zero (high level on IEEE Bus; Low level within 9914).
1 — Logical One (Low level on IEEE Bus; High level within 9914).
X — Don't Care (received message)
X — Must Not Drive (transmitted message)

▶8

Notes to Table 3:

1. L L L L L: Represents the coding for the device listen address.
2. T T T T T: Represents the coding for the device talk address.
3. S S S S S: Represents the coding for the device secondary address.
4. Secondary addresses will be handled via address pass through.
5. OSA will be handled as an invalid secondary address pass through by the MPU.
6. PCG = ACG v UCG v LAG v TAG
7. PPC will be handled in software by the MPU via Unrecognized Address Command Group pass through.
8. PPE, PPD will be handled via pass through next secondary feature.
9. S P P P represents the sense and bit for remote configurable parallel poll.
10. D D D D specify don't care bits that must be sent all zeroes, but need not be decoded by receiving device.
11. PPU is handled via Unrecognized Universal Command Group pass through.
12. TCT will be handled via Unrecognized Addressed Command Group pass through. However, in this case, the device must be in TADS before the pass through will occur.

Interrupt Status Registers 0 and 1

INT0	INT1	BI	BO	END	SPAS	RLC	MAC
GET	UUCG	UACG	APT	DCAS	MA	SRQ	IFC

INT0	An interrupt occurred in register 0
INT1	An interrupt occurred in register 1
BI	A byte has been received
BO	A byte has been output
END	An EOI occurred with ATN false
SPAS	Serial Poll Active State has occurred with rsv set in the Serial Poll register
RLC	A REMOTE/LOCAL change has occurred
MAC	An address change has occurred
GET	A Group Execute Trigger has occurred
UUCG	An Undefined Universal Command has been received
UACG	An Undefined Addressed Command has been received. This bit will also be set on receipt of a secondary command when the pts feature in the Auxiliary Command register is utilized.
APT	A secondary address has occurred
DCAS	Device Clear Active State has occurred
MA	My Address (MLAVMTA)•$\overline{\text{SPSM}}$
SRQ	A Service Request has been received
IFC	An IFC has been received

INT0 is the logical OR of each bit of Interrupt Status Register 0 ANDed with the respective bit of Interrupt Mask Register 0. INT1 is the same but applies to Interrupt Mask and Status Register 1. Reading either Interrupt Status Register will also clear it. The INT line will be cleared only when the interrupt status register which caused the interrupt is read.

Interrupt Mask Registers 0 and 1

		BI	BO	END	IFC	RLC	MAC
GET	UUCG	UACG	APT	DCAS	MA	SRQ	SPAS

The Interrupt Mask Registers 0 and 1 correspond to the Interrupt Status Registers 0 and 1 respectively, with the exception of INT0 and INT1.

Address Status Register

REM	LLO	ATN	LPAS	TPAS	LADS V LACS	TADS V TACS	ulpa

The Address Status Register is used to convey the addressed state of the talker/listener and the remote/local and local lockout condition. This information is derived from the TMS 9914 internal logic states at the time of reading. The ulpa bit is used for dual addressing and indicates the state of the LSB of the bus at last primary addressed time.

Bus Status Register

ATN	DAV	NDAC	NRFD	EOI	SRQ	IFC	REN

The Bus Status Register allows the microprocessor to obtain the current status of the IEEE 488 Bus Management Lines.

Auxiliary Command Register

C/S			f4	f3	f2	f1	f0

The Auxiliary Command Register allows control of additional features on chip and provides a means of inputting some of the local messages to the interface functions. Table 4 lists these messages and commands. If C/S = 1, the feature will be set and if C/S = 0, the feature will be cleared. If C/S = NA, it should be sent as zero.

Table 4. Auxiliary Commands

Function	*Mnemonic*	*C/S*	*f4*	*f3*	*f2*	*f1*	*f0*
Chip Reset	rst	0/1	0	0	0	0	0
Release ACDS holdoff	dacr	0/1	0	0	0	0	1
Release RFD holdoff	rhfd	NA	0	0	0	1	0
Holdoff on all data	hdfa	0/1	0	0	0	1	1
Holdoff on EOI only	hdfe	0/1	0	0	1	0	0
Set new byte available false	nbaf	NA	0	0	1	0	1
Force group execute trigger	fget	0/1	0	0	1	1	0
Return to local	rtl	0/1	0	0	1	1	1
Return to local immediate	rtli	0	0	0	1	1	1
Send EOI with next byte	feoi	NA	0	1	0	0	0
Listen only	lon	0/1	0	1	0	0	1
Talk only	ton	0/1	0	1	0	1	0
Take control synchronously	tcs	NA	0	1	1	0	1
Take control asynchronously	tca	NA	0	1	1	0	0
Go to standby	gts	NA	0	1	0	1	1
Request parallel poll	rpp	0/1	0	1	1	1	0
Send interface clear	sic	0/1	0	1	1	1	1
Send remote enable	sre	0/1	1	0	0	0	0
Request contol	rqc	NA	1	0	0	0	1
Release control	rlc	NA	1	0	0	1	0
Disable all interrupts	dai	0/1	1	0	0	1	1
Pass through next secondary	pts	NA	1	0	1	0	0
Set T1 delay	stdl	0/1	1	0	1	0	1

Address Register

edpa	dal	dat	A5	A4	A3	A2	A1

edpa	enable dual primary addressing	dat	disable the talk function
dal	disable the listen function	A1 – A5	primary device address

The Address Switch Register corresponds to the Address Register. A power-up RESET or a rst command with C/S = 1 will leave the chip in a totally idle state. At this point, the Address Switch Register is read and the value is written into the Address Register. The reset condition is then cleared by sending rst with C/S = 0.

Serial Poll Register

S8	rsv	S6	S5	S4	S3	S2	S1

The Serial Poll register is used to establish the status byte that is sent out when the controller conducts a serial poll. Bits 1 through 6 and 8 contain status information, while bit 7, rsv, is used to enable the SRQ line and to indicate to the controller which device(s) was responsible for making a service request.

Command Pass Through Register

DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

The Command Pass Through Register is used to pass through to the microprocessor any commands or secondary addresses that are not automatically handled in the TMS 9914.

Parallel Poll Register

PP8	PP7	PP6	PP5	PP4	PP3	PP2	PP1

This register contains the status bit that is output when the controller conducts a parallel poll.

Data-In Register

DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

The data-in register is used to move data from the interface bus when the chip is addressed as a listener. Upon receipt of a data byte, the chip will hold NRFD true until the microprocessor reads the data-in register, when NRFD will be set false automatically.

Data-Out Register

DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

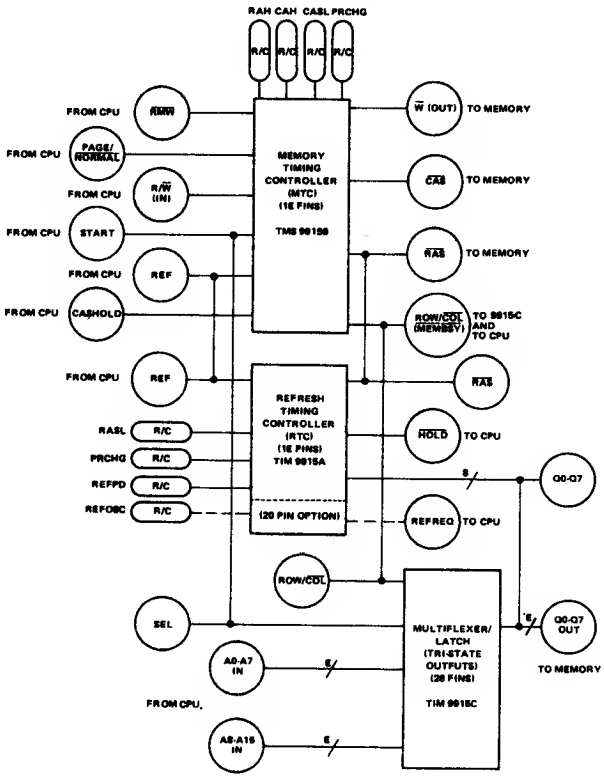
The data-out register is used to move data from the TMS 9914 onto the IEEE std 488-1975 data bus.

After sending a byte out on the bus, the device can take part in a new handshake only after a new byte is placed in the data-out register, when it will be able to send DAV true again.

TIM 9915 - MEMORY TIMING (AND REFRESH) CONTROLLER CHIP SET

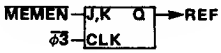
Features of TIM 9915:

- Controls the operation of 4K/16K/64K Dynamic RAMs
- Creates Static RAM Appearance
- Generation and Synchronization of
  - $\overline{RAS}$, $\overline{CAS}$, clocks (and precharge)
  - $\overline{WE}$ Signal
  - Address Multiplexing
  - Refresh (multiple modes)
- Works over wide range of memory speeds
  - Access from 120 ns up to refresh limits
  - Even faster with precision R/C's
- Performs multiple memory cycles
  - Read, Early Write, Read/Write, RMW
- Page-Mode Operation for all memory cycle types
- Selectable Refresh Modes
  - Transparent
  - Cycle Steal
  - Burst Mode
- Refresh Violation Detection
  - Automatic Burst Mode on violation acknowledgement
- Simple Asynchronous START Clock
- Extended $\overline{CAS}$ Data Hold Control via CASHOLD
- All system outputs are
  - Bus Drivers (24mA guaranteed)
  - Tri-State



NOTES: Transparent Refresh Interface for Popular Microprocessors.

TMS 9900



Z80 "MREF" = REF

8080A M1 • T4 = REF

8085 M1 • T4 = REF

DESIGN GOAL

This document describes the design specifications for a product under development. Texas Instruments reserves the right to change these specifications in any manner, without notice.

TMS 9927JL,NL
VIDEO TIMER/CONTROLLER

990/9900 FAMILY MICROCOMPUTER COMPONENTS

- Second sourced by SMC as CRT5027
- TTL Compatibility
- Standard and Non-standard CRT Monitor Capability
- Scrolling Capability
- Interlaced and Non-interlaced Formatting
- Fully Programmable Display Format
  - Characters per data row
  - Data rows per frame
  - Raster scans per data row
  - Raster scans per frame
- Fully Programmable Monitor Format
  - Blanking
  - Horizontal sync
  - Vertical sync
  - Composite sync
- Two Programming Methods
  - Processor controlled
  - PROM on data bus
- Generation of Cursor Video
- N-channel Silicon-Gate Technology

DESCRIPTION

The TMS9927 is a single-chip video timer and controller produced using N-channel silicon-gate MOS technology. This 40-pin package contains the logic to generate all the timing signals for display of video data on standard or nonstandard CRT monitors in both interlaced or noninterlaced format. The only function not on the chip is the dot counter; which, due to high video frequencies, cannot easily be implemented with MOS technology. All the inputs and outputs are TTL compatible.

There are nine 8-bit control registers which are user programmable (see *Table 1*). Seven of the control registers are for horizontal and vertical formatting and two are for cursor address.

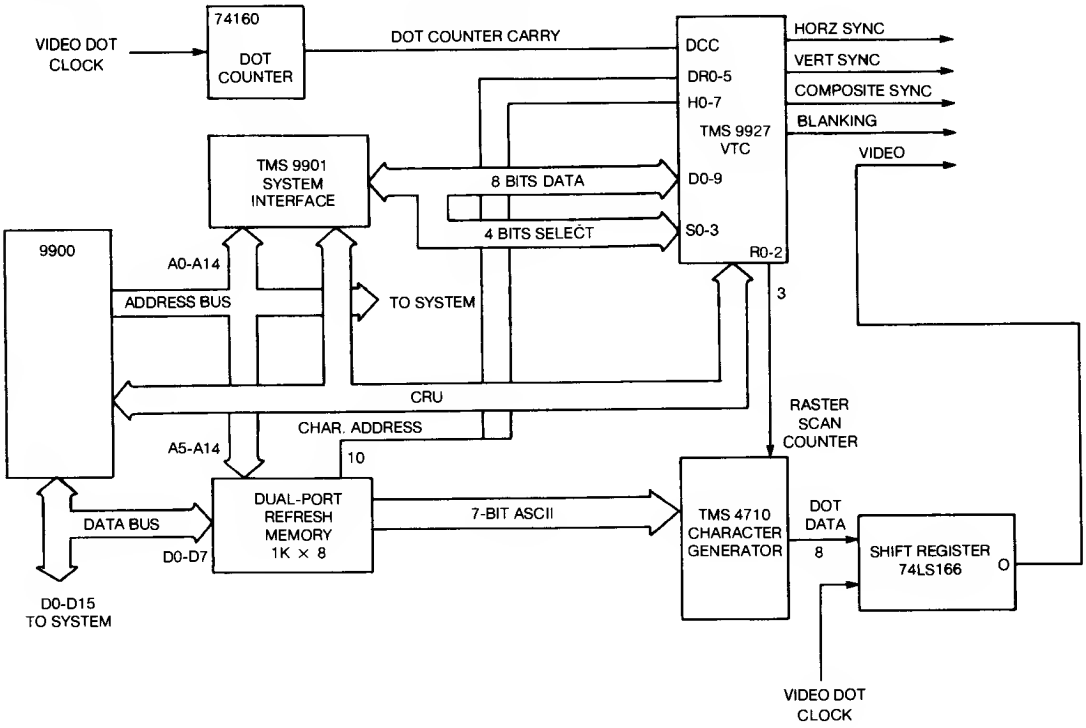


Figure 1. Typical System Interconnect

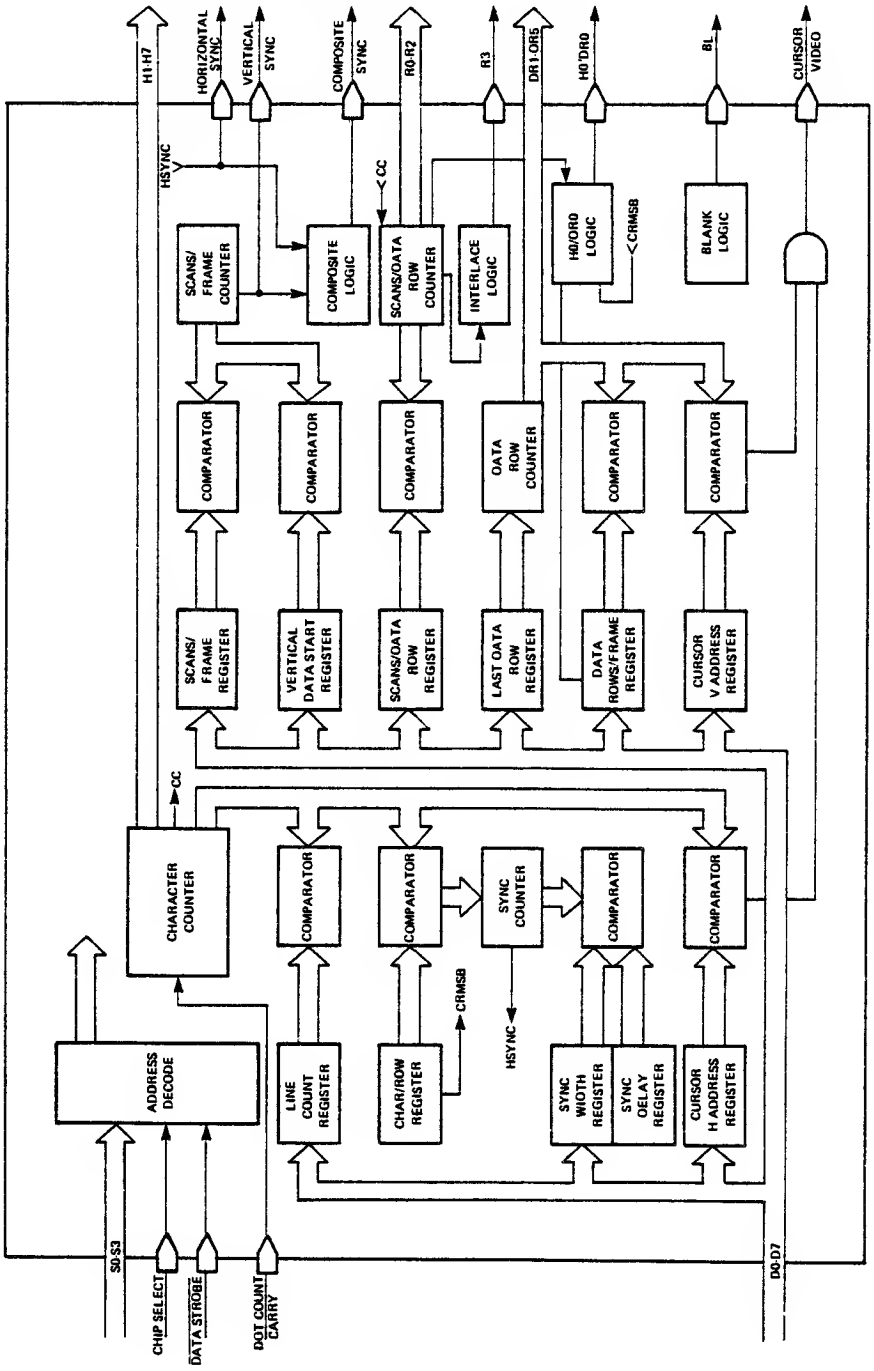
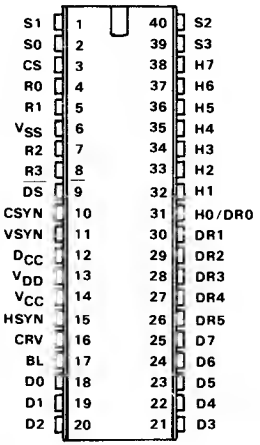


Figure 2. TMS 9927 Architecture.

TMS 9927 PIN FUNCTIONS

Signature	*I/O*	*Description*
D0-7	I/O	Data bus. Input bus for control words from microprocessor or PROM. Bidirectional bus for cursor address.
CS	I	Signals chip that it is being addressed
S0-3	I	Register address bits for selecting one of seven control registers or either of the cursor address registers
$\overline{DS}$	I	Strobes D0-7 into the appropriate register or outputs the cursor character address or cursor line address onto the data bus
DCC	I	Carry from off chip dot counter establishing basic character clock rate.
H7-1	O	Character counter outputs.
R0-2	O	Three most significant bits of the Scan Counter: row select inputs to character generator.
H0/DR0	O	Pin definition is user programmable. Output is MSB of Character Counter if MSB of Characters/Data Row word is a "1", otherwise output is MSB of Data Row Counter
R3	O	Least significant bit of the scan counter. In interlaced mode this bit defines the odd or even field. In this way, odd scan lines of the character font are selected during the odd field and even scans during the even field
DR1-5	O	Data Row counter outputs
BL	O	Defines non active portion of horizontal and vertical scans.
HSYN	O	Initiates horizontal retrace
VSYN	O	Initiates vertical retrace
CSYN	O	Active in non-interlaced mode only. Provides a true RS-170 composite sync waveform.
Vcc	PS	+5 volt Power Supply
V_{DD}	PS	+12 volt Power Supply
V_{SS}	PS	Ground reference



FUNCTIONAL DESCRIPTION

Application Oriented Use

The TMS9927 interfaces to the central processor unit, if one is used, through the communications register unit (CRU) via a TMS9901, as shown in *Figure 2*, or it functions as a mapped memory device. The TMS9901 converts 8 bits of serial CRU data to parallel data to feed the TMS9927 data bus for loading the control registers. The CPU, using the CRU, decodes the high order bits of the address for the TMS9927 chip select and the four low order bits are connected directly to the TMS9927 Video Timer and Controller (VTC) for control register select. The character column (H1-H7) and row lines (DR1-DR5) combine to address the refresh RAM. The refresh RAM outputs the seven-bit ASCII code for the character to be displayed to the character generator, the TMS4710. The character generator uses the raster scan counter (R0-R2) to select which row of the dot matrix to output. A shift register then shifts the dot information out to the video terminal at the dot frequency.

The TMS9927 does have a self-load function as shown in *Figure 3*. It is accomplished by putting the self-load command on the VTC select lines and strobing DATA STROBE ($\overline{DS}$). This causes the TMS9927 to output address information on its row select lines to the control PROM (74S288). The outputs of the control PROM are loaded into the VTC control registers. There are two types of self-load: processor and nonprocessor. The nonprocessor self-load automatically starts the timing chain after load is completed. Processor self-load only causes a self-load and then waits for the start command from the processor. The select signals to the VTC which cause self-load should be applied for the entire duration of self-load.

Table 1. Select Decodes

S0	*S1*	*S2*	*S3*	*Command*	*Description*
0	0	0	0	Load Control Register 0	
0	0	0	1	Load Control Register 1	
0	0	1	0	Load Control Register 2	
0	0	1	1	Load Control Register 3	See *Table 2*
0	1	0	0	Load Control Register 4	
0	1	0	1	Load Control Register 5	
0	1	1	0	Load Control Register 6	
0	1	1	1	Processor Self Load	Command from processor instructing TMS 9927 to enter Self Load Mode
1	0	0	0	Read Cursor Row Address	
1	0	0	1	Read Cursor Character Address	
1	0	1	0	Reset	Resets timing chain to top left of page. Reset is latched on chip by $\overline{DS}$ and counters are held until released by start command.
1	0	1	1	Up Scroll	Increments address of first displayed data row on page. ie; prior to receipt of scroll command—top line = 0, bottom line = 23. After receipt of Scroll Command—top line = 1, bottom line =0.
1	1	0	0	Load Cursor Character Address	
1	1	0	1	Load Cursor Row Address	
1	1	1	0	Start Timing Chain	Receipt of this command after a Reset or Processor Self Load command will release the timing chain approximately one scan line later. In applications requiring synchronous operation of more than one TMS9927 the dot counter carry should be held low during the $\overline{DS}$ for this command.
1	1	1	1	Non-Processor Self Load	Device will begin self load via PROM when $\overline{DS}$ goes low. The 1111 command should be maintained on S0-3 long enough to guarantee self load (Scan counter should cycle through at least once). Self load is automatically terminated and timing chain initiated when the all "1's" condition is removed, independent of $\overline{DS}$. For synchronous operation of more than one TMS 9927, the Dot Counter Carry should be held low when this command is removed.

Note: During Self Load, the scan counter states corresponding to the nine load command addresses will load the appropriate register. Therefore if reseting of the cursor X and Y position registers is required via self load the PROM words for address 1100 and 1101 should be programmed as all zeros.

▶8

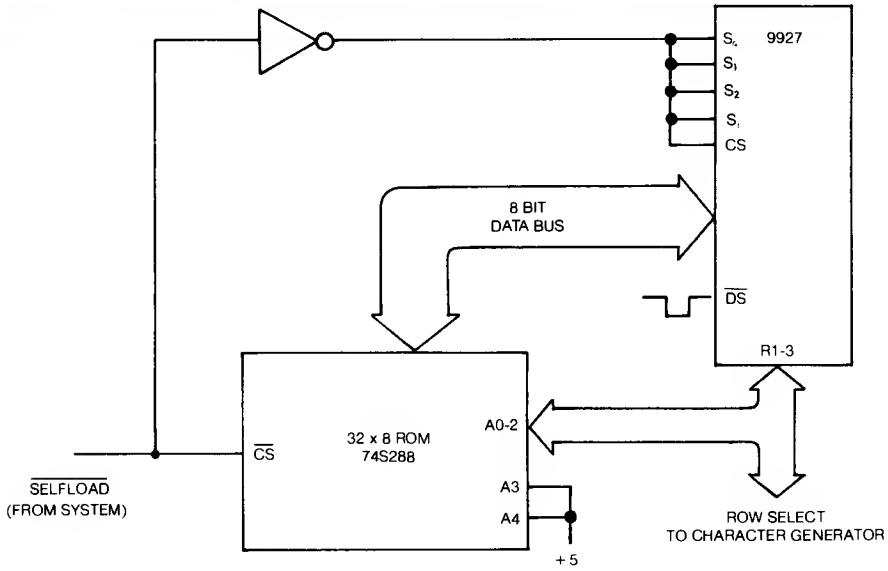


Figure 3. Self-load Command

ARCHITECTURE

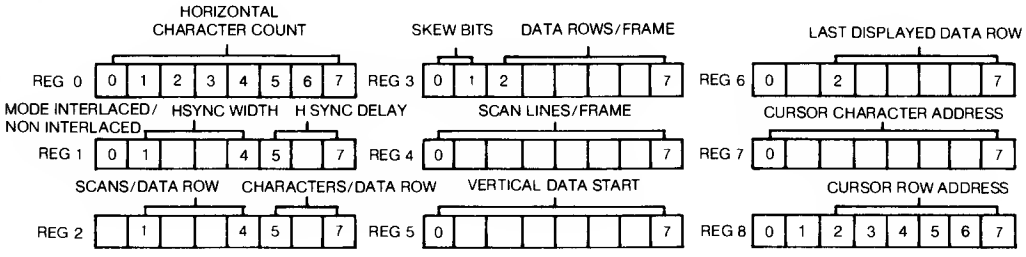
General

The functional block illustrates the architecture of the TMS9927 video timer and controller. The architecture is designed to be as general as possible so that by programming the control registers properly almost any raster scan CRT can be controlled with this chip.

Select Lines

Lines S0-S3 are the select lines. They select the control register for loading via the data bus (DB0-DB7) and also select control functions for the device (see *Table 1*). The bit assignments for the nine control registers are given in *Table 2*. Notice that the cursor line address and the character address can both be read; therefore, the TMS9927 data bus must be bidirectional.

Table 2. TMS 9927 Control Registers.

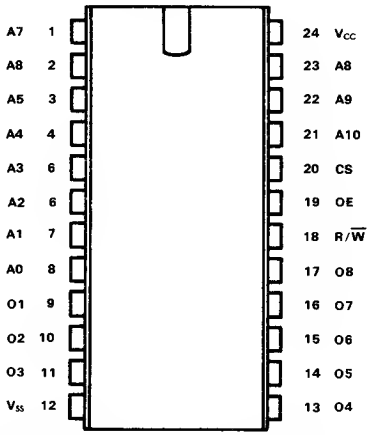


8◀

TMS 9932 JC, NC
COMBINATION ROM/RAM MEMORY

- Single 5 Volt Power Supply, ± 10%
- 15,360 Bits Read Only Memory
- 1024 Bits Random-Access Memory
- All Inputs and Outputs TTL Compatible (No Pull-Up Resistors Needed)
- Maximum Access Time — 400 ns
- Maximum Cycle Time — 400 ns
- Low Power Dissipation of 300 mW (Typical)
- Programmable Chip Select and Output Enable Buffers for Expansion
- N-MOS Silicon-Gate Technology
- Standard 0°C to 70°C Temperature Range

24-Pin Ceramic and Plastic Dual-in-line Packages (Top View)



DESCRIPTION

The TMS 9932 is a 15,360 bit read only memory and a 1024 bit random access memory organized as 1920 words of 8 bit length ROM and 128 words of 8 bit length RAM. The highest 128 addresses will access the RAM, while tbe lower 1920 addresses access the ROM.

The device is fabricated using N-channel silicon gate technology and is completely static, allowing simple interfacing to bipolar and other MOS circuitry with a minimum system parts count.

All inputs can be driven by 7400 series TTL circuits without the use of any external pull-up resistors. Similarly, each output can drive up to two 7400 series TTL circuits without external resistors.

The data outputs are tri-state for OR tying multiple devices on a common bus. A logical zero on the chip select (CS) or the output enable input (OE) forces the input/output buffers into the high impedance state. Chip select and read/write input ($R\overline{W}$) allow data to be written into the RAM while automatically forcing the 1/O buffers into the high impedance state.

The device is supplied in 24 pin dual-in-line plastic and ceramic packages designed for insertion in mounting hole rows on 600-mil centers.

The device is designed for operation over a commercial temperature range from 0°C to 70°C.

OPERATION

Address (A_0 – A_{10})

8

The 11-bit positive-logic address is decoded on-chip to select one of 2048 words of 8-bit length in the memory array. Addresses 0 to 1919 are ROM addresses; addresses 1920 to 2047 are RAM addresses. A_0 is the least-significant bit and A_{10} is the most-significant bit of the word addresses.

Chip Select (CS)

Chip select can be programmed at the factory at the same time the ROM is programmed to be active with either a high or low input signal. This allows for system expansion to use more than one ROM/RAM circuit. When chip select is disabled, data cannot be written into the RAM and the outputs are in the high-impedance state.

Mode Select ($R/\overline{W}$)

The $R/\overline{W}$ input must be high during read and low during write operations to the RAM. Prior to an address change, $R/\overline{W}$ must be in the read state and must remain in that state for a minimum period to eliminate the possibility of data being written into an unwanted position.

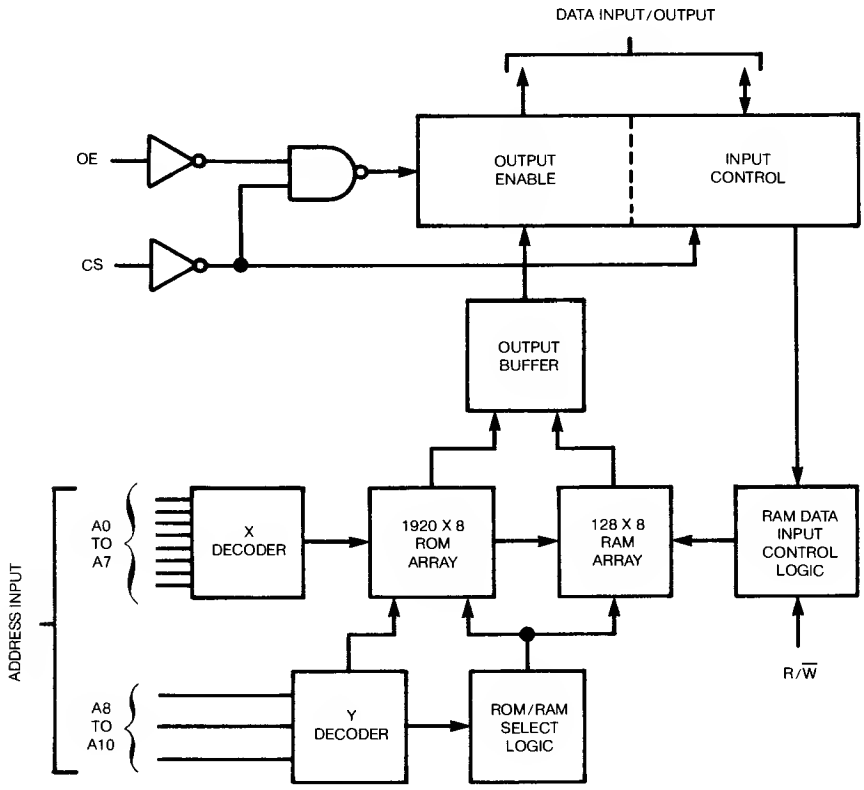
Output Enable (OE)

The output enable input can be programmed, during mask fabrication, to be active with either a high or a low input signal. When output enable is active, all eight outputs are enabled and the eight-bit addressed word can be read. When output enable is not active, all eight outputs are in a high-impedance state.

Data Input/Output (I/O1 – I/O8)

The common input/output terminals are used for both read and write operations. During a write cycle, data must be set up a minimum time before R/$\overline{W}$ goes to the read state (high) to ensure that correct data will enter the addressed memory cell. Also, input data must be held a minimum time after the rise of R/$\overline{W}$.

The output buffers are controlled by output enable and chip select. To read data, output enable and chip select must be valid.



FUNCTION TABLE			
R/$\overline{W}$	CS	OE	OPERATION
L	H	H	Write into RAM (I/O1 to I/O8 = ?)
L	H	L	Write into RAM (I/O1 to I/O8 = Z)
H	H	H	Ream ROM at RAM
X	L	X	Device disabled (I/O1 to I/O8 = Z)
X	X	O	Output disabled (I/O1 to I/O8 = Z)

L = LOW, H = HIGH, X = DON'T CARE, Z = HIGH IMPEDANCE, ? = INDETERMINATE

Functional Block Diagram

Absolute Maximum Ratings Over Operating Free-Air Temperature Range (Unless Otherwise Noted)*

Supply voltage, V_{CC} (see Note 1) . . . . . −0.5 to 7 V
Input voltage (any input) (see Note 1) . . . . . −0.5 to 7 V
Continuous power dissipation . . . . . 1 W
Operating free-air temperature range . . . . . 0° to 70°C
Storage temperature range . . . . . −55°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Retings" may cause permanent damege to the device. This is a stress rating only and functionel operation of the device at these or eny other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to ebsolute maximum-reted conditions for extended periods may affect device reliability.

NOTE 1: Voltage velues are with respect to the ground terminal.

Recommended Operating Conditions

PARAMETER	0°C – 70°C			UNIT
	NIN	NOM	MAX	
Supply voltage, V_{CC}	4.5	5	5.5	V
High-level input voltage, V_{IH}	2		V_{CC}	V
Low-level input voltage, V_{IL} (see Note 2)	−0.5		0.8	V
Reed cycle time, $t_{C(rd)}$		400		ns
Write cycle time, $t_{C(W)}$		400		ns
Write pulse width, $t_{W(W)}$		250		ns
Address setup time, $1_{SU(A)}$		120		ns
Chip select setup time, $t_{SU(CS)}$		350		ns
Data setup time $1_{SU(D)}$		300		ns
Address hold time, $t_{h(A)}$		30		ns
Deta hold time, $t_{h(D)}$		30		ns
Opereting free-air temperature, T_A	0		70	C

NOTE 2: The elgebraic convention where the most negative limit is designeted as minimum is used in this date sheet for logic voltage levels only.

Electrical Characteristics Over Recommended Operating Free-Air Temperature Range (Unless Otherwise Noted)

PARAMETER		TEST CONDITIONS		MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage	I_{OH} = −150μA,	V_{CC} = 4.5 V		2.4		V
V_{OL}	Low-level output voltage	I_{OL} = 3.2 mA	V_{CC} = 5.5 V		0.45		V
I_I	Input current	V_I = 0 to 5.25 V			10		μA
I_{OZH}	Off stete output current, high-level voltage applied	CS at 2.0 V,	V_O = 4 V		15		μA
I_{OZL}	Off-state output current, low-level voltage applied	CS at 2.0 V,	V_O = 0.45 V		−50		μA
I_{CC}	Supply current from V_{CC}	V_{CC} = 5.5 V, I_O = 0 mA	T_A = 0°C to 70°C		55		mA
			T_A = −40° to 125°C		55		
C_i	Input capecitance	V_I = 0 V, f = 1 MHz	T_A = 25°C		4		pF
C_o	Output capacitance	V_O = 0 V, f = 1 MHz	T_A = 25°C		10		pF

†All typical values are at V_{CC} = 5 V, T_A = 25°C.

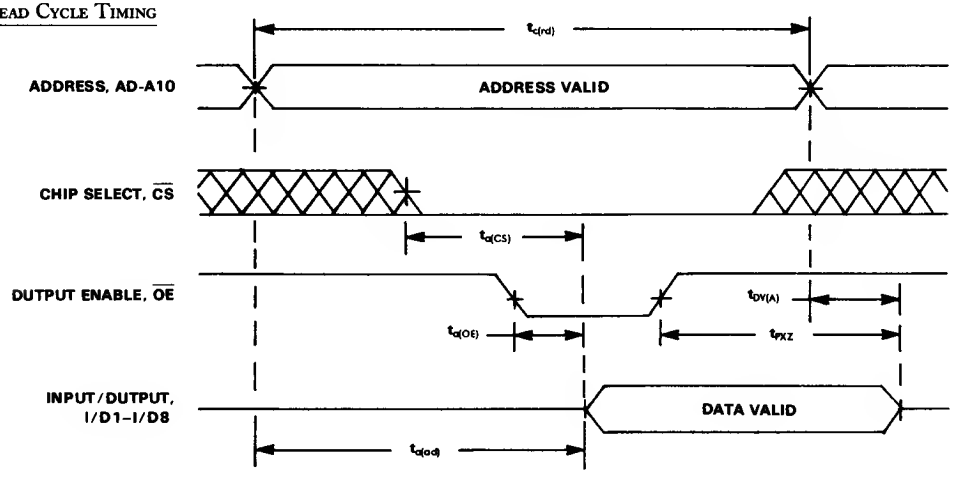
▶8

SWITCHING CHARACTERISTICS OVER RECOMMENDED SUPPLY VOLTAGE RANGE, 2 SERIES 74 TTL LOAD, C_L = 100 pF

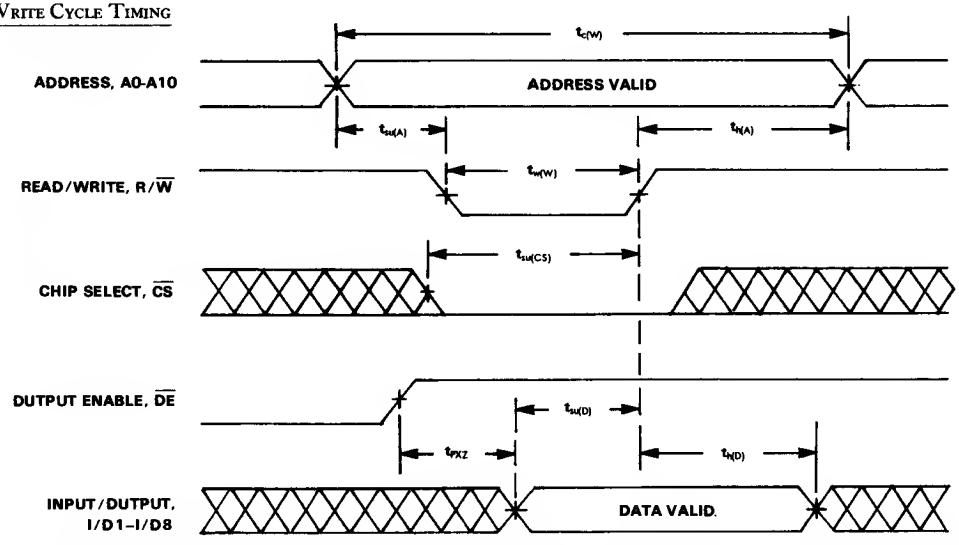
PARAMETER		0° – 70°C			UNIT
		MIN	TYP	MAX	
$t_{a(A)}$	Access time from address		400		ns
$t_{a(CS)}$	Access time from chip select		300		ns
$t_{a(OE)}$	Access time from output enable		100		ns
$t_{DV(A)}$	Previous output date valid after address chenge		40		ns
t_{pxz}	Output disable time from output eneble (see Note 3)		150		ns

NOTE 3: This peremeter defines the delay for the I/O bus to enter the input mode

READ CYCLE TIMING



WRITE CYCLE TIMING



NDTE: FDR MEASURING TIMING REQUIREMENTS AND CHARACTERISTICS V_{IH}=2.0V, V_{IL}=0.50V, t_r=t_f=20ns AND ALL TIMING PDINTS ARE 50% POINTS.

8

SOFTWARE PACKAGE

The TMS 9932 is a fixed program memory in which the programming is performed by TI at the factory during the manufacturing cycle to the specific customer inputs supplied in the format shown. The device is organized as 1920 8 bit words with address locations numbered 0 to 1919. Any 8 bit word can be coded as a 2 digit hexadecimal number between 00 and FF. All stored words and addresses in the format are coded in hexadecimal numbers. In coding, all binary words must be in positive logic before conversion to hexadecimal. I/O1 is considered the least significant bit and I/O8 the most significant bit. Addresses input A0 is least significant and A10 is most significant.

Every card should include the TI Customer Device Number in the form MP XXXX-XXX (8 digit number to be assigned by TI) in columns 71 through 80.

Output enable is customer programmable. Every card should include in column 70 a 1 if the output is to be enabled with a high level input at $\overline{OE}$ or a 0 for enabling with a low level input.

The 1920 coded words must be supplied on 120 cards with 16 two digit hex numbers per card.

CARD	COLUMN	
1	1-9	BLANK
	10	:(ASCII character colon)
	11-12	10 (specifies 16 words per card)
	13	BLANK
	14-16	Hex address of 1st word on 1st card (0th word, address normally 000)
	17-18	BLANK
	19-20	0th word in Hex
	•	
	•	
	•	
	49-50	15th word in Hex
	51-69	BLANK
	70	Output Enable ($\overline{OE}$) Active State
	71-80	Customer Device Number

CARD	COLUMN	HEXADECIMAL INFORMATION
120	1-9	BLANK
	10	:(ASCII character colon)
	11-12	10
	13	BLANK
	14-16	Hex address of 1st word on 120th card (1904th word, address normally 770)
	17-18	BLANK
	19-20	1904th word in Hex
	•	
	•	
	•	
	49-50	1919th word in Hex
	51-69	BLANK
	70	Output Enable ($\overline{OE}$) Active State
	71-80	Customer Device Number

▶8

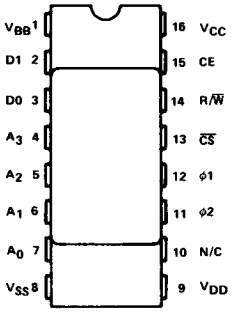
TMS 3064 JL
65,536-BIT CCD MEMORY

- 65,536 x 1 Organization
  (16 Addressable 4096-Bit Loops)
- Performance:

LATENCY TIME AT 5 MHz (MAX)	READ OR WRITE CYCLE (MIN)	READ, MODIFY WRITE CYCLE (MIN)
820 μs	200 ns	300 ns

- Full TTL Compatibility (No Pull-up Resistors Required) on All Inputs Except $\phi 1$, $\phi 2$, and Chip Enable
- Low Power Dissipation:
  280 mW Operating (Typical @ 5 MHz)
  25 mW Recirculating (Typical @ 1 MHz)
  <1 mW Standby (Typical)
- Two-Phase CCD Clocks
- N-Channel Silicon-Gate Technology
- 16-Pin, 400-Mil Dual-in-Line Package

16-PIN CERAMIC
DUAL-IN-LINE PACKAGE

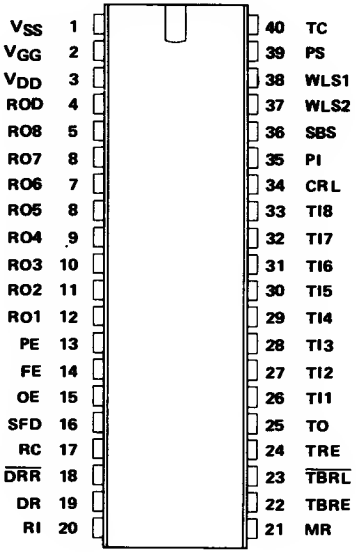


TMS 6011 JC, NC
ASYNCHRONOUS DATA INTERFACE (UART)

MOS
LSI

- Transmits, Receives, and Formats Data
- Full-Duplex or Half-Duplex Operation
- Operation from DC to 200 kHz
- Static Logic
- Buffered Parallel Inputs and Outputs
- Programmable Word Lengths . . . 5, 6, 7, 8 Bits
- Programmable Information Rate
- Programmable Parity Generation/Verification
- Programmable Parity Inhibit
- Automatic Data Formatting
- Automatic Status Generation
- 3-State Push-Pull Buffers
- Low-Threshold Technology
- Standard Power Supplies . . . 5 V, −12 V
- Full TTL Compatibility . . . No External Components

40-PIN CERAMIC AND PLASTIC
DUAL-IN-LINE PACKAGES
(TOP VIEW)



8

990/9900 FAMILY MICROCOMPUTER COMPONENTS

- SBP/TMS 9900 Series Microprocessor Family Peripheral
- 16 Individual, Single-Bit, Software Configurable I/O Ports
- 20/40 mA Current Sinking Outputs
- 28-Pin Package
- Software Compatible with TMS 9901 when used in conjunction with SBP 9961
- TTL Compatible I/O
- Wide Ambient Temperature Operation
  - SBP 9960CJ: 0°C to +70°C
  - SBP 9960EJ: -40°C to +85°C
  - SBP 9960MJ: -55°C to +125°C
  - SBP 9960NJ: -55°C to +125°C (with high-reliability processing)
- I^2L Technology
  - Constant Current Power Source
  - Fully Static Operation
  - Single Phase Edge-Triggering Clock
  - Wide Temperature Stability

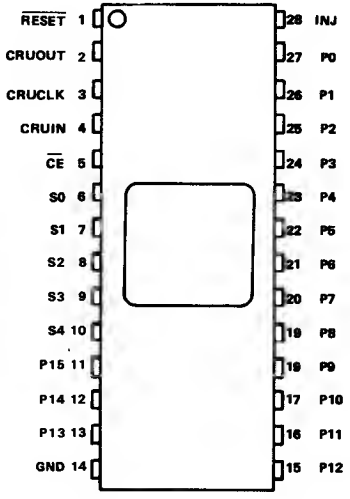
DESCRIPTION

The SBP 9960 CRU I/O Expander is a ruggedized monolithic software-configuration input/output device fabricated with oxide separated Integrated Injection Logic (I^2L) technology. The SBP 9960 provides a flexible and efficient Communications Register Unit (CRU) based interface between the SBP/TMS 9900 series Family of Microprocessors and auxiliary systems functions ranging from bit-oriented sensors and actuatore to byte/word/n-bit-field oriented peripherals.

Under software control, each of the SBP 9960s sixteen individual single-bit I/O ports may be configured to either the input or output mode. I^2L technology enables the SBP 9960s static logic, and TTL compatible I/O, to operate over a very wide ambient temperature range from a single d-c power source with output current sink capability up to 40 mA. When the SBP 9960 is used in conjunction with the SBP 9961 I^2L Interrupt-Controller/Timer, the SBP 9960/SBP 9961 pair form an I^2L systems alternate to the N-channel MOS TMS 9901 Programmable Systems Interface device while maintaining strict compatibility with existing software handlers developed in support of the TMS 9901.

SBP 9960 PIN ASSIGNMENTS AND FUNCTIONS

Signature	*Pin*	*I/O*	*Description*
S0	6	IN	ADDRESS SELECT LINES. The data bit being accessed by the CRU interface is specified by the 5-bit code appearing on S0-S4.
S1	7	IN	
S2	8	IN	
S3	9	IN	
S4	10	IN	
CRUIN	4	OUT	CRU DATA IN (to CPU). Data specified by S0-S4 is transmitted to the CPU by CRUIN. When $\overline{CE}$ is not active, CRUIN is pulled to logic-level high.
CRUOUT	2	IN	CRU DATA OUT (from CPU). When $\overline{CE}$ is active data present on the CRUOUT input will be strobed by CRUCLK and written into the CRU bit specified by S0-S4.
CRUCLK	3	IN	CRU CLOCK (from CPU). CRUCLK specifies that valid data is present on the CRUOUT line.
$\overline{RESET}$	1	IN	POWER-UP RESET. When active (low), $\overline{RESET}$ forces all I/O's (P0-P15) to input mode.
$\overline{CE}$	5	IN	CHIP ENABLE. When active (low), data may be bidirectionally transferred between the SBP 9960 and the CPU.
INJ	28		Supply Current
GND	14		Ground Reference
P0	27	I/O	I/O pins
P1	26	I/O	
P2	25	I/O	
P3	24	I/O	
P4	23	I/O	
P5	22	I/O	
P6	21	I/O	
P7	20	I/O	
P8	19	I/O	
P9	18	I/O	
P10	17	I/O	
P11	16	I/O	
P12	15	I/O	
P13	13	I/O	
P14	12	I/O	
P15	11	I/O	



8

FUNCTIONAL DESCRIPTION

SBP 9960/CPU Interface

The SBP 9960 communicates with the CPU through the Communications Register Unit (CRU) interface as shown in Figures 1 and 3. The SBP 9960s CRU interface consists of: a) five CRU address select lines (S0-S4), b) a single chip enable ($\overline{CE}$), c) a 9960 to CPU serial data-bit line (CRUIN), d) a CPU to 9960 serial data-bit line (CRUOUT), and e) a CPU to 9960 serial data-bit clock (CRUCLK). When $\overline{CE}$ is activated (logic level low), S0-S4 select a specific single-bit I/O port as indicated in Table 1. In the case of an SBP 9960 write operation, the datum is transferred from the CPU to the SBP 9960 via the CRUOUT line. The CRUOUT datum is strobed into the selected single-bit port by CRUCLK. In the case of a SBP 9960 read operation, the selected single-bit port is sampled by the CPU via the CRUIN line.

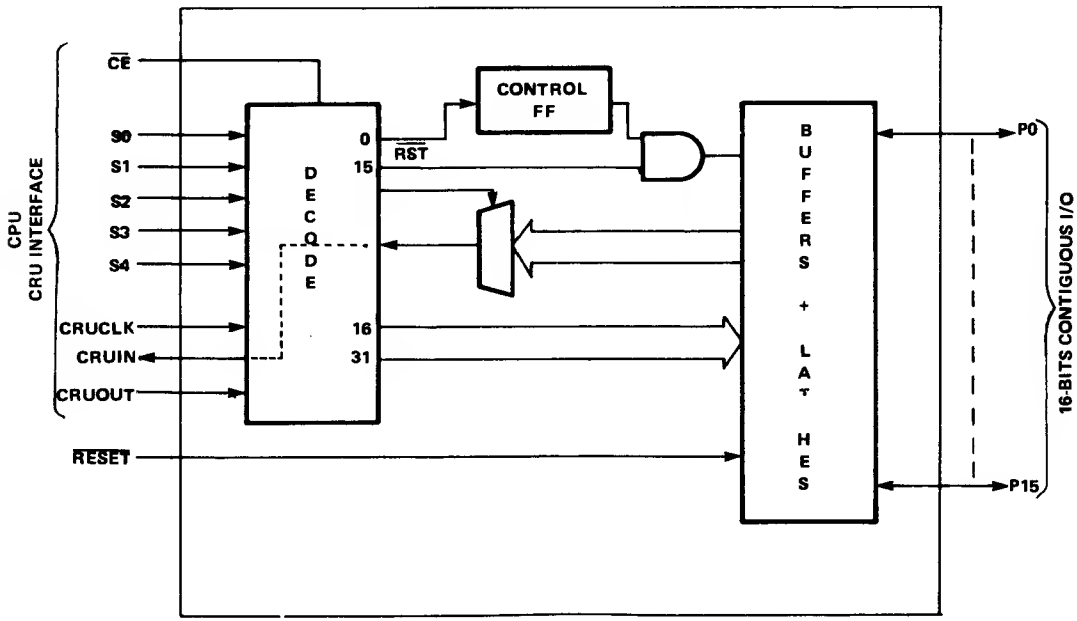


Figure 1.

CRU Bit Assignments

Table 1 describes the SBP 9960's CRU bit assignments. Note that CRU bits 1-14 have been reserved for the SBP 9961 thereby insuring software compatibility between the SBP 9960/SBP9961 pair and the TMS 9901.

Input/Output

One of sixteen SBP 9960, single-bit, combination open-collector-output/resister-divider-input I/O ports is conversationally represented in Figure 2. As a direct result of the open-collector output structure, the data flow direction through the port is determined by the stored logic-level of the associated output-register bit in combination with the data flow direction of the external device serviced by the port. When the ouput-register bit (Q) is at logic-level high, the corresponding package pin (P) is essentially floating and therefore free to be externally pulled to either the high or low logic-level. In other words, when Q is at logic-level high, the ports data flow direction can be either inward, where an external device pulls P to the high or low logic-level; or the data flow direction can be outward, where an external resistor (R) both pulls P to logic-level high and sources current drive into the inputs of external devices. When Q is at logic-level low, the ports unconditional data flow direction is outward, where P has the capacity to sink 20/40*mA of current from external devices. Q can be reset to logic-level low through CPU execution of a SET BIT TO ZERO (SBZ) instruction; Q can be set to logic-level high through: 1) a hardware initiated reset ($\overline{\text{RESET}}$), 2) a software initiated reset ($\overline{\text{RST}}$: CRU BIT 15) preceded by setting the control (CRU BIT 0) to logic-level high, or 3) CPU execution of a SET BIT TO ONE (SBO) instruction. Note that both $\overline{\text{RESET}}$ and RST affect all sixteen single-bit I/O ports while CPU execution of either an SBO or SBZ instruction can be targeted at an individual single-bit port independent of uninvolved ports. Once the data flow direction has been established for each single-bit port, CPU communication with the external devices driven or sensed by each individual port is effected through execution of the CRU instructions: LDCR, STCR, SBO, SBZ, and TM.

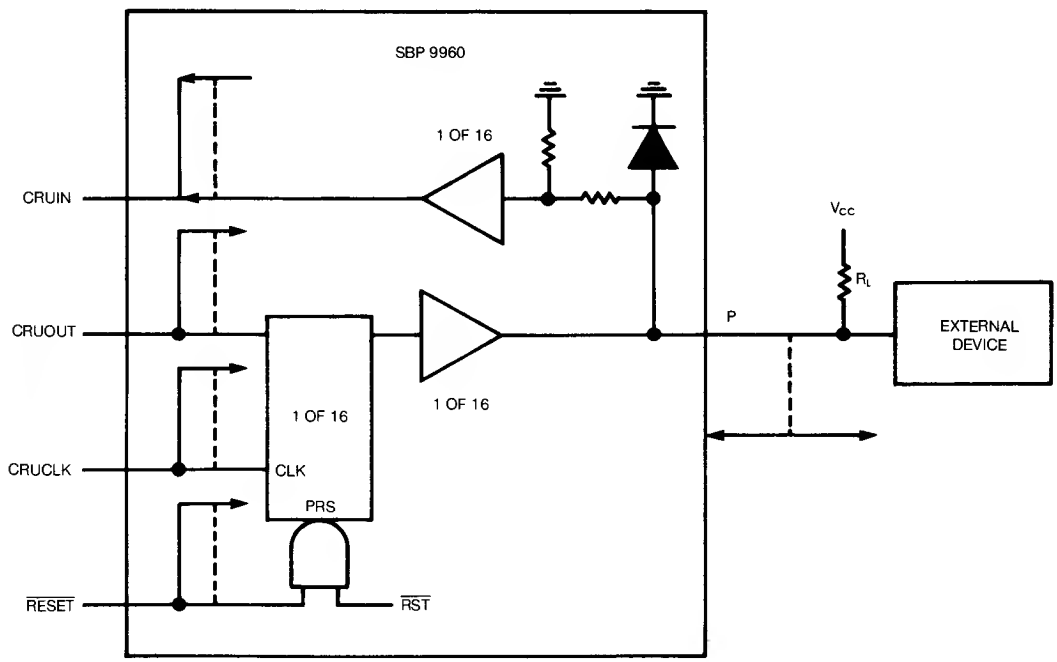


FIGURE 2. 1 of 16 Single-Bit I/O Ports

*Outputs P0, P1, P2, and P3 have extended current sink capability to 40 mA

Table 1. SBP 9960 CRU Bit Assignments

CRU BIT	S0	S1	S2	S3	S4	CRU READ DATA	CRU WRITE DATA
0	0	0	0	0	0	Control Bit	Control Bit
1-14						Note 1	Note 1
15	0	1	1	1	1	"1"	No Operation/ $\overline{RST}$ (2)
16	1	0	0	0	0	P0 Input(3)	P0 Output(4) (5)
17	1	0	0	0	1	P1	P1 (5)
18	1	0	0	1	0	P2	P2 (5)
19	1	0	0	1	1	P3	P3 (5)
20	1	0	1	0	0	P4	P4
21	1	0	1	0	1	P5	P5
22	1	0	1	1	0	P6	P6
23	1	0	1	1	1	P7	P7
24	1	1	0	0	0	P8	P8
25	1	1	0	0	1	P9	P9
26	1	1	0	1	0	P10	P10
27	1	1	0	1	1	P11	P11
28	1	1	1	0	0	P12	P12
29	1	1	1	0	1	P13	P13
30	1	1	1	1	0	P14	P14
31	1	1	1	1	1	P15 Input(3)	P15 Output(4)

NOTES:
(1) Bits 1-14 reserved for SBP 9961 Intervel Timer/Interrupt Controller
(2) Writing a zero tp bit 15 while CONTROL = 1 executes a software reset of the I/O ports.
(3) Data present on the port will be read without affecting the data.
(4) Writing data to the port will both program the port to the output mode end output the data.
(5) These outputs are provided with extended sink-current cepability to 40 mA.

System Operation

During a typical power-up sequence of a SBP 9960-based system, $\overline{RESET}$ should be activated (logic-level low) to force the SBP 9960 to the state where each of the sixteen individual single-bit I/O ports is in the input mode. System software should then configure each single-bit port as required. If a given port must be reconfigured from the input to output mode after power-up, the associated output-register bit must be set to logic-level high through CPU execution of an SBO instruction.

▶8

SBP 9960/SBP 9961 EMULATION OF THE TMS 9901

Figure 3 shows the system configuration of a SBP 9960 functioning in conjunction with a SBP 9961 in emulation of a TMS 9901. Note the common connection of: a) the individual chip enables, and b) the CRU interface lines. For a complete description of the SBP 9961 and the TMS 9901 refer respectively to the *SBP 9961 Interrupt-Controller/Timer Data Manual* and the *TMS 9901 Programmable Systems Interface Data Manual.*

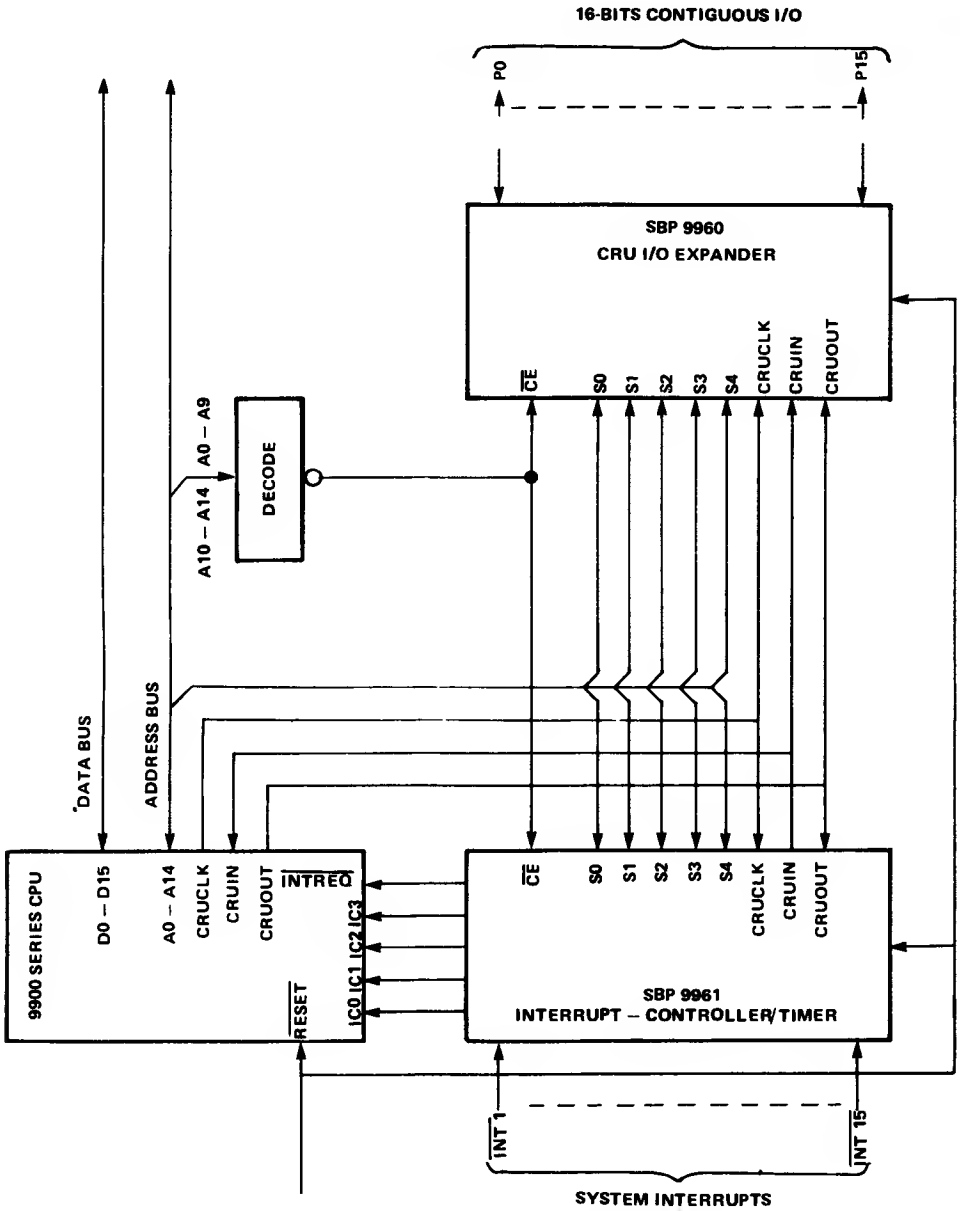


Figure 3. SBP 9960/SBP9961 System Configuration

ELECTRICAL SPECIFICATIONS

Recommended Operating Conditions, Unless Otherwise Noted $I_{cc} = 70$ mA

PARAMETER		MIN	NOM	MAX	UNIT
Supply current, I_{CC}		63	70	77	mA
High-level output voltage, V_{OH}				5.5	V
Low-level output current, I_{OL}				20†	mA
Operating free-air temperature, T_A	SBP 9960MJ, SBP 9960NJ	−55		125	°C
	SBP 9960EJ	−40		85	
	SBP 9960CJ	0		70	

†40 mA on extended drive outputs P0, P1, P2, and P3

Electrical Characteristics (Over Recommended Operating Free-Air Temperature Range, Unless Otherwise Noted)

PARAMETER		TEST CONDITIONS†		MIN	TYP‡	MAX	UNIT
V_{IH}	High-level input voltage			2			V
V_{IL}	Low-level input voltage					0.8	V
V_{IK}	Input clamp voltage	I_{CC} = MIN,	I_I = −12 mA			−1.5	V
I_{OH}	High-level output current	I_{CC} = 70 mA, V_{IL} = 0.8 V,	V_{IH} = 2 V, V_{OH} = 5.5 V			400	μA
V_{OL}	Low-level output voltage	I_{CC} = 70 mA, V_{IL} = 0.8 V,	V_{IH} = 2 V, I_{OL} = 20 mA (40 mA§)			0.5	V
I_I	Input current	I_{CC} = 70 mA,	V_I = 2.4 V		225		μA

†For conditions shown as MAX, use the appropriate value specified under recommended operating conditions.
‡All typical values are at I_{CC} = 70 mA, T_A = 25°C.
§Extended drive outputs only.

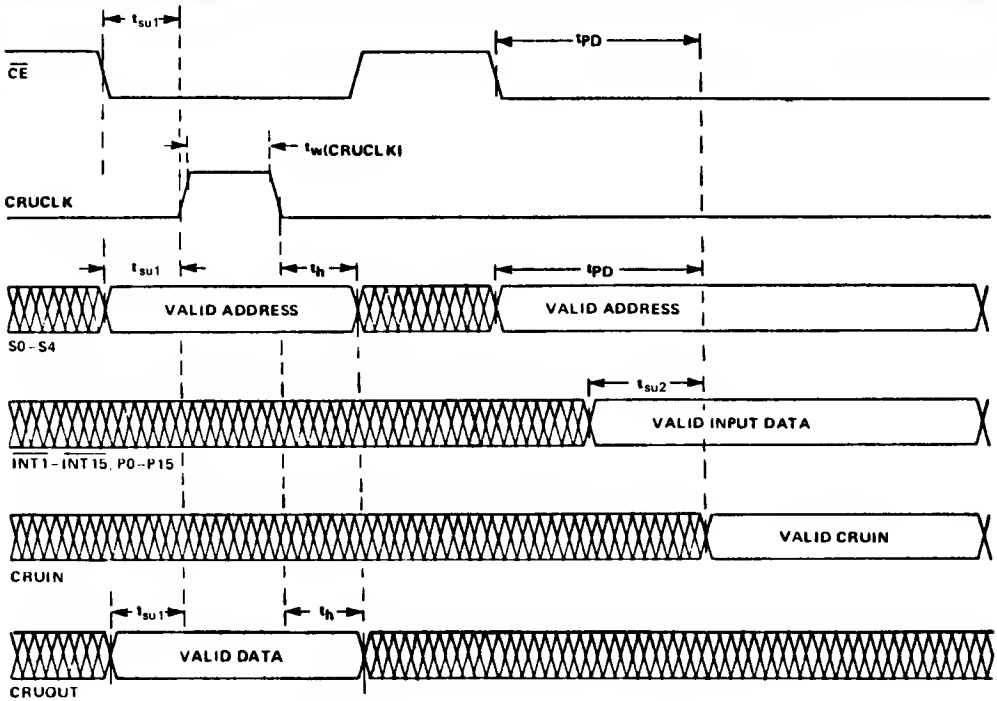
Timing Requirements Over Full Range of Operating Conditions

PARAMETER		MIN	NOM	MAX	UNIT
t_{su1}	Setup time for S0-S4, $\overline{CE}$, or CRUOUT before CRUCLK		200		ns
t_{su2}	Setup time, input before valid CRUIN		200		ns
$t_{w(CRUCLK)}$	CRU clock pulse width		100		ns
t_h	Hold time for Address or Data		0		ns

Switching characteristics Over Full Range of Recommended Operating Conditions

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PD}	Propagation delay, S0-S4 or $\overline{CE}$ to valid CRUIN	C_L = 100 pF, R_L = 300 Ω		300		ns

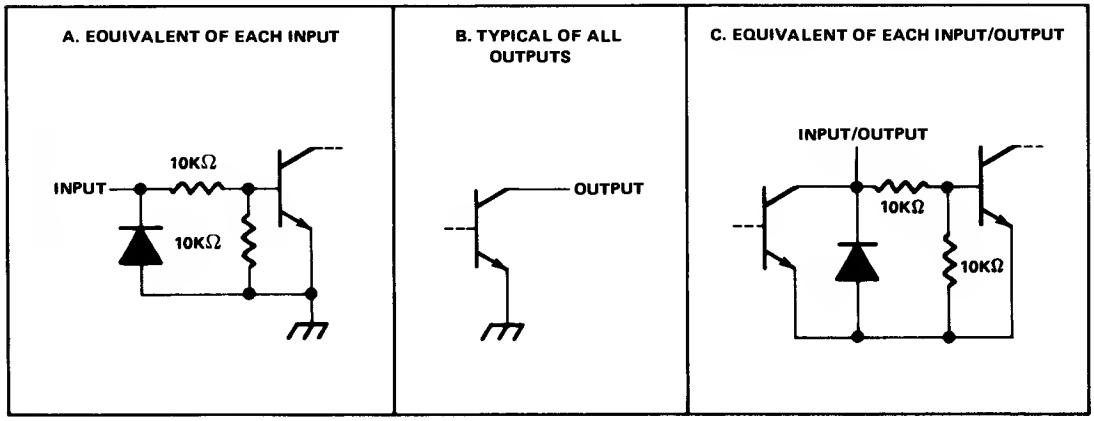
▶8



NOTE 1: ALL TIMING MEASUREMENTS ARE FROM 10% and 90% POINTS

SWITCHING CHARACTERISTICS

INPUT, OUTPUT, AND INPUT/OUTPUT STRUCTURES



8◄

SBP 9961
INTERRUPT-CONTROLLER/TIMER

990/9900 FAMILY MICROCOMPUTER COMPONENTS

- SBP/TMS 9900 Series Microprocessor Family Peripheral
- 15 Dedicated, Maskable, Prioritized, Encoded Interrupts
- 20 mA Current Sinking Outputs
- 40-Pin Package
- Independently Clocked 14-Bit Interval/Event Timer
- Software Compatible with TMS 9901 when used in conjunction with SBP 9960
- TTL Compatible I/O
- Wide Ambient Temperature Operation
  - SBP 9961CJ: 0°C to +70°C
  - SBP 9961EJ: -40°C to +85°C
  - SBP 9961MJ: -55°C to +125°C
  - SBP 9961NJ: -55°C to +125°C (with high-reliability processing)
- I^2L Technology
  - Constant Current Power Source
  - Fully Static Operation
  - Single Phase Edge-Triggering Clock
  - Wide Temperature Stability

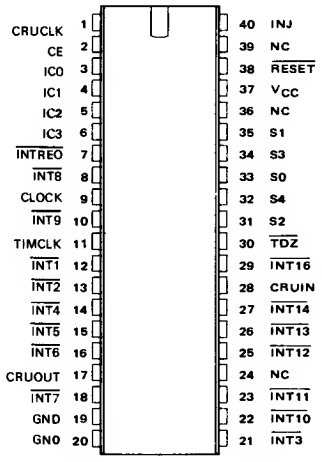
DESCRIPTION

The SBP 9961 Interrupt-Controller/Timer is a ruggedized, monothlithic, Communications Register Unit (CRU) programmable, multifunction systems support device fabricated with oxide separated Integrated Injection Logic (I^2L) technology. The SBP 9961 provides the SBP/TMS 9900 series Family of Microprocessors with a flexible independently clocked interval/event timer plus maskable prioritized interrupt encoding capability. I^2L technology enables the SBP 9961s static logic, and TTL compatible I/O, to operate over a very wide ambient temperature range from a single d-c power source. When the SBP 9961 is used in conjunction with the I^2L SBP 9960 CRU I/O Expander, the SBP 9961/SBP 9960 pair form an I^2L systems alternate to the N-channel MOS TMS 9901 Programmable Systems Interface device while maintaining strict compatibility with existing software handlers developed in support of the TMS 9901.

SBP 9961 PIN ASSIGNMENTS AND FUNCTIONS

Signature	*Pin*	*I/O*	*Description*
S0	33	IN	ADDRESS SELECT LINES. The data bit being accessed by the CRU interface is specified by the 4-bit code appearing on S1-S4. S0 is used as the high order select line when the SBP 9961 is used with the SBP 9960 in emulation of the TMS 9901. Otherwise, tie S0 to logic-level low.
S1	35	IN	
S2	31	IN	
S3	34	IN	
S4	32	IN	
CRUIN	28	OUT	CRU DATA IN (to CPU). Data specified by S0-S4 is transmitted to the CPU by CRUIN. When $\overline{CE}$ is not active, CRUIN is logic-level high.
CRUOUT	17	IN	CRU DATA OUT (from CPU). When $\overline{CE}$ is active, data present on the CRUOUT input will be sampled during CRUCLK and written into the CRU bit specified by S0-S4.
CRUCLK	1	IN	CRU CLOCK (from CPU). CRUCLK specifies that valid data is present on the CRUOUT line.
$\overline{RESET}$	38	IN	POWER-UP RESET. When active (low), $\overline{RESET}$ forces all interrupt masks to "0", and disables the clock.
$\overline{CE}$	2	IN	CHIP ENABLE. When active (low), data transfers may occur between the CPU and the SBP 9961.

SBP 9961
INTERRUPT-CONTROLLER/TIMER



Signature	*Pin*	*I/O*	*Description*
TIMCLK	11	IN	TIMER CLOCK IN. External clock used for the timer decrementer. May be externally tied to the CLOCK input pin.
$\overline{\text{TDZ}}$	30	OUT	TIMER DECREMENTER EQUALS ZERO. Low active pulse indicating that the timers decrementer contains a value of zero (all logic-level lows).
IC0	3	OUT	INTERRUPT CODE LINES (to CPU). IC0 (MSB) through IC3 output the binary code corresponding to the highest priority enabled interrupt most recently asserted.
IC1	4	OUT	
IC2	5	OUT	
IC3	6	OUT	
$\overline{\text{INTREQ}}$	7	OUT	INTERRUPT REQUEST (to CPU). When active (low), $\overline{\text{INTREQ}}$ indicates to the CPU that an enabled interrupt has been asserted, prioritized and encoded.
CLOCK	9	IN	CPU SYSTEM CLOCK. Used by the SBP 9961 to synchronize the interrupt interface ($\overline{\text{INTREQ}}$, IC0-IC3) to the CPU.
$\overline{\text{INJ}}$	40		Supply Current
$\overline{\text{GND}}$	19,20		Ground Reference
$\overline{\text{Vcc}}$	37		Common voltage return/reference for all I/O pull-up resistors.
$\overline{\text{INT1}}$	12	IN	INTERRUPT INPUTS. When active (low), the signal is ANDed with its corresponding mask bit and if enabled sent to the interrupt control section. $\overline{\text{INT1}}$ has highest priority.
$\overline{\text{INT2}}$	13	IN	
$\overline{\text{INT3}}$	21	IN	
$\overline{\text{INT4}}$	14	IN	
$\overline{\text{INT5}}$	15	IN	
$\overline{\text{INT6}}$	16	IN	
$\overline{\text{INT7}}$	18	IN	
$\overline{\text{INT8}}$	8	IN	
$\overline{\text{INT9}}$	10	IN	
$\overline{\text{INT10}}$	22	IN	
$\overline{\text{INT11}}$	23	IN	
$\overline{\text{INT12}}$	25	IN	
$\overline{\text{INT13}}$	26	IN	
$\overline{\text{INT14}}$	27	IN	
$\overline{\text{INT15}}$	29	IN	

8

FUNCTIONAL DESCRIPTION

SBP 9961/CPU Interface

The SBP 9961 communicates with the CPU through the Communications Register Unit (CRU) interface as shown in Figures 1 and 4. The SBP 9961s CRU interface consists of: a) five CRU address select lines (S0-S4), b) a single chip enable ($\overline{CE}$), c) a 9961 to CPU serial data-bit line (CRUIN), d) a CPU to 9961 serial data-bit line (CRUOUT), and e) a CPU to 9961 serial data-bit clock (CRUCLK). When $\overline{CE}$ is activated (logic-level low), S0-S4 selects a specific CRU-bit function as indicated in Table 1. In the case of a SBP 9961 write operation, the datum is transferred from the CPU to the SBP 9961 via the CRUOUT line. The CRUOUT datum is strobed into the selected 9961 CRU-bit function by CRUCLK. In the case of a SBP 9961 read operation, the selected CRU-bit function is sampled by the CPU via the CRUIN line.

Table 1. CRU Bit Assignments

CRU BIT	S0	S1	S2	S3	S4	CRU READ DATA	CRU WRITE DATA
0	0(4)	0	0	0	0	Control Bit	Control Bit(1)
1	0	0	0	0	1	$\overline{INT1}$/TIM1(2)	Mask1/TIM1(3)
2	0	0	0	1	0	$\overline{INT2}$/TIM2	Mask2/TIM2
3	0	0	0	1	1	$\overline{INT3}$/TIM3	Mask3/TIM3
4	0	0	1	0	0	$\overline{INT4}$/TIM4	Mask4/TIM4
5	0	0	1	0	1	$\overline{INT5}$/TIM5	Mask5/TIM5
6	0	0	1	1	0	$\overline{INT6}$/TIM6	Mask6/TIM6
7	0	0	1	1	1	$\overline{INT7}$/TIM7	Mask7/TIM7
8	0	1	0	0	0	$\overline{INT8}$/TIM8	Mask8/TIM8
9	0	1	0	0	1	$\overline{INT9}$/TIM9	Mask9/TIM9
10	0	1	0	1	0	$\overline{INT10}$/TIM10	Mask10/TIM10
11	0	1	0	1	1	$\overline{INT11}$/TIM11	Mask11/TIM11
12	0	1	1	0	0	$\overline{INT12}$/TIM12	Mask12/TIM12
13	0	1	1	0	1	$\overline{INT13}$/TIM13	Mask13/TIM13
14	0	1	1	1	1	$\overline{INT14}$/TIM14	Mask14/TIM14
15	0	1	1	1	1	$\overline{INT15}$/$\overline{INTREQ}$	Mask 15

NOTES:
(1) 0 = Interrupt Mode; 1 = TIMCLK Mode.
(2) Data present on $\overline{INT}$ input (or timer value) will be read regardless of mask value.
(3) While in the Interrupt Mode (Control Bit = 0), writing a "1" into a mask will enable interrupt, "0" will disable.
(4) When the SBP 9961/SBP 9960 pair are used in emulation of the TMS 9901, S0 is used to distinguish between activation of the 9961 (S0 = 0) v.s. the 9960 (S0 = 1).

Interrupt Control

▶8 A block diagram of the SBP 9961s interrupt control section is shown in Figure 2. The interrupt inputs are sampled on the positive-going edge of CLOCK and are ANDed with their respective mask bits. If an interrupt input is active (low) and enabled (MASK = 1), the signal is passed through the priority encoder where the highest priority signal is encoded into a 4-bit binary word as shown in Table 2. This word, along with an interrupt request, is then output to the CPU on the positive-going edge of the next CLOCK.

The output signals will remain valid until either the corresponding interrupt input is removed, the interrupt is disabled (MASK = 0), or a higher priority enabled interrupt becomes active. When the highest priority enabled interrupt is removed, the code corresponding to the next highest priority enabled interrupt is output. If no enabled interrupt is active, $\overline{INTREQ}$ will be pulled to logic-level high with IC0-IC3 retaining the last asserted interrupt code. $\overline{RESET}$ (power-up reset) will force the interrupt code IC0-IC3 to (0,0,0,0) with $\overline{INTREQ}$ pulled high, and will reset all mask bits low (interrupts disabled). Individual interrupts can be subsequently enabled (disabled) by programming the appropriate mask bits. Unused interrupt inputs may be used as data inputs by disabling the interrupt (MASK = 0).

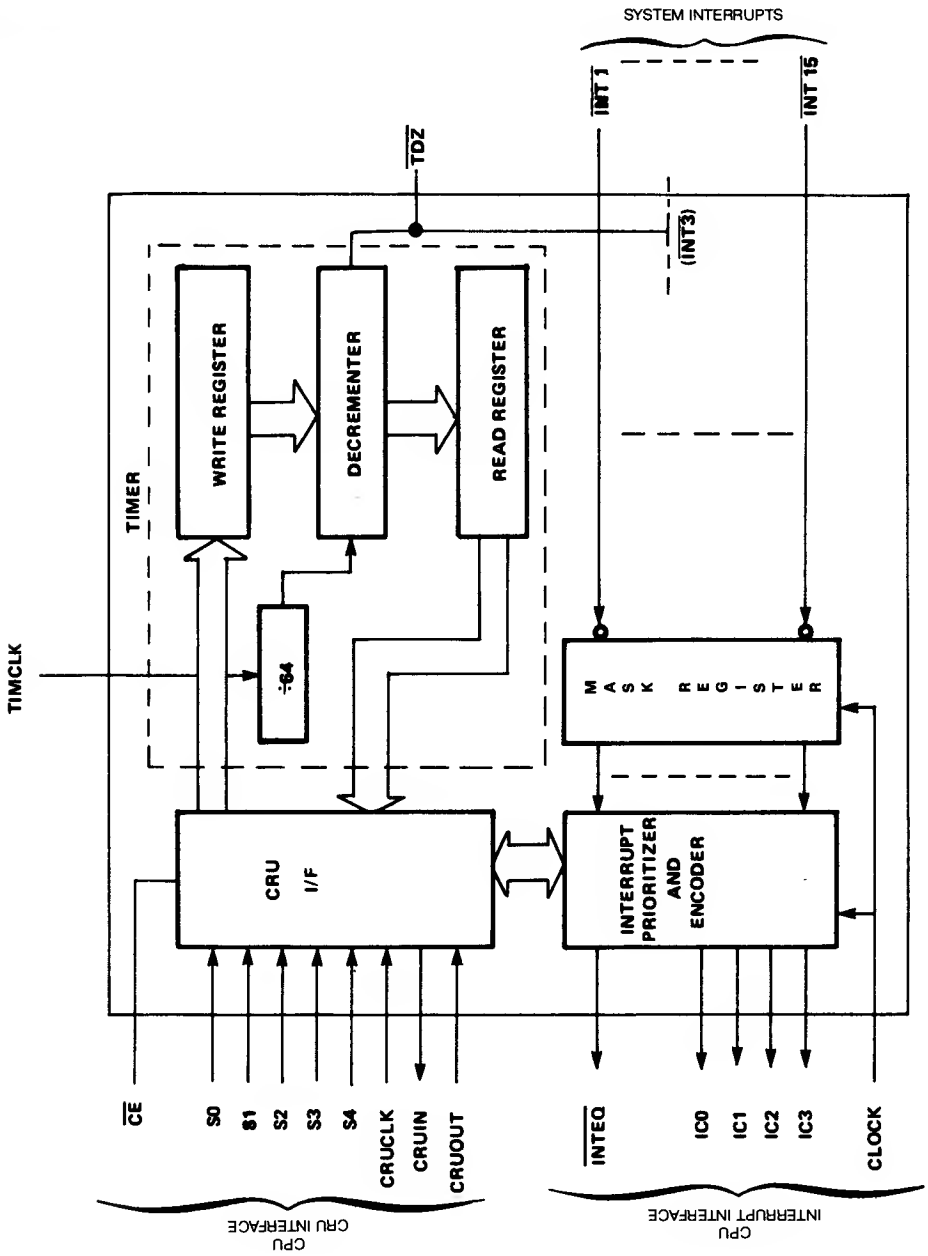


Figure 1. SBP 9961 Block Diagram

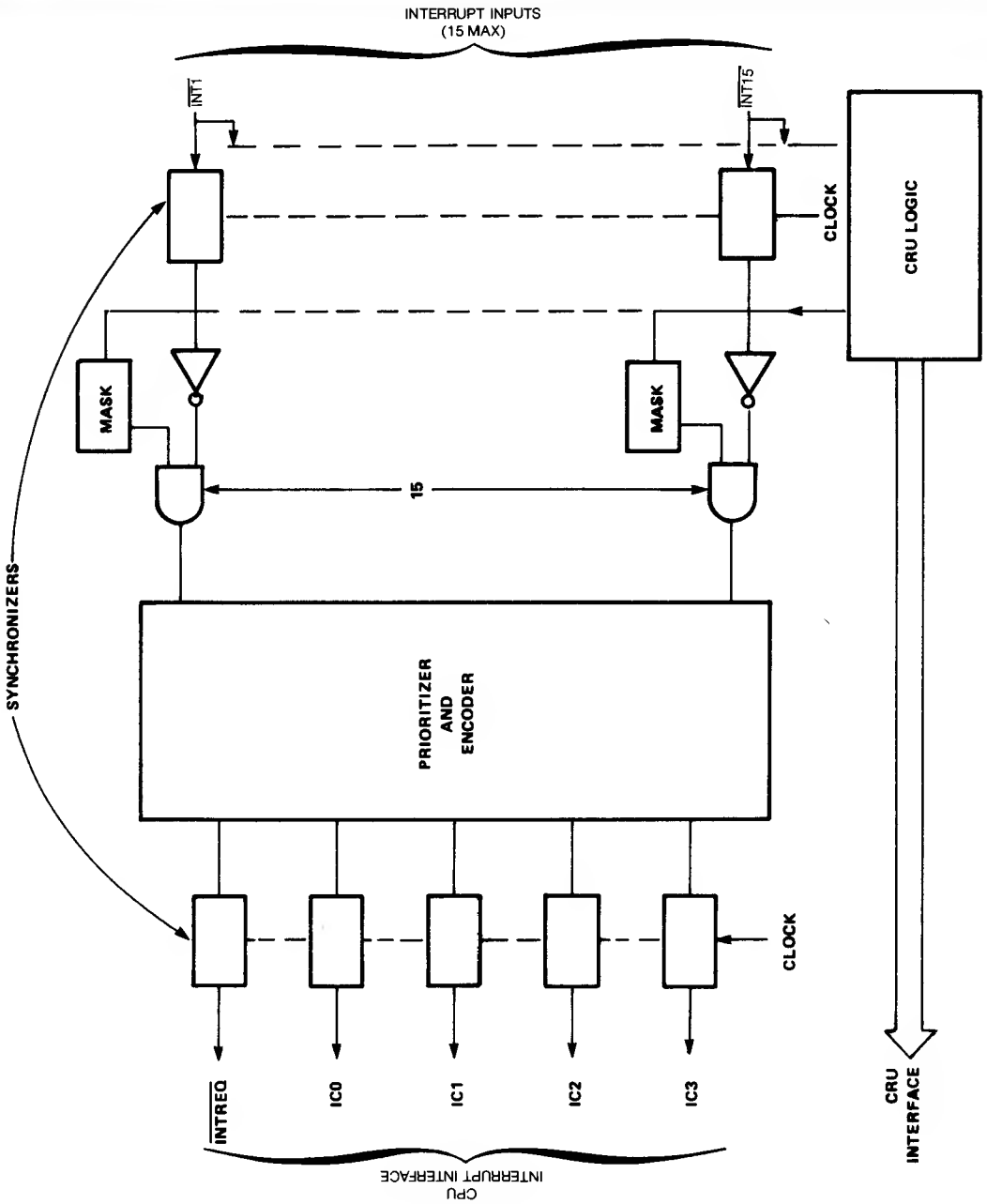


Figure 2. Interrupt Control Logic

Table 2. Interrupt Code Generation

INTERRUPT/STATE	PRIORITY	I_{C0}	I_{C1}	I_{C2}	I_{C3}	$\overline{INTREQ}$
$\overline{INT}$ 1	1 (HIGHEST)	0	0	0	1	0
$\overline{INT}$ 2	2	0	0	1	0	0
$\overline{INT}$ 3/TIMER	3	0	0	1	1	0
$\overline{INT}$ 4	4	0	1	0	0	0
$\overline{INT}$ 5	5	0	1	0	1	0
$\overline{INT}$ 6	6	0	1	1	0	0
$\overline{INT}$ 7	7	0	1	1	1	0
$\overline{INT}$ 8	8	1	0	0	0	0
$\overline{INT}$ 9	9	1	0	0	1	0
$\overline{INT}$ 10	10	1	0	1	0	0
$\overline{INT}$ 11	11	1	0	1	1	0
$\overline{INT}$ 12	12	1	1	0	0	0
$\overline{INT}$ 13	13	1	1	0	1	0
$\overline{INT}$ 14	14	1	1	1	0	0
$\overline{INT}$ 15	15 (LOWEST)	1	1	1	1	0
NO INTERRUPT	—	Note 1	Note 1	Note 1	Note 1	1

(1) IC0-IC3 hold the level code of the previous interrupt

Interval Timer

The SBP 9961s interval/event timer, shown in Figure 3, has the following operational features:

a) Independent clock input TIMCLK
b) Programmable 14-bit decrementer
c) Time-reaches-zero level-3 interrupt
d) Timer reaches zero output pulse $\overline{TDZ}$
e) Able to read the current decremented value and therefore function as an event timer
f) Able to determine the SBP 9961s operating mode and value of $\overline{INTREQ}$.

The SBP 9961 has an independent timer clock input, TIMCLK, which allows the user to define an interval timer clock frequency other than that of the CPU. This, however, does not preclude the user option of connecting TIMCLK to the CPUs CLOCK input and therefore running the interval timer at the CPUs clock frequency. The typical operating range of TIMCLK is 0-5 MHz.

The timer's CRU control bits are shown in Table 1. The SBP 9961 is placed into the timer-access mode by writing a logic-level high to the control bit located at CRU address zero. CRU bits 1-14 are then used to initiate the write-register with the desired start count. Writing a non-zero value to the write-register a) enables the decrementer, b) programs the third priority interrupt ($\overline{INT3}$) as the timer interrupt, and c) disables the influence of external interrupts on the $\overline{INT3}$ input pin. A single LDCR instruction can be used to accomplish the above initialization operation. After the write-register has been initialized with the desired start count, the timer begins decrementing toward zero. Upon reaching zero, the timer issues the level-3 interrupt, outputs the timer-zero pulse $\overline{TDZ}$, and restarts itself with the write-register value. Since the timer interrupt is latched, clearing that interrupt is accomplished by writing either a logic-level low or high to the respective interrupt mask bit at CRU address three. The CRUCLK that accompanies that write operation is the stimulus which resets the timer's interrupt latch. However, in order to retain the current mask value, the appropriate SBZ or SBO CRU-write instruction must be executed unless the mask value itself is to be changed. At any point in the timer's decrement sequence, a timer restart can be accomplished by either reinitiating the entire write-register with an LDCR instruction, or by writing to any individual write-register bit with an SBZ or SBO instruction.

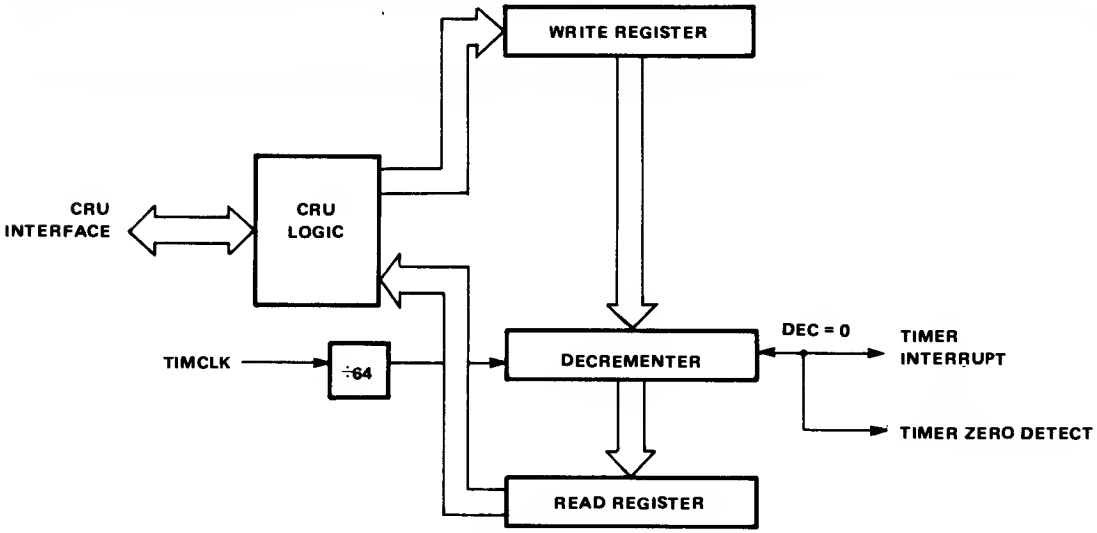


Figure 3. Interval/Event Timer

If the control bit is at logic-level low, the timer's read-register is updated with the current decrementer value after each decrement operation (once every 64 TIMCLK clocks); if the control bit is at logic-level high (timer-access mode), the read-register retains its current value therby ensuring that the read-register is not changed in the event a CRU read operation is executing during a decrement operation. Consequently, the current value of the timer's decrementer can be interrogated by 1) placing the SBP 9961 into the timer-access mode, and 2) performing a CRU-read operation on the timer's read-register through execution of an STCR instruction. The timer, then, can function as an event timer by reading the elapsed time between software events as shown in Table 3. Note that when accessing the timer, all interrupts should be disabled. The timer is disabled by either $\overline{\text{RESET}}$ (power-up reset) or by writing all zeroes to the write-register.

SBP 9961 Status

The SBP 9961s status can be determined by reading the value of the control bit located at CRU address zero. If the control bit has a logic-level low value, then the interrupt masks may be changed and data on the interrupt inputs may be read. However, access to the interval timer is inhibited. If the control bit has a logic-level high value, then the timer may be initiated, restarted, or read. Also, reading CRU address fifteen gives the status of INTREQ where logic-level low indicates activation.

Table 3. Software Examples

ASSUMPTIONS:

- Total of 6 interrupts are used
- RESET has been applied
- System uses timer at maximum interval

```
SYSTEM   LI      R12, CRUBAS     Setup CRU Base Address to point to 9961
SETUP    LDCR    @X,0            Program Timer with maximum interval
         LDCR    @Y,7            Re-enter interrupt mode and enable top 6 interrupts
         (X)---------->FFFF
         (Y)---------->7FXX
         .
         .
         .
         .
         .
         BLWP    CLKVCT          Save Interrupt Mask
         .
         .
         .
CLKPC    LIMI    0               Disable Interrupts
         LI      R12, CRUBASE+1  Set up CRU base
         .
         .
         SBO     -1              Set 9961 into timer-access mode
         STCR    R4, 14          Store read register into R4
         .
         .                       Process Timer Value
         .
         .
         SBZ     -1              Re-enter Interrupt Mode (i.e., Exit Timer-Access Mode)
         RTWP                    Restore Interrupt Mask
         .
         .
         .
CLKUCT   DATA    CLKWP,CLKPC
```

System Operation

During power-up, $\overline{\text{RESET}}$ should be activated (low) to force the SBP 9961 into a known state. $\overline{\text{RESET}}$ will disable all interrupts, disable the timer, and force IC0-IC3 to (0,0,0,0) with $\overline{\text{INTREQ}}$ pulled high. System software should then enable the proper interrupts and program the timer (if used). (See Table 3 for an example.) After initial power up, the SBP 9961 is accessed only as needed to service the timer and enable or disable interrupts.

Figure 4 shows SBP 9961s system configuration. Figure 5 shows the use of a SBP 9961 with a SBP 9960 CRU I/O expander in emulation of the TMS 9901. (See *TMS 9901 Systems Interface Data Manual.*)

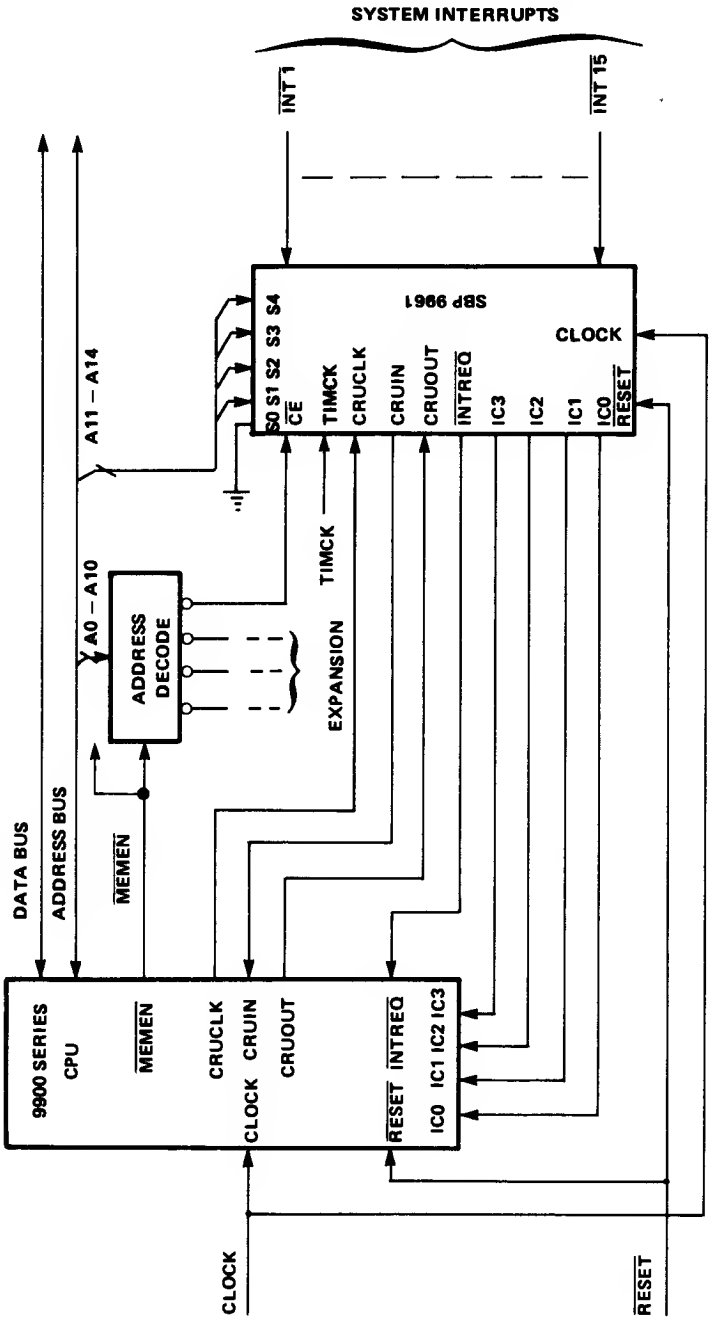


Figure 4. SBP 9961 System Configuration

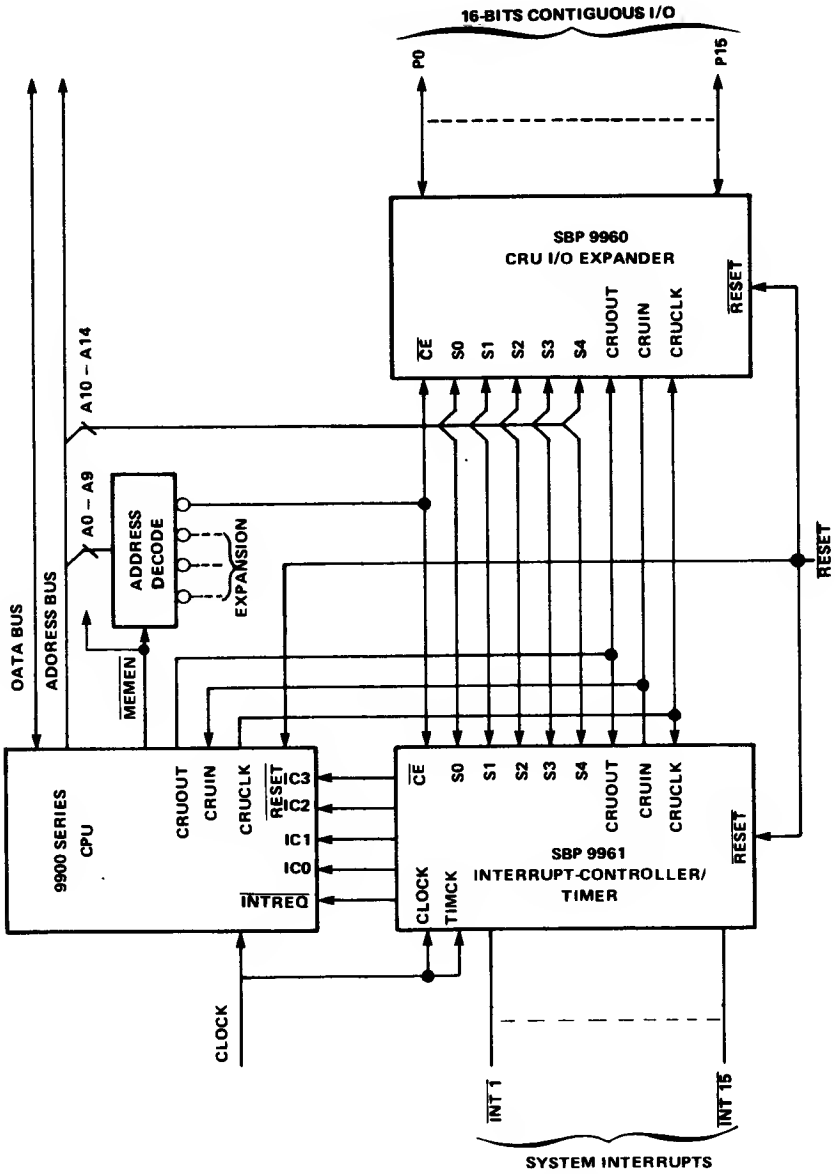


Figure 5. SBP 9961 Configuration with SBP 9960

ELECTRICAL SPECIFICATIONS

Recommended Operating Conditions, Unless Otherwise Noted $I_{CC} = 130mA$

PARAMETER		MIN	NOM	MAX	UNIT
Supply current, I_{CC}		115	130	145	mA
High-level output voltage, V_{OH}				5.5	V
Low-level output current, I_{OL}				20	mA
Operating free-air temperature, T_A	SBP 9961MJ, SBP 9961NJ	−55		125	°C
	SBP 9961EJ	−40		85	
	SBP 9961CJ	0		70	

Electrical Characteristics (Over Recommended Operating Free-Air Temperature Range, Unless Otherwise Noted)

PARAMETER		TEST CONDITIONS†		MIN	TYP‡	MAX	UNIT
V_{IH}	High-level input voltage			2			V
V_{IL}	Low-level input voltage					0.8	V
V_{IK}	Input clemp voltage	I_{CC} = MIN,	I_I = −12 mA			−1.5	V
I_{OH}	High-level output current	I_{CC} = 130 mA, V_{IL} = 0.8 V,	V_{IH} = 2 V, V_{OH} = 5.5 V			400	μA
V_{OL}	Low-level output voltage	I_{CC} = 130 mA, V_{IL} = 0.8 V,	V_{IH} = 2 V, I_{OL} = 20 mA			0.5	V
I_I	Input current	I_{CC} = 130 mA,	V_I = 2.4 V		180		μA

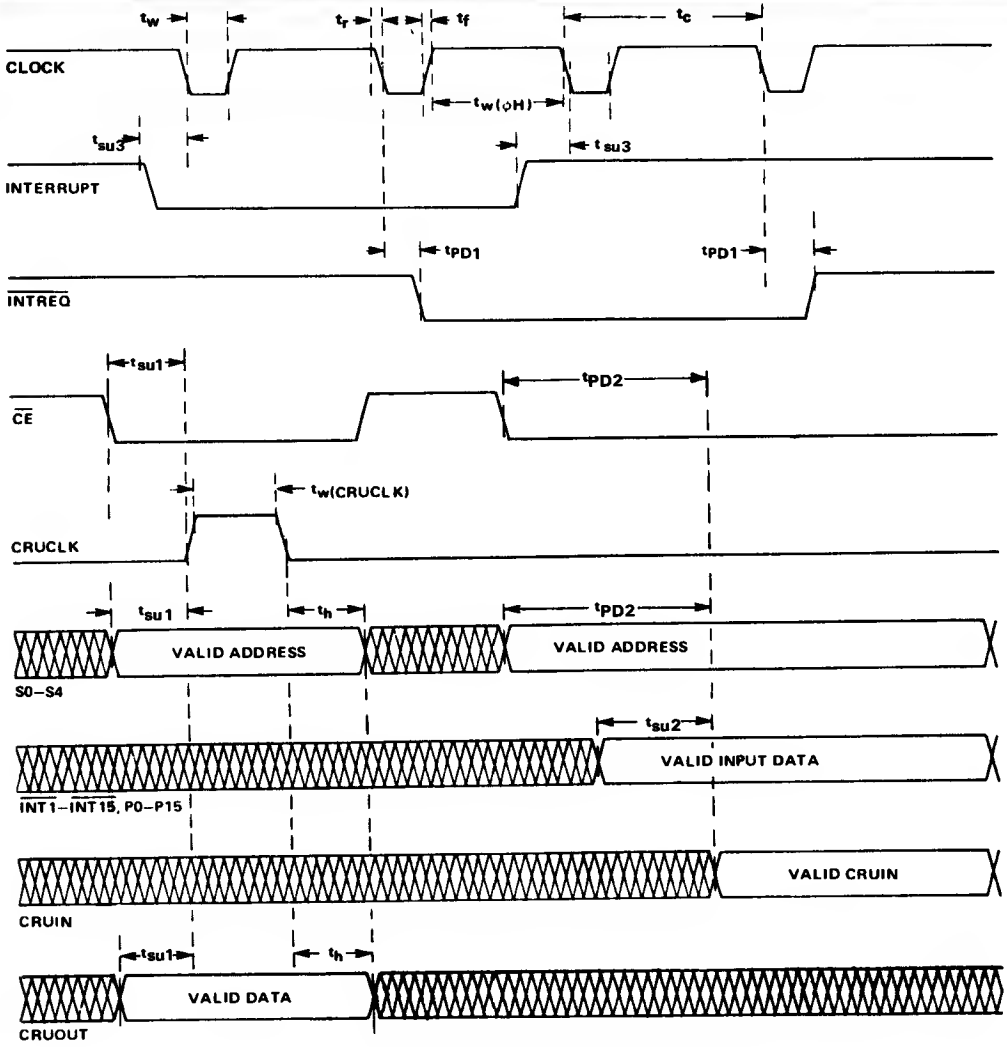
† For conditions shown as MAX, use the appropriate value specified under recommended operating conditions
‡ All typical values are at I_{CC} = 130 mA, T_A = 25°C.

Timing Requirements Over Full Range of Operating Conditions

PARAMETER		MIN	NOM	MAX	UNIT
t_c	Clock cycle time	333			ns
t_r	Clock rise time		10	20	ns
t_f	Clock fall time		10	20	ns
t_{wL}	Clock pulse low width	111			ns
t_{wH}	Clock pulse high width	222			ns
t_{su1}	Setup time for S0-S4, $\overline{CE}$, or CRUOUT before CRUCLK		200		ns
t_{su2}	Setup time, input before valid CRUIN		200		ns
t_{su3}	Setup time, interrupt before clock high		60		ns
$t_{w(CRUCLK)}$	CRU clock pulse width		100		ns
t_h	Address hold time		80		ns
t_{TC}	TIMCLK cycle time		200		ns

Switching Characteristics Over Full Range of Recommended Operating Conditions

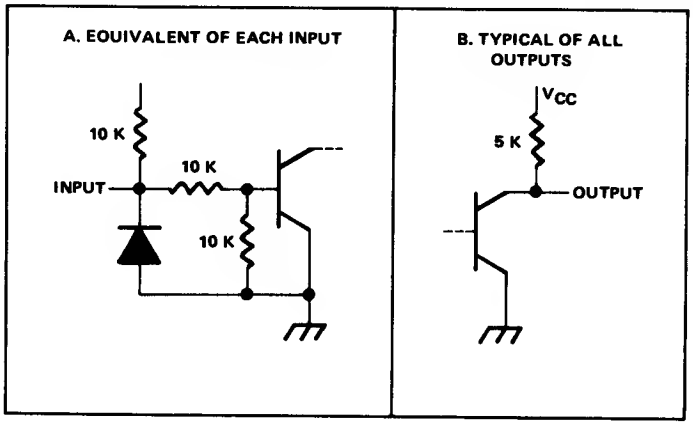
PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PD1}	Propagation delay, ↑ CLOCK to valid $\overline{INTREQ}$, IC0-IC3	C_L = 25 pF, R_L = 5K Ω		150		ns
t_{PD2}	Propagation delay, S0-S4 or $\overline{CE}$ to valid CRUIN	C_L = 25 pF, R_L = 5K Ω		330		ns



NOTE 1: ALL TIMING MEASUREMENTS ARE FROM 10% and 90% POINTS

Switching Characteristics

INPUT AND OUTPUT STRUCTURES



SBP 9964
TIMING CONTROLLER for the SBP 9900A

990/9900 FAMILY MICROCOMPUTER COMPONENTS

- 14-Bit Interval Timer-Event Counter
- $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ Synchronization
- SBP 9900A Clock Generation
- 20-Pin Package
- TTL Compatible Open-Collector I/O

Description

The SBP 9964 is an SBP 9900A peripheral support device which performs general timing and synchronization functions usually implemented with SSI TTL packages.

Internal to the SBP 9964 is a 14-bit interval timer-event counter, an SBP 9900A clock generator and an SBP 9900A $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ signal synchronizer. The interval timer-event counter communicates with the SBP 9900A through the SBP 9900A's Communication Register Unit (CRU) I/O interface. The interval timer-event counter may be efficiently applied to a variety of applications in which the interval between external events, the number of external events, or the initiation of periodic events is desired. $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ synchronizers provide for SBP 9900A compatible synchronization of these signals from asynchronously applied external signals.

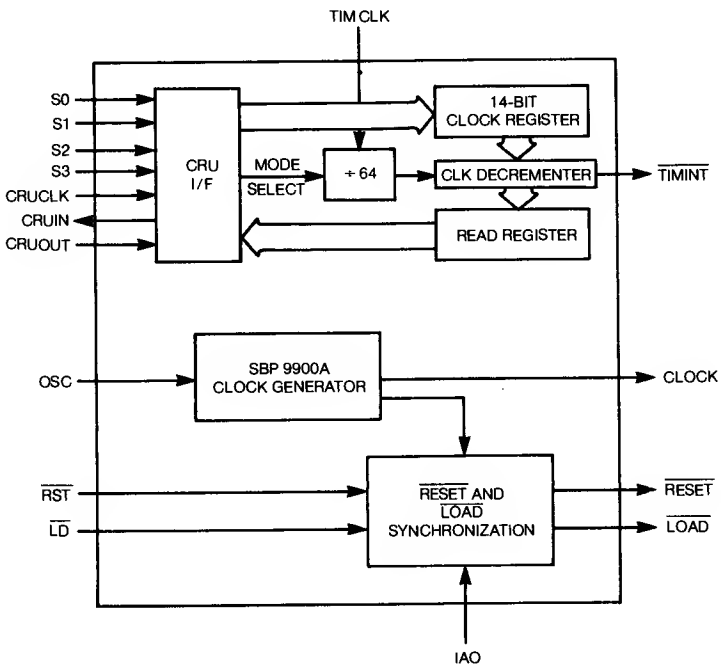


Figure 1. Functional Block Diagram

DESIGN GOAL

This document describes the design specifications for a product under development. Texas Instruments reserves the right to change these specifications in any manner, without notice.

SBP 9965
PERIPHERAL INTERFACE ADAPTER

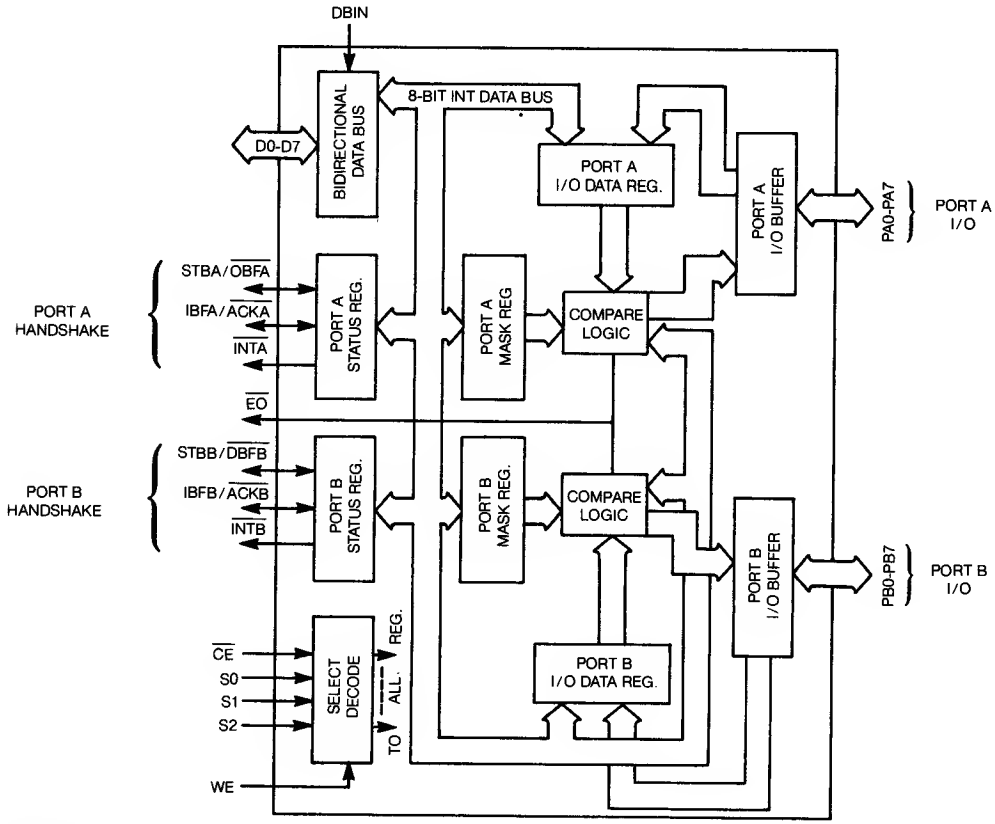
990/9900 FAMILY MICROCOMPUTER COMPONENTS

- Microprocessor Memory-Mapped I/O Peripheral Interface
- Dual 8-Bit Input/Output Peripheral Ports
- Internal Mask Registers and Associated Compare Logic for Character/Data Recognition
- TTL Compatible Open-Collector I/O
- 40-Pin Package

Description

The SBP 9965 Peripheral Interface Adapter is a byte oriented, parallel memory-mapped, input/output interface which interfaces to microprocessor CPU's through the memory bus. Two 8-bit I/O ports with independent handshake lines are provided which allow a variety of byte oriented peripheral devices to be efficiently interfaced to the CPU. High data rates are effected through parallel transfers of data between the CPU and the peripheral device.

Two internal mask registers, one associated with each I/O port, may compare logic which flags the CPU whenever an equal condition exists between I/O and mask register data. This feature is useful for byte string searches or control character recognition.



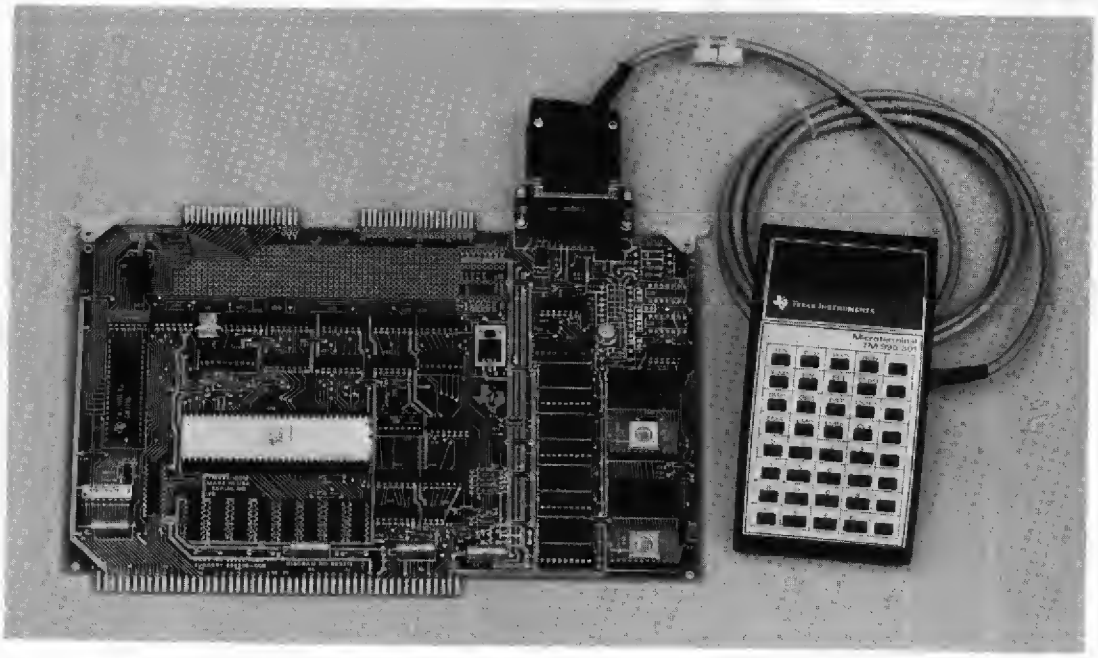
DESIGN GOAL

This document describes the design specifications for a product under development. Texas Instruments reserves the right to change these specifications in any manner, without notice.

Figure 1. Function Block Diagram

TM 990 Series
Microcomputer Modules

8



TM 990/100M Microcomputer Module
with TM 990/301 Microterminal

GENERAL

The TM 990 microcomputer modules are a series of low cost TMS 9900 family microcomputers, assembled on a single printed circuit board. The TMS 9900 16-bit NMOS microcomputer and the TMS 9980, a software compatible 8-bit data bus microprocessor, will be the CPU's for the initial board systems. The TM 990 microcomputers offer a new level of hardware capability incorporating all of the powerful TMS 9900 LSI components. Whether a single CPU unit with on board memory and self contained I/O or an expanded multiboard system is needed, the application can be implemented at the lowest possible systems cost. With its broad line of semiconductor products, Texas Instruments manufactures most of the components utilized on the modules and thus is able to exercise quality control before and after the assembly of the system. This and high volume production ensure that the highest level of reliability and the maximum possible cost savings are passed along to the OEM.

At the heart of any of these systems is, of course, the software compatibility of the entire 990/9900 family. This product line fits squarely into that level of hardware/software integration between the flexibility of the TMS 9900 family component level application and the complete prepackaged system nature of the 990/4 and 990/10 minicomputers. The common instruction set enables the engineer to base hardware decisions solely on considerations of time, design, effort, and cost goals without concern for future compatibility of products or the ability to change his level of integration of the current product should initial considerations change.

The CPU boards, in particular, provide ready-to-use units for the evaluation of 9900 family component and software capability, especially when incorporated with the TM 990/301 microterminal. The OEM will find that these modules provide the best path to market in a time-critical application or in an application whose volume is difficult to assess. In most cases the modules will continue to be the best systems answer for such applications, but if it should not, TI will supply all necessary schematics and artwork to assist in the transition. The modules are completely supported by the AMPL* prototyping system. This floppy disc system, described more fully in another part of this book, has capabilities which make it invaluable for program development and debug as well as for a final systems test unit.

*Trademark of Texas Instruments Incorporated.

MICROCOMPUTER MODULES

The TM 990/100, TM 990/101, and TM 990/180 CPU modules come complete with on-board memory and I/O interface. Each 7½ x 11-inch (190 x 279 mm) board comes with 1k x 16 bits of TMS 2708 EPROM capability that can be expanded to 2k x 16 bits using TMS 2708's or 4k x 16 bits utilizing the jumper-selectable TMS 2716 option on the TM 990/100 and TM 990/101 modules. Static RAM capacity is 256 x 16 (1k x 16 for the TM 990/101) expandable to 512 x 16 (2k x 16 for the TM 990/101). Sixteen bits of parallel I/O are implemented on all three CPU's, as is a RS-232 or TTY serial interface. The TMS 9901, which performs the parallel I/O interface, also enables the user to implement the full interrupt capability of the processor. The TMS 9901 and the TMS 9902 (and for the TM 990/101, another TMS 9902 or a TMS 9903), which handle the serial I/O interface, each have programmable interval timers incorporated on-chip, thereby automatically providing the module user with two interval timers for the TM 990/100 and TM 990/180, and three interval timers for the TM 990/101. The TM 990/101 has two serial I/O parts. The bus structure of the CPU modules makes it possible to expand the system beyond the single board level. Memory, I/O, and special purpose controller boards may be added along with the TM 990/510 card cage for larger system applications.

EIA or teletype terminals can be optionally selected by the user and a differential line driver can be added as a factory option. Additionally, the TM 990/301 microterminal is an extremely low cost hexadecimal terminal option. The microterminal will execute the TIBUG monitor commands and can be used as a computer front panel.

TM 990/100M, TM 990/101, TM 990/180M MICROCOMPUTERS

TM 990/100M

- TMS 9900 16-bit CPU
- Up to 512 x 16 bits of RAM, TMS 4042-2 (2111-1)
- Up to 2k words of EPROM using TMS 2708 or 4k words using TMS 2716
- TMS 9901 programmable system interface
- TMS 9902 asynchronous communications controller
- EIA or TTY terminal interface option
- Prototyping area for custom applications
- Fully expandable bus structure
- Designed to fit the TM 990/510 card cage
- TIBUG operating monitor

TM 990/101

- TMS 9900 16-bit CPU
- Up to 2k x 16 bits of RAM, TMS 4045-45
- Up to 2k words of EPROM using TMS 2708 or 4k words using TMS 2716
- DMA to on-board memory
- TMS 9901 programmable system interface
- Two serial I/O ports, using TMS 9902 asynchronous communications controllers
- Three programmable interval timers
- Edge Triggered Interrupt, with software reset
- CRU addressable L.E.D. and DIP switch for custom applications
- Designed to fit the TM 990/510 card cage

TM 990/180

- TMS 9980 16-bit CPU
- Up to 1k x 8 bits of RAM TMS 4045-45
- Up to 4k bytes of EPROM using TMS 2708
- TMS 9901 programmable system interface
- TMS 9902 asynchronous communications controller
- EIA or TTY terminal interface option
- Prototyping area for customer applications
- Fully expandable bus structure
- Designed to fit the TM 990/510 card cage
- TIBUG operating monitor

SPECIFICATIONS

	TM 990/100M	*TM 990/101*	*TM 990/180M*
CPU:	TMS 9900	TMS 9900	TMS 9980
Instruction set	69 instructions	69 instructions	69 instructions
Bit operation	8, 16, or 32 bits	8, 16, or 32 bits	8, 16, or 32 bits
System clock	3 MHz	3 MHz	2.5 MHz
Interrupts	16 levels—15 may be external	16 levels—15 may be external	6 levels—4 may be external
Interval timers	Two (in TMS 9901 and TMS 9902)	Three (in TMS 9901, TMS 9902, and TMS 9903, or in TMS 9901 and two TMS 9902's)	Two (in TMS 9901 and TMS 9902)
	RESOLUTION / MAXIMUM INTERVAL	RESOLUTION / MAXIMUM INTERVAL	RESOLUTION / MAXIMUM INTERVAL
TMS 9901	21.3 μs / 349 ms	21.3 μs / 349 ms	25.6 μs / 414 ms
TMS 9902	64 μs / 16.4 ms	64 μs / 16.4 ms	76.8 μs / 19.7 ms
TMS 9903		64 μs / 16.4 ms	
Memory:	16-bit word configuration	16-bit word configuration	8-bit byte configuration
On-board EPROM/ROM	1k words, expandable to 4k	1k words, expandable to 4k	2k bytes, expandable to 4k
On-board RAM	256 words, expandable to 512	1k words, expandable to 2k	512 bytes, expandable to 1k
Off-board expansion	Up to 32 k words	Up to 32k words	Up to 16k bytes
Input/Output			
Parallel:	16 lines, expandable to 4k	16 lines (7 dedicated and 9 that may be programmed as inputs, outputs, or interrupts) expandable to 4k	16 lines, expandable to 2k
Serial:	Asynchronous Controller, TMS 9902 5-8 bits/character Programmable data rate, stop bits, parity	Serial Port A: Asynchronous Controller, TMS 9902 Serial Port B: Asynchronous Controller, TMS 9902, or Synchronous Controller, TMS 9903 5-8 bits/character Programmable data rate, stop bits, parity	Asynchronous controller, TMS 9902 5-8 bits/character Programmable data rate, stop bits parity
Baud rates: (bps)	75 300 2400 19,200 110 600 4800 38,400 150 1200 9600	110 600 4800 19,200 150 1200 9600 38,400 300 2400	75 300 2400 19,200 110 600 4800 38,400 150 1200 9600
Interfaces			
Bus: Data and address	3-state, TTL compatible	3-state, TTL-compatible	3-state, TTL compatible
Control	TTL-compatible	TTL-compatible	TTL-compatible
Parallel I/O and interrupts	TTL-compatible	TTL-compatible	TTL-compatible
Serial I/O	RS-232, 20-mA current loop, or differential line driver	Port A; RS-232C, 20-mA current loop, or RS-232C Multidrop Port B: RS-232C terminal, or modem with optional cable	RS-232, 20-mA current loop, or differential line driver

TM 990/100M, TM 990/101, TM 990/180M MICROCOMPUTERS

	TM 990/100M	*TM 990/101*	*TM 990/180M*
Expansion Prototyping Area	Space for one 40-pin DIP and four 16-pin DIP's	Not applicable	Space for one 40-pin DIP and four 16-pin DIP's
Software	TI BUG monitor self-contained in EPROM (TM990/401-1)	TIBUG monitor self-contained in EPROM (TM 990/401-3)	TIBUG monitor self-contained in EPROM (TM 990/401-2)
Mating Connectors			
Bus: 100-pin, 0.125-inch (3,18-mm) centers	TI H4311211-50 (Solder tail), TI H431111-50 (Wire wrap), or Viking 3VH50		
Parallel I/O: 40-pin, 0.100-inch (2,54-mm) centers	TI H421121-20, 3M 3464-0001, or Viking 3VH20/1JN5		
Serial I/O: 25-pin (male)	Cannon DB-25P, Cinch DB-25P, or ITT DB-25P		
Power Requirements		1K RAM 1K EPROM	2K RAM 2K EPROM
	5V ± 3% 1.3A	5V ± 3% 1.6A	1.8A 5V ± 3% 1.3A
	12V ± 3% 0.2A	12V ± 3% 0.2A	0.4A 12V ± 3% 0.2A
	−12V ± 3% 0.1A	−12V ± 3% 0.2A	0.3A −12V ± 3% 0.1A
		−5V regulated on board from −12V	
Operating Temperature Range	0°C to 70°C	0°C to 70°C	0°C to 70°C
Physical Characateristics:			
Width	11 inches (279 mm)	11 inches (279 mm)	11 inches (279 mm)
Height	7.5 inches (190 mm)	7.5 inches (190 mm)	7.5 inches (190 mm)
Depth		0.5 inch (12.7 mm)	
Board thickness	0.062 inch (1,58 mm)	0.062 inch (1,58 mm)	0.062 mm (1,58 mm)
Weight		1 pound (0.45 kg)	
Terminations I/O and Interrupt	4.7 kΩ 5V	10 kΩ 5V	4.7 kΩ 5V
READY B and HOLD B		220 Ω 5V 330Ω	

ORDERING INFORMATION

TM 990/100M-1 — TMS 9900 microcomputer board with TIBUG monitor in two TMS 2708 EPROM's and EIA or TTY serial I/O jumper option.

TM 990/100M-2 — TMS 9900 microcomputer board with unprogrammed TMS 2708 EPROM's and EIA or differential line driver jumper option.

TM 990/100M-3 — TMS 9900 microcomputer board with fully expanded memory (four TMS 2716 EPROM's and eight TMS 4042-2 RAM's) and EIA or differential line driver jumper option.

TM 990/180M-1 — TMS 9980 CPU board with TIBUG monitor in two TMS 2708 EPROM's and EIA or TTY serial I/O jumper option.

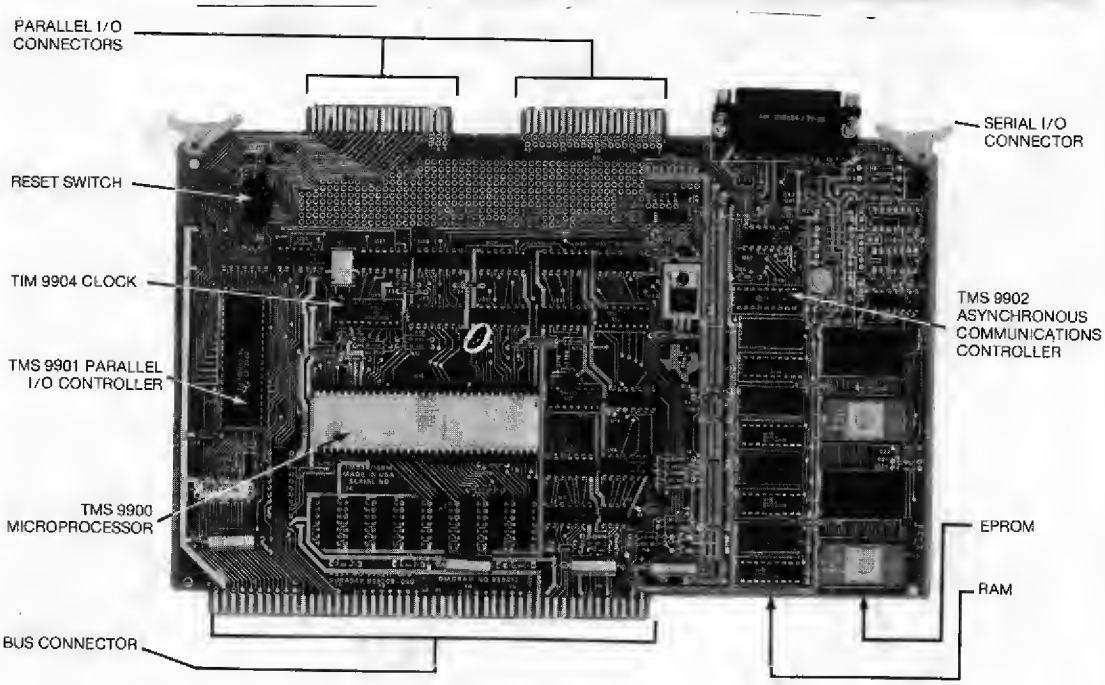
TM 990/180M-3 — TMS 9980 CPU board with four unprogrammed TMS 2708 EPROM's, eight TMS 4042 RAM's, and EIA or differential line driver jumper option.

TMS990/101M OPTIONS

TM990/101-1 TMS 9900 microcomputer board with TIBUG monitor in two TMS 2708 EPROM's and TTY, EIA and microterminal on the local serial port. The remote serial port supports synchronous/asynchronous communications.

TM990/101-2 TMS 9900 microcomputer board with unprogrammed TMS 2708 EPROM's and multidrop, EIA and microterminal options on the local serial port. The remote serial port supports synchronous/asynchronous communications.

TM990/101-3 TMS 9900 microcomputer board with fully expanded memory (four TMS 2716 EPROM's and eight TMS 4045-45 RAM's) and TTY, EIA and microterminal options on the local serial port. The remote serial port supports synchronous/asynchronous communications.



The TM 990/100M is an assembled, tested microcomputer module utilizing the powerful, NMOS 16-bit, TMS 9900 microprocessor as its CPU. With RAM and ROM/EPROM included on board as well as programmable serial and parallel I/O, the TM 990/100M is a powerful single-board microcomputer. Since all address, data, and control lines are brought to the board connectors, the board can be expanded to use the entire capabilities of the TMS 9900 in larger systems.

OPERATION

The TM 990/100M microcomputer is a software compatible member of the TMS 9900/990 family. The TMS 9900 is used as a CPU to provide 16 bits of processing power with a minicomputer instruction set which includes multiply and divide. The TM 990/100M module is designed for 3 MHz operation, utilizing the full 16 levels of prioritized interrupts and the advanced memory-to-memory architecture of the TMS 9900. Additionally, the bus structures are set up to take advantage of the full 64K byte memory addressing capability of the 9900 and the nonmultiplexed memory, I/O and interrupt buses.

MEMORY

The on-board memory includes both an EPROM/ROM section and a static RAM section. Four sockets are available for TMS 2708, TMS 2716 EPROM or TMS 4700, TMS 4732 ROM operation. The assembled price includes two TMS 2708's, or 1K words. Using the available jumper option, all four sockets can be populated with TMS 2716's, providing a maximum on-board EPROM capability of 4K words. The static RAM area consists of two 256-word banks of memory. Four TMS 4042-2 (TMS 2111-1) are populated and four additional sockets are included. The cycle time of this memory section is 0.667 microseconds. The address map is shown in *Figure 1;* the minimum area of EPROM RAM area may not be used for off-board expansion. DMA control lines are also accessible on the bus.

INTERRUPTS AND TIMERS

Fifteen maskable interrupts plus the reset and load trap vectors are implemented. *Table 1* shows the implementation. The TMS 9901 handles all 15 external interrupts which can be generated from either the bus connector or the I/O bus. The TMS 9901 enables each level to be individually maskable under program control. Additionally, level 3 can be programmed to use the interval timer in the TMS 9901. Level 4 can be generated from the TMS 9902 as an interval timer or for three other serial interface conditions (see the *TMS 9902 Data Manual*). Two programmable timers, therefore, are available on board.

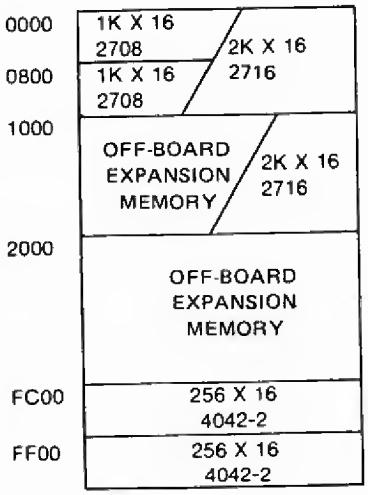


Figure 1. Memory Address Map.

I/O

The serial I/O and the parallel I/O are handled over the dedicated I/O bus of the TMS 9900, the communications register unit (CRU). *Table 2* lists the address assignments within the dedicated 4K CRU address space. The TMS 9902 acts as the controller for this asynchronous serial interface. The character length, baud rate (75 to 38,400), and parity and stop bits are programmable. Three optional types of interface are supported:

- EIA
- 20 mA neutral current loop TTY
- Private wire differential line driver/receiver.

The TM 990/100M board is delivered complete with a 25-pin RS-232 type female connector, and is jumper selectable to support EIA or TTY operation. The differential line driver is normally unpopulated (see *Options*). Also, the TMS 9903 synchronous communications controller can be utilized since the TMS 9902/9903 are socket compatible.

INTERRUPT LEVEL	FUNCTION
0	Reset or PRES
1	External Device
2	External Device
3	Clock or External
4	Serial Int. or Ext.
5–15	External Devices
Load	Restart

Table 1. Interrupts.

BASE ADDRESS (REGISTER 12)	CRU BIT NUMBER	FUNCTION
0080_{16}	$40_{16} \rightarrow 5F_{16}$	On-Board Serial I/O Port (TMS 9902)
0100_{16}	$80_{16} \rightarrow 9F_{16}$	On-Board 16 I/O Parallel Interface, Interrupt Status Register, Interrupt Mask Register, and Interval Timer (TMS 9901)
0000_{16} $00C0_{16}$ 0140_{16}	$00_{16} \rightarrow 3F_{16}$ $60_{16} \rightarrow 7F_{16}$ $A0_{16} \rightarrow FF_{16}$	Reserved for On-Card Expansion
200_{16}	$100_{16} \rightarrow FFF_{16}$	Off-Board CRU

Table 2. CRU Address Map.

The parallel I/O is handled by the TMS 9901; 16 parallel lines are all interfaced to the top edge connector which has 40 pins on 0.100 inch (2.54 mm) centers. Additionally, eight parallel lines are interfaced to the bus connectors. The programmable features of the TMS 9901 permit configuring these lines as I/O lines or interrupts (refer to the *TMS 9901 Data Manual*). All I/O lines are equipped with pullup resistors.

TIBUG

The TIBUG monitor TM 990/401-1 is normally supplied preprogrammed in the populated TMS 2708 EPROM's (see *Options*). Its operation is described in the *User's Manual* or the TM 990 Series literature.

PROTOTYPING AREA

The prototyping area is large enough to accommodate one 40-pin DIP (0.6 inch 15,24 mm centers) plus four 16-pin DIP's (0.3 inch 7,62 mm centers).

OPTIONS

The TM 990/100M-1 board is equipped with two TMS 2708's preprogrammed with the TIBUG monitor, and the serial I/O is jumper selectable as EIA port or a TTY interface. The TM 990/100M-2 board is populated with two blank EPROM's, and a private wire differential line driver interface is populated instead of the TTY interface. The TM 990/100M-3 board is delivered with the maximum memory expansion (512 words of RAM and 4K words of unprogrammed EPROM) and the differential line driver. Other software or accessories, such as the line by line assembler and the microterminal, may be ordered under separate part numbers.

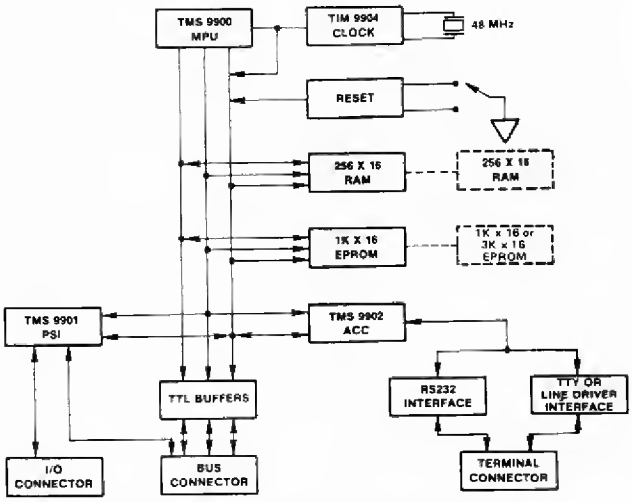
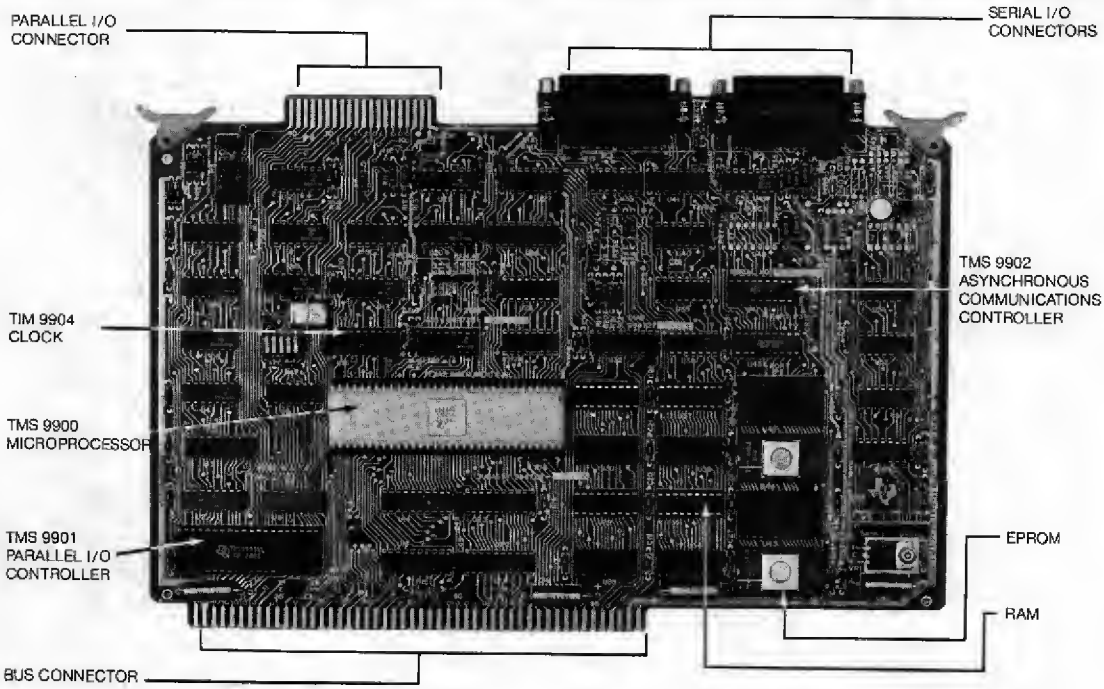


Figure 2. TM 990/100M Block Diagram



The TM 990/101 microcomputer is a member of Texas Instruments line of OEM computer products which take full advantage of Texas Instruments broad based semiconductor technology to provide economical, computer-based solutions for OEM, applications. The CPU, clock, memory, I/O, and bus interface are all contained on a single 7½ x 11 inch (190,5 x 279 mm) printed circuit board.

OPERATION

The TMS 9900 microprocessor is the heart of the TM 990/101. This 16-bit CPU features a memory to memory architecture and a minicomputer instruction set which includes hardware multiply and divide. A total of eight addressing modes, including indirect and pre-indexed addressing, provide powerful software capabilities while the TMS 9900's two-address architecture makes a memory-to-memory add possible without having to load register pairs with addresses or using a dedicated accumulators. The TMS 9900's instruction set is upward compatible with the TI 990 minicomputer family. The TMS 9900 addresses 32K 16-bit words of memory. In addition to DMA and memory mapped I/O, the TMS 9900 performs I/O functions on a separate data structure called the Communications Register Unit (CRU). The CRU consists of 4096 output bits and 4096 input bits. Each bit is separately addressable. Five instructions enable the programmer to perform both single and multibit CRU operations.

MEMORY

The TM 990/101 microcomputer contains up to 4K bytes of static RAM (TMS 4045) on-board. All positions are socketed. Sockets are provided for up to 8K bytes of EPROM (TMS 2716). Convenient jumper options also allow utilization of TMS 2708 1K x 8 bit EPROM's. Provisions are included for deconfiguration of either or both on-board RAM and on-board EPROM, if desired, when used with other TM 990 series Memory Expansion boards. A jumper-selectable wait state for the EPROM is also included.

All memory device locations are socketed. A PROM controls memory address decoding. The RAM is decoded as one bank but the two EPROM pairs are decoded as separate banks, allowing custom placement of EPROM. Such a custom decoding scheme can be done by obtaining a blank SN74S287 PROM, and programming it properly.

CONFIGURATION	MEMORY MAP	ALTERNATE MEMORY MAP
RAM, bank 2	$F000_{16}$-F7FE	0000_{16}-$07FE_{16}$
RAM, bank 1	$F800_{16}$-$FFFE_{16}$	0800_{16}-$0FFE_{16}$
EPROM, all TMS 2708*	0000_{16}-$0FFE_{16}$	$F000_{16}$-$FFFE_{16}$
EPROM, all TMS 2716*	0000_{16}-$1FFE_{16}$	$E000_{16}$-$FFFE_{16}$

*Jumper selectable

INTERRUPTS AND TIMERS

Seventeen interrupt inputs are available on the TM 990/101. All interrupts trap through vectors in memory. Two interrupts are non-maskable interrupts while the others are maskable. There are three interrupt sources on board: Serial I/O Port A, Serial I/O Port B, and the TMS 9901 interval timer. Interrupt 6 may be triggered on either a positive or negative transition. All other interrupts are active low. The 15 maskable interrupts are also automatically prioritized by the microprocessor.

INTERRUPT	LEVEL	VECTOR	DESCRIPTION
PRES	0	0000-0002_{16}	Unmaskable, active low.
RESTART	LOAD	$FFFC_{16}$-$FFFE_{16}$	Unmaskable, active low. May be activated by software (LREX).
INT1-INT5	1-5	0004_{16}-0016_{16}	Maskable, dedicated, active low.
INT6	6	0018_{16}-$001A_{16}$	Maskable, dedicated (+) or (-) edge detect or active low.
INT7-INT15	7-15	$001C_{16}$-$003E_{16}$	Maskable, active low. May be programmed as interrupt, input, or output.

8◀

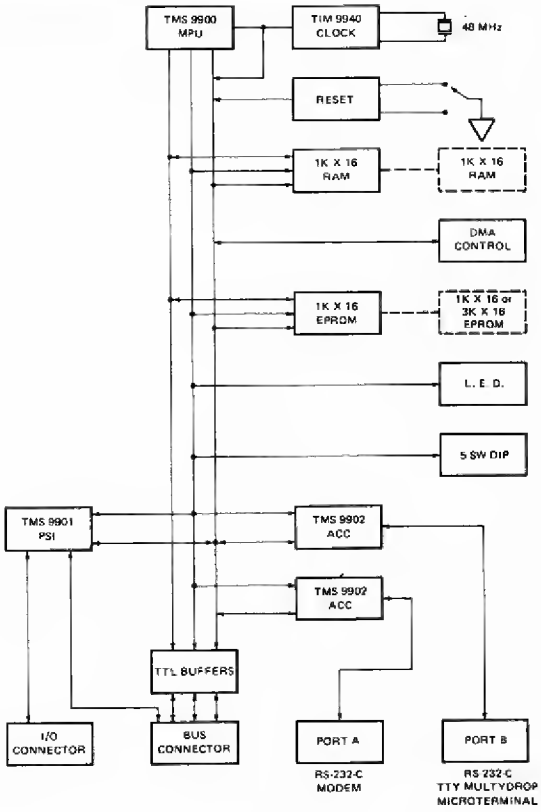
I/O

The TM 990/101 contains up to 16 programmable I/O lines controlled by a TMS 9901 Programmable Systems Interface. Seven lines are dedicated I/O lines while 9 lines may be programmed as either I/O or interrupt inputs. The 16 I/O lines appear on a 40-pin edge connector which mates with ribbon cable or round cable.

Two serial interfaces are available on the TM 990/101. Each port is controlled by a TMS 9902 Asynchronous Communications Controller. Serial communications rates of up to 38,400 baud may be maintained. Port A is compatible with the serial I/O port on the TM 990/100 microcomputer. Port A supports EIA compatible terminals as well as 20 mA neutral current loop teletypes. Port A also supports TI's TM 990/301 Microterminal. A version of the TM 990/101 supports a differential line driver-receiver communications interface in place of the TTY interface. This multidrop interface supports 9600 baud serial communications at distances of up to 10,000 ft. on shielded twisted pairs. Serial Port B supports communications with EIA compatible terminals as well as popular modems such as Bell Type 103J and 212A, using an optional modem cable.

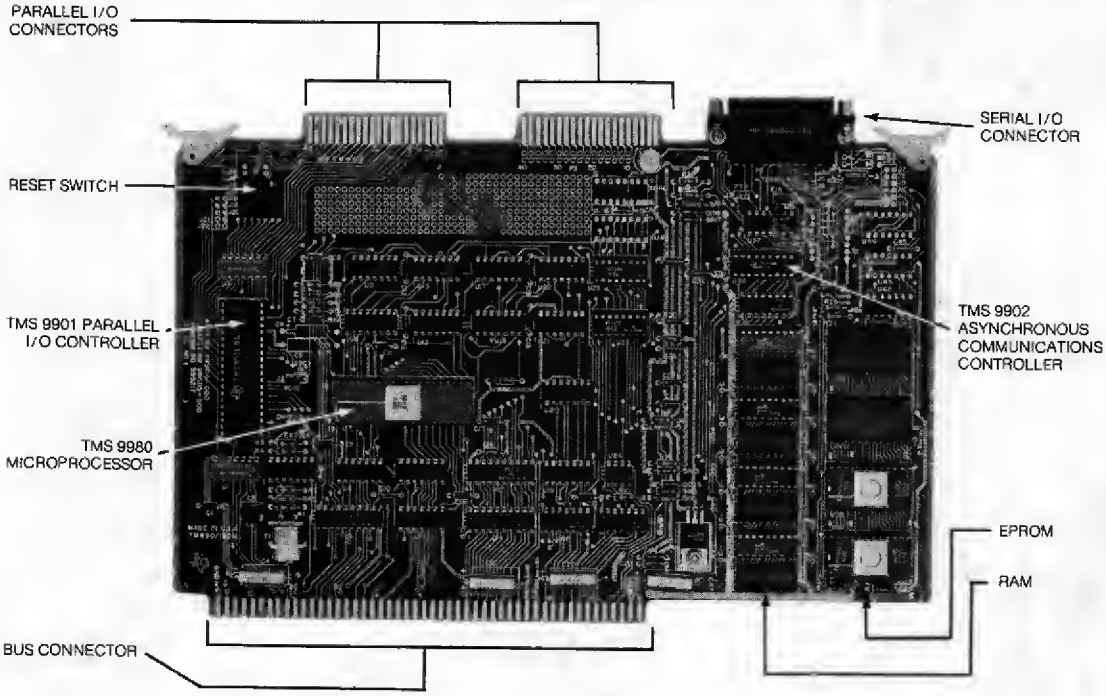
TM 990/101 memory and I/O capacity may be increased by adding Texas Instruments standard I/O and memory expansion cards. Memory may be expanded to 60K bytes by the addition of the TM 990/201 memory expansion boards, leaving 4K bytes open for memory mapped I/O. Parallel I/O and interrupt expansion capability may be increased by the addition of the TM 990/310 48-I/O Data Module.

The development cycle for TM 990/101 based products may be significantly reduced by using Texas Instruments Advanced Microprocessor Prototyping System (AMPL). TMS 9900 emulation as well as 10 MHz trace capability are featured. This floppy disk based software development system permits programs to be edited, assembled, linked, loaded, and executed much faster than conventional paper tape or cassette based systems. TMS 9900 emulation allows development and debugging of software directly on the TM 990/101 while monitoring and controlling this environment from the AMPL prototyping system.



TM 990/101 Block Diagram

8◀



The TM 990/180M is an assembled, tested microcomputer module utilizing the NMOS 16-bit TMS 9980 microprocessor as its CPU. The TMS 9980 utilizes an eight bit data bus which may be the most cost effective solution for smaller byte-dedicated operations. With RAM and ROM/EPROM included on board as well as programmable serial and parallel I/O, the TM 990/180M is a powerful single-board microcomputer. Since all address, data, and control lines are brought to the board connectors, the board can be expanded to use the entire capabilities of the TMS 9980.

OPERATION

The TM 990/180M microcomputer module is a software compatible member of the TMS 9900/990 family. The TMS 9980 is used as a CPU to provide 16 bits of processing power with a minicomputer instruction set which includes multiply and divide. The TM 990/180M module is designed for 2.5 MHz operation, utilizing the full six levels of prioritized interrupts and the advanced memory-to-memory architecture of the TMS 9980. Additionally, the bus structures are set up to take advantage of the full 16K byte memory addressing capability of the TMS 9980 and the nonmultiplexed memory, I/O and interrupt buses. The bus structure is compatible with the other boards of the TM 990 series such as the TM 990/100M board.

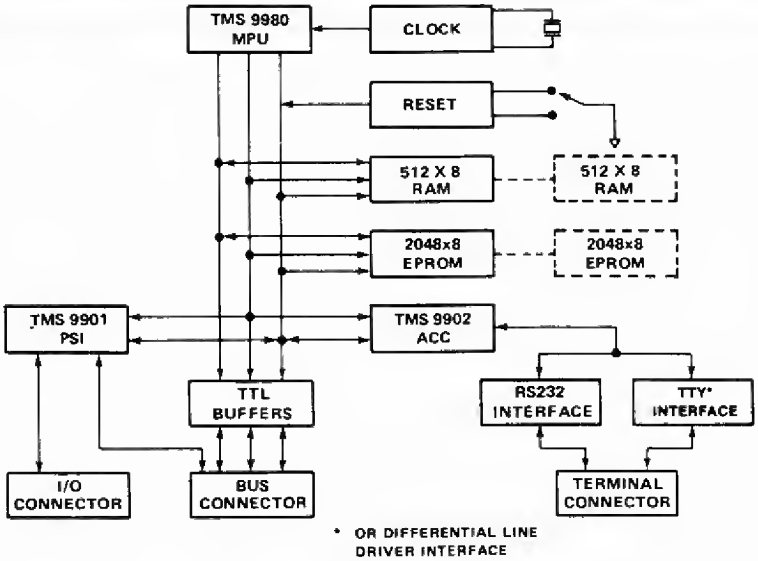


Figure 2. TM 990/180M Block Diagram

MEMORY

The on-board memory includes both an EPROM/ROM section and a static RAM section. Four sockets are available for TMS 2708 EPROM or TMS 4700 ROM operation. The assembled price includes two TMS 2708's or 2K bytes. The static RAM area consists of four 256 byte banks of memory. Four TMS 4042-2 (TMS 2111-1) are populated, and four more sockets are included. The cycle time of this memory section is 1.33 microseconds. The memory address map is shown in *Figure 1.*

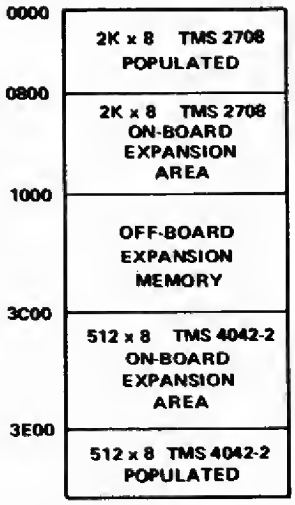


Figure 1. Memory Address Map

INTERRUPTS AND TIMERS

Four maskable interrupts plus the reset and load trap vectors are implemented. *Table 1* shows the implementation. The TMS 9901 handles all four external interrupts which can be generated from either the bus connector or the I/O bus. The TMS 9901 enables each level to be individually maskable under program control. Additionally, level 3 can be programmed to use the interval timer in the TMS 9901. Level 4 can be generated as an interrupt from the TMS 9902. One of the functions that will cause this interrupt is the interval timer. Two programmable timers, therefore, are available on board.

LEVEL FROM TMS 9901	TRAP VECTOR LOCATION	FUNCTION
1	0000_{16}	Reset pushbutton or PRES from the chassis backplane connector.
2	$3FFC_{16}$	Software (LREX) or RESTART from the chassis backplane connector.
3	0004_{16}	Real Time Clock (TMS 9901) or external device.
4	0008_{16}	Serial interface (TMS 9902) or external device.
5	$000C_{16}$	External device.
6	0010_{16}	External device.

Table 1. TMS 9980 Interrupts

INPUT/OUTPUT

The Serial I/O and the parallel I/O are handled over the dedicated I/O bus of the TMS 9900 or the communications register unit (CRU). Table 2 lists the address assignments within the dedicated 4K CRU address space. The TMS 9902 acts as the controller for this asynchronous serial interface. The character length, baud rate (75 to 38,400), parity and stop bits are programmable. Three optional types of interface are supported:

- EIA
- 20 mA neutral current loop TTY
- Private wire differential line driver/receiver.

BASE ADDRESS	CRU BIT NUMBER	FUNCTION
0000_{16}	$000_{16} \rightarrow 03F_{16}$	Reserved for on-board CRU Expansion
0080_{16}	$040_{16} \rightarrow 05F_{16}$	On-board serial I/O (TMS 9902)
$00C0_{16}$	$060_{16} \rightarrow 07F_{16}$	Reserved for on-board CRU Expansion
0100_{16}	$080_{16} \rightarrow 09F_{16}$	On-board parallel I/O interface interrupt status register, interrupt mask register, interval timer (TMS 9901)
0140_{16}	$0A0_{16} \rightarrow 0FF_{16}$	Reserved for on-board CRU Expansion
0200_{16}	$100_{16} \rightarrow 3FF_{16}$	Off-board CRU

Table 2. CRU Address Map

The TM 990/180M board is delivered complete with a 25-pin RS-232 type female connector, and is jumper selectable to support EIA or TTY operation. The differential line driver is normally unpopulated (see *Options*). Also, the TMS 9903 synchronous communications controller can be utilized, since the TMS 9902/9903 are socket compatible.

The parallel I/O is handled by the TMS 9901; 16 parallel lines are all interfaced to the top edge connector which has 40 pins on 0.100 inch (2,54 mm) centers. Additionally, eight parallel lines are interfaced to the bus connectors. The programmable features of the TMS 9901 permit these lines to be set up as I/O lines or interrupts (refer to the *TMS 9901 Data Manual*). All I/O lines are equipped with pullup resistors.

TIBUG

The TIBUG monitor TM 990/401-2 is normally supplied preprogrammed in the populated TMS 2708 EPROM's (see Options). Its operation is described in the TIBUG *User's Manual* or the TM 990-series literature.

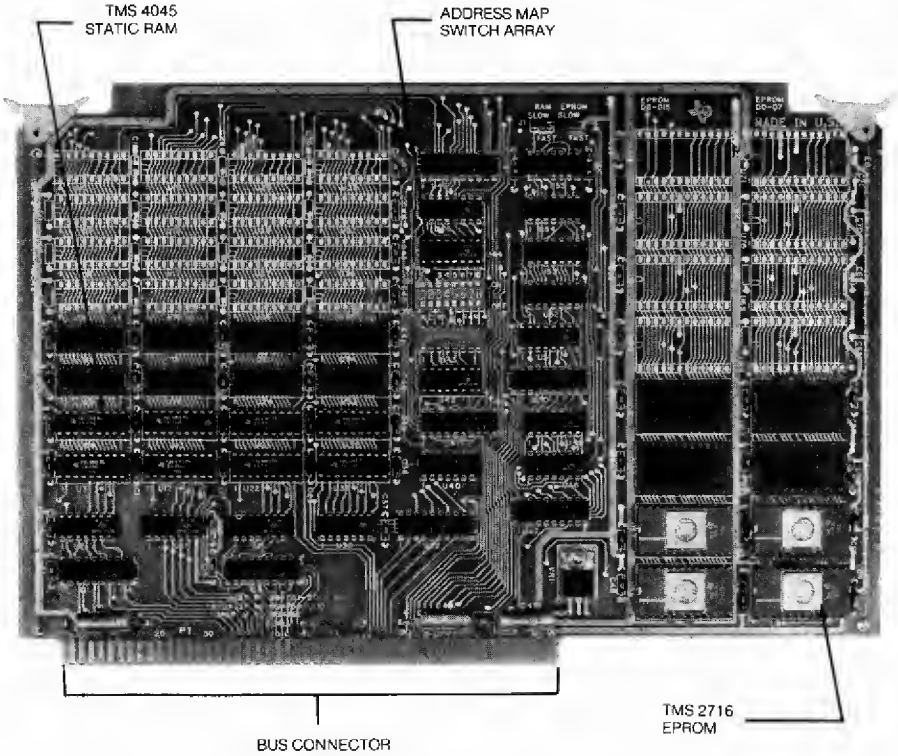
PROTOTYPING AREA

The prototyping area is large enough to accommodate one 40-pin DIP (0.6 inch 15,24 mm centers) plus four 16-pin DIP's (0.3 inch 7,62 mm centers).

OPTIONS

The TM 990/180M-1 board is equipped with two TMS 2708 EPROM's preprogrammed with the TIBUG monitor, and the serial I/O is jumper selectable as an EIA port or a TTY interface. The TM 990/180M-3 board is populated with four TMS 2708 EPROM's (unprogrammed), eight TMS 4042-2 RAM's, and a private wire differential line driver interface instead of the TTY interface. Other software or accessories, such as the line by line assembler and the microterminal, may be ordered under separate part numbers.

TM 990/201 MEMORY EXPANSION BOARD

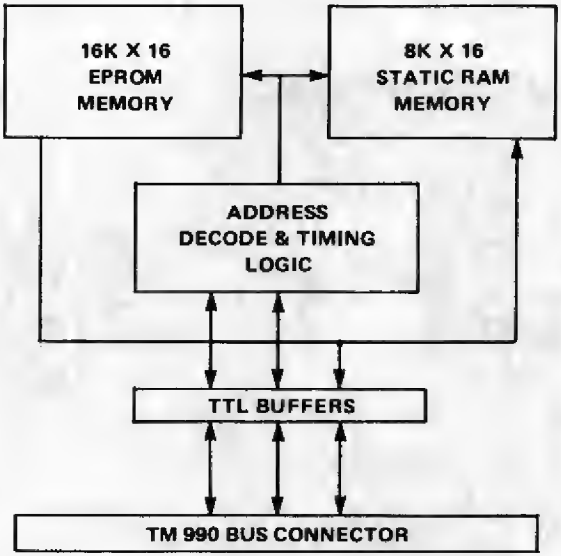


The TM 990/201 is an assembled, tested, memory expansion board designed for use with TMS 9900-based microcomputer modules such as the TM 990/100M. The TM 990/201 contains both static RAM and EPROM memory, expandable to a maximum configuration of 8K x 16 bit words of RAM and 16K x 16 bit words of EPROM. The TM 990/201 does not support the TMS 9980-based TM 990/180M microcomputer.

FEATURES

- Bus-compatible with the TM 990/100M microcomputer module
- 4K words TMS 2716 EPROM, expandable to 16K words
- 2K words TMS 4045 static RAM, expandable to 8K words
- 1 microsecond cycle time (3MHz)
- TTL-compatible interface
- Designed to fit the TM 990/510 card cage.

8◀



DESCRIPTION

The TM 990/201 memory expansion board is a member of Texas Instruments' line of OEM computer products which takes advantage of Texas Instruments' broad based semiconductor technology to provide economical, computer based solutions for OEM applications. The memory expansion board is contained on a 7½ x 11 inch printed circuit board which is fully compatible with the TM 990 board format.

The TM 990/201 features up to 8K x 16 bits of static RAM and up to 16K x 16 bits of EPROM. The static RAM array is composed of Texas Instruments TMS 4045, 1K x 4 static memory devices. The EPROM array comprises Texas Instruments TMS 2716, 2K x 8 EPROM devices. The static RAM array is arranged into four banks of memory, each 2K x 16. The EPROM array is likewise arranged into eight banks, each 2K x 16. Both memory arrays are socketed for convenient memory expansion. The TM 990/201 -41 is half socketed, and the TM 990/201 -42 and TM 990/201 -43 are fully socketed.

The TM 990/201 memory controller logic provides the timing and memory mapping functions necessary to interface the TM 990/201 to 16-bit TM 990/1XX series microcomputers. The memory map is switch selectable for both the RAM and EPROM arrays. Sixteen convenient memory map configurations are possible for each array, and the maps are configured on 2K word address boundaries. The map logic also is designed to accommodate customized memory maps.

TM 990/201 MEMORY EXPANSION BOARD

The TM 990/201 -4X family of memory expansion boards is populated with TMS 4045-45 static RAM's and TMS 2716 EPROM's. Both devices offer 450 nsec access time; consequently, each memory cycle to the TM 990/201 is extended one clock cycle by the insertion of a WAIT state. If faster static RAM's are utilized in the RAM array, the WAIT state in RAM memory cycles can be conveniently removed using only a jumper.

OPTIONS

The TM 990/201 is available in the following three versions.

MODEL NO.	MEMORY POPULATED		MEMORY EXPANSION AREA (EXTRA SOCKETS PROVIDED)	
	EPROM	RAM	EPROM	RAM
TM 990/201-41	4K x 16	2K x 16	4K x 16	2K x 16
TM 990/201-42	8K x 16	4K x 16	8K x 16	4K x 16
TM 990/201-43	16K x 16	8K x 16	—	—

MEMORY CONFIGURATION

Figures 1 and *2* show the memory configurations of RAM and EPROM available on the TM 990/201 memory expansion board.

8

A0-A3 (HEX)	HEX MEMORY ADDRESS	MICROCOMPUTER MEMORY MAP /100	SWITCH NO.	DIP SWITCH CODES* 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	HEX
			5	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	
			6	ON	ON	OFF	OFF	ON	ON	OFF	OFF	ON	ON	OFF	OFF	ON	ON	OFF	OFF	
			7	ON	ON	ON	ON	OFF	OFF	OFF	OFF	ON	ON	ON	ON	OFF	OFF	OFF	OFF	
			8	ON	ON	ON	ON	ON	ON	ON	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	
0	0000-0FFF	EPROM																		
1	1000-1FFF	EPROM (EXPAN.)																		
2	2000-2FFF																			
3	3000-3FFF																			
4	4000-4FFF																			
5	5000-5FFF																			
6	6000-6FFF																			
7	7000-7FFF																			
8	8000-8FFF																			
9	9000-9FFF																			
A	A000-AFFF																			
B	B000-BFFF																			
C	C000-CFFF																			
D	D000-DFFF																			
E	E000-EFFF	MAPPED I/O																		
F	F000-FFFF	RAM																		

*OFF = 1, ON = 0

Figure 1. TM 990/201 Ram Decode Configurations

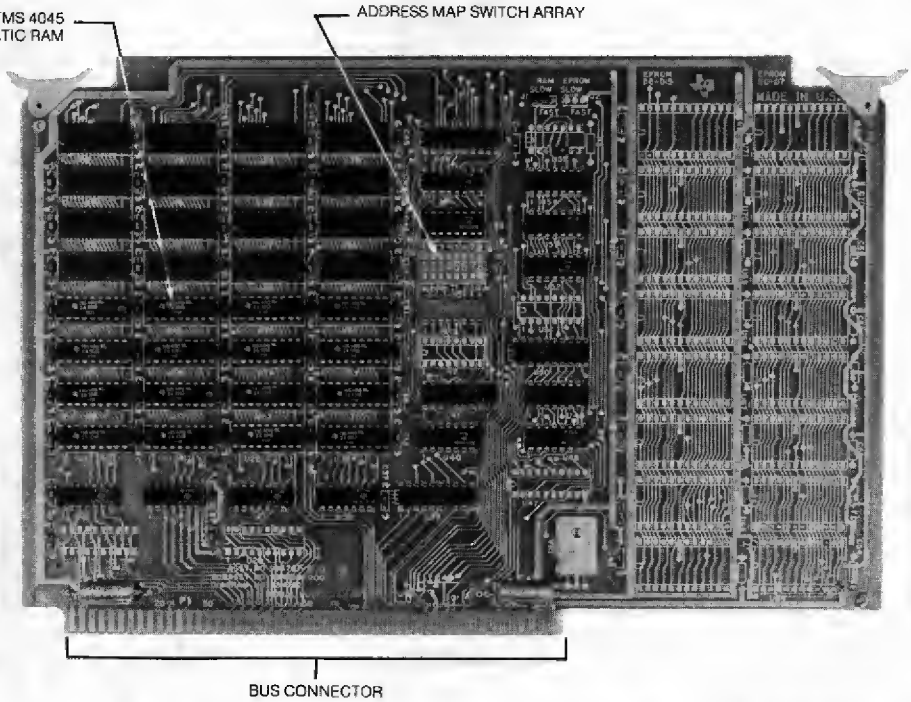
A0-A3 (HEX)	HEX MEMORY ADDRESS	MICROCOMPUTER MEMORY MAP /100	SWITCH NO. 1	2	3	4	SWITCH CODES* 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	HEX
			ON	ON	ON	ON	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	
			ON	ON	ON	ON	ON	ON	OFF	OFF	ON	ON	OFF	OFF	ON	ON	OFF	OFF	ON	ON	OFF	OFF	
			ON	ON	ON	OFF	ON	ON	ON	ON	OFF	OFF	OFF	OFF	ON	ON	ON	ON	OFF	OFF	OFF	OFF	
			ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	
0	0000-0FFF	EPROM																					
1	1000-1FFF	EPROM (EXPAN.)																					
2	2000-2FFF																						
3	3000-3FFF																						
4	4000-4FFF																						
5	5000-5FFF																						
6	6000-6FFF																						
7	7000-7FFF																						
8	8000-8FFF																						
9	9000-9FFF																						
A	A000-AFFF																						
B	B000-BFFF																						
C	C000-CFFF																						
D	D000-DFFF																						
E	E000-EFFF	MAPPED I/O																					
F	F000-FFFF	RAM																					

*OFF = 1, ON = 0

Figure 2. TM 990/201 Eprom Decode Configurations

8◀

TM 990/206 EXPANSION MEMORY BOARD

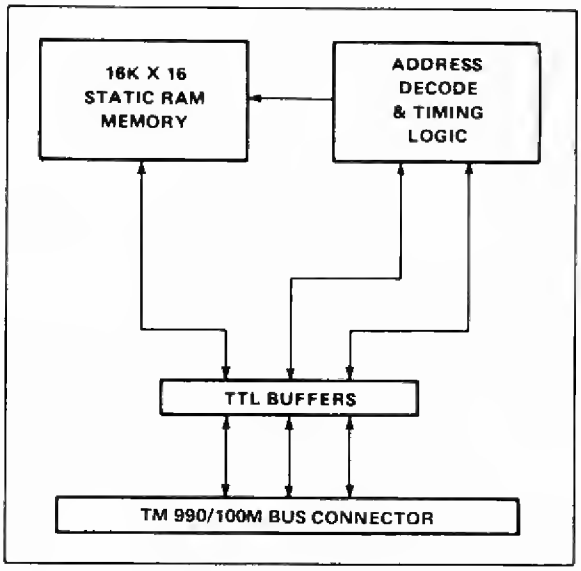


The TM 990/206 is an assembled, tested, RAM expansion memory board designed for use with TMS 9900-based microcomputer modules such as the TM 990/100M. The TM 990/206 contains static RAM memory devices up to a maximum configuration of 8K x 16 words. The TM 990/206 is similar to the popular TM 990/201 memory board, but only the RAM section is populated. The TM 990/206 does not support the TMS 9980-based TM 990/180M microcomputer.

FEATURES:

- Bus compatible with the TM 990/100M microcomputer module
- 4K words of TMS 4045 static RAM, expandable to 8K words
- 1μsec cycle time (3 MHz)
- TTL-compatible interface
- Designed to fit the TM 990/510 card cage.

TM 990/206 EXPANSION MEMORY BOARD



DESCRIPTION

The TM 990/206 expansion memory board is a member of Texas Instruments' line of OEM computer products which take advantage of Texas Instruments' broad based semiconductor technology to provide economical, computer based solutions for OEM applications. The memory expansion board is contained on a 7½ x 11 inch printed circuit board which is fully compatible with the TM 990 board format.

The TM 990/206 features up to 8K x 16 static RAM. The RAM array is composed of Texas Instruments TMS 4045-45 1K x 4 static memory devices. The array is configured into four banks of memory, each bank consisting of 2K words. The RAM array is fully socketed for convenient memory expansion.

The memory controller logic provides the timing and memory mapping functions necessary to interface the TM 990/206 to 16-bit TM 990/1XX series microcomputers. Sixteen convenient, switch selectable, memory map configurations are possible. All maps are configured on 2K word address boundaries.

The TM 990/206-4X family of memory expansion boards is populated with TMS 4045-45 static RAM's, featuring an access time of 450 nsec. For operation with a TM 990/100M microcomputer, each memory cycle to the TM 990/206 is extended one clock cycle by the insertion of a WAIT state. If faster static RAM's are utilized, the WAIT state can be conveniently removed with a jumper.

OPTIONS

The TM 990/206 is available in two versions:

Model	*RAM Population*	*Expansion Area Additional Sockets*
TM 990/206-41	4K×16	4K×16
TM 990/206-42	8K×16	—

MEMORY CONFIGURATION

Figure 1 shows the possible memory configurations for the RAM areas available on the TM 990/206.

A0-A3 (HEX)	HEX MEMORY ADDRESS	MICROCOMPUTER MEMORY MAP /100		SWITCH NO.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	HEX
					SWITCH CODES*																
				5	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	
				6	ON	ON	OFF	OFF	ON	ON	OFF	OFF	ON	ON	OFF	OFF	ON	ON	OFF	OFF	
				7	ON	ON	ON	ON	OFF	OFF	OFF	OFF	ON	ON	ON	ON	OFF	OFF	OFF	OFF	
				8	ON	ON	ON	ON	ON	ON	ON	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	
0	0000-0FFF	EPROM			8K WORDS (32 4045's)								4K WORDS (16 4045's)								
1	1000-1FFF	EPROM (EXPAN.)												RBLK0							
2	2000-2FFF					RBLK0								RBLK1							
3	3000-3FFF																				
4	4000-4FFF																				
5	5000-5FFF					RBLK4															
6	6000-6FFF																				
7	7000-7FFF																				
8	8000-8FFF																				
9	9000-9FFF																				
A	A000-AFFF																				
B	B000-BFFF																				
C	C000-CFFF																				
D	D000-DFFF																				
E	E000-EFFF	MAPPED I/O																			
F	F000-FFFF	RAM																			

*OFF = 1, ON = 0

Figure 1. TM 990/206-4X Ram Decode Configuration

TM 990/206 EXPANSION MEMORY BOARD

SPECIFICATIONS

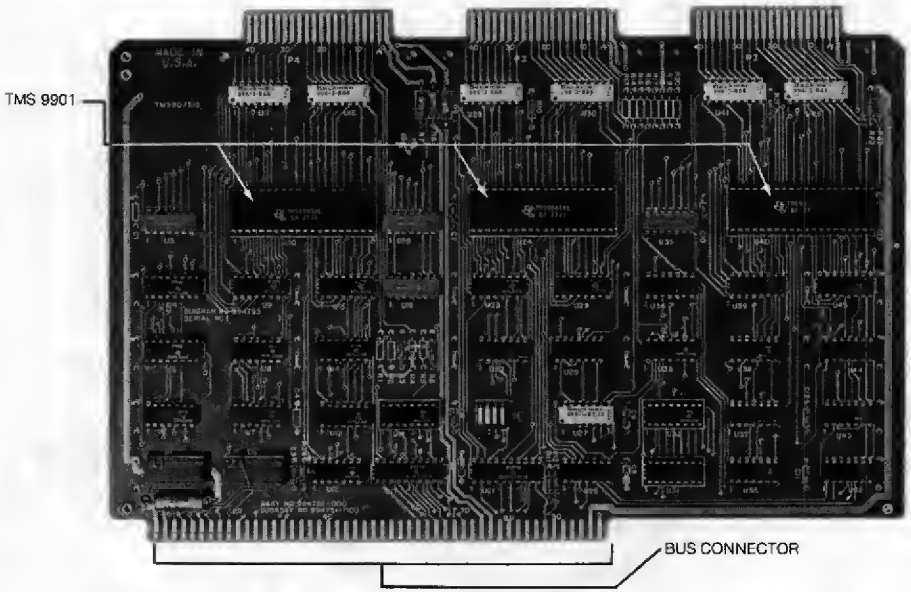
	TM 990/201			*TM 990/206*	
Memory Configuration	TMS 4045-45, 1K x 4 static RAM TMS 2716, 2K x 8 EPROM			TMS 4045-45, 1K x 4 static RAM	
Typical Power Requirements for Various Model Numbers	−41	−42	−43	−41	−42
5V ± 5%	1.0A	1.4A	2.15A	1.3A	2.15A
12V ± 5%	0.16A	0.225A	0.475A	Not required	Not required
−12V ± 5%	0.05A	0.125A	0.225A	Not required	Not required
Cycle Time					
Memory Device Access Time	450 ns	300 ns		200 ns	150 ns
Memory Cycle Time @ 3 MHz	1.0 μs	0.667 μs		0.667 μs	0.667 μs
Bus Interface					
Data and Address	Three-state TTL-compatible				
Control	TTL-compatible				
Mating Connector 100 pin, 0.125 inch (3,175 mm) centers	TI H431111-50 (wire wrap), TI H431121-50 (solder tail), or Viking 3VH50/1CN5 (pierced tail)				
Operating Temperature Range	0°C to 70°C				
Physical Characteristics					
Width	11 inches (279,4 mm)				
Height	7½ inches (190,5 mm)				
Board thickness	0.062 inch (1,575 mm)				
Component height	0.40 inch (10,16 mm)				

ORDERING INFORMATION

TM 990/201-41	4K x 16 EPROM, 2K x 16 RAM, half socketed
TM 990/201-42	8K EPROM, 4K RAM, fully socketed
TM 990/201-43	16K EPROM, 8K RAM, fully socketed
TM 990/206-41	4K x 16 RAM, sockets for 8K x 16 memory
TM 990/206-42	8K x 16 RAM, fully socketed

8

TM 990/310 I/O EXPANSION MODULE



The TM 990/310 is a fully assembled, fully tested, input/output expansion module compatible with all TM 990 family microcomputer modules. The TM 990/310 offers a maximum I/O expansion capability of 48 I/O points, programmable as either inputs or outputs.

FEATURES

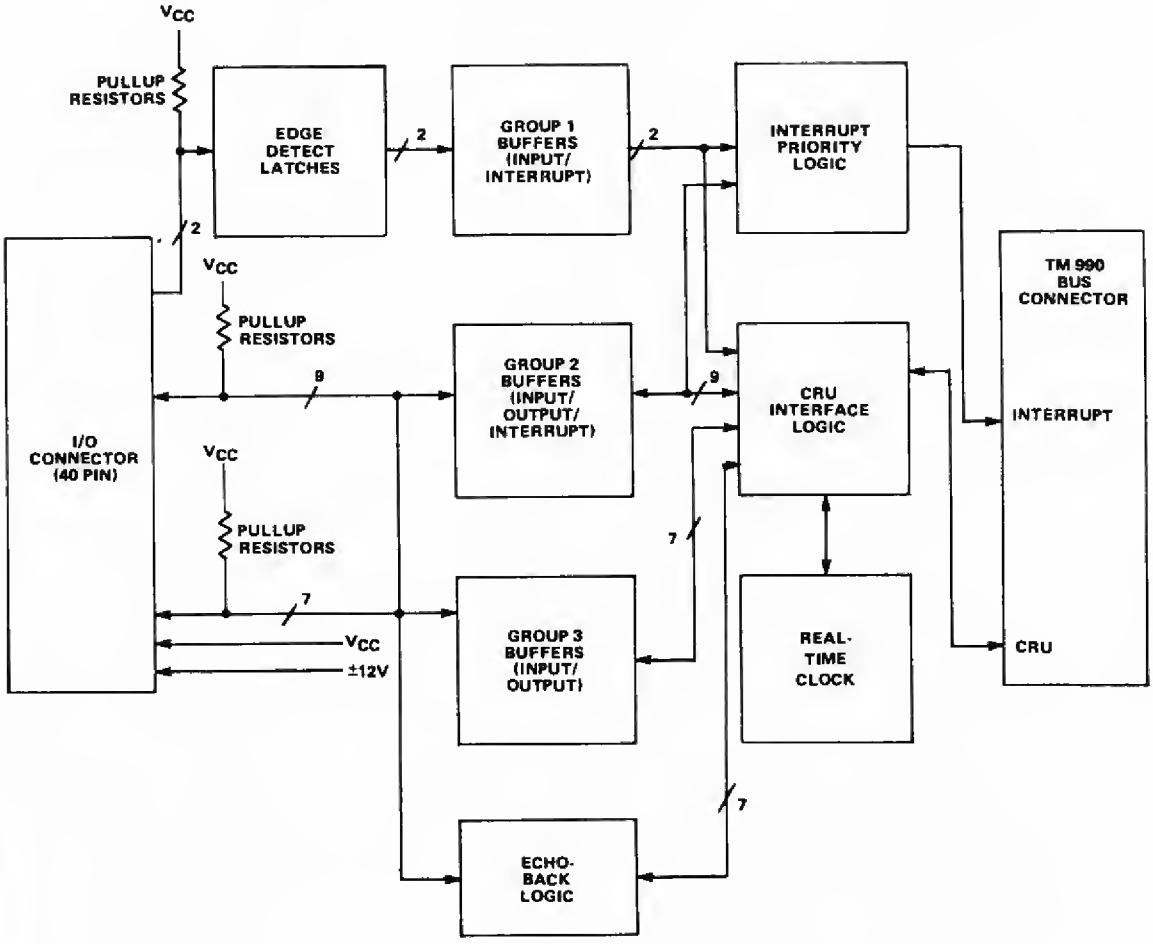
- Compatible with the TM 990 microcomputer module CRU bus
- Designed to fit the TM 990/510 card cage
- Inputs/outputs are TTL-compatible
- May be used with solder, wire wrap, or ribbon cable edge connectors
- Up to 27 I/O lines may be programmed as prioritized, unlatched interrupts
- Three (+) and three (-) edge-triggered and latched, prioritized interrupt inputs are provided (in addition to 48 I/O lines)
- Contains three real-time clocks (or event timers)
- I/O lines are provided with echo-back feature

OPERATION

The TM 990/310 input/output expansion module is implemented using the TM 990 printed circuit format. The TM 990/310 uses three TMS 9901 LSI, programmable, systems interface chips to control I/O. The extreme versatility and low cost of the TM 990/310 module makes it usable in a wide variety of I/O applications. Inputs and outputs may be mixed in any proportion, and any number of interrupts may be utilized, up to a maximum of 33. The interrupt priority encoding scheme also permits use of the module as an interrupt expander for the TM 990/100M microcomputer family.

TM 990/310 I/O EXPANSION MODULE

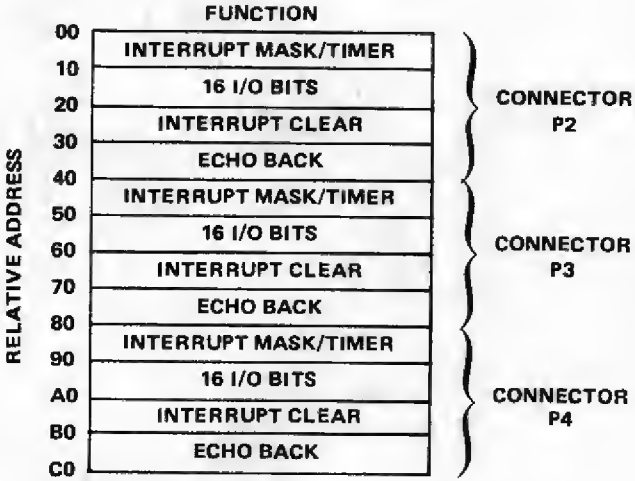
The TM 990/310 expansion module contains three I/O logic groups, each of which interfaces to separate connectors with 16 I/O lines. (Signal and ground are routed for each I/O line, and each line is equipped with pullup resistors.) Each I/O group may be programmed as 16 inputs, 16 outputs, nine interrupts, or any combination thereof. Each of the output lines is equipped with an echo-back feature which enables the user to read back each bit as it is written to a given output point. In addition, each connector contains a rising edge detect interrupt input and a falling edge detect interrupt input, along with +5 volts, +12 volts, and −12 volts power supply connections.



TM 990/310 Expansion Module, 16 I/O Lines,
Logic Group Block Diagram
(One of three groups)

ADDRESSING

The TM 990/310 I/O expansion module is addressed via the dedicated CRU interface over the system bus connector. Each I/O bit can be addressed individually; or up to 16 parallel ports can be addressed. Each 16-bit I/O line logic group has an addressing block of 64 bits, and each group can be stacked back to back. Each connector appears exactly the same; the functions and relative addresses for one TM 990/310 is shown below. The CRU address map permits addressing of 4K individual addresses. Any CRU bit beginning with 100_{16} can be addressed; the first FF_{16} bits are dedicated to the microcomputer module.



CRU ADDRESS MAP

▶8

SPECIFICATIONS

Input/Output	48 bits programmed as inputs, outputs, or up to 27 unlatched interrupts
Interrupts	33 maximum [six are (+) or (−) edge-detect latches] output of priority encoders may be jumpered to three levels of the 15 external TM 990 interrupts levels.
Interval Timers	Three 14-bit timers
Resolution	21.3μs
Maximum interval (for 3 MHz CPU clock)	349μs
Input Levels	
High-level input voltage	2.0 V nominal
Low-level input voltage	0.8 V nominal
Maximum input voltage range	−0.3 V to +10 V
Input current	10 kΩ (± 10%) pullup to V_{cc}
Edge Detect Interrupts	
Positive-going threshold voltage	1.9 V maximum
Negative-going threshold voltage	0.5 V minimum
Hysteresis	0.4 V minimum, 0.8 V typical
Maximum input voltage range	−0.3 V to +5.5 V
High-level input current	−1.29 mA maximum at 2.7 V
Low-level input current	−2.85 mA maximum at 0.4 V
Outputs	
High-level output voltage	2.4 V minimum at −200 μA 2.0 V minimum at −600 μA
Low-level output voltage	0.4 V maximum at 2.2 mA
Input/Output Connectors	
40 Pin (3 each)	TI H31120 (wire wrap), Viking 3 VH20/1JN5 (solder tail), 3M 3464-0001 (ribbon cable), or equivalents.
100 pin	TI H43111150 (wire wrap), TI H431121-50 (solder tail), Viking 3 VH50/ICN5 (pierced tail), or TM 990/510 card cage
Power Requirements	5 V ± 5%, 800 mA (typical)
Temperature Range	
Operating	0°C to 70°C
Storage	−65°C to 150°C
Physical Characteristics	
Width	11 inches (279,4 mm)
Height	7½ inches (190,5 mm)
Board thickness	0.062 inch (1,575 mm)
Component height	0.5 inch (12,7 mm) maximum

8◀

FEATURES

- Hexadecimal pushbutton keyboard
- Register, Memory, or CRU Display and entry keys
- Operations under TIBUG monitor
- 4 digit hexadecimal display of address and data
- Execution, single instruction and conversion keys

8 The TM 990/301 is a microterminal designed to interface with the TM 990 series of microcomputer modules. The microterminal's communications link to the TM 990 CPU module is via the EIA type cable and the serial terminal interface. The TM 990/301 performs the front panel functions of the microcomputer system, giving the programmer the ability to display and change register and memory information. This low cost terminal offers the capability to enter short programs in hexadecimal or alter a section of a longer sequence.

OPERATION

The TM 990/301 operates under control of the TM 990/401 TIBUG monitor. The data rate utilized is 110 baud. Once the CPU board is initialized, depressing the clear (CLR) key transfers TIBUG monitor control to the microterminal. The TIBUG software will enter a wait routine unless performing a function defined by the microterminal. Depressing the run (RUN) key will cause the CPU module to begin program execution and it will ignore other key depressions until the halt (HALT/SIE) key is depressed. If the CPU is halted, depression of the single instruction execution key (HALT/SIE) causes execution of the next instruction.

The display of the microterminal is divided into two 4 hexadecimal digit banks. The left bank displays address register information and the right bank displays data registers (see *Figure 1*).

KEY	*FUNCTION*
CLR	Clear — blank all displays — initialize software
RUN	Run — TM 990 CPU begins program execution; "RUN" is displayed in data digits.
HALT/SIE	Halt/Single Instruction Execution — If in run mode, halts CPU execution — address of next instruction displayed in address digits. If CPU is halted, one single instruction will be executed. Address display indicates address of next instruction; data display indicates contents of that location.
O—F	Hexadecimal digits (0—15) — data entry. F/– also indicates negative.
EPC	Enter Program Counter — Enter 4 digits, key depressions alters active program counter, data display indicates entered value.
DPC	Display Program Counter — Active PC register indicated in data display.
EST	Enter Status Register — Enter 4 digits — key depressions alters active status register data display indicates entered value.
DST	Display Status Register — Active status register indicated in data display.
EWP	Enter Workspace Pointer — Enter 4 digits — key depression alters active workspace pointer — data display indicates entered value.
DWP	Display Workspace Pointer — Active WP indicated in data displayed.
EMA	Enter Memory Address — Enter 4 digits — key depression will shift display of digit from data display to address display. Contents of the new memory address will then be indicated in the data display.
EMD	Enter Memory Data — After EMA function has been executed, enter 4 digits — the data display indicates the new data — key depression alters the data at the displayed memory address.

8◀

KEY	*FUNCTION*
EMDI	Enter Memory Data/Increment — Functions the same as EMD — after key depression of EMDI and data update the address display will automatically increment by 2 and the new addresses contents will be indicated by the data display. To increment the address register without altering data contents, depress EMDI key without entering new digit information.
DCRU	Display CRU Data — Enter 4 digits — the first digit specifies the CRU bit count; the remaining 3 digits specify the CRU address. Key depression shifts the entered digits to the address display and indicates the contents of that address in the data display. All 16 bits will be displayed.
ECRU	Enter CRU Data — After DCRU function has been executed, enter 4 digits — the new data is now indicated by the data display — key depression alters the data at the specified CRU address — only the number of bits specified will be altered.
H→D	Hexadecimal to Decimal Conversion — Enter 4 digits — key depression will indicate the decimal equivalent in 5 rightmost display digits.
D→H	Decimal to Hexadecimal Conversion — Enter 6 digits — the first digit designates the sign F/ — = negative, 0 = positive) the remaining 5 are decimal data — key depression displays hexadecimal equivalent in 4 right digits.

▶8

SPECIFICATIONS

Display

8 digit hexadecimal display
4 left digits indicate address register
4 right digits indicate data register

Keyboard

16 data keys
16 function keys
8 keys not connected

Interface: Serial Asynchronous Interface
Signals Include
/HALT
TERMINAL DATA IN
TERMINAL DATA OUT
+12 V
GND
−12 V
+5

Power Requirements: Supplied through cable
+12 V @ 50 mA
−12 V @ 20 mA
− 5 V @ 150 mA

Operating Temperature Range
0°C to 50°C

Physical Dimensions
Height: 5.8 inches
Width: 3.2 inches
Thickness: 1.38 inches
Cable Length: 6 ft.

Ordering Information
TM 900/301

Microterminal compatible with all TM 990 series Microcomputer modules.

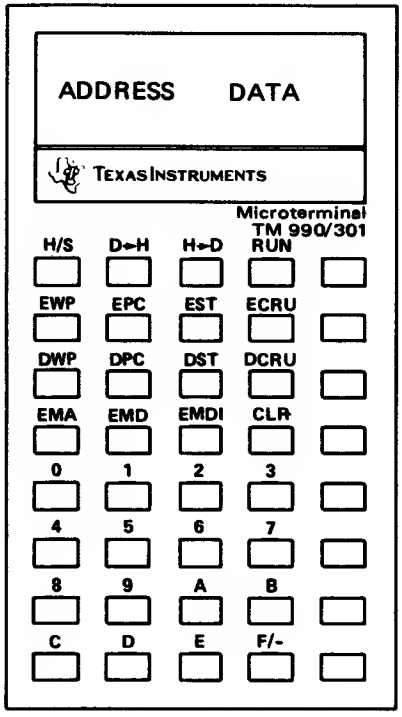


Figure 1. Microterminal Keyboard and Display.

8◀

TIBUG

TM 990/401 (TIBUG) is a comprehensive, interactive debug monitor which is included in the basic price of the CPU modules. (The OEM optionally may order the board with blank EPROM). TIBUG includes 13 user commands plus six user accessible utilities. TIBUG operates with 110, 300, 1200, and 2400 baud terminals. The user's manual for each CPU board includes a complete description of the use of TIBUG as well as a complete listing of the monitor. The TM 990/100M board TIBUG software is slightly different from the TM 990/180M board TIBUG software. Therefore, the TM 990/401-1 software is compatible with the TM 990/100M board, and the TM 990/401-2 software is compatible with the TM 990/180M board, and the TM 990/401-3 is compatible with the TM 990/101M board.

TIBUG Commands

B Executive and breakpoint on a specified address.
C Inspect/change the communications register unit.
D Dump memory to cassette or paper tape in 990 compatible tag format compatible with PX990, the TMS 9900 software development system.
E Execute
F Find word/byte
H Hexadecimal arithmetic
L Load memory from cassette or paper tape (compatible with TMS 9900 prototyping system formats).
M Inspect/change memory
R Inspect/change user program counter, workspace pointer, and status register.
S Execute user program in single/multiple steps.
W Inspect/change user register file.

User Accessible Utilities

- Read a character from the UART
- Write a character to the UART
- Hexadecimal numeric input
- Four-digit hexadecimal numeric output
- Single-digit hexadecimal numeric output
- ASCII message output

TM 990/402 LINE BY LINE ASSEMBLER

Line By Line Assembler

TM 990/402 is a line by line assembler supplied preprogrammed into a TMS 2708 EPROM Kit for immediate use in the system. These EPROM's insert into the extra sockets not required for the TIBUG monitor. It is an extremely useful tool for assembly language input of short programs or easy patching of longer programs.

The line by line assembler can be directly accessed from the TIBUG monitor by utilizing the "R" command and entering the proper program counter value. A complete User's Manual is included with the TMS 990/402 and a source listing can also be obtained. The TM 990/402-1 software is compatible with both the TM 990/100M and TM 990/101M board, and the TM 990/402-2 software is compatible with the TM 990/180M board.

Line By Line Assembler Input Commands

$ Convert symbolic constants from ASCII to hexadecimal and store in memory.
+,- Enter numeric constant
/ Change program counter
ESC Return control to monitor

Assembly Instruction.

Enter the instruction mnemonic and operand field — all allowable TMS 9900 instructions and addressing modes are recognized. Displacements are allowed on either an absolute basis or relative with respect to the current instruction designated by $-n. The displacement range is + 254, -256 bytes.

Assembler Error Messages

*S Syntax error
*D Displacement error jump target address
*R Range error — current field has erroneous input

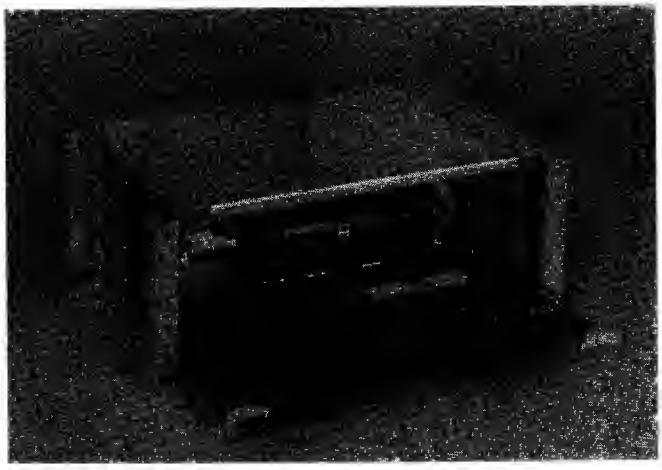
8◀

CARD CAGE

The TM 990/510 is an OEM card cage with four slots on 1 inch centers. The backpanel contains the address bus, data bus, interrupt and control lines to permit memory, I/O and DMA expansion of CPU modules. A 10-terminal barrier strip is mounted on the backpanel to permit connection of the following signals as the system requires:

- Reset
- Restart
- Power Down Interrupt
- ± 5V
- ± 12V
- ± 15 V
- GND

The outside dimensions of the OEM card cage are 5 inches (127 MM) high, 12.5 inches (317,5mm) wide, and 8 inches (203,2 mm) deep.



TM 990/510 OEM Card Cage

EXTENDER BOARD

The TM 990/511 is a connector bus compatible extender board. The extender board has a printed circuit tab and a connector with card guide at its edges.

PROTOTYPING BOARD

The TM 990/512 is a universal prototyping card. The printed circuit board is designed to accommodate 0.3, 0.4, 0.6 and 0.9 inch (7,62, 16,16, 15,24, and 22,86 mm) wide, dual in line, IC packages or their equivalent soldertail or wirewrap sockets. The TM 990/512 has GND and 5 V planes, two power strips for ± 12 V, plus two power strips for user-selectable voltage option.

CONNECTORS AND CABLES

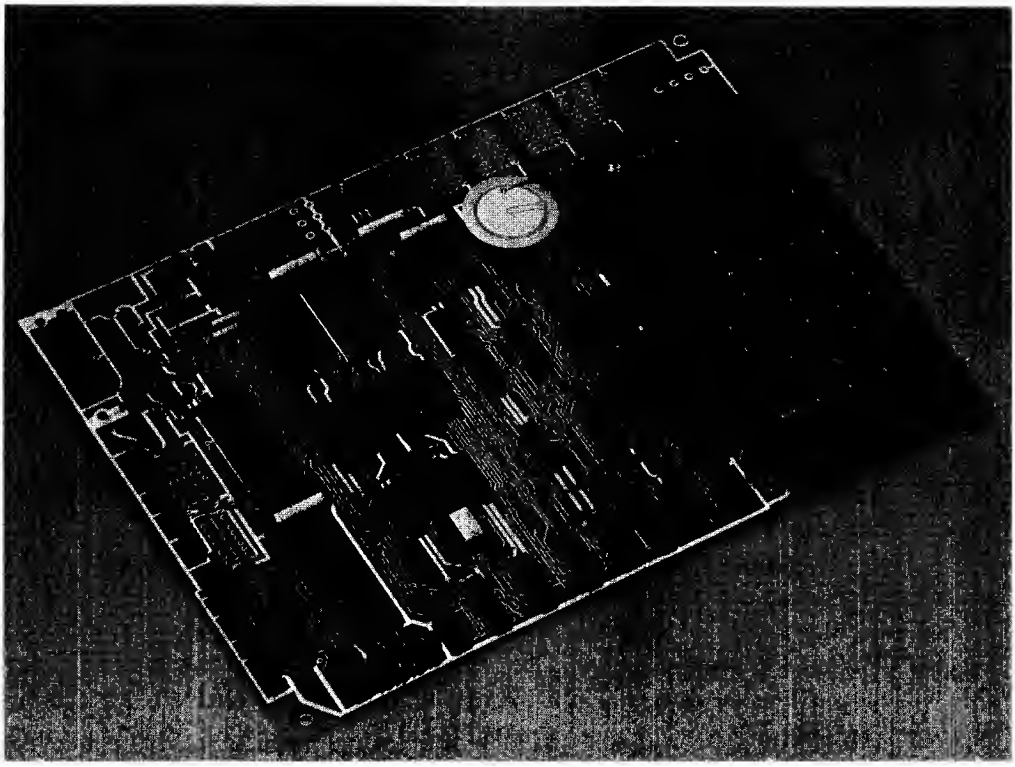
8◀

The bottom edge bus connector is a 100-pin printed circuit tap on 0.125 inch (3,175 mm) centers, compatible with the TM 990 bus specification. Two top edge, 40-pin, printed circuit tab connectors for I/O or interrupt are on 0.10 (2,54 mm) centers. Additionally, the CPU modules come mounted with a 25-pin EIA connector, and the universal prototyping boards have space for mounting the same. The TM 990/501 is a connector kit offered for all of the connectors. The parts supplied in this kit are readily available, multisource connectors. The connectors available through TI as part of the TM 990/501 kit are

QTY	*Description*	*Part Number*
1	25 pin male connector	AMP DB-25P or ITT Cannon DB-25P
1	40 pin solder eyelet female edge connector	Viking 3VH20/1JN5 or TI H421121-20
1	100 pin solder wire wrap female edge connector	Viking 3VH50/1CN5 or TI H431111-50
1	Hood for 25 pin connector	AMP 206478-3

Various cable kits are also offered. TM 990/502 is a general purpose EIA cable compatible with terminals which have female RS-232 connectors. TM 990/503 is an EIA cable compatible with 743 KSR or 745 terminals. The TM 990/504 is a standard current loop TTY cable. The TM 990/505 is compatible with the 733 ASR.

TM990/189 UNIVERSITY MICROCOMPUTER BOARD



TM990/189 University Microcomputer Board

The TM990/189 is a self-contained microcomputer system (available in kit form or fully assembled and tested) which is designed primarily as a learning tool for the engineer, student or hobbyist. It can be used as an aid in the instruction of microcomputer fundamentals, machine and assembly level language programming and microcomputer interfacing as well as demonstrating the power of the 9900 family 16-bit architecture. The board utilizes the powerful NMOS 16 bit TMS 9980 microprocessor as its CPU. Additionally, this extremely economical board has the following exciting features: 8◀

SOFTWARE

- Unibug—user interactive software debug monitor, ROM resident.
- Line by Line Assembler with forward references—assembles 9900 instructions into machine level code and allows for the use of labels.

TEXTS

- Tutorial text—suitable as a 3-hour university course outline or as a stand-alone self-paced programmed learning text. Includes experiments, applications, problems and solutions.
- Hardware reference manual—describes theory of operations, connection of hardware options (for example, expansion memory, I/O expansion, audio cassette interface, and RS232C or TTY options) and kit assembly procedures.
- System Design Handbook—design handbook featuring application and design references to all members of the TMS 9900 family.

HARDWARE

- Power Supply—included as a standard or available as an option.
- Display—10 digit seven segment display.
- Keyboard—45 keys, full alpha-numeric keyboard.
- ROM—4K bytes of dedicated ROM, sockets for additional 2k on-board expansion.
- RAM—1K bytes RAM including sockets for additional 1K on-board expansion.
- Memory Expansion—50-pin connector provided to expand memory to 16K bytes.
- I/O Expansion—50-pin connector is provided with all standard CPU bus signals accessible, including extra pins for user-defined functions.
- EIA Connector—25-pin standard EIA connector is provided for interface to RS232C or 20 mA TTY loop.
- DMA—Direct Memory Access. HOLD and HOLD ACKNOWLEDGE are brought out to external pins.
- Visual Indicators—7 LED's are provided: 3 dedicated and 4 user-defined.
- Acoustical Indicator—Piezoelectric disk for audio reference.
- Audio Cassette—single audio cassette interface.
- I/O—16 I/O lines are provided.

ORDERING INFORMATION

TM990/189 K	Kit form less power supply
TM990/189 M	Assembled and tested microcomputer module.
TM990/519	Power supply for TM990/189 K or TM990/189 M including all interconnecting cables.

Memory

8

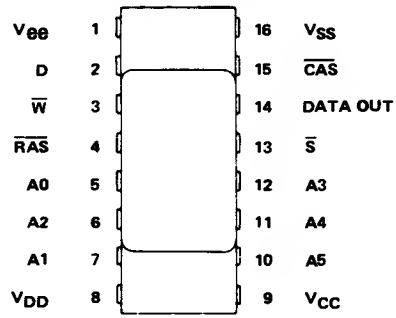
TMS 4027 JL, NL
4096-BIT DYNAMIC RANDOM-ACCESS MEMORY

- **4096 X 1 Organization**
- **Industry Standard 16-Pin 300-Mil Package Configuration**
- **10% Tolerance on All Supplies**
- **All Inputs Including Clocks TTL Compatible**
- **Three-State Fully TTL-Compatible Output Latched and Valid Into Next Cycle**
- **3 Performance Ranges:**

	ACCESS TIME ROW ADDRESS (MAX)	ACCESS TIME COLUMN ADDRESS (MAX)	READ OR WRITE CYCLE (MIN)	READ, MODIFY-WRITE† CYCLE (MIN)
TMS 4027-15	150 ns	100 ns	320 ns	330 ns
TMS 4027-20	200 ns	135 ns	375 ns	420 ns
TMS 4027-25	250 ns	165 ns	375 ns	480 ns

- **Page-Mode Operation for Faster Access Time**
- **Low-Power Dissipation**
  - **– Operating 460 mW (max)**
  - **– Standby 27 mW (max)**
- **1-T Cell Design, N-Channel Silicon-Gate Technology**
- **Refresh time: 2 ms**

16-PIN CERAMIC DUAL-IN-LINE PACKAGE (TOP VIEW)



PIN NAMES

A0–A5	Address Inputs
$\overline{CAS}$	Column address strobe
D	Data input
DATA OUT	Deta output
$\overline{RAS}$	Row address strobe
$\overline{S}$	Chip select
$\overline{W}$	Write enable
V_{BB}	–5 V power supply
V_{CC}	+5 V power supply
V_{DD}	+12 V power supply
V_{SS}	0 V ground

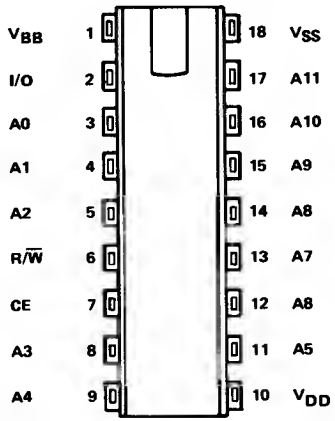
TMS 4050 JL, NL
4096-BIT DYNAMIC RANDOM-ACCESS MEMORIES

- 4096 x 1 Organization
- 18-Pin 300-Mil Package Configuration
- Multiplexed Data Input/Output
- 3 Performance Ranges:

	ACCESS TIME (MAX)	READ OR WRITE CYCLE (MIN)	READ, MODIFY WRITE CYCLE (MIN)
TMS 4050	300 ns	470 ns	730 ns
TMS 4050-1	250 ns	430 ns	660 ns
TMS 4050-2	200 ns	400 ns	600 ns

- Full TTL Compatibility on All Inputs (No Pull-up Resistors Needed)
- Registers for Addresses Provided on Chip
- Open-Drain Output Buffer
- Single Low-Capacitance Clock
- Low-Power Dissipation
  - 420 mW Operating (Typical)
  - 0.1 mW Standby (Typical)
- N-Channel Silicon-Gate Technology
- Refresh time: 2 ms or less

18-PIN CERAMIC AND PLASTIC DUAL-IN-LINE PACKAGES (TOP VIEW)



VDD +12V power supply
VSS 0V ground
VBB – 5V power supply

8

TMS 4051 JL, NL
4096-BIT DYNAMIC RANDOM ACCESS MEMORIES

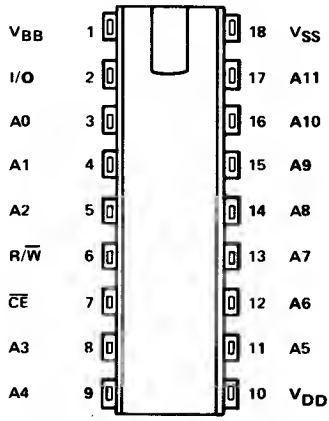
MOS LSI

- 4096 x 1 Organization
- 18-Pin 300-Mil Package Configuration
- Single Low-Capacitance TTL-Compatible Clock
- Multiplexed Data Input/Output
- 2 Performance Ranges:

	ACCESS TIME (MAX)	READ OR WRITE CYCLE (MIN)	READ, MODIFY WRITE CYCLE (MIN)
TMS 4051	300 ns	470 ns	730 ns
TMS 4051-1	250 ns	430 ns	660 ns

- Full TTL Compatibility on All Inputs (No Pull-up Resistors Needed Except with $\overline{CE}$)
- Registers for Addresses Provided on Chip
- Open-Drain Output Buffer
- Low-Power Dissipation
  - 460 mW Operating (Typical)
  - 60 mW Standby (Typical)
- N-Channel Silicon-Gate Technology
- Refresh time: 2 ms or less

18-PIN CERAMIC AND PLASTIC DUAL-IN-LINE PACKAGES (TOP VIEW)



V_{DD} +12V power supply
V_{SS} 0V ground
V_{BB} − 5V power supply

TMS 4060 JL, NL
4096-BIT DYNAMIC RANDOM-ACCESS MEMORIES

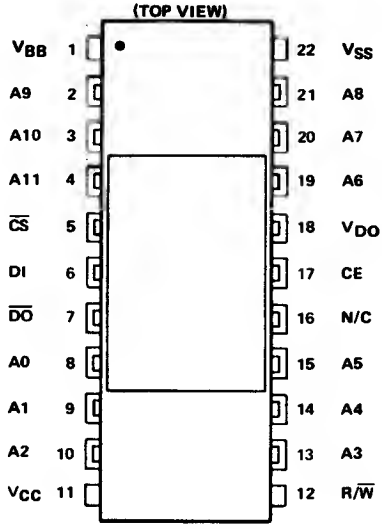
MOS LSI

- 4096 x 1 Organization
- 3 Performance Ranges:

	ACCESS TIME (MAX)	READ OR WRITE CYCLE (MIN)	READ, MODIFY WRITE CYCLE (MIN)
TMS 4060	300 ns	470 ns	710 ns
TMS 4060-1	250 ns	430 ns	640 ns
TMS 4060-2	200 ns	400 ns	580 ns

- Full TTL Compatibility on All Inputs Except CE (No Pull-Up Resistors Needed)
- Low Power Dissipation
  - 400 mW Operating (Typical)
  - 0.2 mW Standby (Typical)
- Single Low-Capacitance Clock
- N-Channel Silicon-Gate Technology
- 22-Pin 400-Mil Dual-in-Line Package
- Refresh time: 2 ms or less

22-PIN CERAMIC ANO PLASTIC DUAL-IN-LINE PACKAGES (TOP VIEW)



V_{CC} + 5V power supply
V_{SS} 0V ground
V_{DD} +12V power supply
V_{BB} − 5V power supply

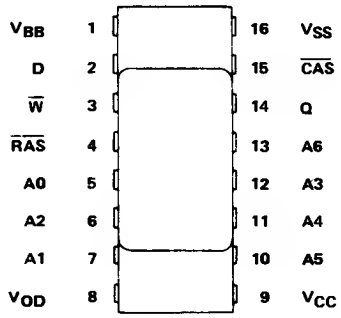
TMS 4116 JL
16,384-BIT DYNAMIC RANDOM-ACCESS MEMORY

- 16,384 X 1 Organization
- 10% Tolerance on All Supplies
- All Inputs Including Clocks TTL Compatible
- Unlatched Three-State Fully TTL-Compatible Output
- 3 Performance Ranges:

	ACCESS TIME ROW ADDRESS (MAX)	ACCESS TIME COLUMN ADDRESS (MAX)	READ OR WRITE CYCLE (MIN)	READ, MODIFY-WRITE† CYCLE (MIN)
TMS 4116-15	150 ns	100 ns	375 ns	375 ns
TMS 4116-20	200 ns	135 ns	375 ns	375 ns
TMS 4116-25	250 ns	165 ns	410 ns	515 ns

- Page-Mode Operation for Faster Access Time
- Common I/O Capability with "Early Write" Feature
- Low-Power Dissipation
  - Operating 462 mW (max)
  - Standby 20 mW (max)
- 1-T Cell Design, N-Channel Silicon-Gate Technology
- 16-Pin 300-Mil Package Configuration
- Refresh time: 2 ms or less

16-PIN CERAMIC
DUAL-IN-LINE PACKAGE
(TOP VIEW)



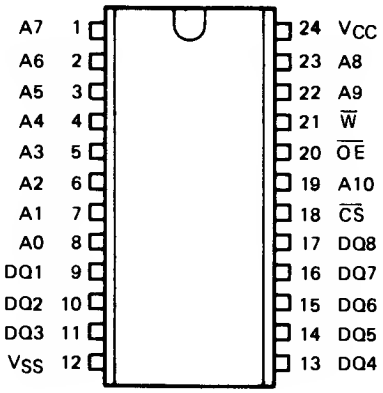
PIN NOMENCLATURE			
A0-A6	Address Inputs	$\overline{W}$	Write Enable
$\overline{CAS}$	Column address strobe	V_{BB}	−5 V power supply
D	Data input	V_{CC}	+5 V power supply
Q	Data output	V_{DD}	+12 V power supply
$\overline{RAS}$	Row address strobe	V_{SS}	0 V ground

8◂

TMS 4016 JL, NL
2048-WORD BY 8-BIT STATIC RAM

- **2K × 8 Organization**
- **Single +5 V Supply (±10% Tolerance)**
- **Fully Static Operation (No Clocks, No Refresh)**
- **JEDEC Proposed Standard Pinout**
- **24-Pin 600 Mil Package Configuration**
- **Plug-in Compatible with 16K 5V EPROMs**
- **8-Bit Output for Use in Microprocessor-Based Systems**
- **Max Access/Min Cycle Times Down to 150 ns**
- **Tri-State Outputs with $\overline{CS}$ for Or-ties**
- **$\overline{OE}$ Eliminates Need for External Bus Buffers**
- **Common I/O Capability**
- **All Inputs and Outputs Fully TTL Compatible**
- **Fanout to Series 74, Series 74S, or Series 74LS TTL Loads**
- **N-Channel Silicon-Gate Technology**
- **Power Dissipation Under 495 mW Max**
- **Guaranteed dc Noise Immunity of 400 mV with Standard TTL Loads**

TMS 4016
24-PIN CERAMIC AND PLASTIC DUAL-IN-LINE PACKAGE
(TOP VIEW)



PIN NOMENCLATURE	
A0-A10	Addresses
DQ1-DQ8	Data In/Data Out
$\overline{CS}$	Chip Select
$\overline{OE}$	Output Enable
$\overline{W}$	Write Enable
V_{SS}	Ground
V_{CC}	+5 V Supply

description

The TMS 4016 static random-access memory is organized as 2048 words of 8 bits each. Fabricated using proven N-channel, silicon-gate MOS technology, the TMS 4016 operates at high speeds and draws less power per bit than 4K static RAMs. It is fully compatible with Series 74, 74S, or 74LS TTL. Its static design means that no refresh clocking circuitry is needed and timing requirements are simplified. Access time is equal to cycle time. A chip select control is provided for controlling the flow of data-in and data-out and an output enable function is included in order to eliminate the need for external bus buffers.

Of special importance is that the TMS 4016 static RAM has the same standardized pinout as TI's compatible EPROM family. This, along with other compatible features, makes the TMS 4016 directly plug-in compatible with the TMS 2516 (or other 16K 5V EPROMs). No modifications are needed. This allows the microprocessor system designer complete flexibility in partitioning his memory board between read/write and non-volatile storage. A more detailed explanation of this compatibility is given on the reverse side.

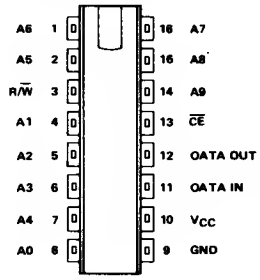
PREVIEW
This document contains the design specifications for a product under development. Specifications may be changed in any manner without notice.

TMS 4033, 4034, 4035, 4039, 4042, 4043, JL, NL 1024-BIT AND TMS 4036 NL 512-BIT STATIC RAMs

- No clocks — No refresh
- Input/Output fully TTL compatible
- Three-state output for OR-Tie capability
- Single 5-volt supply
- Simple, fully decoded addressing
- Reliability, same process as TI's industry standard 4K RAMs
- Economy, high volume production techniques and choice of plastic or ceramic packaging
- Super low standby power (typical 2-3 mW)
- Wide range of speeds for design optimization
- Easy to use 8-bit byte organization (64 x 8) plus industry standard 256 x 4 and 1024 x 1

TMS 4033, TMS 4034, TMS 4035

16-PIN CERAMIC AND PLASTIC DUAL-IN-LINE PACKAGES (TOP VIEW)

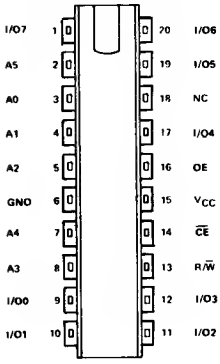


- 1024 x 4 x 1 Organization
- 225 mW Typical power dissipation

Part Number	Max. Access/ Min. Cycle
TMS 4033/2102 1 JL, NL	450 ns
TMS 4034/2102-2 JL, NL	650 ns
TMS 4035/2102 JL, NL	1000 ns

TMS 4036

20 PIN PLASTIC DUAL IN LINE PACKAGES (TOP VIEW)

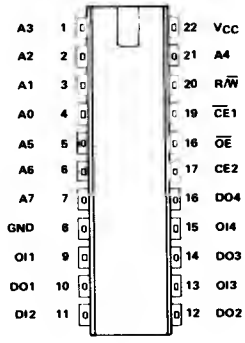


- 64 x 8 Organization
- 250 mW Typical power dissipation
- On-chip multiplexed I/O for microprocessor-oriented systems

Part Number	Max. Access/ Min. Cycle
TMS 4036 NL	1000 ns
TMS 4036 1 NL	650 ns
TMS 4036 2 NL	450 ns

TMS 4039

22 PIN CERAMIC AND PLASTIC DUAL-IN-LINE PACKAGES (TOP VIEW)

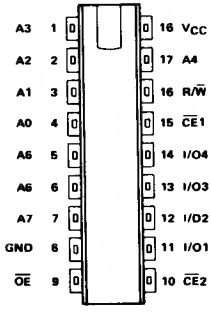


- 256 x 4 Organization
- 175 mW Typical power dissipation
- Separate input and output
- 2 chip enables for maximum control

Part Number	Max. Access/ Min. Cycle
TMS 4039/2101 JL, NL	1000 ns
TMS 4039-1/2101-2 JL, NL	650 ns
TMS 4039 2/2101-1 JL, NL	450 ns

TMS 4042

18 PIN CERAMIC AND PLASTIC DUAL IN LINE PACKAGES (TOP VIEW)

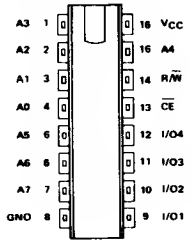


- 256 x 4 Organization
- 175 mW Typical power dissipation
- Common I/O for Bus-oriented systems

Part Number	Max Access/ Min Cycle
TMS 4042/2111 JL, NL	1000 ns
TMS 4042 1/2111 2 JL, NL	650 ns
TMS 4042 2/2111 1 JL, NL	450 ns

TMS 4043

16 PIN CERAMIC AND PLASTIC DUAL IN LINE PACKAGES (TOP VIEW)



- 256 x 4 Organization
- 175 mW Typical power dissipation
- Provides common I/O and highest packing density

Part Number	Max. Access/ Min. Cycle
TMS 4043/2112 JL, NL	1000 ns
TMS 4043 1/2112 2 JL, NL	650 ns
TMS 4043 2 JL, NL	450 ns

TMS 4044 JL, NL; TMS 40L44 JL, NL; TMS 4046 JL, NL; TMS 40L46 JL, NL 4096-WORD BY 1-BIT STATIC RAMs

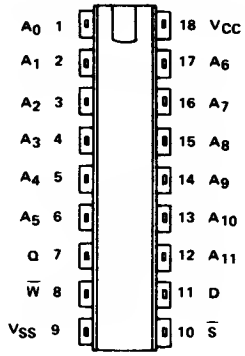
- **4096 × 1 Organization**
- **Single +5 V Supply (±10% Tolerance)**
- **High Density 300-mil 18- and 20-Pin Packages**
- **Fully Static Operation (No Clocks, No Refresh, No Timing Strobe)**
- **4 Performance Ranges:**

	ACCESS TIME (MAX)	READ OR WRITE CYCLE (MIN)
TMS 4044/L44-45, TMS 4046/L46-45	450 ns	450 ns
TMS 4044/L44-25, TMS 4046/L46-25	250 ns	250 ns
TMS 4044/L44-20, TMS 4046/L46-20	200 ns	200 ns
TMS 4044-15, TMS 4046-15	150 ns	150 ns

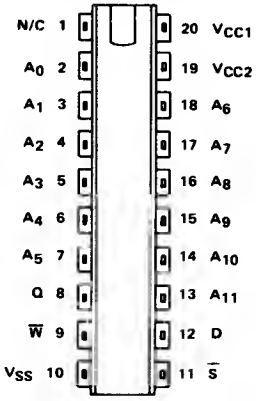
- **400 mV Guaranteed DC Noise Immunity with Standard TTL Loads — No Pull-Up Resistors Required**
- **Common I/O Capability**
- **3-State Outputs and Chip Select Control for OR-Tie Capability**
- **Fan-Out to 2 Series 74, 1 Series 74S, or 8 Series 74LS TTL Loads**
- **Low Power Dissipation**

	MAX (OPERATING)	MAX (STANDBY)
TMS 4044	440 mW	156 mW
TMS 40L44	275 mW	96 mW
TMS 4046	440 mW	13 mW
TMS 40L46	275 mW	13 mW

TMS 4044/TMS 40L44
18-PIN CERAMIC AND PLASTIC DUAL-IN-LINE PACKAGES
(TOP VIEW)



TMS 4046/TMS 40L46
20-PIN CERAMIC AND PLASTIC DUAL-IN-LINE PACKAGES
(TOP VIEW)



PIN NAMES

A_0-A_{11}	Addresses
D	Data In
Q	Data Out
$\overline{S}$	Chip Select
V_{CC} (TMS 4044/L44)	+5 V Supply
V_{CC1} (TMS 4046/L46)	+5 V Supply (array only)
V_{CC2} (TMS 4046/L46)	+5 V Supply (periphery only)
V_{SS}	Ground
$\overline{W}$	Write Enable

TMS 40L45 JL, NL; TMS 40L47 JL, NL
1024-WORD BY 4-BIT STATIC RAMs

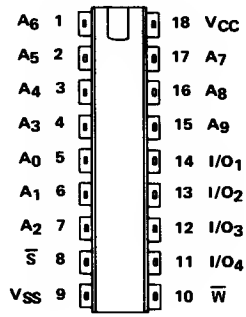
- 1024 x 4 Organization
- Single 10% Tolerance 5-V Supply
- High Density 300-mil 18- and 20-Pin Packages
- Fully Static Operation (No Clocks, No Refresh, No Timing Strobe)
- 3 Performance Ranges:

	ACCESS TIME (MAX)	READ OR WRITE CYCLE (MIN)
TMS 40L45-25, TMS 40L47-25	250 ns	250 ns
TMS 40L45-30, TMS 40L47-30	300 ns	300 ns
TMS 40L45-45, TMS 40L47-45	450 ns	450 ns

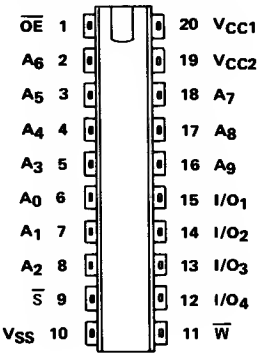
- 400-mV Guaranteed Noise Immunity With Standard TTL Loads – No Pull-Up Resistors Required
- Common I/O With Three-State Outputs and Chip Select Control for OR-Tie Capability
- Fan-Out to 1 Series 74 or 74S TTL Load – No Pull-Up Resistors Required
- Low Power Dissipation
  - 250 mW *Typical
  - 370 mW *Maximum
- Standby Power Dissipation (TMS 40L47)
  - 12 mW Typical
  - 24 mW Maximum

PIN NAMES	
A_0-A_9	Addresses
I/O_1-I/O_4	Data input/output
$\overline{OE}$	Output Enable
$\overline{S}$	Chip Select
V_{CC} (TMS 40L45)	+5-V Supply
V_{CC1} (TMS 40L47)	+5-V Supply (array only)
V_{CC2} (TMS 40L47)	+5-V Supply (periphery only)
V_{SS}	Ground
$\overline{W}$	Write Enable

TMS 40L45
18-PIN CERAMIC AND PLASTIC
DUAL-IN-LINE PACKAGES
(TOP VIEW)



TMS 40L47
20-PIN CERAMIC AND PLASTIC
DUAL-IN-LINE PACKAGES
(TOP VIEW)

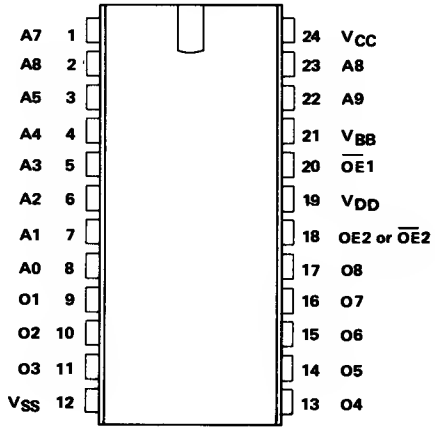


TMS 4700 JL, NL
1024-WORD BY 8-BIT READ-ONLY MEMORY

MOS LSI

- 1024 x 8 Organization
- All Inputs and Outputs TTL-Compatible
- Maximum Access Time . . . 450 ns
- Minimum Cycle Time . . . 450 ns
- Typical Power Dissipation . . . 310 mW
- 3-State Outputs for OR-Ties
- Output Enable Control
- Silicon-Gate Technology
- 8-Bit Output for use in Microprocessor Based Systems
- Pin-compatible with TMS 2708, TMS 27L08, and Intel 2308
- Inputs require external pull-up resistors for TTL-compatibility

24-PIN CERAMIC AND PLASTIC DUAL-IN-LINE PACKAGES (TOP VIEW)



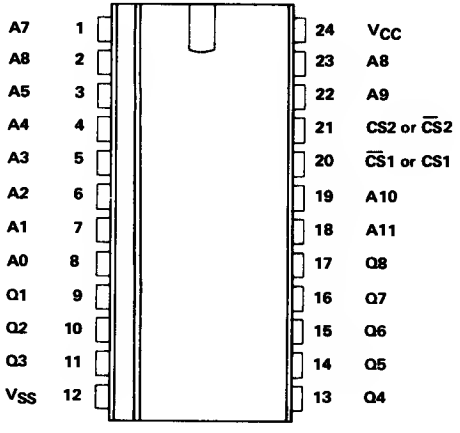
V_{BB}	−5V power supply
V_{CC}	+5V power supply
V_{DD}	+12V power supply
V_{SS}	OV ground

TMS 4732 JL, NL
4096-WORD BY 8-BIT READ-ONLY MEMORY

MOS LSI

- 4096 x 8 Organization
- All Inputs and Outputs TTL-Compatible
- Fully Static (No Clocks, No Refresh)
- Single 5 V Power Supply
- Maximum Access Time . . . 450 ns
- Minimum Cycle Time . . . 450 ns
- Typical Power Dissipation . . . 580 mW
- 3-State Outputs for OR-Ties
- Pin Compatible with TMS 4700, TMS 2708 and Intel 8316B
- Two Output Enable Controls for Chip Select Flexibility
- N-Channel Silicon-Gate Technology

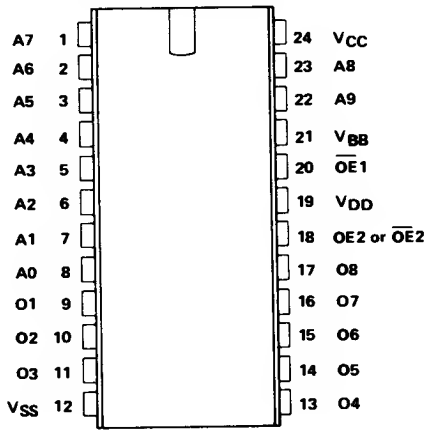
24-PIN CERAMIC AND PLASTIC DUAL-IN-LINE PACKAGES (TOP VIEW)



TMS 4710 JL, NL COMPLETE ASCII CHARACTER SET GENERATOR 5x7 CHARACTER, 8x8 BLOCK

- TMS 4710 (Standard TMS 4700 8K ROM)
- Full Upper and Lower Case ASCII Character Generator
- Ideal for Video Terminal Applications
- Fully Static Operation
- Block Size 8 x 8
- Character Size 5 x 7
- 1024 x 8 Organization
- All Inputs and Outputs TTL-Compatible
- Maximum Access Time . . . 450 ns
- Minimum Cycle Time . . . 450 ns
- Typical Power Dissipation . . . 310 mW
- 3-State Outputs for OR-Ties
- Output Enable Control
- Silicon-Gate Technology
- 8-Bit Output for use in Microprocessor Based Systems

24-PIN CERAMIC AND PLASTIC
DUAL-IN-LINE PACKAGES
(TOP VIEW)



TYPES SBP 8316, SBP 9818
16,384 I²L READ-ONLY MEMORIES

- Mask Programmable I²L ROM
- Fully TTL Compatible Inputs/Outputs
- Programmable Options Include:
  - User Selectable Speed/Power Operation:
    - Wide Range for Injector Current Supply Operation (SBP 9818)
    - Resistor Options for 5-Volt Supply Operation (SBP 8316)
  - Choice of Outputs:
    - Open-Collector for V_{CC} or INJ Operation
    - Internal 10K Ω Pull-Up Resistors to V_{CC} (SBP 8316)
  - Choose Any Combination of Up to 3 Boolean Variables for Chip Select or 2 Boolean Variables with Latched Outputs
- Industry Standard Pin Assignments in 24-Pin Plastic or C-DIP Packages
- Choice of Temperature Ranges:
  - SBP 8316CN, SBP 9818CN for 0 to 70°C Applications
  - SBP 8316MJ, SBP 9818MJ for −55°C to 125°C Applications
- Single + 5-V power supply for the SBP8316
- Injector current of 500 mA maximum for the SPB9818

J OR N PACKAGE
(TOP VIEW)

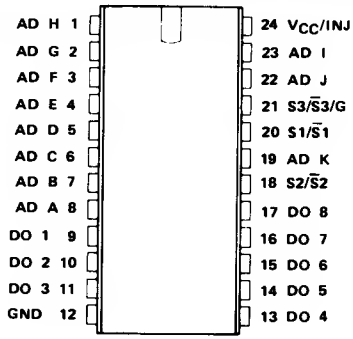
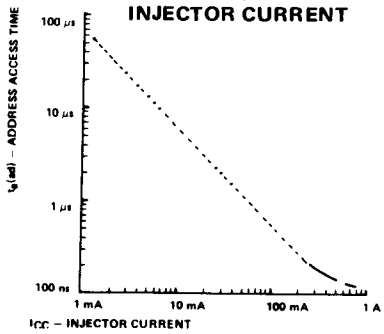


FIGURE 1
ADDRESS ACCESS TIMING
VS.
INJECTOR CURRENT

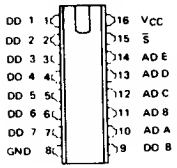


SERIES 54S/74S PROGRAMMABLE READ-ONLY MEMORIES

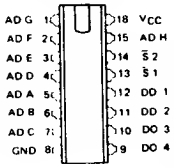
- Titanium-Tungsten (Ti-W) Fuse Links for Fast, Low-Voltage, Reliable Programming
- All Schottky-Clamped PROM's Offer:
  - Fast Chip Select to Simplify System Decode
  - Choice of Three-State or Open-Collector Outputs
  - P-N-P Inputs for Reduced Loading on System Buffers/Drivers
- Single 5-V power supply
- Full Decoding and Chip Select Simplify System Design
- Applications Include:
  - Microprogramming/Firmware Loaders
  - Code Converters/Character Generators
  - Translators/Emulators
  - Address Mapping/Look-Up Tables

TYPE NUMBER (PACKAGES)		BIT SIZE (ORGANIZATION)	OUTPUT CONFIGURATION	TYPICAL PERFORMANCE	
−55°C to 125°C	0°C to 70°C			ADDRESS ACCESS TIME	POWER DISSIPATION
SN54S188(J, W)	SN74S188(J, N)	256 bits (32 W x 8 B)	open-collector	25 ns	400 mW
SN54S288(J, W)	SN74S288(J, N)		three-state		
SN54S287(J, W)	SN74S287(J, N)	1024 bits (256 W x 4 B)	three-state	42 ns	500 mW
SN54S387(J, W)	SN74S387(J, N)		open-collector		
SN54S470(J)	SN74S470(J, N)	2048 bits (256 W x 8 B)	open-collector	50 ns	550 mW
SN54S471(J)	SN74S471(J, N)		three-state		
SN54S472(J)	SN74S472(J, N)	4096 bits (512 W x 8 B)	three-state	55 ns	600 mW
SN54S473(J)	SN74S473(J, N)		open-collector		
SN54S474(J, W)	SN74S474(J, N)	4096 bits (512 W x 8 B)	three-state	55 ns	600 mW
SN54S475(J, W)	SN74S475(J, N)		open-collector		

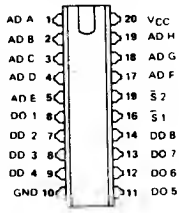
256 BITS (32 WORDS BY 8 BITS) 'S188, 'S288



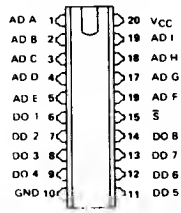
1024 BITS (256 WORDS BY 4 BITS) 'S287, 'S387



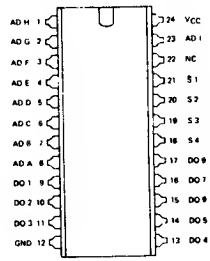
2048 BITS (256 WORDS BY 8 BITS) 'S470, 'S471



4096 BITS (512 WORDS BY 8 BITS) 'S472, 'S473



4096 BITS (512 WORDS BY 8 BITS) 'S474, 'S475



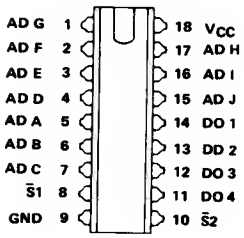
Pin assignments for all of these memories are the same for all packages.

8◀

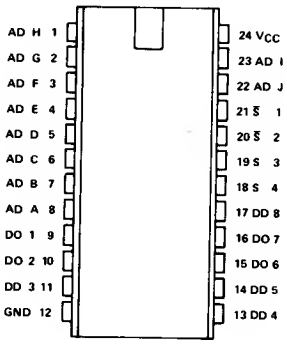
- Titanium-Tungsten (Ti-W) Fuse Links for Fast Low-Voltage Reliable Programming
- Full Decoding and Chip Select Simplify System Design
- Single 5-V power supply
- Fast Chip Select to Simplify System Decode
- Choice of Three-State or Open Collector Outputs
- PNP Inputs for Reduced Loading on System Buffers/Drivers
- Applications Include:
  Microprogramming/Firmware Loaders
  Code Converters/Character Generators
  Translators/Emulators
  Address Mapping/Look-Up Tables

TYPE NUMBER (PACKAGES)		BIT SIZE (ORGANIZATION)	OUTPUT CONFIGURATION	TYPICAL PERFORMANCE		
				ACCESS TIMES		POWER DISSIPATION
−55°C to 125°C	0°C to 70°C			ADDRESS	SELECT	
SN54S476(J)	SN74S476(J,N)	4096 bits	three-state	35 ns	15 ns	475 mW
SN54S477(J)	SN74S477(J,N)	1024 W x 4 B	open-collector			
SN54S478(J)	SN74S478(J,N)	8192 bits	three-state	45 ns	20 ns	600 mW
SN54S479(J)	SN74S479(J,N)	1024 W x 8 B	open-collector			

SN54S/74S476 3-S OUTPUTS
SN54S/74S477 0-C OUTPUTS
4096 BITS
(1024 WORDS BY 4 BITS)
'S476, 'S477



SN54S/74S478 3-S OUTPUTS
SN54S/74S479 0-C OUTPUTS
8192 BITS
(1024 WORDS BY 8 BITS)
'S478, 'S479



▸8

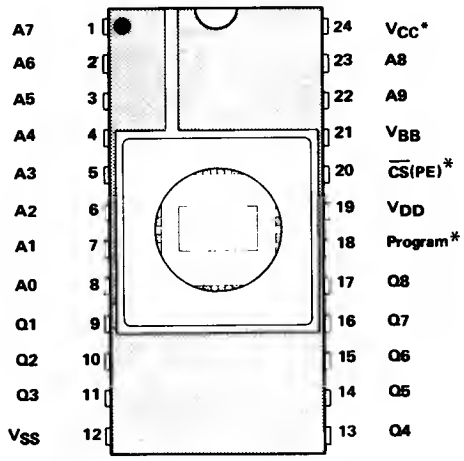
MAXIMUM DELAY TIMES

TYPE	ADDRESS	EN	DISABLE
SN54S'	75 ns	40 ns	40 ns
SN74S'	60 ns	30 ns	30 ns

TMS 2708 JL, TMS 27L08 AND TMS 2716 JL 8K AND 16K ERASABLE PROGRAMMABLE ROMs

- 2708 JL and 27L08 JL – 1024 X 8 Organization
- 2716 JL 2048 X 8 Organization
- All Inputs and Outputs Fully TTL-Compatible
- Static Operation (No Clocks, No Refresh)
- Maximum Access Time . . . 450 ns
- Minimum Cycle Time . . . 450 ns
- 3-State Outputs for OR-Ties
- N-Channel Silicon-Gate Technology
- 8-Bit Output for Use in Microprocessor-Based Systems
- Low Power
  TMS 27L08 . . . 245 mW (Typical)
  TMS 2716 . . . 315 mW (Typical)
- 10% Power Supply Tolerance (TMS 27L08 Only)
- Plug-Compatible Pin-Outs Allowing Interchangeability/Upgrade to 16K With Minimum Board Change

24-PIN CERAMIC
DUAL-IN-LINE PACKAGE
(TOP VIEW)



*For 2716 JL Only:

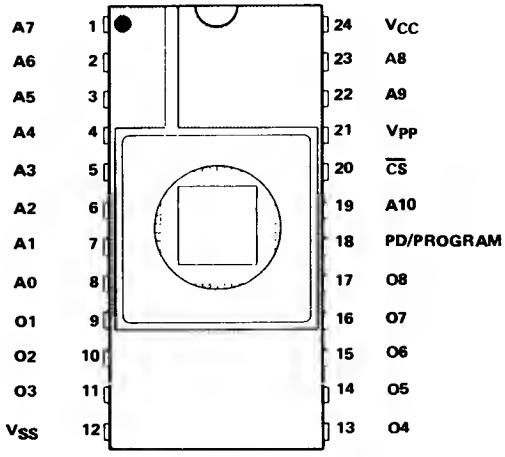
Pin:	
18	$\overline{CS}$ (Program)
20	A10
24	V_{CC} (PE)

V_{BB}	−5V power supply
V_{CC}	+5V power supply
V_{DD}	+12V power supply
V_{SS}	OV ground

TMS 2516 JL 16K ERASABLE PROGRAMMABLE READ-ONLY MEMORIES

- 2048 x 8 Organization
- Single +5 V Power Supply
- All Inputs and Outputs Fully TTL-Compatible
- Maximum Access Time . . . 450 ns
- Minimum Cycle Time . . . 450 ns
- 3-State Outputs for OR-Ties
- 8-Bit Output for Use in Microprocessor Based Systems
- N-Channel Silicon-Gate Technology
- Low Power: 525 mW Maximum Active Power
  132 mW Maximum Standby Power
- Guaranteed d.c. Noise Immunity with Standard TTL Loads – No Pull-Up Resistors Required
- Interchangeable with Intel 2716

24-PIN CERAMIC
DUAL-IN-LINE PACKAGE
(TOP VIEW)

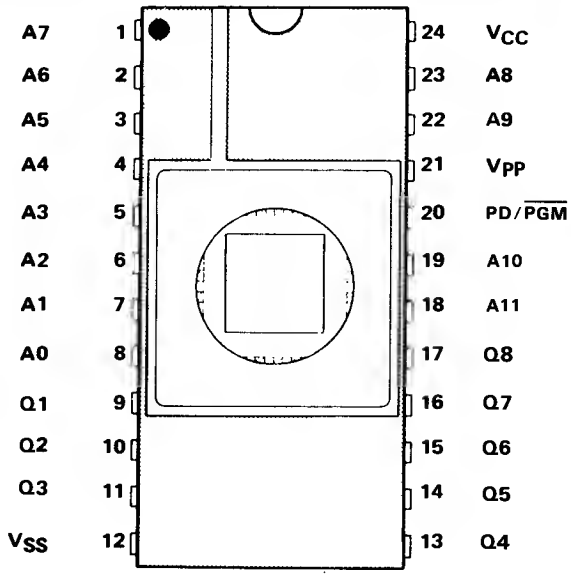


8◂

TMS 2532 JL
32K ERASABLE PROGRAMMABLE ROMs

- 4096x8 Organization
- Single +5V Power Supply
- Pin Compatible with all 8K and 16K EPROMs
- Plug-in Compatible with the TMS 4732 32K ROM
- All Inputs and Outputs Fully TTL Compatible
- Static Operation (No Clocks, No Refresh)
- Maximum Access Time . . . 450ns
- Minimum Cycle Time . . . 450ns
- 3-State Outputs for OR-Ties
- 8 Bit Output for Use in Microprocessor Based Systems
- N-Channel Silicon-Gate Technology
- Low Power: 840mW Maximum Active
  132mW Maximum Standby
- Guaranteed DC Noise Immunity with Standard TTL Loads—No Pull-Up Resistors Required
- Interchangeable with Intel 2716

24-PIN CERAMIC
DUAL-IN-LINE PACKAGE
(TOP VIEW)



Mechanical Data

8

NUMBERING SYSTEM

Factory orders for circuits should include the complete part-type numbers.

Example: TM S 4030 N L 30

TI Prefix

Product Status
- S Standard devices
- X Prototype or experiement
- C Custom designs

Unique Product Identification Number*

Package
- F Flat
- J Ceramic dual-in-line
- N Plastic dual-in-line
- U Unencapsulated (beam lead, etc.)

Temperature Range
- C −25°C to 85°C
- L 0°C to 70°C
- M −55°C to 125°C
- R −55°C to 85°C
- S Special range

Speed (Access Time)**

−15	150 ns	−30	300 ns
−20	200 ns	−45	450 ns
−25	250 ns		

MANUFACTURING INFORMATION

Die-attach is by standard gold silicon eutectic or by conductive epoxy.

Thermal compression gold wire bonding is used on plastic packaged circuits. Typical bond strength is 5 grams. Bond strength is monitored on a lot-to-lot basis. Any bond strength of less than 2 grams causes rejection of the entire lot of devices. On hermetic devices either thermal compression or ultrasonic wire bonding is used. All hermetic MOS LSI devices produced by TI are capable of withstanding 5×10^{-7} atm cc/sec inspection and may be screened to 5×10^{-8} atm cc/sec fine leak, if desired by the customer, for special applications.

All packages are capable of withstanding a shock of 3000 g. All packages except the 64-pin package are capable of passing a 20,000 g acceleration (centrifuge) test at the Y-axis. Final specifications for the 64-pin package are not available at this printing. Pin strength is measured by a pin-shearing test. All pins are able to withstand the application of a force of 6 pounds at 45° in the peel-off direction.

DUAL-IN-LINE PACKAGES

A pin-to-pin spacing of 100 mils has been selected for standard dual-in-line packages.

TI uses two basic types of hermetically sealed ceramic dual-in-line packages. The first type is the side-brazed package cap and tin-plated leads. The second is the cerdip which consists of a ceramic base and cap sealed with a low-temperature glass and tin-plated leads.

*Inclusion of an "L" within the identification number indicates the device operates in the low power range (e.g., 27L08, 40L45).

▸8

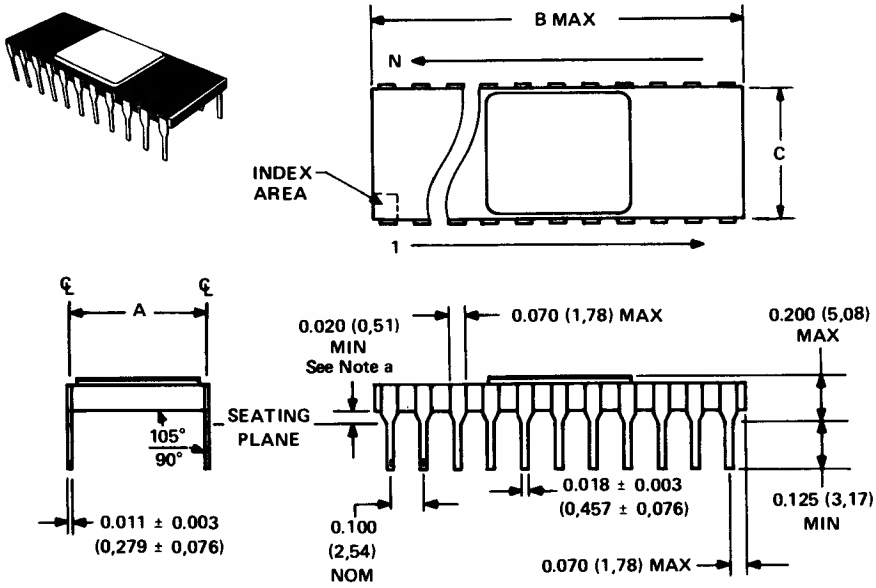
**On some parts DRAMs {-1 250 ns, -2 200 ns} SRAMS {-1 650 ns, -2 450 ns} 9900 FAMILY {-30 3MHz, -40 4 MHz}

MECHANICAL DATA

The following dual-in-line packages are available in plastic or ceramic:

Distance Between Rows	*Number of Pins*								
	8	10	16	18	20	22	24	28	40
300 mils	X†	X†	X	X	X				
400 mils						X			
600 mils							X	X	X

CERAMIC PACKAGES WITH SIDE-BRAZED LEADS AND METAL OR EPOXY OR GLASS LID SEAL

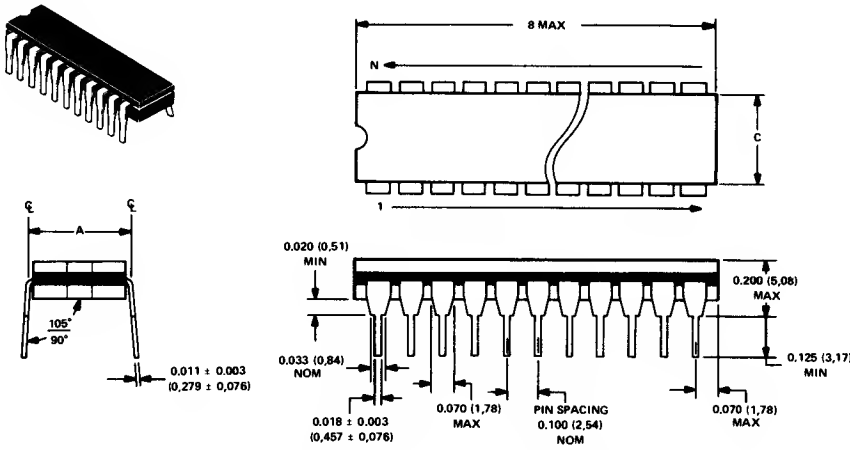


NOTES: a. This minimum spacing is valid for printed circuit board mounting with 0.033 (0,84) diameter holes for the leads.
b. All linear dimensions are in inches and parenthetically in millimeters. Inch dimensions govern.

DIM \ PINS	16	18	20	22	24	28	40
A ± 0.010 (0,26)	0.300 (7,62)	0.300 (7,62)	0.300 (7,62)	0.400 (10,16)	0.600 (15,24)	0.600 (15,24)	0.600 (15,24)
B MAX	0.840 (21,4)	0.910 (23,1)	1.020 (25,9)	1.100 (28,0)	1.290 (32,8)	1.415 (36,0)	2.020 (51,3)
C NOM	0.290 (7,4)	0.290 (7,4)	0.290 (7,4)	0.390 (9,9)	0.590 (15,0)	0.590 (15,0)	0.590 (15,0)

8◀

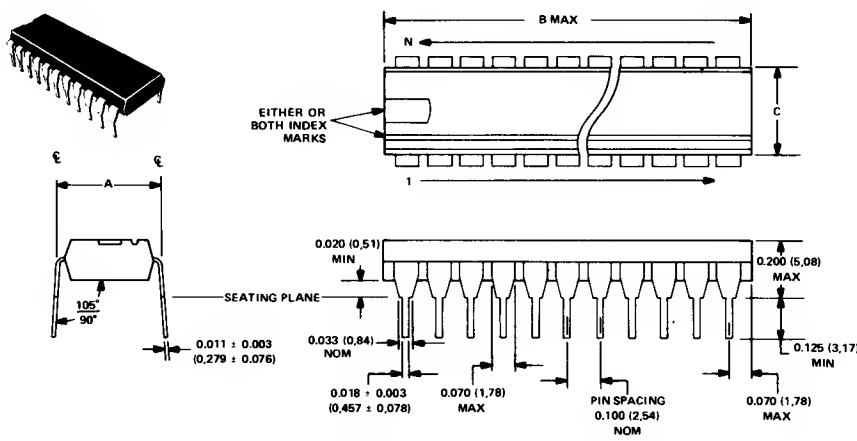
Cerdip Packages



ALL LINEAR DIMENSIONS ARE IN INCHES AND PARENTHETICALLY IN MILLIMETERS. INCH DIMENSIONS GOVERN.

DIM \ PINS	16	18	20	22	24	28	40
A ± 0.010 (0,26)	0.300 (7,62)	0.300 (7,62)	0.300 (7,62)	0.400 (10,16)	0.600 (15,24)	0.600 (15,24)	0.600 (15,24)
B MAX	0.785 (20,0)	0.920 (23,4)	1.070 (27,2)	1.100 (28,0)	1,290 (32,8)	1.460 (37,1)	2.090 (53.1)
C MAX	0.288 (7,3)	0.288 (7,3)	0.288 (7,3)	0.388 (9,86)	0.560 (14,2)	0.560 (14,2)	0.560 (14,2)

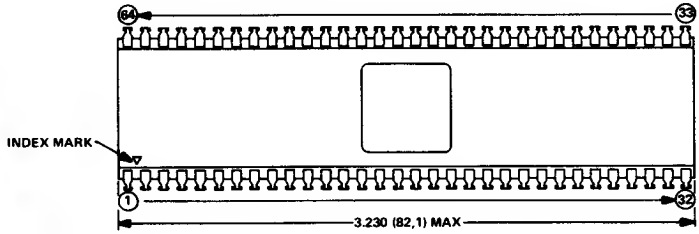
Plastic Packages

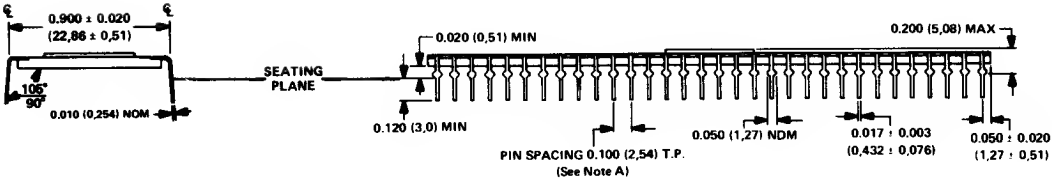


ALL LINEAR DIMENSIONS ARE IN INCHES AND PARENTHETICALLY IN MILLIMETERS. INCH DIMENSIONS GOVERN.

DIM \ PINS	8	16	18	20	22	24	28	40
A ± 0.010 (0,26)	0.300 (7,62)	0.300 (7,62)	0.300 (7,62)	0.300 (7,62)	0.400 (10,16)	0.600 (15,24)	0.600 (15,24)	0.600 (15,24)
8 MAX	0.390 (9,9)	0.870 (22,1)	0.920 (23,4)	1.070 (27,2)	1.100 (28,0)	1.290 (32,8)	1.440 (36,6)	2.090 (53,1)
C NOM	0.250 (6,4)	0.250 (6,4)	0.250 (6,4)	0.265 (6,7)	0.350 (8,9)	0.550 (14,0)	0.550 (14,0)	0.550 (14,0)

CERAMIC PACKAGES WITH TOP-BRAZED OR SIDE-BRAZED LEADS AND METAL OR EPOXY OR GLASS LID SEAL





ALL LINEAR DIMENSIONS ARE IN INCHES AND PARENTHETICALLY IN MILLIMETERS, INCH DIMENSIONS GOVERN.

8

Software

8

TMSW 101MT
TMS 9900 TRANSPORTABLE CROSS-SUPPORT

- TMS 9900 Cross Assembler
- TMS 9900 Simulator
- ROM Utility

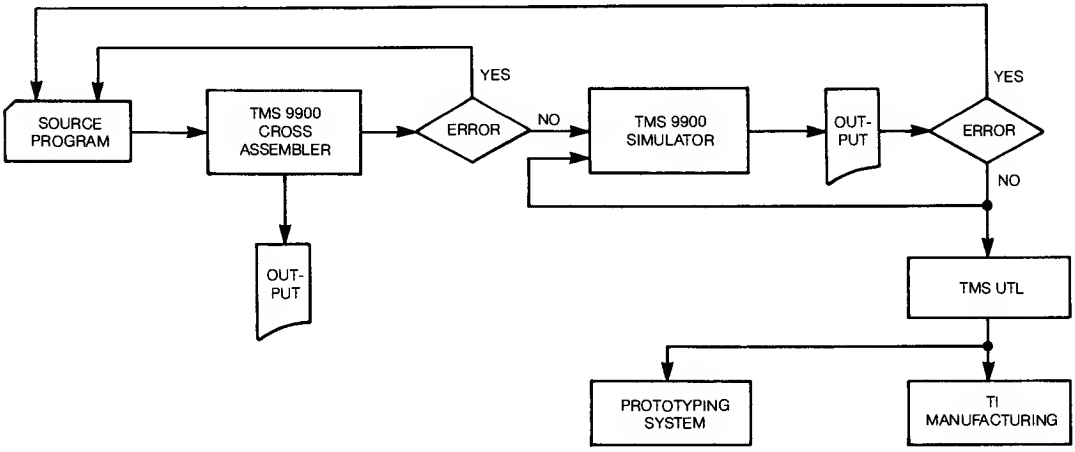


Figure 1

TMS 9900 CROSS ASSEMBLER DESCRIPTION

Before the advent of assemblers and other programming aids, the computer programmer was required to manually generate the particular bit patterns that constitute a program. This tedious task is now performed by symbolic assembler program such as the TMS 9900 Assembler. The Assembler permits the programmer to refer to data, memory addresses and machine actions symbolically when creating a source program. The TMS 9900 Assembly Language source is translated by the TMS 9900 Cross Assembler into relocatable linkable TMS 9900 Object module format. Both the source input and the object output are fully compatible with the FS 990 Prototype Development System and nationally available timesharing services (GE, NCSS, and TYMSHARE).

TMS 9900 SIMULATOR DESCRIPTION

The TMS 9900 Simulator is compatible with and has extensions to the Simulator on GE, NCSS, and TYMSHARE. The Simulator runs either in batch mode or in an interactive mode for maximum effectiveness. The microprocessor system simulation is specified by the designer from a keyboard/display device in the interactive mode. The output, such as instruction trace can be viewed on a CRT or printed out when in the batch mode.

The Simulator accepts object modules plus "link-control" statements from the Assembler as shown in Figure 1. Then the load module plus debug and control statements are sent to a "Run Processor" that performs the application program's execution. By executing instructions in software just as the TMS 9900 microprocessor executes instructions in hardware, the program logic is verified and the performance is measured. To set up a target system's characteristics, control language statements initialize memory, I/O ports, I/O linkages, and processor clock frequency. The control language also allows multiple breakpoints, full instruction trace, snapshots, memory and register inspection/changes. Interrupt controls and total run/stop commands are also provided. By exercising total software control, a program can be thoroughly checked before the hardware is running.

TMS UTL, ROM UTILITY DESCRIPTION

When the application program has been satisfactorily verified, the object module is accessed by the ROM Utility program, TMS UTL, for translation into a format acceptable for production of a gate placement program (prepatory to mass production). Alternatively, the utility can generate a BNPF or hexadecimal formatted file that is an input to a PROM programmer (Data I/O, etc.) to produce a PROM or EPROM version of the program. In all, there are 12 acceptable input formats and 12 output formats in support of the TMS 9900 microprocessors. Table I indicates the valid input/output translations supported by the utility for the TMS 9900.

***Table I** — TMSUTL Format Paths*

	AVAILABLE OUTPUT FORMAT							
AVAILABLE INPUT FROM	1	2	3	4	5	6	7	8
1) Hexadecimal #2 (PROM)	YES	YES	YES	YES	NO	NO	YES	YES
2) Hexadecimal #2 (ROM)	YES	YES	YES	YES	NO	NO	YES	YES
3) BNPF	YES	YES	YES	YES	YES	YES	YES	YES
4) SN74S271 & SN74S371 ROM/HILO Format	YES	YES	YES	YES	NO	NO	YES	YES
5) TMS 9900 Standard Absolute Object of Cross Support System (Assembler or Loader/Simulator) & Prototyping System	YES	YES	YES	YES	YES	YES	YES	YES
6) TMS 9900 Compressed Absolute Object of Prototyping System	YES	YES	YES	YES	YES	YES	YES	YES
7) TMS 4700 ROM	YES	YES	YES	YES	NO	NO	YES	YES
8) TMS 4800 ROM	YES	YES	YES	YES	NO	NO	YES	YES

OPERATING ENVIRONMENT

The programs are written to conform to ANSI STANDARD X3.0 (1966) 16-bit FORTRAN and are designed to execute on any minicomputer with the following minimum characterics.

- ANSI STANDARD X3.0 (1966) 16-bit FORTRAN Compiler
- Two's compliment arithmetic
- Disc capacity for up to 7 simultaneously active sequential files, with two being in rewindable and re-readable media
- A 16k to 20k word user program memory partition

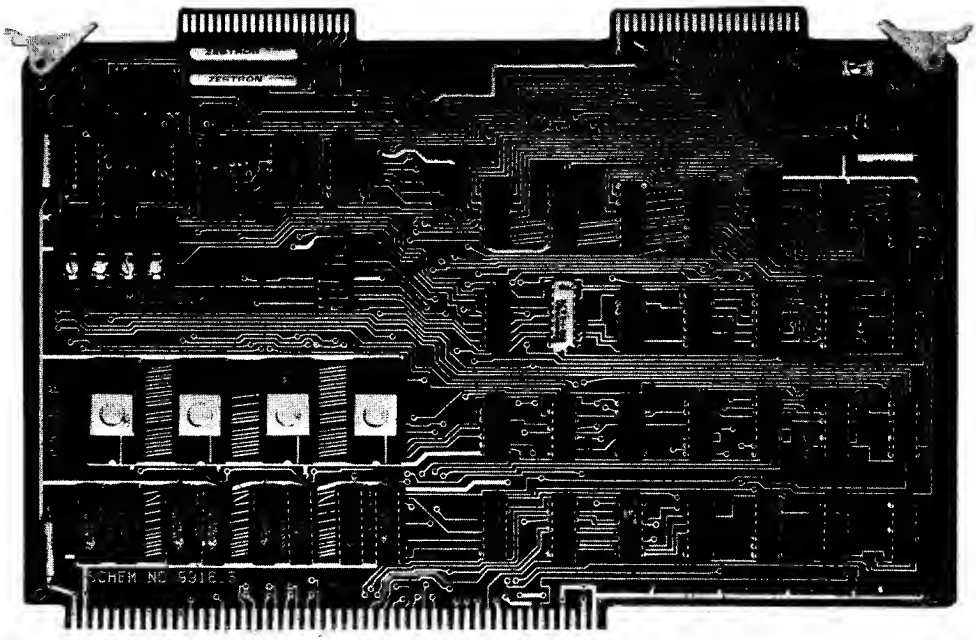
8◀

PACKAGING

The TMS 9900 Transportable Cross Support package is composed of three distinct products: TMS 9900 Cross Assembler, Simulator, and ROM Utility. The part number for the package is TMSW 101MT. The product name is TMS 9900 *Transportable Cross-Support Software.* TMSW 101T is manufactured only on *half-inch, 9 track PE encoded* (IBM compatible) magnetic tape recorded at *1600 BPI.* The tape is *unlabeled, unblocked,* with *80 ASCII bytes per data record* and contains *128 files.* The first file on the tape is a data file which contains a one-time description for each file on the tape. Each file is terminated by an EOF mark except for the last file which is terminated with a double EOF to indicate *end-of-logical tape.*

Included in the shipping package is a *User Manual* for each of the three programs and an *Installation Manual* (4 manuals, total).

TM 990/302
SOFTWARE DEVELOPMENT BOARD



- Dual or Single audio-cassette interface
- EPROM programming options:
  TMS 9940, TMS 2716, TMS 2708,
  TMS 2532, TMS 2516
- Software development aids residing in ROM:
  - Symbolic Assembler
  - Text Editor
  - EPROM programmer
  - Relocating loader
  - I/O Scheduler/Handler
  - Debugger
- Optional POWER BASIC development software residing in EPROM (16k bytes)
- 4K × 16 EPROM or preprogrammed ROM
- 2K × 16 RAM
- Memory expandability for additional performance (TM 990/201 or TM 990/206 memory expansion boards)
- EIA communication with other computers

DESCRIPTION

The TM 990/302 is an assembled, tested module for developing assembly language software to be used on 990/9900 family microprocessor based systems. The TM 990/302, a bus-compatible member of the TM 990 microcomputer module family, provides dual audio cassette interfaces, both static RAM and ROM memory, and hardware circuitry for the programming of read-only memory devices. Used in conjunction with either the TM 990/100M or TM 990/101M microcomputer modules, the TM 990/302 provides a complete standalone software development system offering support for program generation, editing, assembly, debugging, and EPROM programming at an extremely attractive cost. Figure 1 is a system block diagram of the TM 990/302. The TM 990/100 and TM 990/101 memory map incorporating the Software Development Board ROM and RAM are shown in Figure 2.

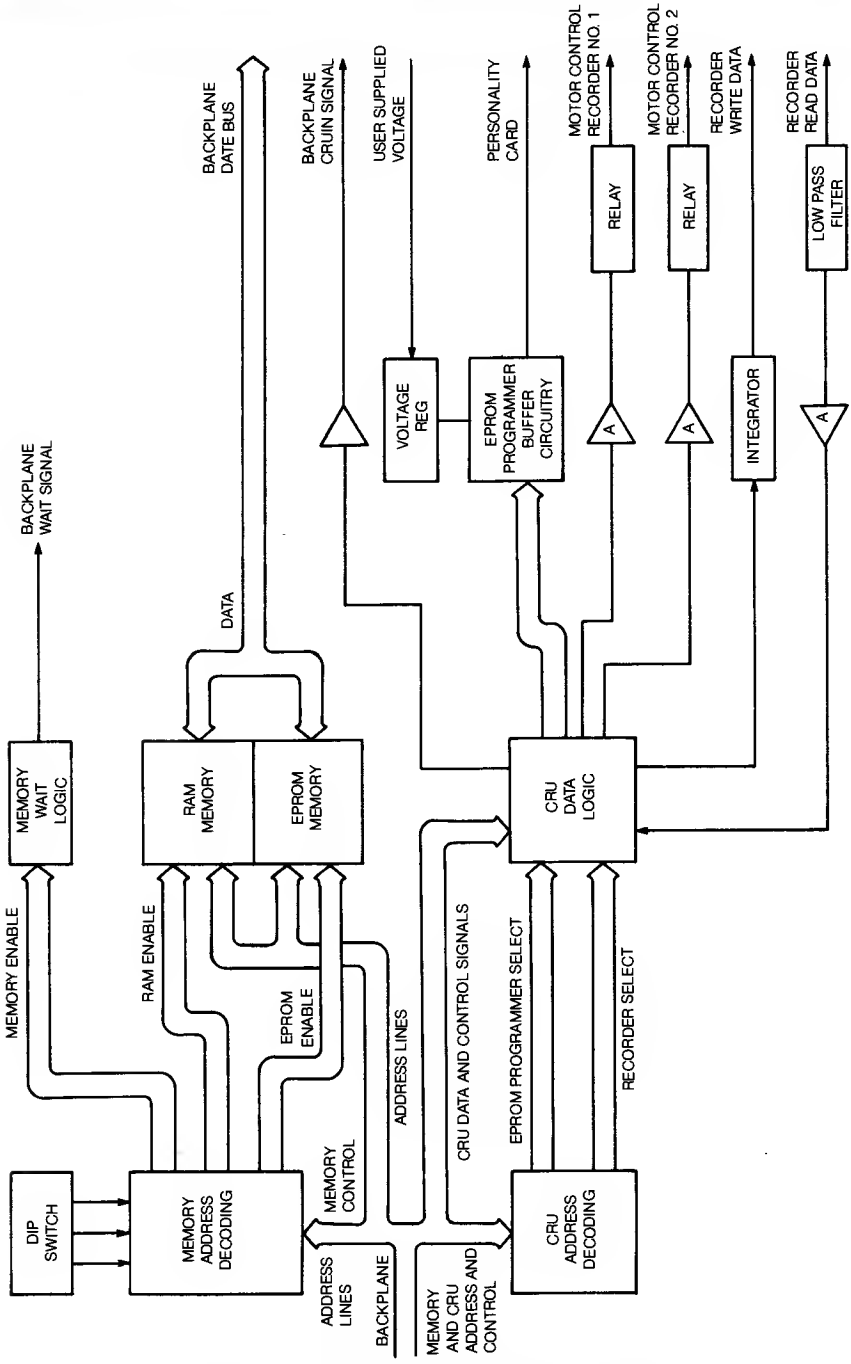


Figure 1. TM 990/302 System Functional Block Diagram

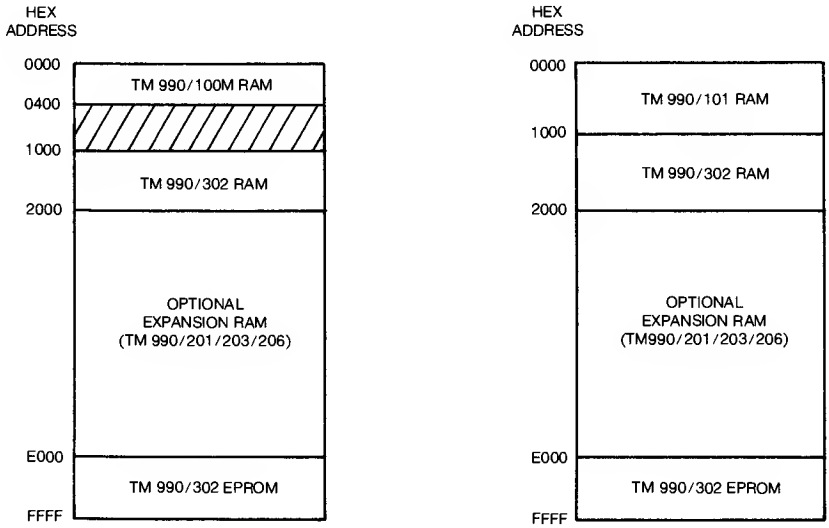


Figure 2. System Memory Map

SOFTWARE

Figure 3 is a typical software development cycle using the TM 990/302 Software Development board.

SOURCE TEXT EDITING

The text editor provides the means for initial source code entry or program update. Initial source inputs will be from the user's terminal. Source programs on audio cassette will be updated with changes made from the user's terminal. The size of the text editor buffer is determined at initialization as a function of the total available RAM.

The text editor operates on the source code text in a line mode. Text editor commands with their respective functions are:

▶8

D	Delete lines n thru m
I	Insert at line n with optional line-number autoincrement by m
K	Keep (store) buffer and print new top line in the buffer
G	Get buffer and print new bottom line of the buffer
P	Print lines n thru m
Q	Flush the input file until end of input file and return to executive.
R	Resequence output line numbers, n is initial line number and m is the increment.

To create or update the source program, the text.editor provides manipulation of individual lines of code and entry with automatic line number indexing. The designer may delete, insert, print, resequence, and interactively check 9900 instruction syntax from his keyboard. The text editor handles programs of any length by segmenting the code into "buffer" blocks. It controls buffer loading and storage into cassette-tape memory. The buffer is enlarged by plugging in memory-expansion cards, which also expand the amount of target system memory available for execution.

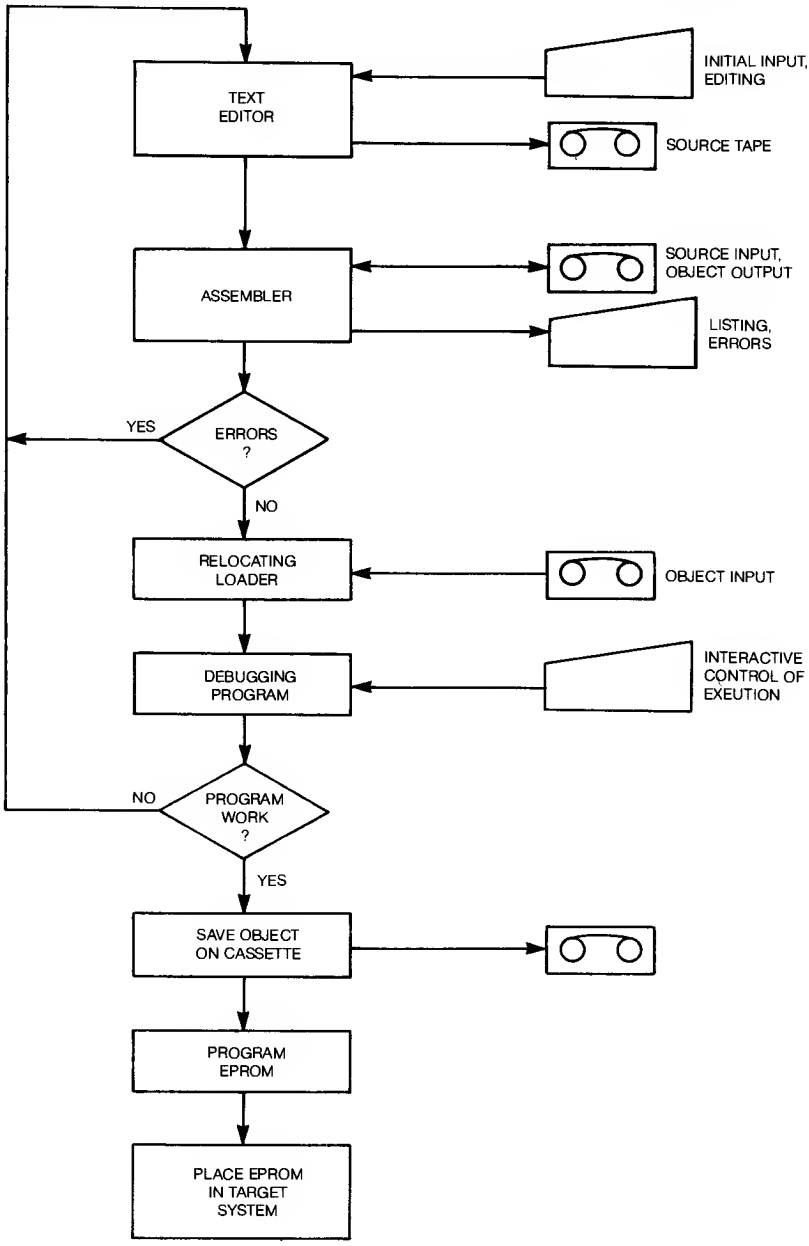


Figure 3. Typical Software Development Execution

TM 990/302 SOFTWARE DEVELOPMENT BOARD

ASSEMBLING SOURCE

The next step in program development is a two-pass assembly of TMS 9900, SBP 9900, TMS 9980, TMS 9985, TMS 9940 instruction sets into absolute standard 9900 object code. This two-pass assembler allows four-character symbolic addressing. The assembly listing output, including error messages, is routed to a user chosen device.

DEBUGGING

Seven debug commands aid program development after the loader program puts the assembled object into memory. Multi-step trace, software breakpoints and data inspection changes are featured.

Debug Commands:

SB	Set software breakpoint and execute
IM	Inspect/change memory
IC	Inspect/change CRU
IR	Inspect/change registers
RU	Record Program Execution Path Change
ST	Single step for 1 or more instructions with or without trace
DM	Dump memory to specified cassette in object format

EPROM PROGRAMMING

After debug, the EPROM programmer can be invoked to program EPROM's, read back EPROM's into memory, or compare EPROM contents to memory. Byte and word serial formats are available. The EPROM programmer is able to program the following EPROMS: 2708, 2716, 2516, 2532, and 9940.

HARDWARE FEATURES

- Two audio cassette interfaces
- 4K × 16 programmed ROM's
- 2K × 16 RAM
- One decode PROM for upgrading TM 990/100M microcomputer board RAM maps
- TMS 2716, TMS 2708, TMS 2532, TMS 2516 and TMS 9940 EPROM programming personality card.

When the TM 990/302 board is used as a software development system, the equipment configuration could include:

- TM 990/302 Software Development Board
- A TM 990/100M or TM 990/101 microcomputer board
- TM 990/510/520 (4 or 8 slot) card cage
- User supplied power supply: +12V, +5V, −12V, +30V to +52V (for EPROM Programmer) TM 990/518 Power Supply
- User supplied audio cassette player/recorder
- Power Basic Interpreter (16k byte) optional
- User supplied terminal: current loop TTY, TI silent 700, or equivalent.

▶8

Table 1 lists the hardware characteristics of the development system when used with either of the two microcomputer boards.

TM 990/302 SOFTWARE DEVELOPMENT BOARD

Table 1. TM 990/302 System Specifications

Item	*TM 990/302 with TM 990/100M*	*TM 990/302 with TM 990/101M*
Microprocessor	TMS 9900, 16 bit	TMS 9900, 16 bit
Timers	2	3
Serial ASCII	1(110-19,200 baud)	2(110-19,200 baud)
I/O ports	1 16-bit port	1 16-bit port
RAM (min-max)	2.25K-2.25K words	4K words
Non-volatile memory	5K-8K words ROM	4K-8K words ROM
Programmable memories	2708, 2716, 2516 2532, 9940	2708, 2716, 2516 2532, 9940
Cassette interfaces	2	2
Operating Temperature	0-55°C	0-55°C
Physical Characteristics Width: 11 inches (279.4mm) Height: 7½ inches (190.5mm)		
Power Requirements +5V 1.5A −12V 50mA +12V 50mA 30-52V 100mA EPROM programming voltage		
Ordering Information TM 990/302 Software Development Board		

8

TM 990/40DS
DESIGN AID FOR TMS 9940 MICROCOMPUTER

- EPROM Programmer for TMS 9940E
- Assembler
- Debug Monitor
- Trial in System Emulation
- Can be used with EPROM and/or Mask ROM Version of TMS 9940

DESCRIPTION

The TM 990/40DS is a low cost development system for the first 16-bit single-component microcomputer, the TMS 9940. To make the TM990/40DS as cost effective as possible, the traditional front panel assembly of lights and switches is not used. Instead an input/output system is provided which enables the programmer to use a standard EIA terminal such as an ASR-33 Teletype or a Texas Instruments Silent 700 terminal. The TM 990/40DS helps the programmer:

- Generate executable software
- Program this software to be on the EPROM version of the TMS 9940, and
- Test the program to be in the user's target system using the Trial In-system Emulation (TISE) Feature.

▶8

SOFTWARE GENERATION

To help generate the software the TM 990/40DS features an assembler and monitor. The Line by Line Assembler (LBLA) is a single pass assembler that assembles the user's program written in TMS 9940 instructions and stores machine code in memory. As each source line is assembled, the resulting machine code is placed in the user's RAM on the TM 990/40DS. The user can then implement the TIBUG II monitor to test and debug the software prior to using the EPROM programmer.

TIBUG II is a debug monitor which provides an interactive interface between the user and the TM 990/40DS. The TIBUG II monitor provides software routines that accomplish special tasks. These routines, listed in Table I, facilitate software development using the TM 990/40DS. All communications with TIBUG II occur via a 20mA current loop or RS-232-C device.

PROGRAMMING THE EPROM OF THE TMS 9940

This is accomplished by three of the TIBUG II commands: PP, CP, and VP. These are described in Table I. These commands allow for the programming of the EPROM from the user's RAM memory of the TM 990/40DS, the copying from the TMS 9940E EPROM to the RAM memory, and the verification of the EPROM and RAM memories.

TRIAL IN-SYSTEM EMULATION (TISE)

Once the user's program has been assembled and debugged using the TM 990/40DS TIBUG II monitor, the program can be tested in the user's target system using the TISE feature of the TM 990/40DS. This feature allows emulation of most of the TMS 9940's operations, utilizing the TM 990/40DS memory. Using TISE, a three-foot 40-conductor cable is connected to the edge connector of the TM 990/40DS. The other end of the cable contains a 40-pin male connector that is plugged into the user's system at the socket that will contain the TMS 9940.

Table I — TIBUG II Commands

Command	*Description*
ZA	Assembler — used to call up the Line by Line Assembler.
LP	Load Object Program from cassette/paper tape into the TM 990/40DS user memory.
DP	Dump the TM 990/40DS memory onto cassette/paper tape in TMS 9900 absolute object format.
SB	Set Breakpoint. This command sets breakpoints that allow programs to be executed from one memory address to another. From 1 to 16 addresses can be entered as breakpoints. When a breakpoint address occurs, execution stops and contents of the Program Counter, Status Register and the Workspace Pointer are displayed.
CB	Clear Breakpoint. Clears breakpoints previously set.
IM	Inspect/Change Memory, Memory Dump. Memory inspect/change "opens" a memory location, displays it, and gives the option of changing the data in the location. Memory dump directs a display of memory contents from "start address" to "stop address".
IR	Inspect/Change Hardware Registers (Workspace Pointer, Program Counter, Status Register). These three registers are displayed and may be changed.
IW	Inspect/Change user workspace. This command is used to display contents of the entire workspace register file or display one register at a time allowing the user to change the register contents.
IC	Inspect/Change CRU. The CRU register is displayed and may be changed. NOTE: The CRU is a bit oriented I/O interface through which both input and output bits can be directly addressed individually or in fields of 1 to 16.
FB	Find Byte in memory. A memory field is searched for a value. The memory addresses that contain the value are printed out.
FW	Find Word in memory. Same as Find Byte in memory except the value is a word.
DH	Decimal to Hexadecimal conversion provides the user the capability of converting decimal numbers to hexadecimal.
HD	Hexadecimal to Decimal conversion, provides user the capability of converting hexadecimal numbers to decimal.
HA	Hexadecimal Arithmetic. Two hexadecimal numbers are entered and their sum and difference are printed out.
PP	Program EPROM. Used to load the TMS 9940's EPROM area with the user's program.

TM 990/40DS DESIGN AID FOR TMS 9940 MICROCOMPUTER

Command	*Description*
CP	Copy EPROM program. The contents of the EPROM section of the TMS 9940 is transferred into the user's RAM area of the TM 990/40DS.
VP	Verify PROM. The contents of the EPROM of the TMS 9940 is compared to the contents of the user's RAM area in the TM 990/40DS. This verifies that the correct program is in the TMS 9940.
MV	Move Block of memory. A starting and ending address is given for the block of memory to be transferred along with the starting address of the destination.
EX	Execute. Following execution of this command, program execution begins at the value presently in the program counter.
RU	Run in multistep mode. From 1 to 64K instruction executions followed with WP, PC, Status Printout.
HE	Help command. This command brings up a listing of all the TIBUG II commands as reference for the user.
TM	Texas Instruments 733 ASR run at 1200 Baud. This command is used to alert TIBUG II that the terminal being used is a 1200 Baud terminal other than a Texas Instruments 733 ASR,
TT	Self Test. The self test is a software routine used to test the TMS 9940s. The routine is loaded into the TMS 9940's RAM space and executed by the TM 990/40DS. Once completed, the program transmits the results of the test to the system terminal.

TM 990/40DS PARTS LIST

The TM 990/40DS consists of 3 boards (described below), a TM 990/510 card cage, power supply, Trial In-System Evaluation (TISE) cable, EPROM programmer cable, a serial interface cable, and a chassis.

Three boards make up the TM 990/40DS System (See Figure 1). First is the TM 990/100M-4 microcomputer board with 1k × 8 bits of RAM and 8k × 8 bits of EPROM. This board is the central processing unit which controls I/O to the emulator boards and also to peripheral devices.

The two emulator boards are the TM 990/901 and TM 990/902 which handle the emulator functions of the TM 990/40DS during software development.

ORDERING INFORMATION TM 990/40DS DESIGN AID FOR TMS 9940 MICROCOMPUTER

The TM 990/40DS may be ordered through any TI authorized distributors under the following Part Number: TM 990/40DS.

▶8

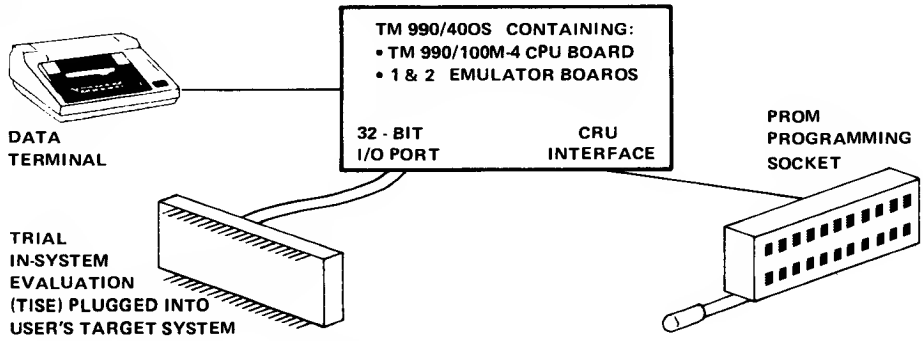


Figure 1. TM 990/40DS System Diagram

TM990/450, 451, 452, TMSW201F POWER BASIC FAMILY

POWER BASIC FEATURES

- Software Based in ROM or Floppy Diskettes
- For Use with Single or Multi-Board Systems
- Bit, Character and Word Oriented I/O
- Multiple Process Execution
- Interprocess Communication through Common Variables
- Automatic Minimum Memory Configuration
- Real Time Clock and Interrupt Processing
- Extended Arithmetic Capability
  - Multi-Dimensional Arrays
  - Multi-Argument Functions
  - 48 Bit Real Precision

A FAMILY OF PRODUCTS

The POWER BASIC Family is a set of software that edits and translates BASIC language statements into 9900 instructions and executes these statements to solve a particular algorithm.

Texas Instruments family of POWER BASIC language interpreters brings the features of BASIC to the industrial user of microprocessors providing a selection of products to meet the requirements for evaluation, development, and application in economical yet versatile packages. These additions continue the development of a comprehensive line of software development tools to support 990/9900 family components and systems to meet a wide range of application requirements.

The POWER BASIC family members are provided in read-only memory devices and on floppy diskettes. They can operate on a variety of hardware configurations ranging from stand-alone component based systems using 9900 family microprocessors to full-feature minicomputers such as the FS990. *Figure 1* illustrates the solution of real world problems in real time provided by a TM990/101M-10 module.

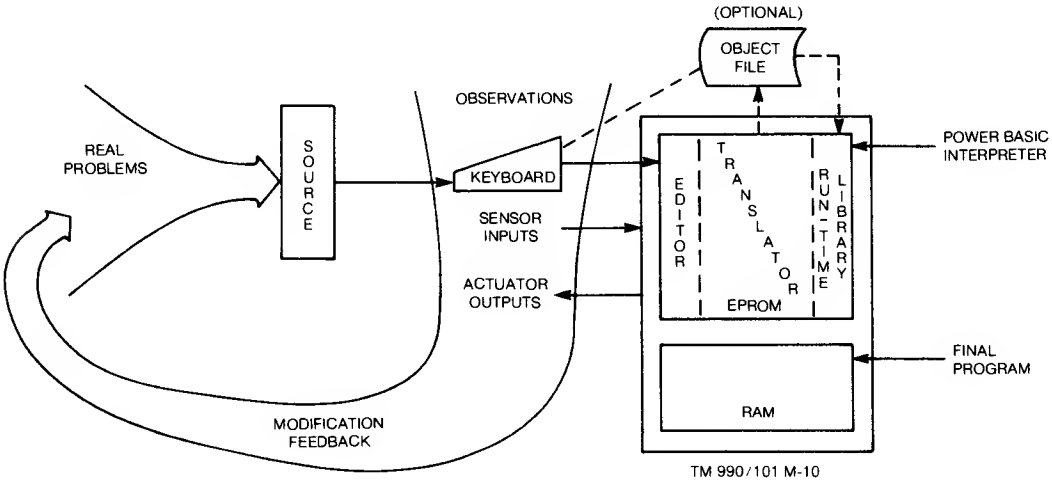


Figure 1. Real World Problem-Solving in Real Time with a TM990/101M-10 Module

TM990/450, 451, 452, TMSW201F POWER BASIC FAMILY

APPLICATIONS

POWER BASIC is used to solve problems in data acquisition and control, data communications, information analysis, and sequencing of external events. Current applications include: intrusion alarm monitoring, navigational computation, numerical control, data reduction and analysis, inventory and payroll management, point-of-sale accounting, simulation and forecasting, and data base manipulation. Also used for education in programming and the structure of algorithmic processes.

APPLICATION FEATURES

POWER BASIC for Increased Speed

- TI Breaks the 20 Second Barrier

The 16 bit architecture and instruction set of the 9900 family of microprocessors is well suited for efficient execution of POWER BASIC programs. Results compared with benchmark programs reported in *KILOBAUD* magazine (October, 1977) show POWER BASIC to be faster than all others reported — less than 20 seconds on benchmark 7.

POWER BASIC for Cost Effectiveness

- Single Board, ROM Resident, Configurable

For production systems written in POWER BASIC, a CONFIGURATOR program is provided which analyzes the POWER BASIC application program and produces a minimum memory load module for that application.

Insertion of assembly language subroutines provides additional problem solving flexibility.

POWER BASIC for Flexible I/O

- Direct Manipulation of I/O and Memory

Control of information into and out of a 9900 family microprocessor based system is easily accomplished using a special variety of assignment statements supported by POWER BASIC. Internal and external data and control signals may be received and transmitted by the application program using a special set of POWER BASIC system defined functions. These special functions, as well as assembly language subroutines, support direct manipulation of either CRU or memory-map interfaced devices. In many cases, use of the system defined memory and CRU functions will be adequate substitutes for assembly language, improving the reliability of the application program by using pre-tested software and increasing the productivity of the design engineer.

POWER BASIC for Multi-Processing (Eval. BASIC)

- Multiple Process Execution and Communication

Within the POWER BASIC command and control structure, special provision has been made for independent execution of programs through a FOREGROUND/BACKGROUND mode switch. Once a program has been started in the FOREGROUND mode, a switch can be activated to provide a new program environment while relegating the currently executing POWER BASIC application program to the BACKGROUND mode of operation. After this has been accomplished, communication between FOREGROUND and BACKGROUND programs is supported through the use of shared dedicated variables. These unique variables may be modified and tested by either program independently, offering full capabilities for interaction and control.

TM990/450, 451, 452, TMSW201F POWER BASIC FAMILY

POWER BASIC FAMILY MEMBERS

Part No.	*Media*	*Name*	*Description*
TM990/450 TM990/101M-10	EPROM device kit TM990/101M	Evaluation POWER BASIC	Reduced memory version (8K Byte) designed to offer evaluation tools for exploring POWER BASIC applications. ROM kit executes stand-alone on TM990/100M, 101M modules.
TM990/451	ROM device kit	Development POWER BASIC	Expanded memory version (12K Byte) providing capability for design, development, debug, and complete programming of POWER BASIC programs. Executes on TM990/302 module interfaced with TM990/100M, 101M CPU modules.
TM990/452	EPROM device kit	Development BASIC Software Enhancement package	Enhancements (4K byte) to Development POWER BASIC. Provides utilities for use with TM990/302 module (EPROM programmer, cassette interfaces, etc.). Executes on TM990/302 module interfaced with TM990/101M CPU or with TM990/100M CPU and TM990/100M CPU and TM990/201 Memory module.
TMSW201F	FS990 diskette	Configurable POWER BASIC	Fully expanded version including complete diskette file support and a CONFIGURATOR program which reduces the size of POWER BASIC programs for execution.

A REAL SOLUTION TO A PROBLEM

To the industrial designer of microprocessor based electronic equipment, Texas Instruments POWER BASIC Family offers a versatile alternative to the use of assembly language in implementing application programs. Designed to provide a selection of products to meet a broad range of feature and cost requirements, POWER BASIC delivers productivity improvements and architecture independence which impact development costs and minimize project risks. Packed with improved features, POWER BASIC makes the solution of complex system problems a straight-forward process, eliminating unnecessary design details, while providing the kind of performance mandated by state-of-the-art semiconductor technology and minicomputer architecture.

Texas Instruments is committed to provide the most advanced microprocessor system development tools, making the 990/9900 family of microprocessors the low risk choice for designers of electronic equipment. The TI POWER BASIC Family reaffirms that commitment and is indicative of the quality support which the user can expect from Texas Instruments in the future.

Additional details on the individual products which comprise the POWER BASIC Family are available from your Texas Instruments sales representative and authorized distributors.

CHAPTER 9

Applications

INTRODUCTION

This chapter is devoted to examples of applications of the 9900 family of components. Throughout this book many details of the 9900 family of CPU's, peripherals, microcomputer modules, software and software development system support have been discussed. However, these have been somewhat isolated general discussions and not directed to a particular application. This chapter has solutions of specific problems — from the beginning concept to the final machine code — to give you examples of how someone else has approached the problem and to help you understand the concepts behind the approach and the details of the solution.

Three applications are included. They are:

1. A SIMULATED INDUSTRIAL CONTROL APPLICATION

   A 9900 microprocessor based microcomputer is used in a system simulating the control of industrial manufacturing processes. Solutions to the problems of interfacing between industrial power levels and computer logic levels, both at the input and the output, are demonstrated, as well as basic concepts of computer control.

2. A LOW-COST DATA TERMINAL

   Direct comparison is made showing how the characteristics of the 9940 single chip 16-bit microcomputer are used to significantly reduce the package count of an intelligent terminal designed with an 8080 8-bit processor. At the same time the performance-cost ratio of the end equipment is improved.

3. A FLOPPY DISK CONTROLLER

   The design of a complex system used for the control of a floppy diskette memory is described. All the details of how a 9900 family microprocessor is used to arrive at a problem solution are included.

A Simulated Industrial Control Application

9

INTRODUCTION

Controlling motors, relays, solenoids, actuators; sensing limit switches, photo-electric outputs, push-button switches are real world problems encountered in controlling industrial manufacturing. This application simulates such conditions. It develops the application of a TMS9900 microprocessor (using the 990/100M microcomputer module of Chapter 3) and interconnecting hardware to automating industrial control requirements. This example includes the description of interface hardware to couple industrial power levels to and from the microcomputer system. It illustrates the use of an EIA/TTY terminal for interactive program entry and control, a line-by-line assembler for inexpensive program assembly, and the techniques of interrupt driven processing.

No motors, actuators, or solenoids are actually being controlled, but by sensing switches for logical voltage inputs and by turning lights on and off, the industrial control inputs and loads are simulated and the means demonstrated to accomplish the control.

As a logical extension of the first encounter application of Chapter 3, this application is written for "hands-on" operation to develop basic concepts and show that the 9900 family of microprocessors is ideally suited for industrial control applications. Each program step is described as the subprograms are developed and the total program is assembled into machine code.

Excitement comes from actually getting a microprocessor system doing useful things. This application is designed for that purpose. Let it demonstrate how easy it is to begin applying the 9900 family of microprocessors.

INITIAL SYSTEM SETUP WITH AN EIA TERMINAL

To begin, look at *Figures 1* and *6*. The system uses the same TM990/100M-1 microcomputer module shown in *Figure 3-12* and interconnected in *Figure 3-14*. It is a complete microcomputer with 256 16-bit words of RAM, 1024 16-bit words of ROM, and interface circuits to handle parallel and serial I/O. In *Figure 3-14* it has power supplied to it through P1, the 100 pin edge connector as specified in *Figure 3-17*. P2 interconnects the TM990/301 microterminal which is being used as an input terminal for programming, editing, and debugging. The output board *(Figure 3-9)* with a 7 segment LED display is connected to the microcomputer through P4. The program *(Table 3-2)* sequenced the elements f, b, e and c of the LED display on and off, either fast or slow, depending on the position of the control switch.

▶9 *Table 3-2* was "assembled-by-hand." In the examples that follow, a ROM resident "line-by-line" assembler will be used. This is a low-cost, effective way of providing machine code. However, a different terminal is required so that print out of the code can be obtained. Therefore, in this application the microterminal attached to the TM990/100M microcomputer is replaced with a keyboard terminal with EIA/TTY interconnection. Refer to *Figure 1*.

INITIAL SYSTEM SETUP WITH AN EIA TERMINAL

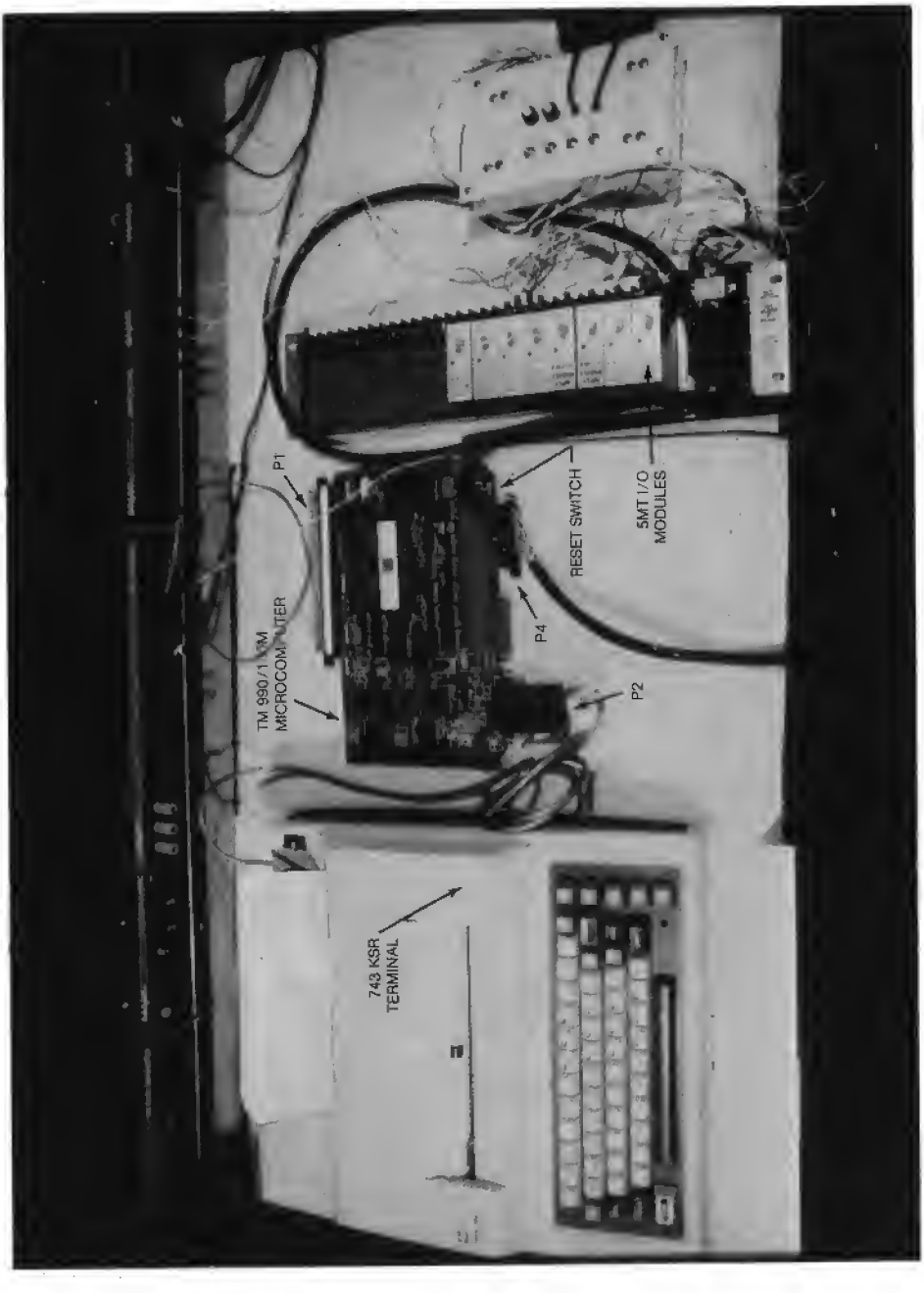


Figure 1. Picture of System Set-up

A 743 KSR terminal is chosen for this purpose. A special cable is required to interface the terminal to the microcomputer through P2. The cable connections are as follows:

TM 990/100M-1 P2 Pin	*743 Terminal P1 Pin*	*Description*
1	9	Protective Gnd
2	13	Transmit data
3	12	Receive data
7	1	Signal Gnd
8	11	Request to send
20	15	Data Terminal Ready

If a preassembled cable is desired, a TM990/503 can be purchased for the purpose.

If the TM990/100M-1 microcomputer was used for the Chapter 3 First Encounter, power was supplied to the microterminal from the TM990/100M module by jumpers installed across the pins J13, J14 and J15 *(Figure 3-12* and *3-13).* These should now be removed; the microterminal disconnected from P2; and the 743 KSR terminal connected to P2 with the referenced cable. Connect ac power to the 743 terminal with a separate cord. Return the jumpers to the spare positions on the board J16, J17, and J18 *(Figure 3-13).* If P1 is to be wired to supply power, use *Figure 3-17* for the connections. *Figure 1* shows the 743 terminal in place instead of the microterminal. It also shows the I/O interface components that will be used for this application connected to P4. If familiar with a 743 terminal, skip the next discussion and go on to the description of the I/O interface components (5MT interface modules).

For those not familiar with the operation of a 743 terminal, reconnect the output board of *Figure 3-9* to P4 and proceed thru the following steps:

1. Turn on the power supplies, the −12V, +12V and +5V, in that order.
2. Turn on the terminal and place it "on line."
3. The system is now ready to receive a program.
4. The terminal uses the TIBUG interactive monitor (TM990/401-1) resident on the TM990/100M-1 in the U42 and U44 sockets. It must be initialized. To do this, press the RESET toggle switch on the TM990/100M *(Figure 1)* and the character "A" or a carriage return (CR) on the terminal. The terminal responds:

```
TIBUG REV.A
?
```

▶9

5. The question mark is the TIBUG prompt symbol saying "what's next?" To enter code or data into memory, press the M (Memory Inspect and Change) command key followed by the address in Memory where the program or routine is to start followed by a (CR). The terminal printout looks like this:

```
?M  FE00 (CR)
```

6. TIBUG responds with the address and the data located at that address such as:

```
FE00=ABCD
```

If the data is not correct and is to be changed, type in the correct data and press either of these options:

A. (CR) to return to TIBUG
B. The space bar to increment to the next memory word location.
C. A minus (−) character to return to the previous word location.

The complete sequence is illustrated here:

```
?M FE00  (CR)
FE00=ABCD        02E0 (Space)
FE02=3D04        FF20 (Space)
FE04=FC36          —  (minus)*
FE02=FF20             (Space)
FE04=FC36        0201 (Space)
FE06=0032             (CR)
?                     *requires pressing "NUM" key
```

7. After an M and the starting address FE00 and a (CR), the total program of *Table 3-2*, should be entered by entering the correct machine code at each address and then pressing the space bar. At the end of the program, exit the memory inspect and change mode by pressing (CR). The terminal responds with the familiar "?". If an error occurs, press (CR), then M and the address at which the error occurred; then repeat the input code.

8. Now the program is ready to run. However, the workspace pointer and the program counter may have to be set; at least the program counter, because it controls where the program starts. The register inspect and change command R is pressed. TIBUG responds with the contents of the workspace pointer. Press the space bar and TIBUG comes back with the program counter contents. Either of these can be changed in the same manner as memory.

Change the contents of the PC to the first address of the program to be run, then type a (CR) and the program is ready to be executed. The total routine looks like this:

```
?R
W=0020 (Space)
P=0846 FE00 (CR)
?
```

The program counter is now set at the starting address of the program of *Table 3-2*, FE00. Usually as the program proceeds, it will set the workspace pointer as needed; thus, no change is made to W in the above routine.

9. The Execute Command, E, runs the program:

```
?E
```

It runs until the RESET switch is pressed. After RESET, the program counter must be reset to FE00. This is done with a (CR), then R, then (Space), then FE00, then (CR), then E to start again.

The necessary details of interfacing and operating the 743 KSR have now been covered. Further information on commands may be obtained by referring to the TM990/100M user's guide. Operation with a 745 KSR acoustical terminal is possible but an EIA/ auxiliary coupler cable kit (Part #983856) must be obtained from a TI Digital Systems Division distributor.

SIMULATING CONTROL OF AN ASSEMBLY LINE

Coupling the KSR-745 terminal to the TM990/100M microcomputer provides a more interactive terminal than the 301 microterminal so that the hardware can be expanded to simulate general kinds of input and output requirements encountered in light-manufacturing assembly lines. In addition, the "assembling" of the program is made easier by using a "line-by-line" assembler, which requires an EIA compatible terminal for this interaction.

Now, obviously, the output board shown in *Figure 3-9,* which contained only simple logic level inverters and an LED display, will not be adequate to provide the reaction power levels that are required for the simulated application. Therefore, new interface modules are needed.

5MT Interface Modules

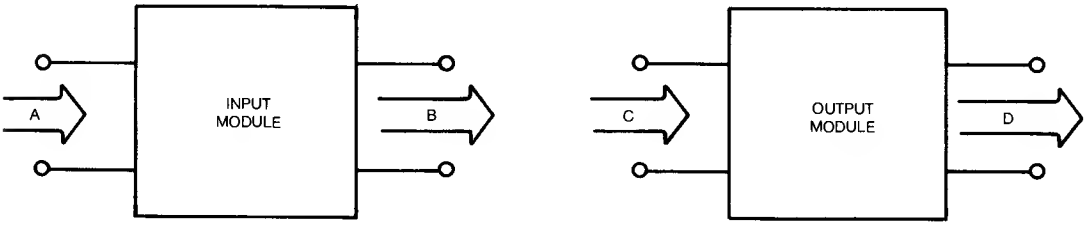
A means must be provided in the system to change input signals from push buttons, limit switches, cam switches, or transducers that are at voltage levels of 90-132 volts ac or 3 to 28 volts dc to standard TTL low-level logic signals between 0 and +5 volts.

In like fashion, means must also be provided in the hardware system to change the low-level logic output signals into power signals up to 28 volts dc or 90 to 132 volts ac. The concept is shown in *Figure 2.*

Texas Instruments supplies modules which meet these requirements. They are called the 5MT I/O modules that are part of a 5TI Control system. A simplified set of specifications for the basic modules is contained in *Table I.*

▶9 The I/O modules are solid-state devices incorporating optical coupler isolation between input and output of 1500 volts for excellent noise immunity. Internal protection is provided to guard against external voltage transients. Each module has an LED status

indicator located at the low-level logic side of the module to help in set-up and troubleshooting. The I/O modules operate from 0-60°C and are designed for 100 million operations. The modules are shown in *Figure 3* with a 5MT43 mounting base which accepts 16 plug-in modules and provides all of the wiring terminals. A logic interface module which mounts on the 5MT mounting base is also shown in *Figure 3.* It provides a serial interface between the 5MT mounting base and a 5TI sequencer. It is not necessary for this application, but is very necessary if other 5TI components are interconnected in the system.



INPUT A	OUTPUT B
90-132VAC	LOW-LEVEL LOGIC
3-28VDC	LOW-LEVEL LOGIC

INPUT C	OUTPUT D
LOW-LEVEL LOGIC	90-132VAC
LOW-LEVEL LOGIC	3-28VDC

***Figure 2.** Input/Output Modules*

CATALOG NO.	TYPE OF DEVICE	RATING		TURN ON TIME (ms)	TURN OFF TIME (ms)
		VOLTAGE	CURRENT		
5MT11-A05L	AC Input	90-132 Vac Input Voltage	35 mA Max	8 Typ. 8.3 Max	12 Typ. 8.3 Max
5MT12-40AL	AC Output	90-132 Vac Output Voltage	3 Amps Continuous (40°C)	4 Max	4 Max
5MT13-D03L	DC Input	3-28 Vdc Input Voltage	30 mA Max	2 Max	2 Max
5MT14-30CL	DC Output	10-28 Vdc Output Voltage	1 Amp Continuous (60°C)		
5MT43	Mounting Base Holds Up to 16 Modules				

***Table 1.** 5MT Module Selection Table*

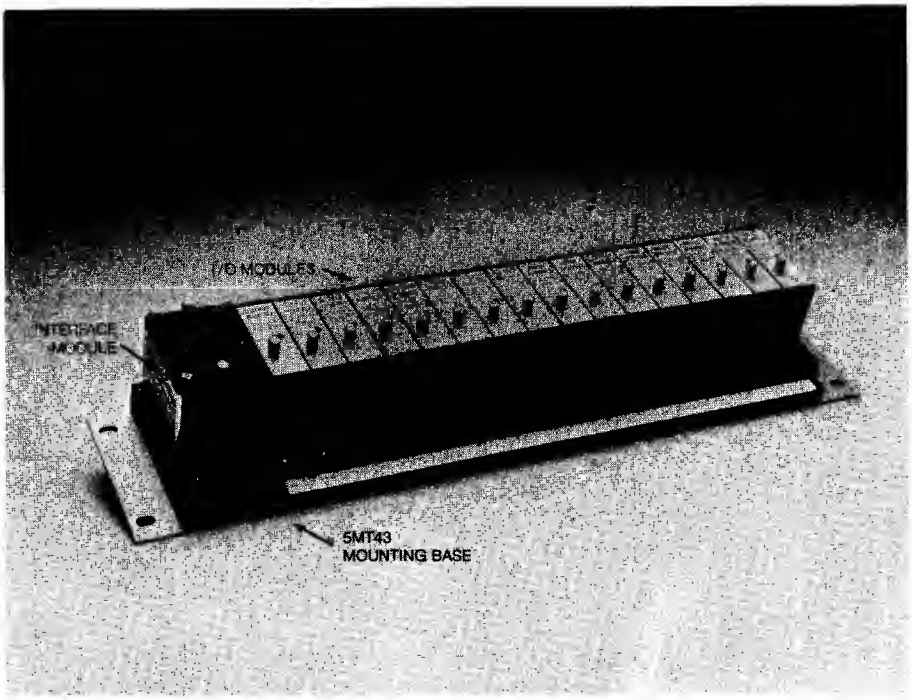


Figure 3. I/O Modules and Mounting Base

The 5MT43 mounting base interfaces with the TM 990/100M-1 microcomputer with a cable to P4, the same 40 pin edge connector that was used for the output board of *Figure 3-9.* The cable connections and hardware required are shown in *Figure 4.* This cable may be wired from scratch or a TM 990/507 cable can be purchased for the purpose. With this cable in place (J1 to the 5MT43 base and J4 to P4 on the TM 990/100M microcomputer module), the major components will be ready to simulate the industrial application. Of course, the additional parts must be purchased:

1 — 5MT43 Mounting Base
2 — 5MT11-A05L Input Modules
2 — 5MT12-40AL Output Modules
2 — 5MT13-D03L Input Modules
2 — 5MT14-30CL Output Modules
1 — TM990/507 Cable (or this can be fabricated as per *Figure 4*)

(Equivalent circuits of 5MT modules are provided in *Figure 5* in case these are to be simulated.)

WIRE LIST

J1	J4	SIGNAL
1	10	MODULE 5
2	18	MODULE 4
3	14	MODULE 2
4	20	MODULE 0
5	24	MODULE 7
6	28	MODULE 9
7	32	MODULE 11
8	36	MODULE 13
9	40	MODULE 15
10	13	GROUND 2
11	19	GROUND 0
12	9	GROUND 5
13	23	GROUND 7
14	27	GROUND 9
15	31	GROUND 11
16	35	GROUND 13
17	39	GROUND 15
21	16	MODULE 3
22	22	MODULE 1
23	12	MODULE 6
24	26	MODULE 8
25	30	MODULE 10
26	34	MODULE 12
27	38	MODULE 14
28	15	GROUND 3
29	21	GROUND 1
30	17	GROUND 4
31	11	GROUND 6
32	25	GROUND 8
33	29	GROUND 10
34	33	GROUND 12
35	37	GROUND 14
36	# 24 GA., STRANDED FOR MODULE V_{CC} (7 – 9V_{dc}@.6A) TERMINATION CAN BE # 6 SPADE LUG, BANANA PLUG, ETC.	

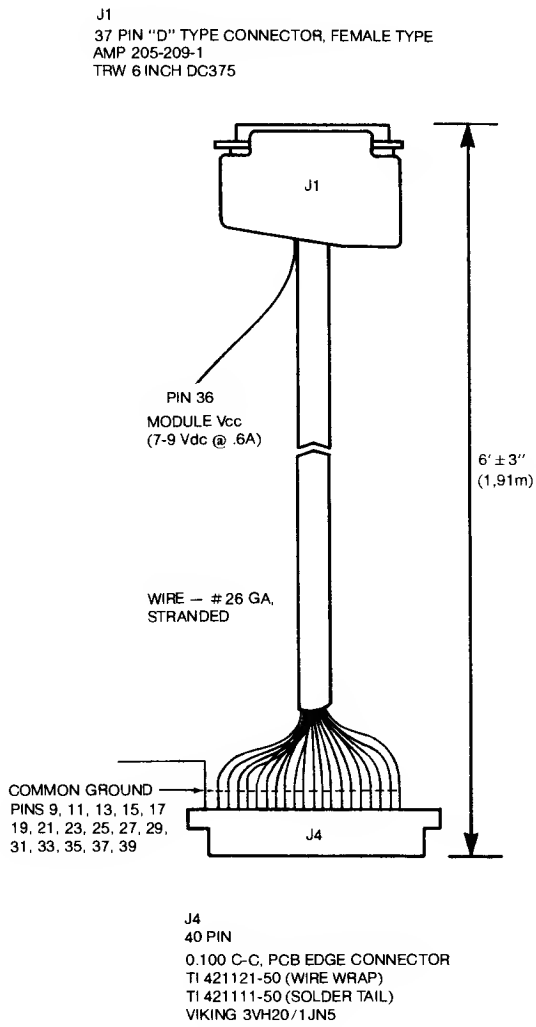


Figure 4. 5MT Interface Cable

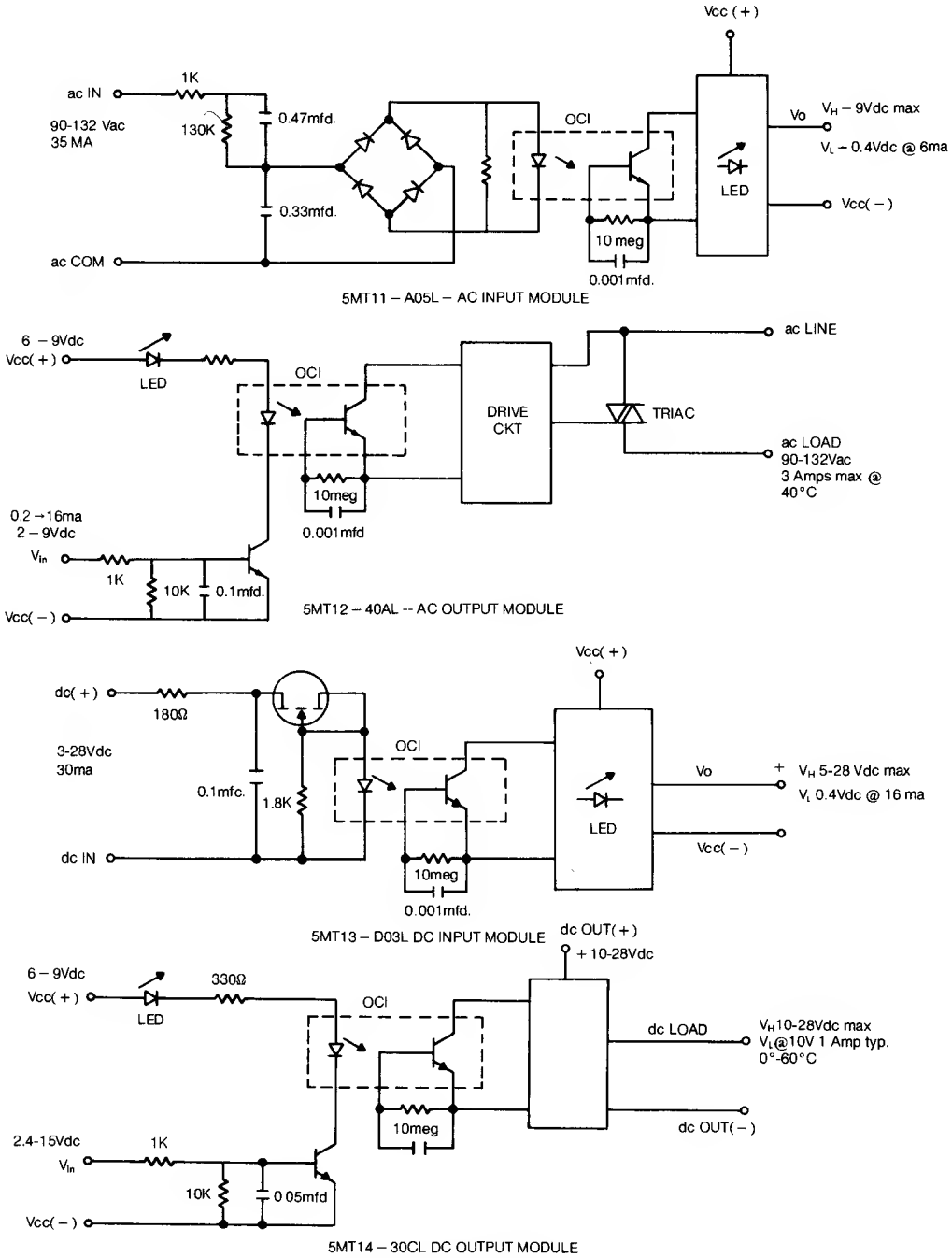


Figure 5. Equivalent Circuits for 5MT Modules

Demonstration Example

The industrial control example, shown in concept form in the block diagram of *Figure 6* is intended to give the reader an insight into the use of a microcomputer based system. Even though no motors, actuators, solenoids, positioning valves, etc. are actually energized, the application demonstrates the means to do it. It also uses real world control voltages in its operation. There will be three modes of operation. To add interest, the system will be programmed so that the user can select the mode of operation.

In the first mode of operation *(Figure 6),* the system is to be programmed to accept inputs and switch a corresponding output according to the state of the input. Switches are going to apply input industrial level dc voltages to the dc input modules and input industrial level ac voltages to the ac input modules. Output lights powered by industrial level dc and ac voltages will be activated corresponding to the state of the input signal. Such a mode of operation simulates switch closures on the assembly line requesting an output reaction.

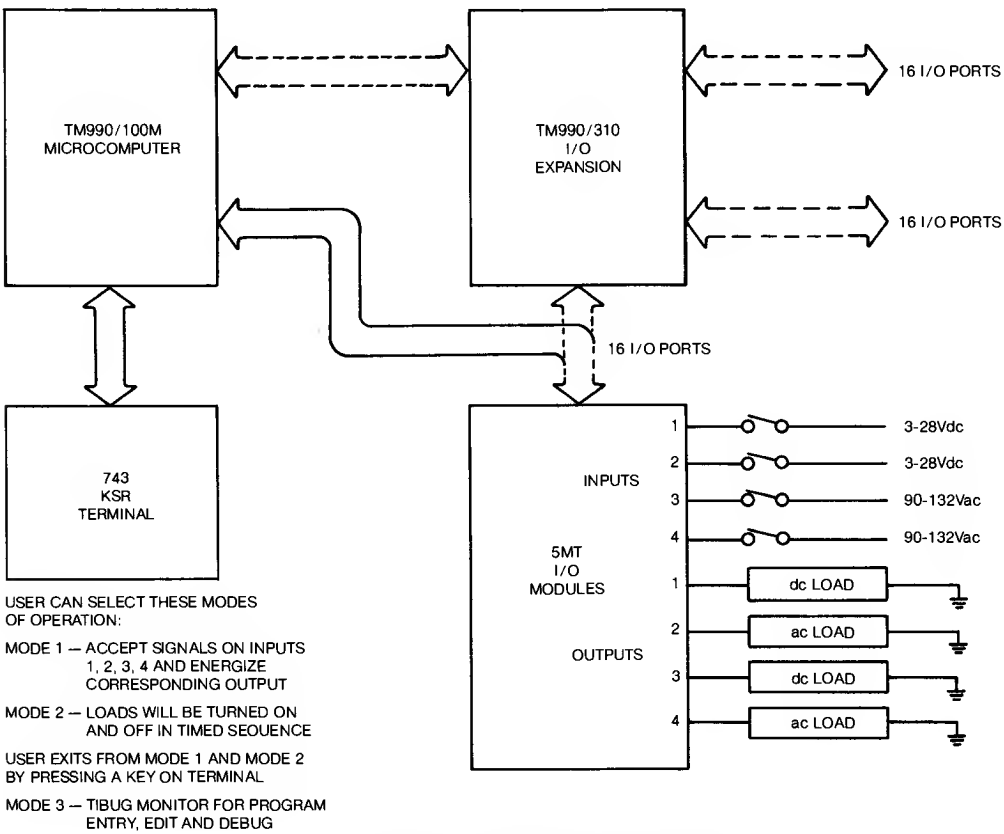


Figure 6. *Application Block Diagram*

The second mode of operation is very similar to the light sequence of Chapter 3. However, with the 5MT modules controlling either + 12Vdc light bulbs or 110Vac light bulbs, it demonstrates a different means of timed sequence control. It uses the real time clock in the TMS9901 in the microcomputer module for a much greater precision. The system is to be programmed so the time can be varied easily. There is to be an added feature in the first and second mode. The system has a routine that allows the user to choose the mode of operation by selecting a key on the keyboard.

A third mode returns the system to the TIBUG interactive monitor. In this mode, the program can be edited, debugged or added to and initial conditions can be changed.

Lets see how this can be accomplished.

The TM990/100M Microcomputer Module

Figure 7 is a much more detailed block diagram of the TM990/100M microcomputer. Four areas are of particular interest:

1. More details on the TMS9901;
2. Details on the TMS9902—this device was not discussed at all in Chapter 3;
3. The addition of a TM990/310 module to the system to obtain I/O expansion; and
4. Expansion of resident RAM and ROM.

Note in particular that the TM990/100M-1 comes populated with 256 words of RAM and 1K words of ROM (which is the TIBUG EPROM resident monitor). Also note the address bus goes to the I/O interface units. Thus, I/O is selected with addresses in the same fashion as memory words. In addition, the four busses—address, control, data and CRU are available for off-board expansion. This is the way I/O expansion through the TM990/310 module is controlled. 512 words of RAM can be provided on the board. Further expansion is possible with off-board memory. Additional ROM, expandable on the board to 4K, will be used when the line-by-line assembler (LBLA) is used.

TMS9901

The TMS9901, programmable system interface, shown in *Figure 7* was previously shown in the block diagram of *Figure 3-17.* Only one portion of it was used to control output signals and detect an input signal. Now all of the functions will be examined in more detail.

9 The block diagram of the TMS9901 in *Figure 8* will be used to identify the major functions.

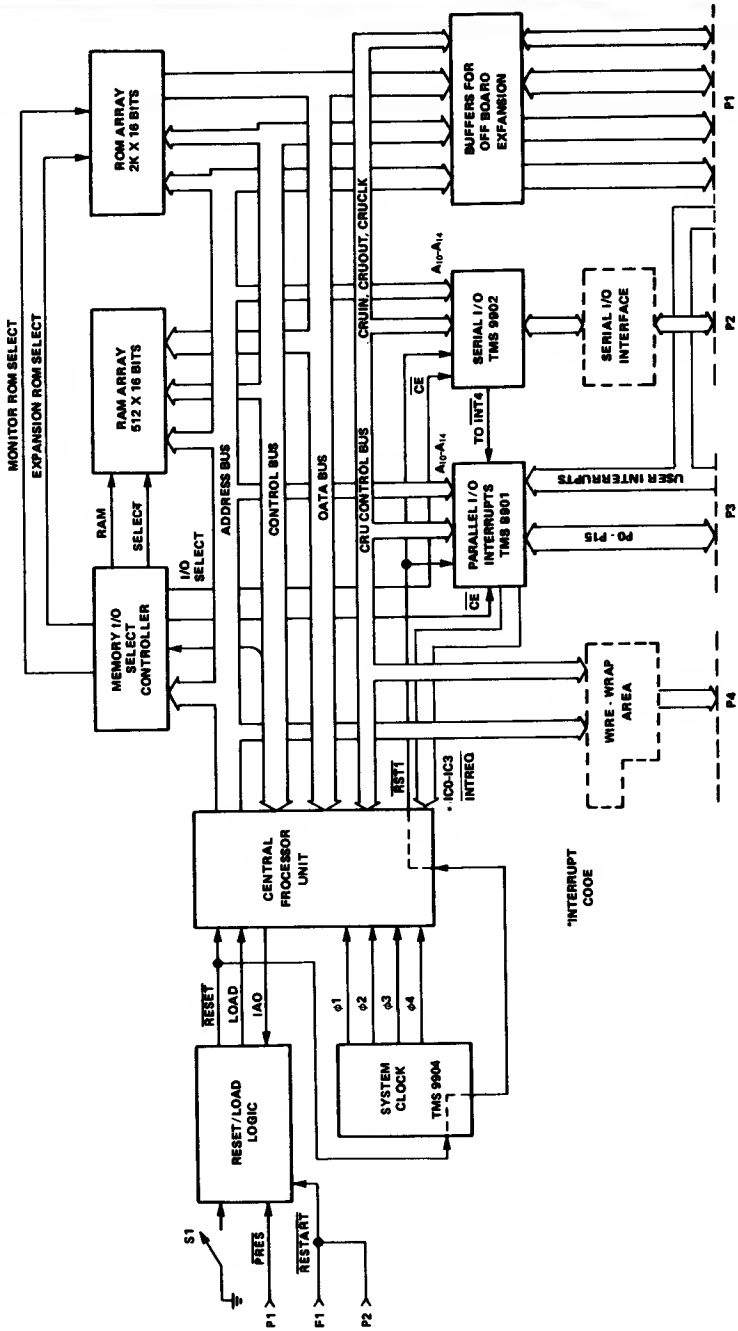


Figure 7. TM 990/100M Block Diagram

First of all, since the TMS9901 is a programmable systems interface, as shown in *Figure 7,* it is designed to handle parallel input and output signals. The input signals are either data inputs or special signals called interrupts. Interrupts are special signals because they interrupt the main program routine of the microcomputer and ask for service from the microcomputer to do some selected priority subroutine or subprogram. In *Figure 8,* the data output paths and input paths and the interrupt paths are identified. The 22 pins are programmable and divide into three groups as follows:

***Table 1.** Programmable Pin Functions*

GROUP	*NAME*	*IN*	*OUT*	*INT*	*COMMENT*
1.	$\overline{\text{INT}}$ 1	X		X	Principally inputs but may be used as interrupts
	$\overline{\text{INT}}$ 2	X		X	
	$\overline{\text{INT}}$ 3	X		X	
	$\overline{\text{INT}}$ 4	X		X	
	$\overline{\text{INT}}$ 5	X		X	
	$\overline{\text{INT}}$ 6	X		X	
2.	$\overline{\text{INT}}$ 7/P15	X	X	X	Fully programmable as inputs, outputs or interrupts
	$\overline{\text{INT}}$ 8/P14	X	X	X	
	$\overline{\text{INT}}$ 9/P13	X	X	X	
	$\overline{\text{INT}}$ 10/P12	X	X	X	
	$\overline{\text{INT}}$ 11/P11	X	X	X	
	$\overline{\text{INT}}$ 12/P10	X	X	X	
	$\overline{\text{INT}}$ 13/P9	X	X	X	
	$\overline{\text{INT}}$ 14/P8	X	X	X	
	$\overline{\text{INT}}$ 15/P7	X	X	X	
3.	P6	X	X		Programmable as inputs or outputs.
	P5	X	X		
	P4	X	X		
	P3	X	X		
	P2	X	X		
	P1	X	X		
	P0	X	X		

In addition to the input/output function, the TMS9901 also has incorporated a clock function. This was identified in *Figure 8,* but is further detailed in *Figure 9.* This real time clock will be used in this application as an interval timer for the Mode 2 light sequence. To provide this function, the clock register is loaded with a value, (just like in Chapter 3); however, now the register automatically decrements after it is loaded. When it has decremented to zero, an interrupt signal is sent out to be processed by the interrupt path of the TMS9901. It won't be used for this application, but an elapsed time counter can be implemented by reading the value of the clock read register (*Figure 9*) periodically to determine how much time has elapsed from an established start.

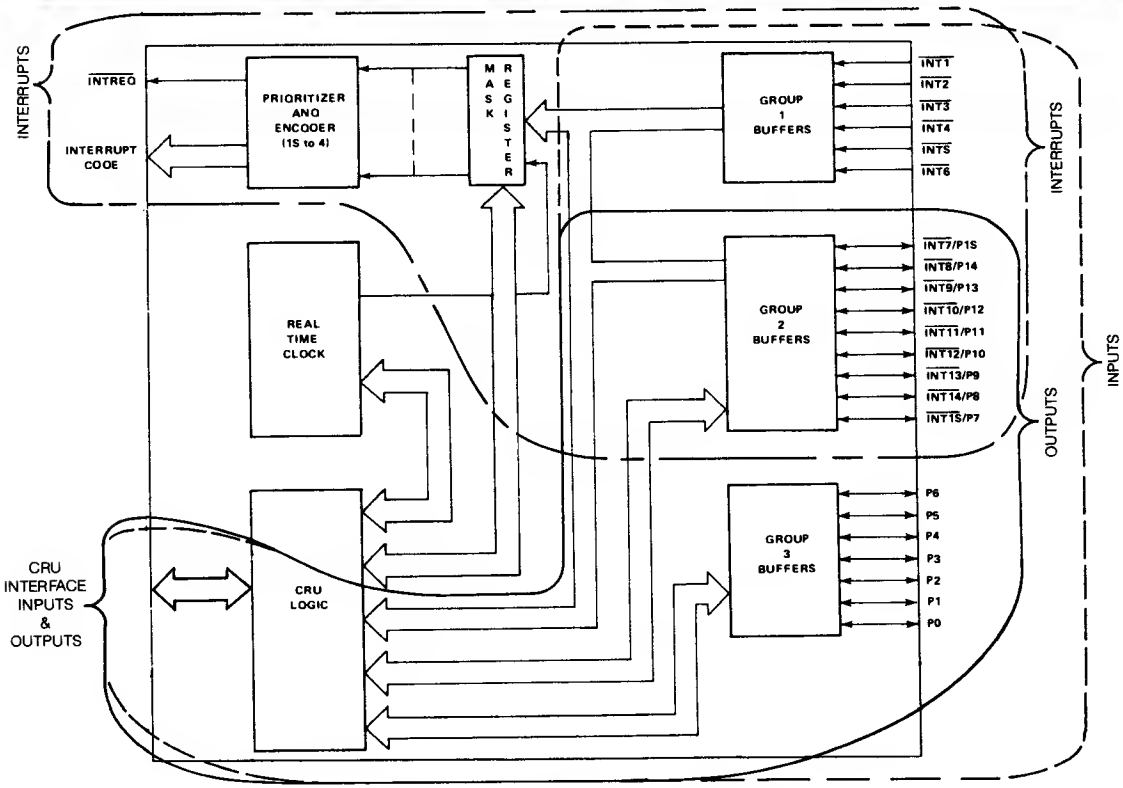


Figure 8. TMS 9901 Block Diagram

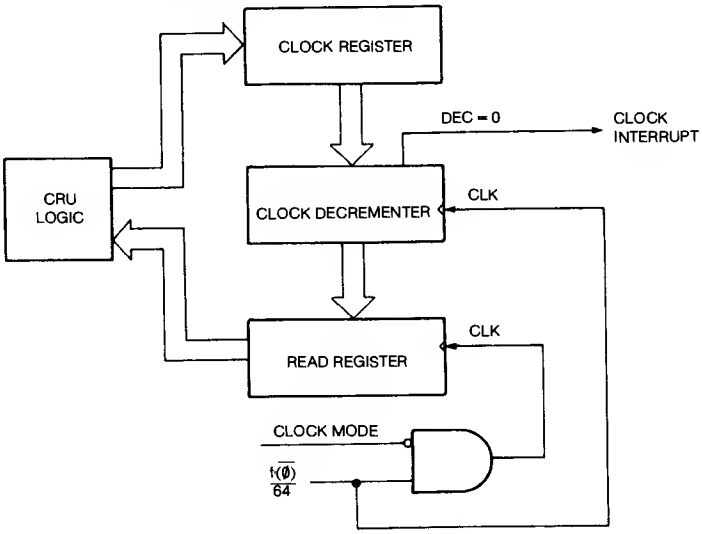


Figure 9. Real Time Clock

Interface with the 9900

It is important to understand the communications channels between the TMS 9901 and the 9900 microprocessor in the microcomputer. Basic concepts need to developed to understand how the algorithm for this application is programmed.

The communications channels are shown in *Figure 10.* They are presented in somewhat different form than shown previously in Chapter 3.

The main data link between the 9900 and the 9901 and subsequent inputs and outputs is via a serial data link. The line CRUIN transfers data from the 9901 to the 9900 in serial format. Again in serial format, the line CRUOUT transfers data from the 9900 to the 9901. The transfer of data out is synchronized by the signal CRUCLK, which comes from the 9900 and specifies that data is valid on the CRUOUT line. Remember that CRU means Communications Register Unit.

In order to manipulate data from the CRU to and from the inputs and outputs and the real time clock of the 9901, five CRU instructions are included in the instruction set. They are:

1.	SBO	Set bit to one
2.	SBZ	Set bit to zero
3.	TB	Test bit
4.	LDCR	Load CRU Register
5.	STCR	Store CRU Register

In Chapter 3, it was demonstrated how individual bits could be selected and set to a "1" or a "0" by using the SBO and SBZ instructions. If this hasn't been reviewed, it would be helpful to do so.

Not only can individual bits be manipulated, but data can also be transferred in blocks of from one to 16 bits. The multiple bit instructions LDCR, "Load CRU Register", and STCR, "Store CRU Register", are used for this purpose. Since this application requires the use of these multiple-bit instructions, further time will be spent explaining them in more detail.

Basic Concepts

Figure 11 summarizes the basic concept of the programmable input-output capability of the 9900 family. In this example, a microcomputer, the TM990/100M, which contains a 9901, and a TM990/310 module, which contains 3 additional 9901's are used. Such an arrangement expands the I/O capabilities by 48 inputs or outputs.

Industrial control applications like the one that is being simulated normally require many inputs and outputs. Much more capability is available because I/O could be expanded to 4096 ports by adding more units and continuing the example of *Figure 11.*

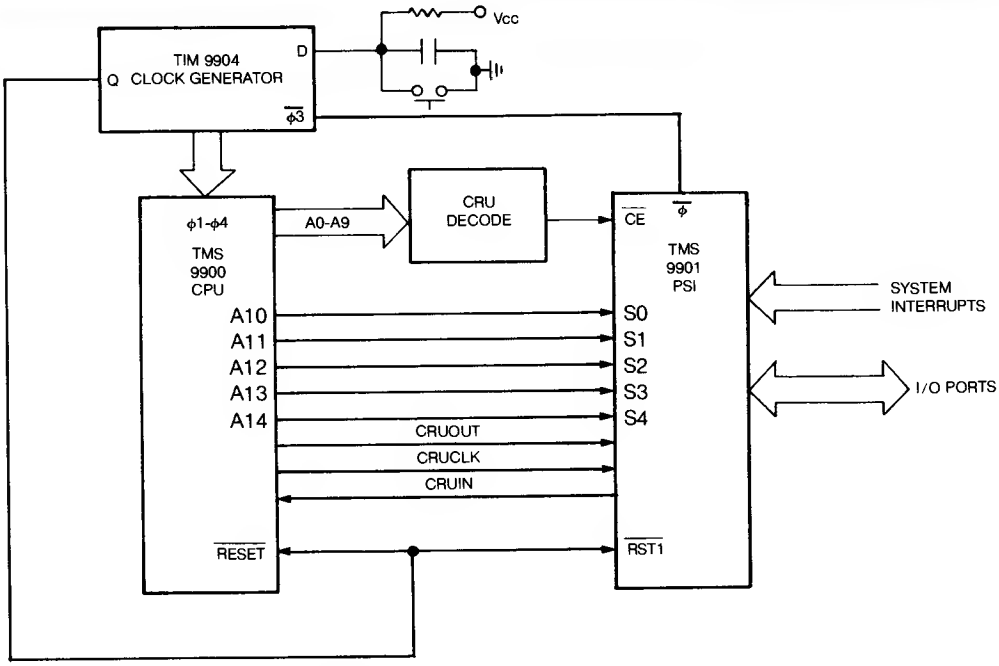


Figure 10. TMS 9900-TMS 9901 Interface

As shown, the data moves over CRUIN and CRUOUT in a serial format from the 9900 to the 9901, or vice versa. When the instruction LDCR is used, the data is flowing from the 9900 to the 9901 over CRUOUT. The first bit to arrive serially (the least significant bit) is latched in the zero bit position of the 9901 determined by the CRU select bit, subsequent bits that arrive are then placed in bits, 1, 2, 3-12, 13, 14, 15 at each CRUCLK pulse. Such is the case if 16-bits are being processed. Any number of bits from 1 to 16 may be processed at the user's discretion. When flowing out on CRUOUT, the transfer rate is determined by CRUCLK. When flowing in on CRUIN, the 9900 microprocessor transfers the data present on the inputs during ϕ_1 of clock cycle 2 of the machine cycles.

What determines where the bit position starts? The select bits on S_0-S_4 in the 9901 (*Figure 10* and *11*) are distributed as A_{10} thru A_{14} from the 9900. Since this address is distributed to each 9901 shown, and since CRUOUT goes to each 9901, the data out would tend to be latched in each 9901. This is prevented by the chip enable (CE) signal. The only CE that is active low is the one decoded from the corresponding base address for the correct 9901. Bits A_0 thru A_9 provide the additional address information. For example, if in *Figure 11* the 9901 on the TM990/100M board is to be used for the I/O, then hardware base address 0080_{16} is used. If the second 9901 on the TM990/310 module is used, the hardware base address is 0140_{16}.

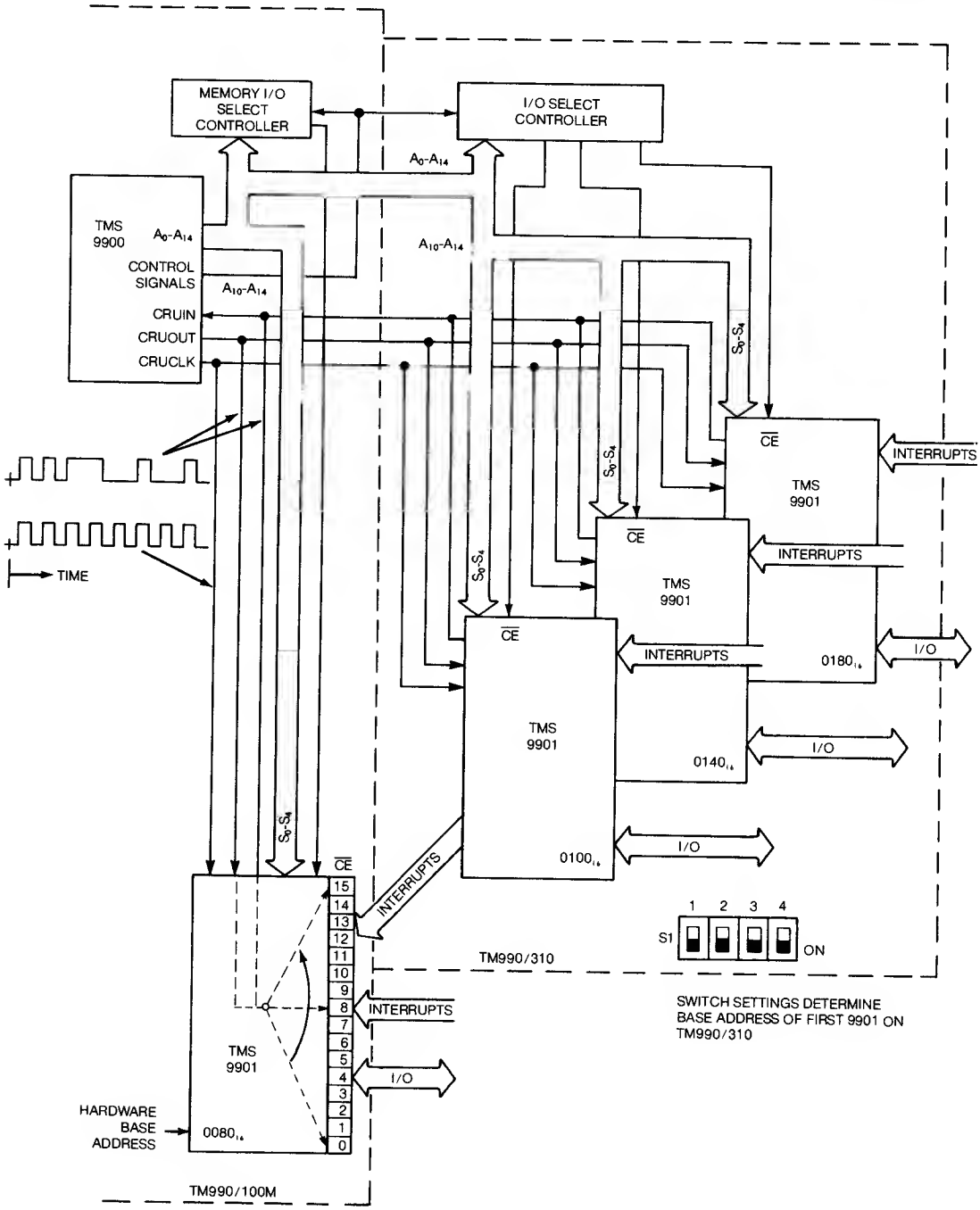


Figure 11. Basic Concept of Programmable I/O

In *Figure 3-23,* for the single bit instructions SBO, SBZ, and TB, the effective CRU bit address is obtained by adding a signed displacement to the 9901 base address. For the multiple bit instructions, the effective CRU bit address is computed in the same way; however, the base address is the address of the first bit. From there, the address is incremented by the number of multiple bits to be transferred. The LDCR instruction format contains a C field which specifies the number of multiple bits to be transferred. For example:

```
LDCR   R1,9
```

would instruct the microcomputer to send out (output) the 9 least significant bits of register R1. The 9 would be in the C field of the instruction format. Before the LDCR instruction in the program, there is an instruction that loaded the software base address of the particular 9901 to be used into the correct workspace register 12. Recall that WR12 is the register where the software base address is always located for a CRU instruction. This will become clearer as a specific example is discussed later. What is important is that the software base address for the 9901 must be loaded into workspace register 12. However, this is not completely straightforward. For example, if the 9901 on the TM990/100M microcomputer is to be addressed with a LDCR or STCR instruction, the 0080_{16} hardware base address must be displaced to the software base address 0100_{16} when it is loaded into WR12. This is necessary because bit 15 of WR12 is not used in the calculation of the effective CRU bit address. The concept, described in *Figure 3-23,* is shown again in *Figure 12.*

It is probably obvious that the STCR instruction operates in the reverse of the LDCR. The data from the input pins on the selected 9901 is incremented bit by bit and sent to the CRU in the 9900 over CRUIN. The final result of a STCR instruction is that the 9900 processor stores the input data in RAM in a specified location called out in the instruction. In like fashion, when LDCR is used the data transferred to the output is obtained from a RAM location called out in the instruction. This is a distinct advantage in that it need not be a register. The specifics on the data transfers are shown in *Figure 13.*

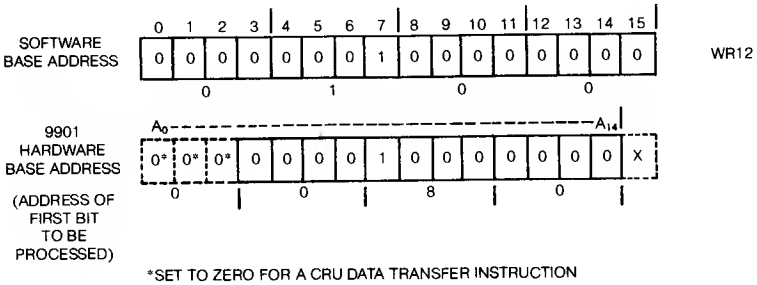


Figure 12. *9901 Base Address*

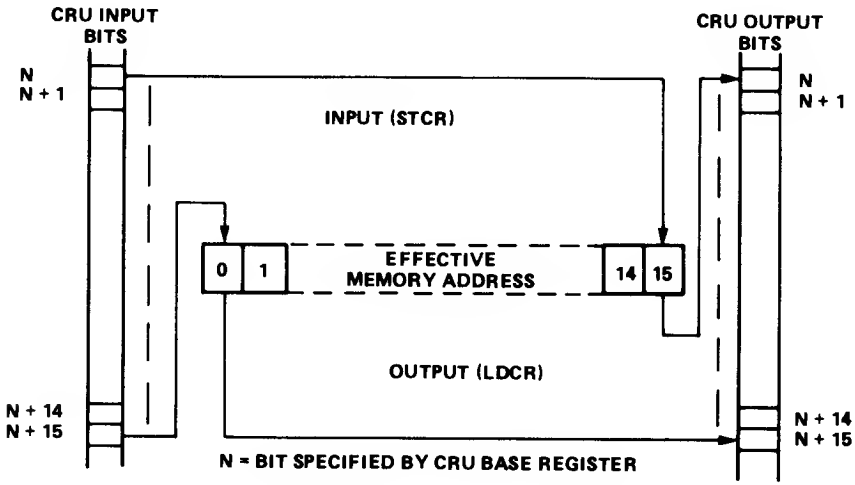


Figure 13. LDCR / STCR Data Transfers

Interrupts

Another form of input is the special one called interrupt, so named because it asks the microcomputer to interrupt the program routine presently in process.

In *Figure 8,* it was pointed out that there are only certain lines on which an interrupt is accepted. Group 1 of the 9901 pins may be used for 6 interrupts. Up to 15 interrupt signals can be programmed by using Group 2 pins.

What value do interrupts have? First, they allow external events to interrupt the current program so that the program can provide service to an external device. In so doing certain pieces of data must be saved in order to return to the same point in the program that was interrupted. This allows the program to continue correctly after the interrupt has been serviced. Secondly, interrupts provide quick response. Third, they provide a priority to be established for time critical events. Certain interrupts are more important than others. The user decides the priority. To set up priorities for interrupt signals, a means is provided to honor the priority established. In the 9900 system family, this is called enabling a valid interrupt through a "masking" of interrupts.

Masking means to enable or disable. *Figure 14* shows that the TM990/100M microcomputer module has two levels of masking. One mask must be enabled to pass the interrupt signals through the 9901 and another must be enabled at the 9900 microprocessor. The value in bits 12, 13, 14 and 15 of the status register set the priority level of the interrupt mask in the 9900. Any interrupt equal to or higher than the priority level is enabled and allowed to interrupt the microcomputer.

Masking

Figure 15 is a block diagram of the 9901 control logic illustrating how the masking is accomplished. In order to enable an interrupt, MASK must equal 1 for the particular interrupt pin. When several interrupts are present at the same time, the control logic encodes the enabled interrupt inputs and sends to the 9900 microprocessor a code that represents the highest level of interrupt that has been enabled. $\overline{\text{INT}}$ 1 is the highest level, $\overline{\text{INT}}$ 2 is next and so on down to 15. In addition, an $\overline{\text{INTREQ}}$ active low signal is also sent to the 9900. The code sent on lines IC0 through IC3 is shown in *Table 2.* Level zero is used by $\overline{\text{RESET}}$ and will be covered later.

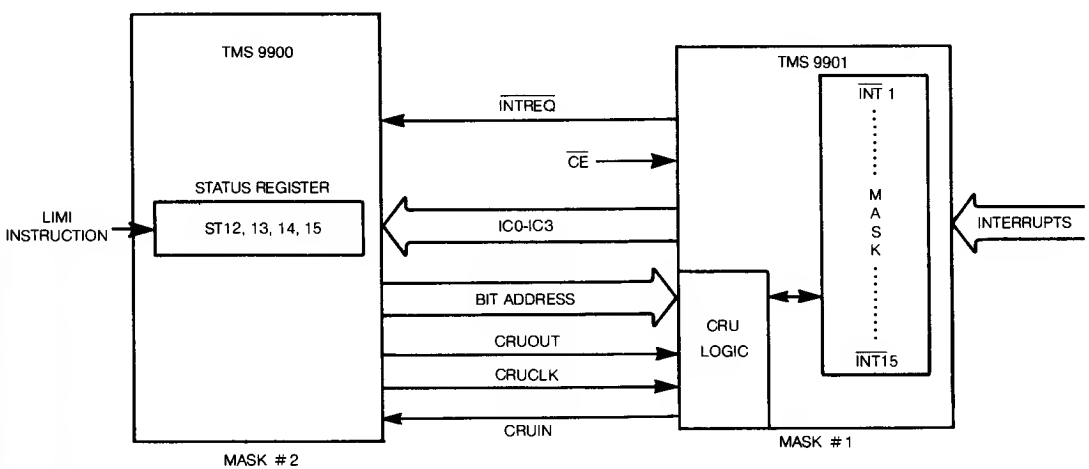


Figure 14. Interrupt Masking

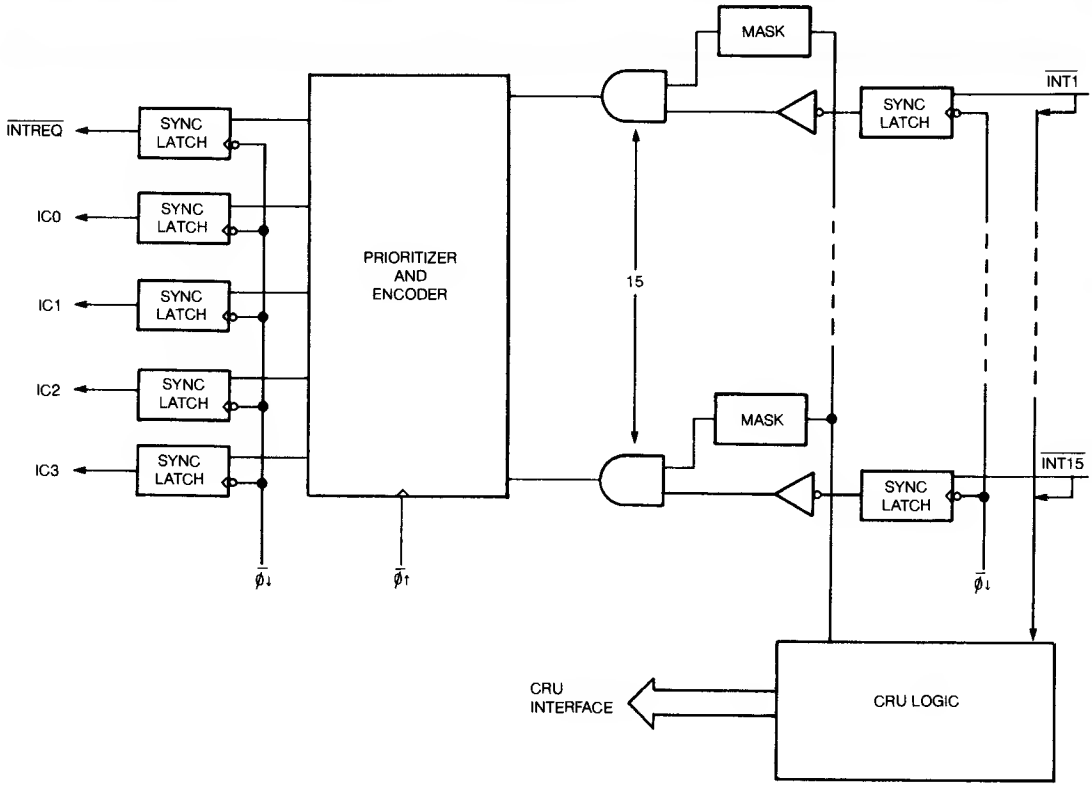


Figure 15. Interrupt Control Logic

The code on IC0 thru IC3 is compared to the status bits ST12, 13, 14 and 15 in the status register of the 9900. The priority level loaded into the interrupt mask of the 9900 enables that level and all higher priority levels as well. If the interrupt level set up in ST12, 13, 14 and 15 is higher than the interrupt level received, the interrupt is not enabled. If the interrupt received is higher in level than the priority level, then the interrupt is enabled and all higher level interrupts as well. This is shown in *Figure 16.*

The code on IC0-IC3 is as follows:

Table 2. Interrupt Code Generation

INTERRUPT/STATE	*PRIORITY*	*IC0*	*IC1*	*IC2*	*IC3*	*$\overline{INTREQ}$*
$\overline{INT}$ 1	1 (HIGHEST)	0	0	0	1	0
$\overline{INT}$ 2	2	0	0	1	0	0
$\overline{INT}$ 3/CLOCK	3	0	0	1	1	0
$\overline{INT}$ 4	4	0	1	0	0	0
$\overline{INT}$ 5	5	0	1	0	1	0
$\overline{INT}$ 6	6	0	1	1	0	0
$\overline{INT}$ 7	7	0	1	1	1	0
$\overline{INT}$ 8	8	1	0	0	0	0
$\overline{INT}$ 9	9	1	0	0	1	0
$\overline{INT}$ 10	10	1	0	1	0	0
$\overline{INT}$ 11	11	1	0	1	1	0
$\overline{INT}$ 12	12	1	1	0	0	0
$\overline{INT}$ 13	13	1	1	0	1	0
$\overline{INT}$ 14	14	1	1	1	0	0
$\overline{INT}$ 15	15 (LOWEST)	1	1	1	1	0
NO INTERRUPT		1	1	1	1	1

The output signals will remain valid until the corresponding interrupt input is removed, or an interrupt service routine disables (MASK = 0), or a higher priority enabled interrupt becomes active. When the highest priority enabled interrupt is removed, the code corresponding to the next highest priority enabled interrupt is output. If no enabled interrupt is active, all CPU interface lines ($\overline{INTREQ}$, IC0-IC3) are held high.

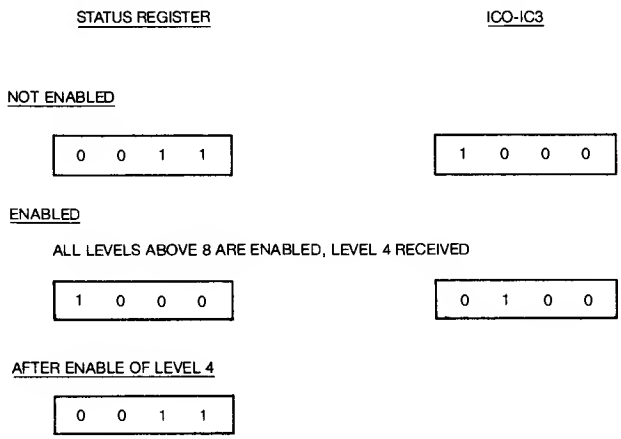


Figure 16. Interrupt Mask at 9900

Remember to enable an interrupt, say $\overline{\text{INT}}$ 1, a "1" must be placed in the latch (MASK = 1) for the CRU bit (pin) associated with that interrupt. Likewise, to disable an interrupt, a "0" must be placed in the latch (MASK = 0) associated with the pin receiving the particular interrupt.

To mask any of the interrupts from 1 through 15, the 9901 must be in the interrupt mode. The zero select bit of the 9901 is the control bit for this. As shown in *Figure 23,* if this control bit is a zero, the 9901 is in the interrupt mode. If it is a "1", the 9901 is in the clock mode.

Enabling or disabling the mask in the 9901 for the interrupts may be accomplished by individual bit instructions SBO and SBZ or by a multiple bit LDCR instruction.

All masks can be disabled simultaneously by performing a hardware ($\overline{\text{RESET}}$) or software ($\overline{\text{RST}}$ 2) reset.

Signals appearing on the inputs to the 9901 will be accepted as interrupt signals by the 9901 if the masks are enabled. The priority code for the highest priority level interrupt simultaneously received will be sent to the 9900 via the code lines, IC0-IC3, as well as the signal $\overline{\text{INTREQ}}$. If the interrupt mask in the 9900 has the level enabled, the interrupt is accepted and serviced.

Saving Items on Interrupt

When an interrupt occurs, data pertinent to the "state of the machine" must be saved. This provides a return to the interrupted program so that the program can continue to execute properly. For example, when an interrupt occurs, the CPU suspends its current program routine to do the subroutine called for by the interrupt. How does it do this? As any program executes, the "state of the machine" at any time is determined by the value in the program counter, the value in the workspace pointer, the value in the status register, and the contents of the registers in the workspace register file. Each of these is saved through a "context switch" when an interrupt occurs. Full details are available in Chapter 4. A brief summary will be covered here for convenience.

Interrupt Vectors — Context Switching

To execute an interrupt, here's what happens. There are special places in memory reserved for the address that contains a new workspace pointer for a given interrupt. In addition, in the next word following there is a new program counter value. These special places in memory are called interrupt vector traps and the two addresses — one for workspace and the other for the program counter — have the name "interrupt vector."

Figure 17 illustrated the process. A valid interrupt is received and its level points to its vector. The vector contains a new workspace pointer and a new program counter value. The program shifts and points to the new workspace. In the new workspace, the microprocessor stores the old workspace pointer in R13, the old program counter in R14 and the old status register in R15. These old contents are always put in the same place in the new workspace — R13, R14 and R15.

After all this occurs, the program counter with its new value executes the interrupt subroutine. The last instruction in this subroutine, RTWP, is an instruction to return to the interrupted routine. RTWP — "Return with Workspace Pointer" — returns to the interrupted routine by loading the contents of R13 into the workspace pointer (R13→WP), R14 into the program counter (R14→PC), and R15 into the status register (R15→ST) and then executes the instruction pointed to by the program counter. In so doing, the system has returned to the interrupted program at the point of interruption and begins execution using the old workspace. This is illustrated in *Figure 18.*

Note: When the interrupt priority level comes into the 9900 and the interrupt is enabled, a number one less than the interrupt level received is placed in the interrupt mask in the status register as shown in *Figure 16* to prevent lower level interrupts from occurring during the servicing of the present interrupt. If a higher priority interrupt occurs, a second interrupt context switch takes place after at least one instruction is executed for the first interrupt routine. This means that an interrupt service routine may begin with a LIMI instruction which can load an interrupt mask in the 9900 which disables other interrupts. Completion of the second interrupt passes control back to the first interrupt using the RTWP instruction.

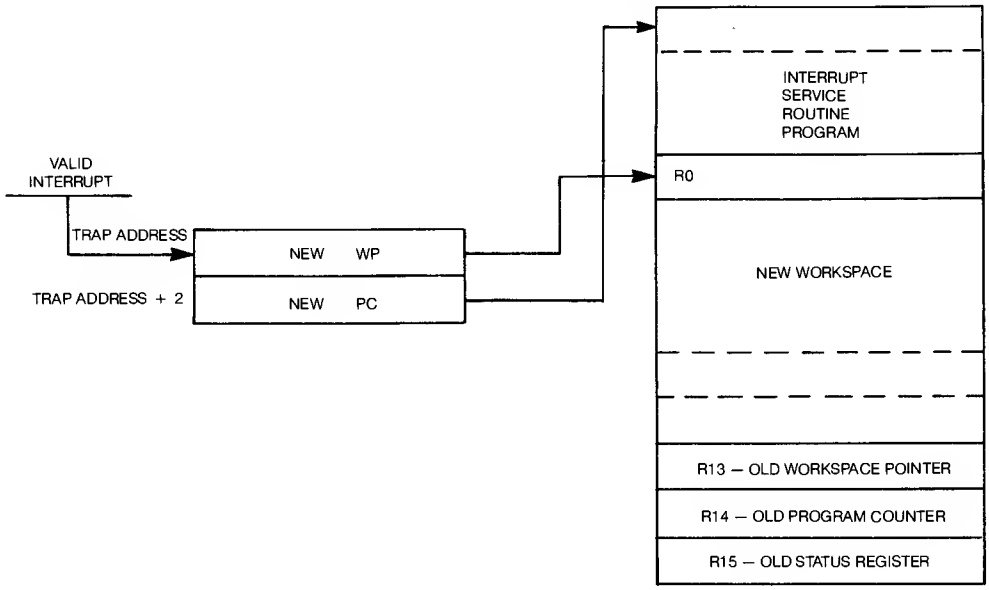


Figure 17. Interrupt Context Switch — New Workspace and Saving Old WP, PC, and ST Data

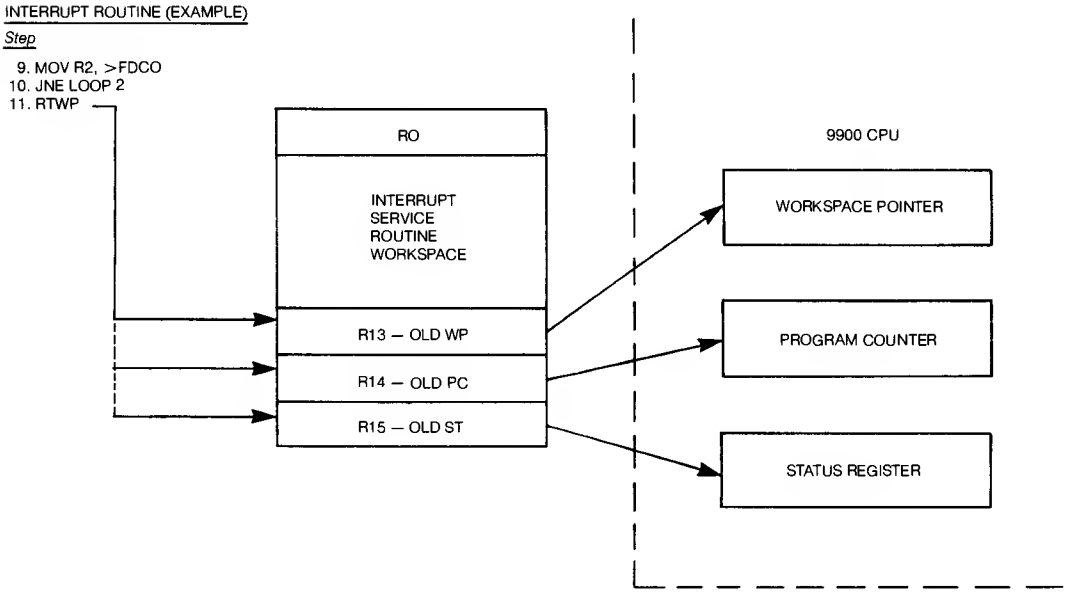


Figure 18. Interrupt Context Switch Returning to Interrupted Program

Memory Map and Interrupt Vectors

In *Figure 19,* the memory map of the TM990/100M microcomputer module is shown. Note that the first words of memory from hexadecimal addresses 0000_{16} to $07FE_{16}$ are dedicated memory. Addresses 0000_{16} to $003E_{16}$ are reserved for the 16 interrupt transfer vectors. These are detailed further in Figure 20. Each interrupt vector has two words of memory — one for the workspace pointer, one for the program counter.

There are two interrupt vectors, $\overline{INT}$ 3 and $\overline{INT}$ 4 that will be of particular interest for they have important use in the program for this application.

Notice that interrupt 0 in *Figure 20* is used for $\overline{RESET}$ and that values have already been placed in the vector locations for interrupt 3 and interrupt 4.

When an $\overline{INT}$ 3 level is received, it points to the interrupt 3 vector. The context switch occurs and at $000C_{16}$ it obtains the value $FF68_{16}$ for the workspace pointer and at $000E_{16}$ the value $FF88_{16}$ for the program counter. The context switch operations store the old context registers in the new workspace pointed to by $FF68_{16}$. Then the interrupt service routine begins by executing the instruction pointed to by $FF88_{16}$. Since there are valid reserved locations for only two memory words at the $FF88_{16}$ location, the instruction pointed to by $FF88_{16}$ and $FF8A_{16}$ must branch to another section of memory where the remaining interrupt service routine is located.

A similar sequence of events occurs when an $\overline{INT}$ 4 level interrupt signal is received, except that the workspace pointer value is $FF8C_{16}$ and the program counter value is $FFAC_{16}$.

The remaining interrupt vectors do not have values. These would be programmed into EPROM locations by the user as the need arises.

For the interrupt 3 and 4 service routines, 16-word workspaces are provided, pointed to by $FF68_{16}$ and $FF8C_{16}$. These are reserved and must be noted by the programmer.

The microcomputer must always start from initial conditions. These are usually started by a reset. The vector space required for the initial value of the workspace pointer and the program counter resides in the reserved memory spaces 0000_{16} for WP and 0002_{16} for PC, as shown in *Figure 20.* The 16 interrupt vectors at 0000_{16} to $003E_{16}$ are in read only memory and cannot be changed unless the read only memory is reprogrammed.

As the extended application program is written, it must be remembered that the TIBUG monitor needs workspaces. The space from $FFB0_{16}$ to $FFFB_{16}$ is reserved for this purpose. This is noted because this space cannot be used for data or program memory in the application.

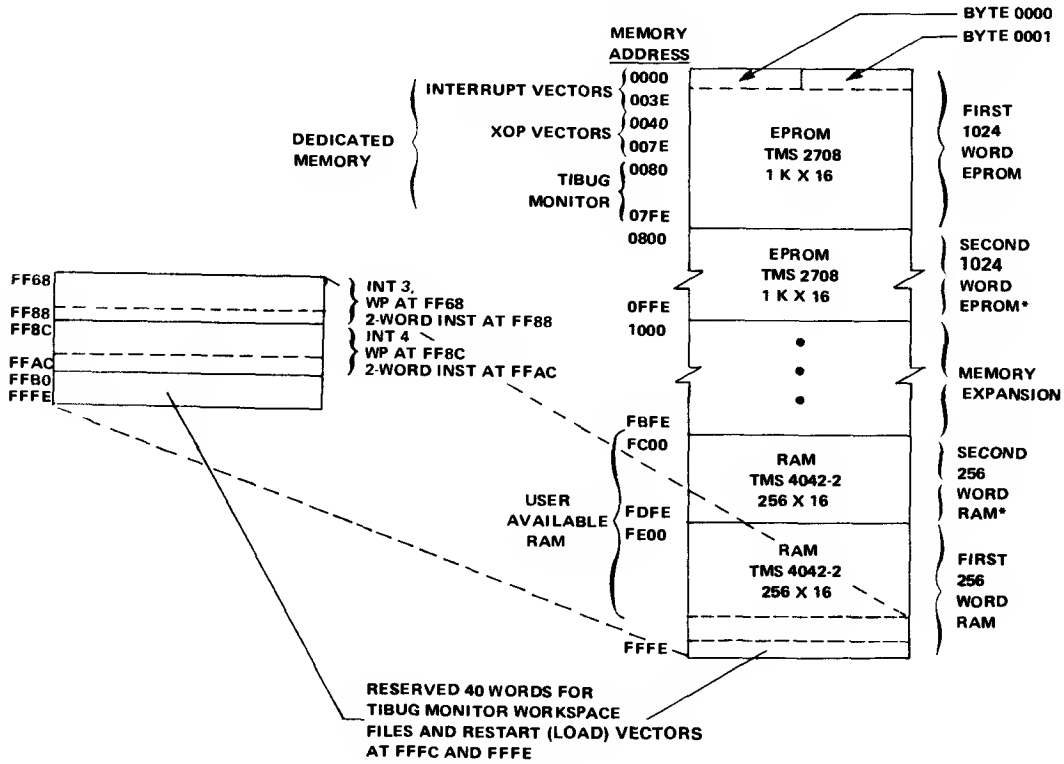


Figure 19. Memory Map

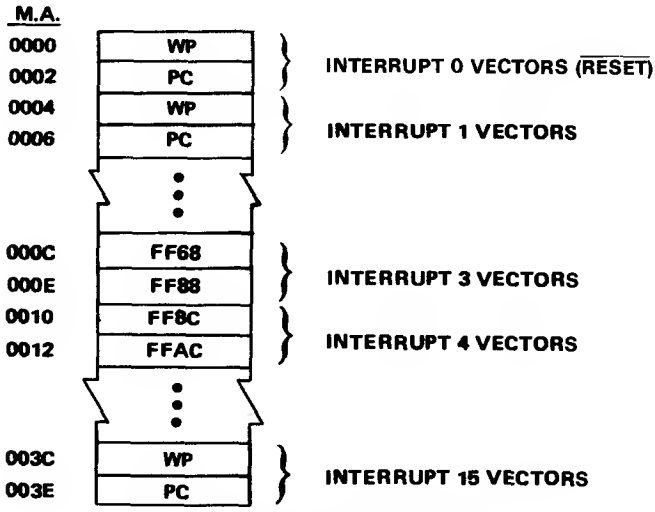


Figure 20. Interrupt Trap Locations

Extended Operations (XOP's)

Refer to *Figure 19* which shows the read-only memory space reserved for software interrupt vectors. Memory words from 0040_{16} to $007E_{16}$ are XOP vectors. As with interrupts, each XOP vector has a word containing a workspace pointer value and a next word containing a program counter value.

XOP instructions point to XOP vectors which point to new workspace pointer and program counter values in a similar way to what was just described for interrupts.

An instruction calling for an XOP (extended operation) is a means of switching from the main program to a subroutine. It has a special calling sequence and it functions as though the routine were a single instruction added to the 9900 set of operation codes, hence the name "extended operation".

For example, the TIBUG monitor in the microcomputer contains seven XOP routines that perform input/output functions with the terminal. These are as follows:

XOP	*Description*
8	Write one hexadecimal character to terminal
9	Read hexadecimal word from terminal
10	Write 4 hexadecimal characters to terminal
11	Echo character
12	Write one character to terminal
13	Read one character from terminal
14	Write message to terminal

Two of these XOPs are used in the extended application example. XOP 11 is used to read a character from the terminal and at the same time print it at the terminal. XOP 14 is used to print out instructions to explain how the program operates. Some of these XOPs call other XOPs. Further detail on XOPs can be obtained in Chapter 5 and 6.

Printing a Message

A message at the beginning of the program which will be developed for this application tells the user to select the mode of operation. XOP 14 is used to write the message. The instruction

```
XOP @MSG1,14
```

is used. XOP 14 identifies that the subtask is "Write message to terminal". A context switch takes place. The vector at location 14 of the reserved XOP vector memory space provides the WP and the PC values. The PC value provides the first subtask instruction and the subroutine continues until the subtask is complete and the program returns to the main program.

Suppose the message identified with the label MSG1 is "THIS IS A SAMPLE." Its coding would look like the following:

LINE	ADDRESS	CODE	MESSAGE	ASCII CODE	
0	MSG1	5448	$THIS IS A SAMPLE.	A	41
				E	45
1		4953		H	48
				I	49
2		2049		L	4C
				M	4D
3		5320		P	50
				S	53
4		4120		T	54
5		5341			
6		4D50		SPACE (SP)	20
7		4C45		LINE FEED (LF)	0A
8		2E20			
9		0D0A	+>0D0A	CARRIAGE RETURN (CR)	0D
10		0000	+>0000	. (PERIOD)	2E

Note that line 9 contains a carriage return and a line feed and has the code 0D0A. The message beginning at location MSG1 is preceded by a dollar sign and terminated with a byte containing all binary zeroes. The + > 0D0A is a code recognized by the line-by-line assembler that is loaded directly into memory. It is initiated by typing the (+) before the desired number. The dollar sign indicates that a comment is being entered. Such XOPs are very useful in calling subroutines prepared to accomplish specific terminal functions.

Selecting a Mode

XOP 11 will be used to make the choice of the mode of operation. ECHO CHARACTER means that whatever key is pressed on the terminal will be read into a designated workspace register and then sent back from the register and printed on the terminal.

The one instruction,

```
XOP R5,11
```

accomplishes this. If a key is pressed, the terminal reads the character, places it in workspace register 5 and then prints the character on the terminal. The XOP subroutine was provided by the TIBUG monitor but it all was accomplished with one instruction — thus, the "extended operation."

TMS9902

The TMS9902, asynchronous communications controller provides an interface between the EIA terminal (serial asynchronous communications channel) and the 9900 in the TM990/100M microcomputer module. The block diagram of the microcomputer was shown in *Figure 7.* A simplified one is shown in *Figure 21a.* Note that the interface to the CPU (TMS9900) is the same as for the 9901. Note also the line $\overline{INT}$ 4 going from the 9902 to the 9901; this interrupt line will be important in this application.

All of the discussion that pertained to the 9901 and the addressing of the I/O bits also applies to the 9902. It has the same address bits A_{10}—A_{14} used for addressing the CRU bits inside the 9902 through S_0—S_4. It has the same CRU control bus signals for communication over the CRU serial data link.

A base address and $\overline{CE}$ select the 9902 over other I/O units that might be available in the system (in this case, only 9901s are present). The hardware base address 0040_{16} identifies the 9902 contained in the microcomputer. The software base address of 0080_{16} is loaded into WR12. This is added to the appropriate displacement to arrive at the effective CRU bit address desired as described for the 9901.

In this extended application, pressing a key on the terminal while the system is in mode 1 or mode 2 will switch the system back to the command mode. The user then selects a new mode of operation. This is a common way to use a terminal and the 9902 must be programmed to accomplish it. The arrangement is as shown in *Figure 21a.*

First, the 9902 must recognize that a character has been generated by the terminal and received by the 9902. Second, the output signal line $\overline{INT}$ from the 9902 must be enabled so it can pass the signal to the 9901 input $\overline{INT}$ 4. Since the 9901 receives this signal as an interrupt, then interrupt masks at the 9901 and the 9900 must be enabled. With these steps accomplished, the main program of the processor is interrupted and the operation mode is shifted.

Figure 21b shows that $\overline{INT}$ will be active in the receive mode if RBRL = 1 and RIENB = 1. RBRL will be a "1" when the Receive Buffer Register has received a character and stored it. This happens when a key is pressed. The 9902 is enabled by making RIENB (Receiver Interrupt Enable) a "1". *Figure 22* identifies that CRU bit 18 must be made a "1" to make RIENB = 1. A CRU SBO instruction with a displacement of 18 will set CRU bit 18 to a "1" if the software base address has previously been loaded in WR12.

Since $\overline{INT}$4 is the desired interrupt level, it is enabled in the 9900 by placing this level in its interrupt mask. This is accomplished with an instruction LIMI 4 which loads the value 4 into the status register.

With the $\overline{INT}$4 enabled at the 9901 by placing a "1" in the CRU bit mask corresponding to the input for $\overline{INT}$4, the 9901 sends the interrupt code to the 9900 over IC0-IC3 when the $\overline{INT}$ signal is received from the 9902. Since $\overline{INT}$4 is enabled in the 9900, the signal path is complete and the operating mode shifts.

$\overline{INT}$4 executes a context switch and finds its new workspace pointer is $FF8C_{16}$ and its new PC is $FFAC_{16}$.

In all the discussion, only the enabling of interrupts has been covered. It must be stressed that similar instructions in many cases must be included in the programming to disable an interrupt once it has been enabled.

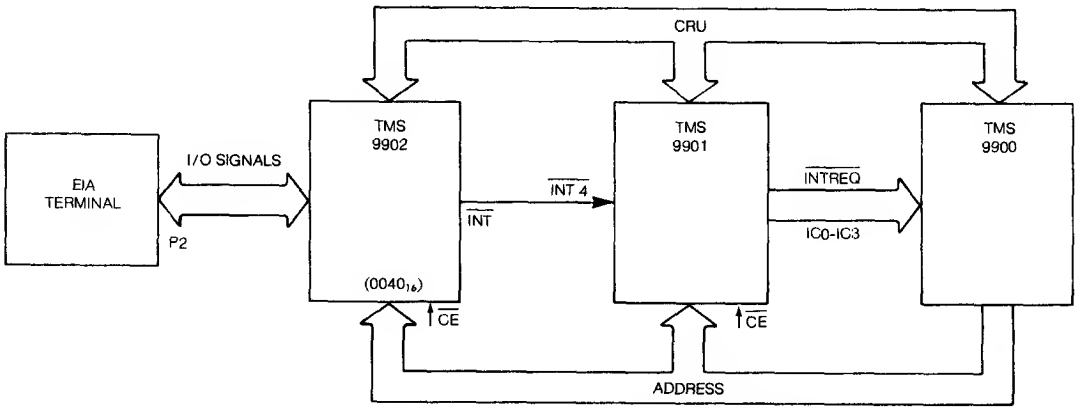


Figure 21a. Simplified Block Diagram Showing TMS 9902 Interface

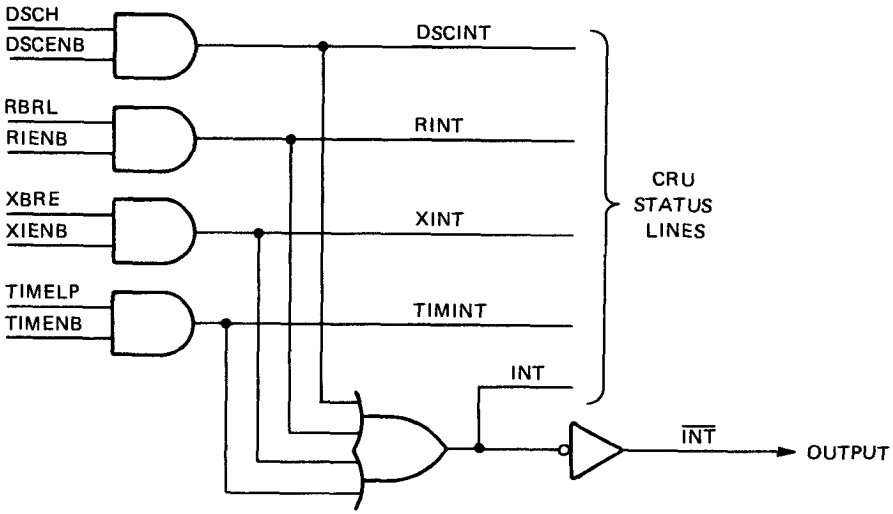


Figure 21b. $\overline{INT}$ Output Generation

ADDRESS$_2$ S0 S1 S2 S3 S4	ADDRESS$_{10}$	NAME	DESCRIPTION
1 1 1 1 1	31	RESET	Reset device.
	30-22		Not used.
1 0 1 0 1	21	DSCENB	Data Set Status Change Interrupt Enable.
1 0 1 0 0	20	TIMENB	Timer Interrupt Enable
1 0 0 1 1	19	XBIENB	Transmitter Interrupt Enable
1 0 0 1 0	18	RIENB	Receiver Interrupt Enable
1 0 0 0 1	17	BRKON	Break On
1 0 0 0 0	16	RTSON	Request to Send On
0 1 1 1 1	15	TSTMD	Test Mode
0 1 1 1 0	14	LDCTRL	Load Control Register
0 1 1 0 1	13	LDIR	Load Interval Register
0 1 1 0 0	12	LRDR	Load Receiver Data Rate Register
0 1 0 1 1	11	LXDR	Load Transmit Data Rate Register
	10-0		Control, Interval, Receive Data Rate, Transmit Data Rate, and Transmit Buffer Registers

Figure 22. TMS 9902 ACC Output Bit Address Assignments

PROGRAMMING THE 9901 I/O

The discussion, previously quite general, now gets more specific, focusing on how the program will have to be written to satisfy the requirements of the application. Since all input and output signals must go through the 9901, let's begin there. Refer to *Figure 23.*

Note that there are multiple functions for the pins on the 9901. The pins are referenced to establish the link between Group 1, Group 2 and Group 3 which were mentioned previously in the text. Note that all the functions are referenced to a select bit number from 0 to 31. Select bit zero is addressed when the 9901 base address is called. For example, the instruction:

SBO 0

addresses select bit zero in the 9901 and will set this bit, called the control bit, to a "1". Because it was bit zero, there was no additional displacement value added to the base address. However, as was done in Chapter 3, 10_{16} will be added to the 9901 hardware base address in the microcomputer when P_0 thru P_{15} are being used as data inputs and data outputs. This makes the base address point to select bit 16 as indicated in *Figure 23.* It makes the assignment of I/O bit 0 correspond to P_0, bit 1 to P_1, bit 2 to P_2, etc.

Figure 23 shows how select bit zero, the control bit, controls the mode of the 9901. When it is a "0", the 9901 is in the interrupt mode; when it is a "1", the 9901 is in the clock mode. The 9901 must be in the interrupt mode to mask interrupt inputs; it must be in the clock mode to use the internal clock.

NOTES	SELECT BIT	S0 S1 S2 S3 S4	PIN NO.	PIN FUNCTION WHEN BEING READ BY A CRU INSTRUCTION		PIN FUNCTION WHEN BEING SET OR "WRITTEN TO" BY A CRU INSTRUCTION	
9901	MODE			INTERRUPT	CLOCK	INTERRUPT	CLOCK
Base Address	0 (control bit)	0 0 0 0 0		0	1	0	1
	1	0 0 0 0 1	17	$\overline{\text{INT}}$ 1	CLK 1	MASK 1	CLK 1
	2	0 0 0 1 0	18	$\overline{\text{INT}}$ 2	CLK 2	MASK 2	CLK 2
	3	0 0 0 1 1	9	$\overline{\text{INT}}$ 3	CLK 3	MASK 3	CLK 3
	4	0 0 1 0 0	8	$\overline{\text{INT}}$ 4	CLK 4	MASK 4	CLK 4
	5	0 0 1 0 1	7	$\overline{\text{INT}}$ 5	CLK 5	MASK 5	CLK 5
	6	0 0 1 1 0	6	$\overline{\text{INT}}$ 6	CLK 6	MASK 6	CLK 6
	7	0 0 1 1 1	*34	$\overline{\text{INT}}$ 7	CLK 7	MASK 7	CLK 7
	8	0 1 0 0 0	*33	$\overline{\text{INT}}$ 8	CLK 8	MASK 8	CLK 8
	9	0 1 0 0 1	*32	$\overline{\text{INT}}$ 9	CLK 9	MASK 9	CLK 9
	10	0 1 0 1 0	*31	$\overline{\text{INT}}$ 10	CLK 10	MASK 10	CLK 10
	11	0 1 0 1 1	*30	$\overline{\text{INT}}$ 11	CLK 11	MASK 11	CLK 11
	12	0 1 1 0 0	*29	$\overline{\text{INT}}$ 12	CLK 12	MASK 12	CLK 12
	13	0 1 1 0 1	*28	$\overline{\text{INT}}$ 13	CLK 13	MASK 13	CLK 13
	14	0 1 1 1 0	*27	$\overline{\text{INT}}$ 14	CLK 14	MASK 14	CLK 14
	15	0 1 1 1 1	*23	$\overline{\text{INT}}$ 15	ΔINTREQ	MASK 15	$\overline{\text{RST}}$ 2
I/O Ports → Address	16	1 0 0 0 0	38	P0 INPUT		P0 OUTPUT	
	17	1 0 0 0 1	37	P1 INPUT		P1 OUTPUT	
	18	1 0 0 1 0	26	P2 INPUT		P2 OUTPUT	
	19	1 0 0 1 1	22	P3 INPUT		P3 OUTPUT	
	20	1 0 1 0 0	21	P4 INPUT		P4 OUTPUT	
	21	1 0 1 0 1	20	P5 INPUT		P5 OUTPUT	
	22	1 0 1 1 0	19	P6 INPUT		P6 OUTPUT	
	23	1 0 1 1 1	*23	P7 INPUT		P7 OUTPUT	
	24	1 1 0 0 0	*27	P8 INPUT		P8 OUTPUT	
	25	1 1 0 0 1	*28	P9 INPUT		P9 OUTPUT	
	26	1 1 0 1 0	*29	P10 INPUT		P10 OUTPUT	
	27	1 1 0 1 1	*30	P11 INPUT		P11 OUTPUT	
	28	1 1 1 0 0	*31	P12 INPUT		P12 OUTPUT	
	29	1 1 1 0 1	*32	P13 INPUT		P13 OUTPUT	
	30	1 1 1 1 0	*33	P14 INPUT		P14 OUTPUT	
	31	1 1 1 1 1	*34	P15 INPUT		P15 OUTPUT	

▶9

*COMMON

Δ INVERTED FROM INTREQ

Figure 23. 9901 Select Bit Assignments

Interrupt Mode

Select bit outputs 1 through 15 become MASK bits 1 through 15 when writing to these bits to enable (MASK = 1) or disable (MASK = 0) interrupts. Enabled interrupts received on the inputs will be decoded by the prioritizer and encoder of *Figure 15.*

Clock Mode

To set or read the self-contained clock, the 9901 must be in the clock mode. Using the CRU, the clock is set to a total count by writing a value to select bits 1 through 14.

Reading the clock is accomplished by a CRU instruction to read select bits 1 through 14. Another read instruction without switching the 9901 out of the clock mode will read the same value.

The clock is reset by writing a zero value to the clock or by a system reset.

In the clock mode, select bits 1 through 14 become CLK bits 1 through 14.

Data Inputs and Outputs

Select bits 16 through 31 are used for data inputs and outputs. All I/O pins are set to the input mode by a reset. To set a select bit as an output, just write data to that pin. The data will be latched and can be read with a CRU read instruction without affecting the data. Once an I/O port is programmed to be an output, it can only be programmed as an input by a hardware or software reset. This can be done two ways.

1. Receiving a hardware reset, $\overline{\text{RESET}}$.
   (Operating the RESET switch on the microcomputer.)
2. Writing a "0" to select bit 15 of the 9901 while in the clock mode will cause a software $\overline{\text{RST2}}$ and force all I/O ports to the input mode.

The status of the 9901 can be evaluated by checking (reading) the control bit. Testing select bit 15 in the interrupt mode can indicate if an interrupt has been received. If one has, INTREQ will be high because $\overline{\text{INTREQ}}$ is low.

After a hardware $\overline{\text{RESET}}$, or a software reset $\overline{\text{RST2}}$, all interrupts $\overline{\text{INT1}}$ through $\overline{\text{INT15}}$ are disabled, all I/O ports will be in the input mode, the code on IC0-IC3 will be 0000, $\overline{\text{INTREQ}}$ will be high and the 9901 will be in the interrupt mode.

Examples of Programming

Setting the Control Bit

If the interrupt and clock modes of the 9901 are to be controlled, load the base address in WR12 (100_{16} for 9901 on microcomputer board) and set select bit zero to the respective value:

```
LI  R12,>100        LOADS>100 INTO WR12
SBZ  0              9901 TO INTERRUPT MODE
SBO  0              9901 TO CLOCK MODE
```

Enabling or Disabling Interrupt Level

Interrupt levels are enabled or disabled by setting the MASK to a "1" or a "0" value, respectively. As an example, after a reset, the 9901 would be in the interrupt mode. Now interrupts 2, 5, 6 and 8 are to be enabled. The instruction:

LDCR R2,9

will do this as shown in *Figure 24.* The contents of workspace register 2, 0164_{16} from bit 15 thru 7 are read into select bits 0 thru 8 to enable interrupt levels 2, 5, 6 and 8. Of course, WR12 had to be loaded with the software base address using a

LI R12,>100

instruction, as an example, and WR2 would have been loaded in a similar fashion.

In like fashion, the same levels could be disabled by writing "0" to bits 2, 5, 6 and 8 with an LDCR instruction, or programming a software $\overline{RST}$2, or by using the single bit CRU instructions.

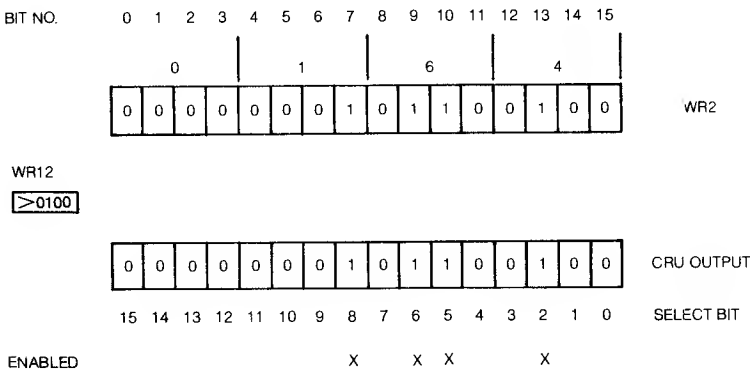


Figure 24. Enabling Interrupt Levels 2, 5, 6 and 8 with an LDCR Instruction

For example,

SBZ 2
SBZ 5
SBZ 6
SBZ 8

would set each bit to a "0". Previously WR12 was set to 0100_{16} to reference the 9901 on the microcomputer module.

Setting the Output Bits

Similar single bit or LDCR CRU instructions can be used to set the output bits.

LDCR R2, 0 would read out the value of WR2 to the output pins P_0 through P_{15} (the 0 in the LDCR R2, 0 means all 16 bits will be written to the output). WR12 has previously been loaded with 0120_{16}. This is shown in *Figure 25.*

A routine of loading 9901 I/O INPUTS and storing 9901 I/O OUTPUTS with a 743 KSR terminal would look like the following, after pressing the RESET toggle switch on the microcomputer module and a carriage return on the terminal:

```
TIBUG REV A
?M FE00 (CR)

 ADDRESS        OP CODE        MNEMONIC        COMMENT
FE00=XXXX       02E0   (SP)   LWPI  >FF20      ;WP=>FF20
FE02=XXXX       FF20   (SP)
FE04=XXXX       020C   (SP)   LI  R12,>120     ;9901 SOFTWARE BASE
                                               ADDRESS = >120
FE06=XXXX       0120   (SP)
FE08=XXXX       0200   (SP)   LI  R0,>F0F0     ;CRU DATA
FE0A=XXXX       F0F0   (SP)
FE0C=XXXX       3000   (SP)   LDCR  R0,0       LOAD 9901 I/O
                                               PORTS WITH R0
FE0E=XXXX       3400   (SP)   STCR  R0,0       STORE 9901 I/O
                                               PORTS IN R0
FE10=XXXX       0460   (SP)   B  @>80          ;RETURN TO TIBUG
FE12=XXXX       0080   (CR)
?
```

The XXXX shown are don't care contents at the respective memory addresses which are changed as the op codes are entered. (SP) is a space bar command and (CR) is a carriage return.

9◀

BIT NO.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
	0				1				6				4				
	0	0	0	0	0	0	0	1	0	1	1	0	0	1	0	0	WR2
WR12 >0120																	
	0	0	0	0	0	0	0	1	0	1	1	0	0	1	0	0	CRU OUTPUT
SELECT BIT	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
9901 OUTPUT	P_{15}	P_{14}	P_{13}	P_{12}	P_{11}	P_{10}	P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0	

Figure 25. Output From WR 2 with LDCR Instruction

Programming the 9901 Clock

In *Figure 9*, the clock function of the 9901 was described. The clock register must be loaded with a value to set its total count and enable the clock. When the register is decremented to zero, it generates a level 3 interrupt (INT 3) as the elapsed time signal.

Access is gained to the clock by setting select bit zero to a "1" which puts the 9901 in the clock mode. All select bits 1 thru 15 are then in the clock mode and become the access for setting the clock count. CLK bit 15 is used for software reset. Therefore, the clock count is set by the value on select bits 1 through 14. An example is shown in *Figure 26*. The maximum value that can be loaded into 14 bits (all ones) would be 16,383. The rate at which the clock decrements the value is f(ϕ)/64. If f is 3 MHz, then the rate is approximately 46,875 Hz. The time interval is equal to the value in the clock register times 1/46,875. With the maximum value, the maximum interval is 349 milliseconds.

If 25 millisecond intervals are required, then the clock register would have to be loaded with 46,875 × 0.025 = 1172. This is equivalent to 0494_{16}. The least significant bit of the register value must be a 1 to set the control bit, therefore 0494_{16} is moved over a bit position and the register is loaded with 0929_{16}. A LDCR instruction is used for loading the value and the sequence of steps is shown in *Figure 26*.

▶9

The software is as follows:

```
LI  R12,>0100        ;SET 9901 ON MODULE SOFTWARE ADDRESS=>0100
LI  R1,>0929         ;LOAD CLOCK VALUE INTO R1, SET CLOCK MODE
LDCR  R1,15          ;MOVE TIMER VALUE AND CONTROL BIT TO 9901
```

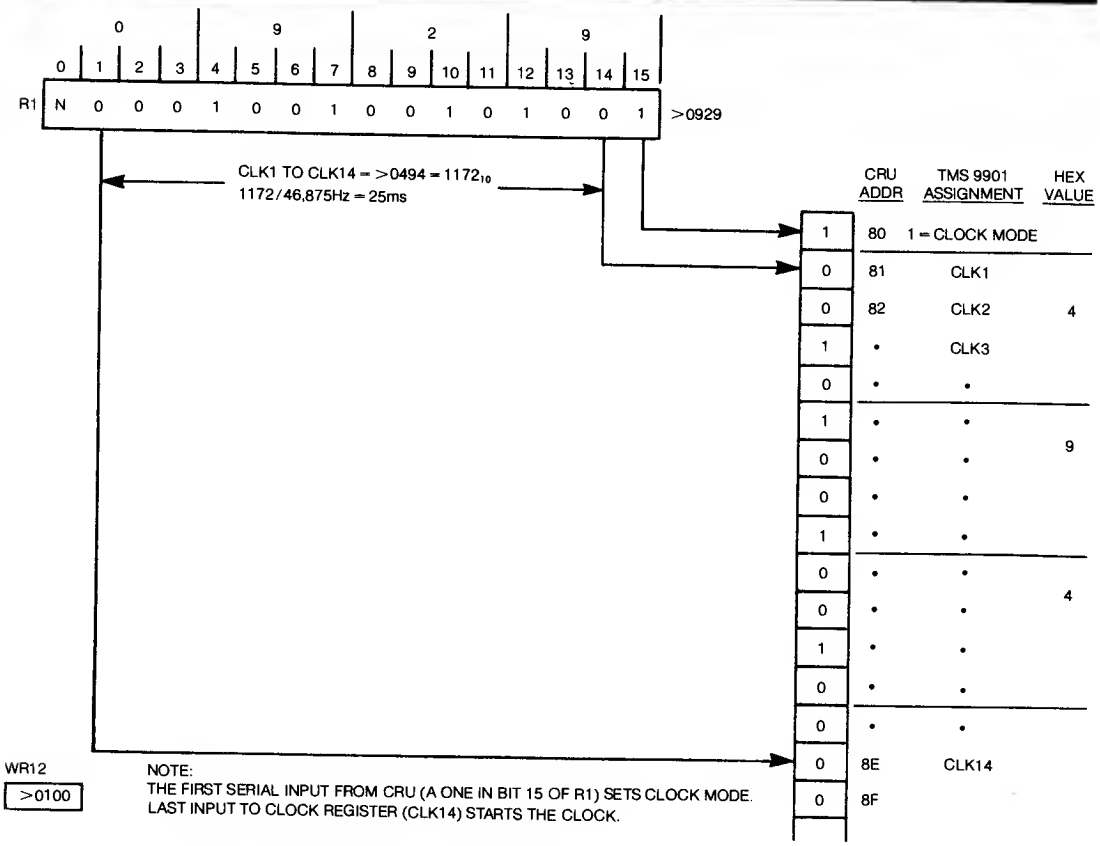


Figure 26. Enabling and Triggering TMS 9901 Interval Timer

Enabling Clock Interrupt

When the clock decrements to zero, a level 3 interrupt is given. The interrupt level 3 mask needs to be enabled on the 9901 and the 9900 CPU. The interrupt mask on the 9901 is enabled by setting the control bit to a logical "0" (interrupt mode) and then setting select bit 3 to a "1" (write a "1" to bit 3). The interrupt mask on the 9900 is enabled by loading the appropriate value (in this case, 3) into the interrupt mask. When 3 is loaded into the 9900 with a LIMI 3 instruction, all higher priority levels are also enabled.

The software is:

```
LI  R12,>0100      ;SET BASE ADDRESS TO 9901 ON BOARD, >0100
SBZ 0              ;9901 TO INTERRUPT MODE
SBO 3              ;ENABLE INTERRUPT 3 AT 9901
LIMI 3             ;LOAD 9900 INTERRUPT MASK
```

Putting Some Pieces Together

Some of the pieces can now be combined to provide a larger program. It looks like this:

```
        LI  R12,>0100       ;SET SOFTWARE BASE ADDRESS OF 9901=0100
        CLR  R0             ;INITIALIZE INTERRUPT INDICATOR, R0 SET TO ZERO
        LI  R1,>0929        ;CLOCK COUNT 0494 AND CLOCK MODE IN R1
        LDCR  R1,15         ;SET CLOCK COUNT ENABLE TIMER
        SBZ  0              ;9901 TO INTERRUPT MODE
        SBO  3              ;ENABLE INT 3 AT 9901
        LIMI  3             ;LOAD 9900 INTERRUPT MASK
LOOP 2  CI  R0, >FFFF       ;HAS INT 3 OCCURED?
        JNE  LOOP 2         ;IF NO, GO TO LOOP 2
```

When the timer gives an interrupt 3, a context switch occurs; the interrupt 3 vector PC points to $FF88_{16}$ which contains an instruction to get to the interrupt routine:

```
        B  @CLKINT          ;BRANCH TO INTERRUPT ROUTINE IDENTIFIED BY CLK INT
```

The branch then takes the program to:

```
CLKINT  LI  R12,>0100       ;SET SOFTWARE BASE ADDRESS OF 9901=0100
        SBZ  3              ;DISABLE INTERRUPT 3
        SETO  *R13          ;SET PREVIOUS R0 TO FFFF
        RTWP                ;RETURN TO PROGRAM
```

Thus, if an interrupt 3 has not occured, the program remains in Loop 2 until it does. When $\overline{INT}$ 3 occurs a context switch to the interrupt subroutine causes R0 to be changed from all zeros to all ones. R0 will now equal $FFFF_{16}$ and the program proceeds to the step after JNE Loop 2, which, as will be seen later, is a count down.

FROM BASIC CONCEPTS TO PROGRAM

As with the Chapter 3 application, converting the idea to program starts with solidifying the basic concept, then developing acceptable flow charts, and then programming the algorithm for the problem solution. As with hard-wired logic design, the place to start is with a block diagram. The one used in *Figure 6* will be expanded with a bit more detail and will be the concept diagram *(Figure 27).*

The terminal, the microcomputer module and the interface modules with their respective inputs and outputs will constitute the system. Later on the TM900/310 module will be added to show the I/O expansion capability. This will only involve plugging the interface modules into one of the additional 9901 outputs on the 310 board
▶9 (P4 in this case) and changing the CRU base address to select the chosen 9901. It will be assumed that the power and all interconnections have also been made through P1 to the microcomputer and 310 module as shown in *Figure 27.* There is a special power

supply required for supplying the interface modules. This is the +8V shown in *Figure 27.* 110Vac is supplied separately for the terminal and the industrial level voltages of 12 volts dc and 110Vac are supplied separately, as they would be in a user facility.

The physical arrangement of the interface modules is important to the program for the problem solution. Therefore, I/O positions 0 thru 7 are identified. Positions 0 thru 3 are input positions; positions 4 thru 7 are output positions. Signals received on input position 0 will cause reaction at output position 4. Correspondingly for input 1 and output 5, input 2 and output 6, and input 3 and output 7. Thus, the program will be written to sense input 1 and set output 5 to correspond.

Switches S1 through S4 represent industrial level input voltages, either dc or ac. Lights L1 and L3 represent industrial dc loads; L2 and L4 represent industrial ac loads.

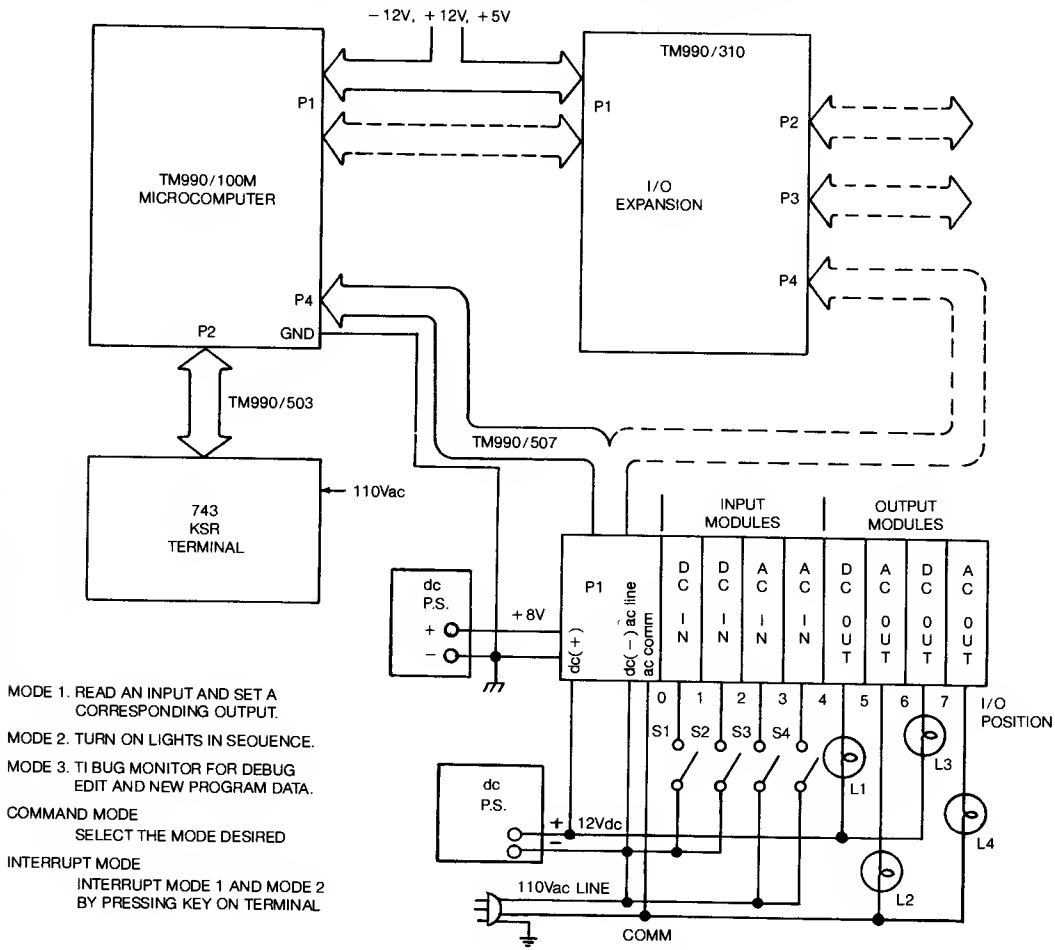


Figure 27. Concept Flow Diagram

Flow Charts for the Program

Software design is really little different from hardware design in the execution of good engineering practice.

The task from overall concept stage is divided into subsystems — in the case of software, subprograms or subroutines. *Figure 28* identifies subprograms for the extended application which are detailed in flow charts so that basic functions can be identified.

The flow charts are separated according to the functions that are to be implemented. Operation in Mode 1 simulates sensing four industrial level inputs 0 through 3 and reacting to these inputs by providing output voltages to four corresponding loads, 4 through 7. The flow chart identifies that inputs will be sensed and a corresponding output will be set to match the input state or value.

The four output loads, in this case light bulbs, will be turned on and off in sequence and held in each of these states for a set time (variable by the program). This is Mode 2 operation. The flow chart shows the major functions. After all four lights are turned off and on, the sequence starts over. The clock in the 9901 will be used to provide the time interval.

There is an operating Mode 3 but it will be contained in the mode called the COMMAND Mode. In Mode 3 the operation of the system is under the control of the TIBUG Monitor which is contained in the 1K words of EPROM resident in the microcomputer. It is used for inputting the original program and editing and changing the program as the need may be.

The flow chart for the Command Mode starts with initial setup of the system. Certain registers and certain locations in memory are loaded with data used throughout the program. A print-out of general information and specific instructions follows. Since the user will make a choice, instructions identify that a one (1) key is to be pressed on the terminal to operate in Mode 1; a two (2) key to operate in Mode 2; and a Q for Mode 3. The character pressed by the user is then examined and the appropriate operating mode selected. If none of the operating mode characters are received the system waits in the command mode until one is received.

On the flow chart for the COMMAND mode A and B connect with the respective points on the MODE 1 and MODE 2 flow charts.

Recall that the system is to have a provision for the user to command an escape from the continuous operation in Mode 1 or Mode 2. This happens by interrupting Mode 1 or Mode 2 operation by pressing a key on the terminal. The first blocks in the flowcharts of MODE 1 and MODE 2 provide the means for accomplishing the interrupt. When a key is pressed on the terminal, this initiates an interrupt signal output from the 9902. This interrupt must be enabled to pass to the 9901 and the 9900 so that it will cause the return to the COMMAND MODE. The generation of the signal in the 9902 is flowcharted under the heading INTERRUPT MODE of *Figure 28.*

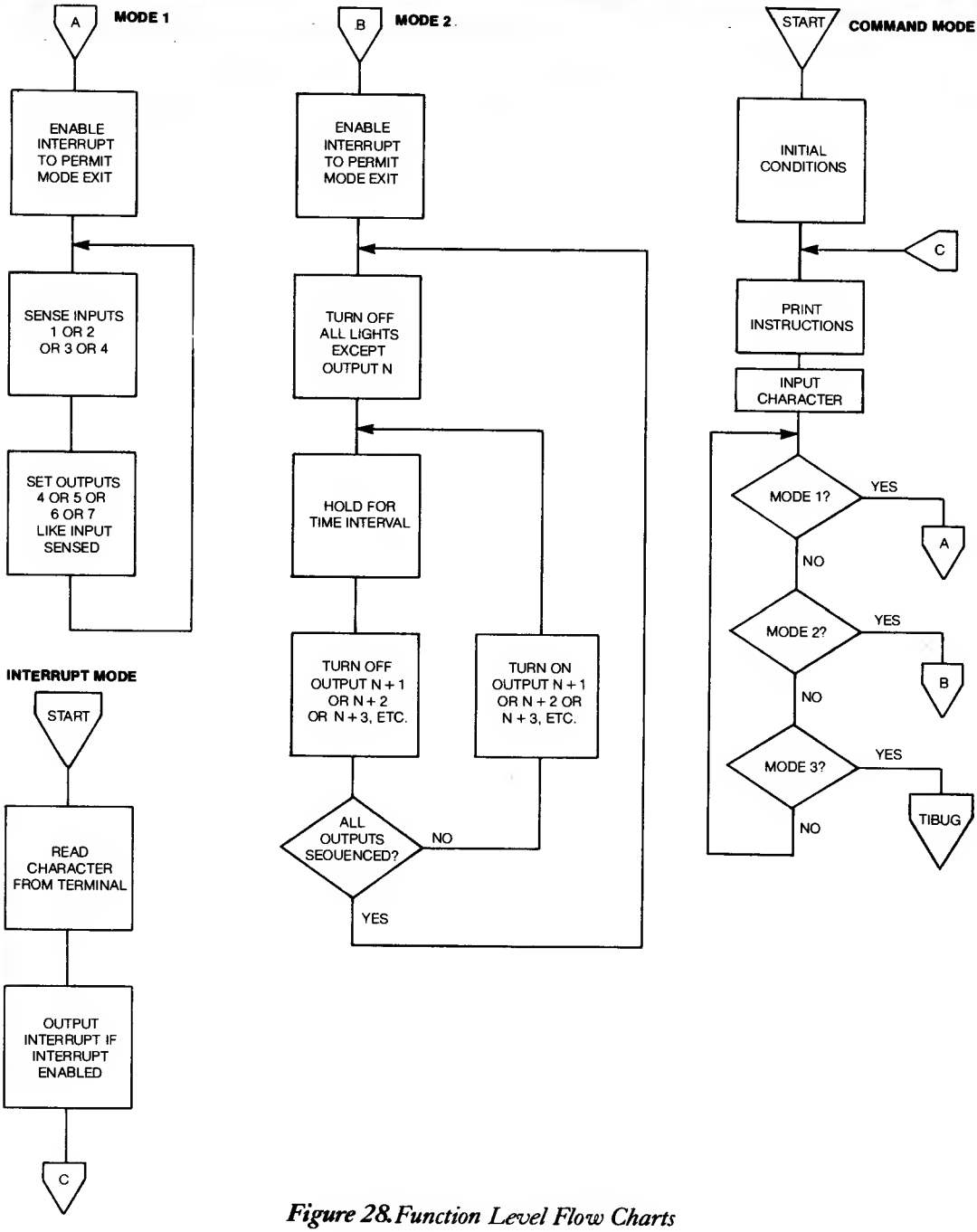


Figure 28. Function Level Flow Charts

Writing the Program

Memory Space

All elements are now in place to write the program. First, it is necessary to decide what locations are to be used in memory for the program, for the workspace and for data. Refer to *Figure 29.*

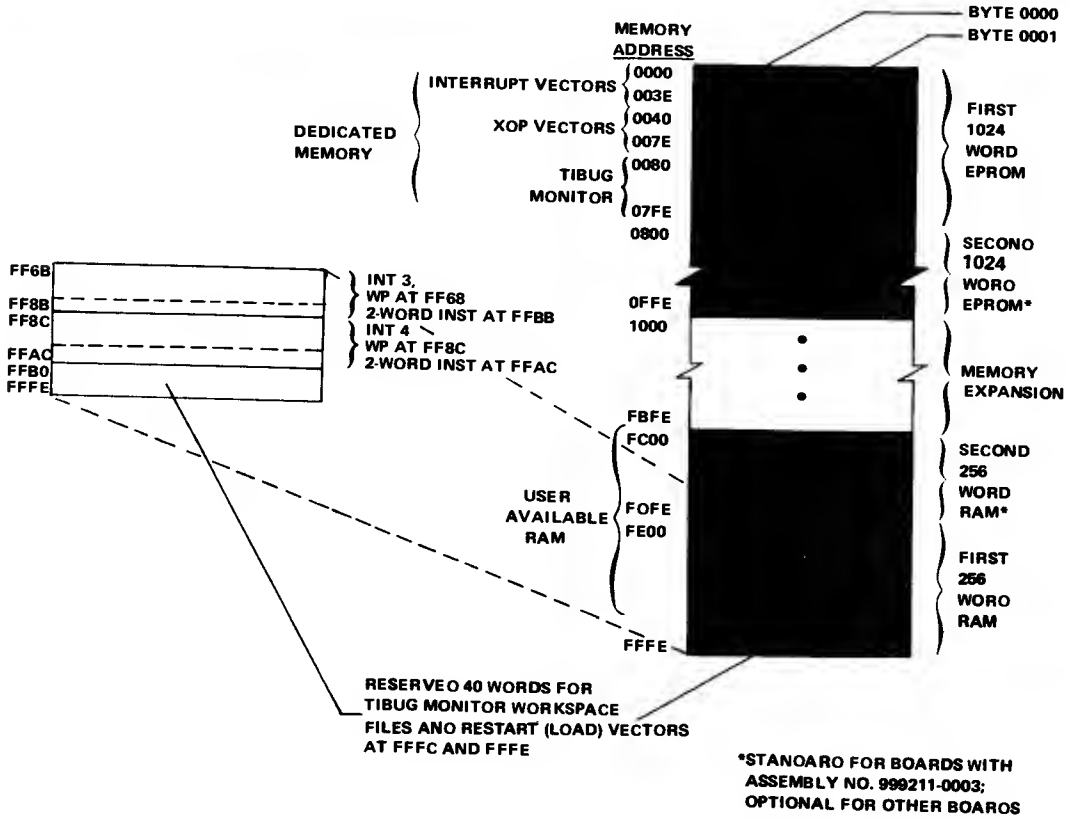
For this application more memory space is required than for Chapter 3's First Encounter. Thus, additional RAM units are installed on the microcomputer board at locations U33, U35, U37 and U39 (4042 Units). This expands the available RAM space to $FC00_{16}$ and this is the location for the start of the program.

Incidentally, while available memory is being discussed, note the address of the TIBUG monitor, 0080_{16}. This memory location must be referenced when returning to the TIBUG Monitor in Mode 3. The TIBUG workspace located at $FFB0_{16}$ has already been discussed. This space must be reserved.

One more point — the second 1K of EPROM starting at location 0800_{16} will be populated with the Line-by-Line Assembler (LBLA) resident in EPROM. This will be used for assembly of the program. The socket locations on the board are U43 and U45 and the product number is TM990/402-1. Normally, the LBLA would start assemblying at address $FE00_{16}$, however, by using a /FC00 command the start location is changed to $FC00_{16}$.

The Command Mode

A more complete flow chart is shown in *Figure 30* for the Command Mode. The program begins with initialization of registers. When writing the first draft of the program, labels are used for ease of writing. For later drafts and when a LBLA is used, the labels are replaced with actual addresses. INPUT1 will be the label for the start of Mode 1. BLINKR will be the label for the start of Mode 2. COMODE labels the message that asks the user to select the mode.



DEDICATED MEMORY

ADDRESS (HEX)	PURPOSE
0000-0003	Level zero interrupt vector ($\overline{\text{RESET}}$)
000C-000F	$\overline{\text{INT3}}$ vectors (TMS 9901 timer)
0010-0013	$\overline{\text{INT4}}$ vectors (TMS9902 timer)
0040-0047	Vectors for XOP's 0 and 1 (Microterminal I/O)
0060-007F	Vectors for XOP's 8 to 15 (TIBUG utilities)
0080-07FF	TIBUG monitor
FFB0-FFFB	Four overlapping monitor workspaces
FFFC-FFFF	Restart (load) vector

9◀

Figure 29. Memory Map with Fully Populated 990/100M-1 Module

The Command Mode program is as follows:

```
COUNT    +>929          ;SET UP 9901 CLOCK
BASE1    +>100          ;SET UP 9901 CRU BASE
BASE2    +>120          ;SET UP 9901 I/O BASE
START    LWPI >FF20     ;SET WP AT FF20
         LI  R1,>1E00   ;SBZ OP COOE TO R1
         LI  R2,>1000   ;SBO OP COOE TO R2
         LI  R3,>1F00   ;TB OP COOE TO R3
         XOP  @MSG1, 14 ;PRINT HEAOER @MSG1
COMOOE   XOP  @MSG2, 14 ;ASK FOR MOOE WITH MSG2
         XOP  R7, 11    ;REAO CHAR FROM TER TO R7
         CI  R7,>3100   ;IS CHAR A 1?
         JEQ  INPUT1    ;IF YES GO TO MOOE 1
         CI  R7,>3200   ;IS CHAR A 2?
         JEQ  BLINKR    ;IF YES TO TO MODE 2
         CI  R7,>5100   ;IS CHAR A Q?
         JNE  COMOOE    ;IF NO KEEP LOOPING
         B  @>80        ;IF YES GO TO TIBUG
```

To initialize registers, the values for the TMS 9901 clock interval, TMS 9901 CRU software base address and TMS 9901 I/O software base address are loaded directly into memory spaces by using a (+) in front of the data. 0929_{16} is placed in the 9901 for a 25ms interval. Recall that the module 9901 has a base address of 0100_{16} for select bit zero and 0120_{16} so that select bit 16 activates P0 when input or output bit 0 is addressed, as discussed previously. Note that the workspace is set up at $FF20_{16}$.

The machine codes for SBZ, SBO and TB are loaded into workspace registers one, two and three, respectively. As discussed previously, an XOP 14 is used to print the header and instructions for use of the program. The messages are labeled with MSG1 and MSG2 and are located at the end of the program and will be discussed later. Next an XOP is used to read a character from the terminal and load the ASCII code into R7. This is then compared with the ASCII codes for the number one, two and the letter Q to determine the character. Depending on what character is received, the program jumps to the proper area in memory to execute the correct mode of operation. The entry point to the TIBUG monitor is 0080_{16} and a branch to this location will execute the monitor.

Mode 1 Operation

Figure 31 shows the flow chart for Mode 1 Operation. The label INPUT1 begins the operation. The first function sets up the system so that the 9902 will generate an interrupt when a received character fills the receiver buffer (RBRL = 1). Recall that the interrupt generated by the 9902 must be enabled by making RIENB = 1. This is accomplished by making the 9902 select bit 18 equal to "1". The enabled interrupt from the 9902 is wired to the $\overline{\text{INT}}$4 input of the 9901. Thus, as previously discussed, level 4 interrupts must be enabled both at the 9901 and the 9900.

The software looks like this:

```
INPUT1   RSET                ; PUT 9901 INTD INPUT MODE
         LIMI  4             ; ENABLE 9900 INT1-INT4
         LI  R12,>B0         ; LDAD R12 W/9902 BASE ADDR
         STCR  R7,0          ; CLEAR 9902 RCV BUFFER
         SBD  18             ; ENABLE 9902 RCV INT
         MDV  @BASE1,R12     ; SET 9901 BASE ADDR TD  >100
         SBD  4              ; ENABLE 99D2 INT AT 9901
```

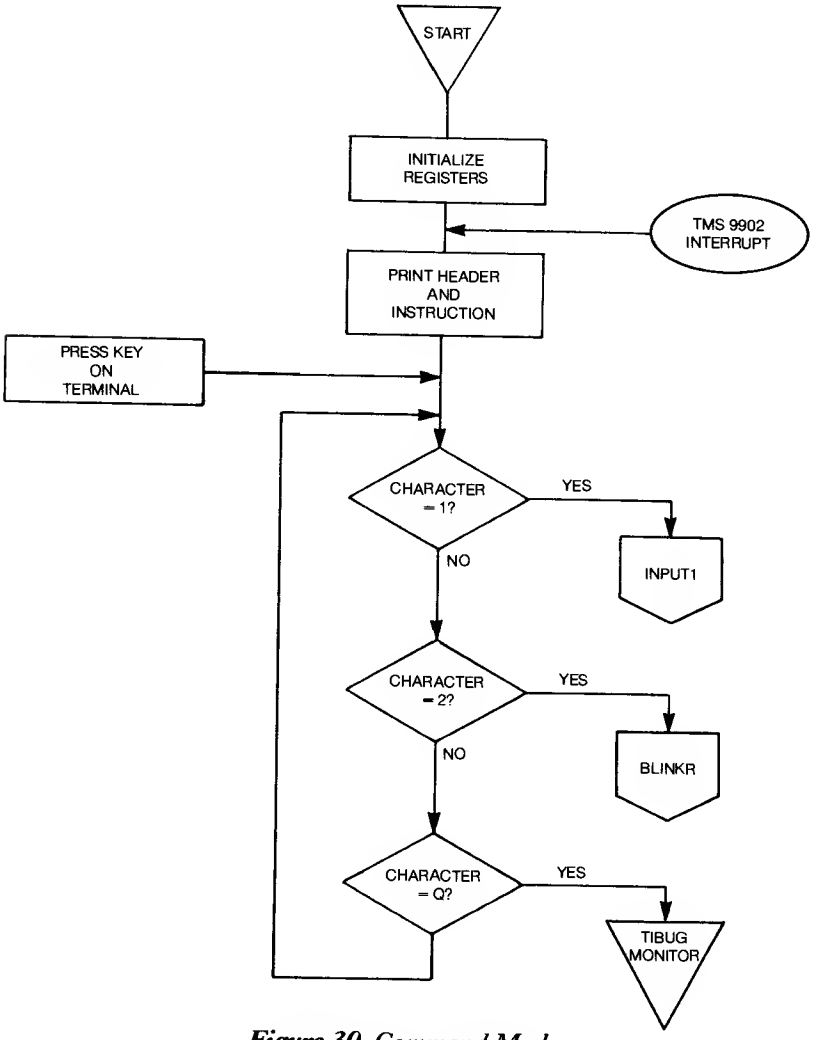


Figure 30. Command Mode

First the 9901 is reset to put it into the input mode. Then the 9900 interrupt mask is set to 4 to allow interrupts 1 thru 4 to be acknowledged. To enable select bit 18 of the 9902, the software base address is loaded into WR12 and an SBO 18 instruction sets the bit to "1" for the enable. The 9902 receiver buffer is read into R7 with the STCR instruction which resets the buffer for receipt of a character. WR12 is set with the software base address for the 9901, and then select bit 4 is set to a "1". These steps enable the 9901 interrupt level 4 to clear the complete path for generating an interrupt when a character is received from the terminal.

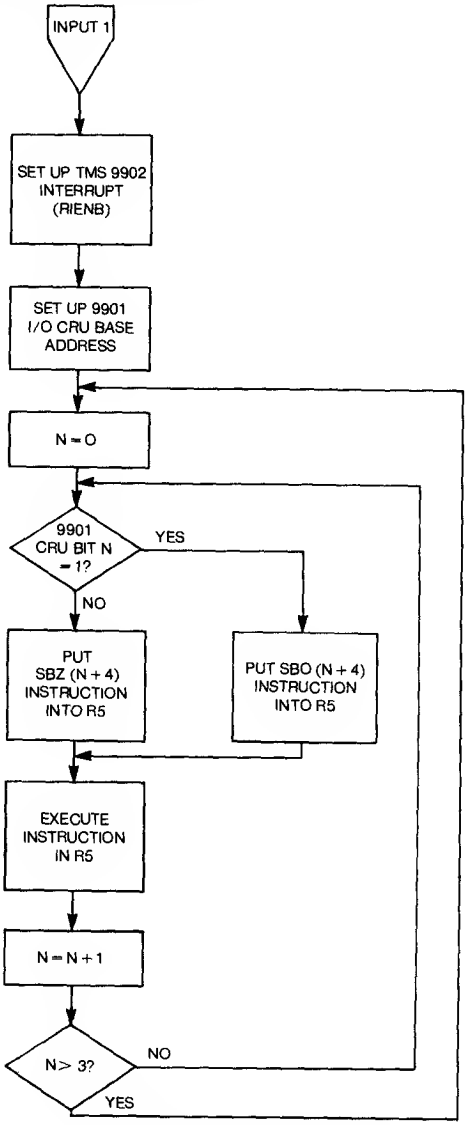


Figure 31. Mode 1 Operation

Checking the Inputs—Setting the Outputs

Figure 31 shows that with N = 0 the CRU is testing the zero input bit of the 9901. If it is a "1", then the N + 4 bit (I/O bit 4) will be set to a "1" to correspond. One will be added to N (0 + 1 = 1), which will be less than 3 and the cycle is repeated: the second time with N = 1, the next time with N = 2 and the next with N = 3. With N = 3, N + 1 will be greater than three and everything is reinitialized and the sequence starts over with input bit zero again. So the procedure is to check each input bit and set the corresponding output bit. The software is as follows:

```
        MOV  @BASE2,R12     ; SET 9901 BASE ADDR TO >120
INIT1   CLR  R4             ; R4 CONTAINS CRU BIT TO BE
                            ; TESTED
INDEX1  MOV  R4,R5          ; MOVE CRU BIT TO R5
        SOC  R3,R4          ; R4 CONTAINS TB INST (R3)
        X  R4               ; EXECUTE TB SPECIFIED BY R4
        JEQ  HIGH           ; IF CRU BIT=1 GO TO HIGH
LOW     MOV  R5,R4          ; RELOAD CRU BIT INTO R4
        AI  R5,>4           ; SHIFT CRU BIT OVER BY 4
        SOC  R1,R5          ; R5 CONTAINS SBZ OP CODE (R1)
XECUTE  X  R5               ; EXECUTE OP CODE SPECIFIED BY R5
        INC  R4             ; INCREMENT TO NEXT CRU BIT
        CI  R4,>3           ; IS CRU BIT >3?
        JGT  INIT1          ; IF YES REINITIALIZE
        JMP  INDEX1         ; START TESTING NEXT CRU BIT
HIGH    MOV  R5,R4          ; RELOAD CRU BIT INTO R4
        AI  R5,>4           ; SHIFT CRU BIT OVER 4
        SOC  R2,R5          ; R5 CONTAINS SBO OP CODE (R2)
        JMP  XECUTE         ; GO EXECUTE SBO INST
```

Input bits 0-3 correspond to output bits 4-7 respectively. R4 contains the value of the select bit to be tested (the program starts with bit zero). R4 is moved to R5 to preserve the contents of R4. R3 contains the machine code for TB. Actually it contains the machine code for the instruction TB 0 (Test bit 0). By doing a set ones correspondence (SOC) between R3 and R4, the machine code for the TB instruction is combined with the value of the select bit to be tested so that R4 contains the instruction — "test the select bit previously specified by R4." More specifically, R4 = TB (R4).

An X of R4 will execute this instruction. Using this procedure allows R4 to contain the bit position separate from the TB instruction which is in R3. The bit position in R4 or R5 can also be combined with the SBO and SBZ op codes located in R2 and R1 to allow execution of the SBO or SBZ instructions on the select bits specified by R4 or R5. The procedure is the same as for the TB instruction.

If the bit tested is a zero, R4 is reloaded from R5 with the original value of the select bit to be tested, which is still in R5. R5 plus 4 is combined with R1 using a SOC R1, R5 instruction. The selected output bit will be set to zero when the resulting SBZ instruction in R5 is executed. Thus, an N + 4 output is set to zero, if the corresponding N bit was a zero.

R4 is incremented to the next bit and is tested to determine if its value is greater than 3. When it is not, the program jumps to label INDEX1 and tests the next bit in the same sequence as the first and sets the corresponding output bit. Now, suppose this bit is a "1" instead of a "0" as for the preceding bit. The program jumps to the label HIGH, reloads R4, adds 4 to R5, and now executes an SOC R2, R5 to set the N + 4 output to one when the SBO instruction in R5 is executed.

When input bit 3 is tested, the test of R4 + 1 will show its value is greater than 3 and the program is reinitialized and the procedure starts over. To exit the loop, any key on the keyboard is pressed which produces a level 4 interrupt. The level 4 interrupt comes from the 9902 and the system enters the command mode as shown in *Figure 30.*

Mode 2 Operation *(Figure 32)*

Mode 2 operation sequences the loads simulated by light bulbs. The flowchart is shown in *Figure 32.* It has a time interval of 25ms set up by the 9901 real time clock. A program loop multiples the 25ms times R6 to obtain the total time interval; with R6 = 4, each total time interval is 100ms. The time interval can also be varied by changing the initial value 0929_{16} set into the clock register of the 9901. The value in R4 determines the number of light bulbs (loads) that are going to be turned on, held for 100ms, turned off, and started through the sequence again. As with mode 1, pressing a key on the terminal causes a return to the Command Mode.

It is worthy to note, even though the 9901 is in the input mode when reset, outputs 4,5,6 and 7 are such that all light bulbs are on. Thus, the function of turning off outputs 5, 6 and 7 and leaving 4 on starts the program after the CRU base address is set. In actual industrial applications it may be necessary to put additional inverters between the output of the microcomputer and the 5MT modules so that the reset condition has all loads off.

Recall that when the 9901 clock register is decremented to zero it puts out a $\overline{INT3}$ signal. This interrupt causes a context switch to occur and sets the old workspace R0 to $FFFF_{16}$. When this happens the time interval has ended.

Interrupt 4 from TMS 9902

The software for Mode 2 starts as follows to set up the interrupt 4 from the 9902:

```
BLINKR    RSET                ; SET 9901 TO THE INPUT MODE
          LIMI  4             ; ENABLE 9900 INT1-INT4
          LI  R12,>80         ; SET UP 9902 BASE ADDR
          STCR  R7,0          ; CLEAR 9902 RCV BUFFER
          SBO  1B             ; ENABLE 9902 RCV INT
```

The reset at BLINKR sets the 9901 to the input mode and turns on the loads on outputs 4, 5, 6 and 7.

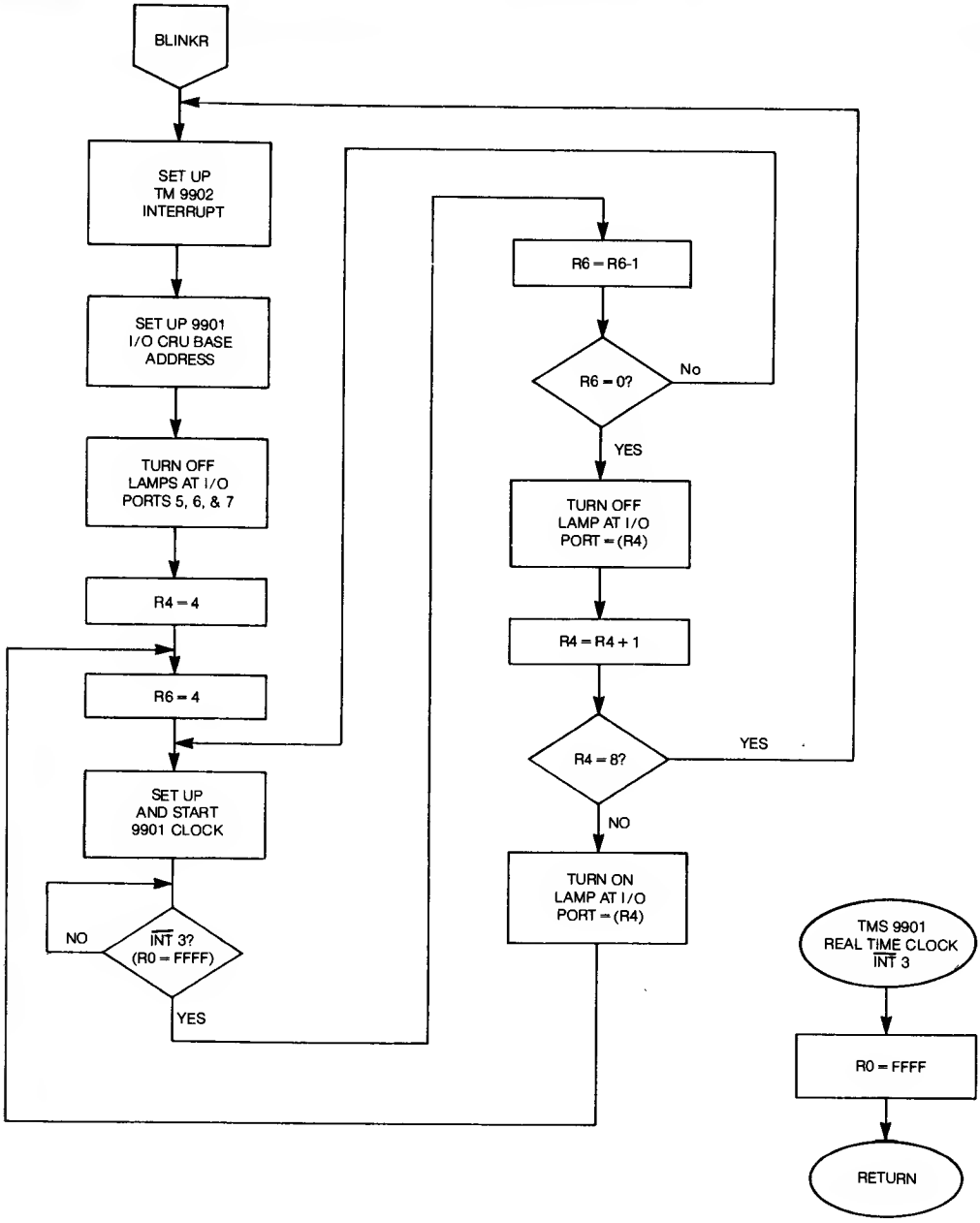


Figure 32. Mode 2 Operation

The next 7 instructions after the 9901 software base address is set at 0120_{16} are concerned with turning off outputs 5, 6 and 7. These start with INT2 and continue through the next 6 instructions after LOOP 1.

```
        MOV  @BASE2, R12      ; SET 9901 BASE ADDR=>120
INT2    LI   R4,>5            ; R4 CONTAINS CRU BIT POS 5
LOOP1   MOV  R4,R5            ; MOV POS 5 TO R5
        SOC  R1,R5            ; R5 CONTAINS SBZ OP CODE (R1)
        X    R5               ; EXECUTE SBZ SPECIFIED BY (R5)
        INC  R4               ; R4=R4+1
        CI   R4,>B            ; HAS CRU BIT 7 BEEN SET=0?
        JNE  LOOP1            ; IF NO GO TO LOOP 1
```

Lamp 4 remains on.

Register 4 must now be loaded with the output position from which the sequence starts—in this case 4.

```
        LI   R4,>4            ; SET OUTPUT BASE BIT
```

Timing Loop

R6 is set equal to 4 so that the overall time interval is 100ms. This starts the timing loop at INDEX2. The 5 instructions following TIMER set up the 9901 clock to count a 25ms interval and then cause a level 3 interrupt. Note that the 9901 must be put into the interrupt mode and the level 3 interrupt enabled. Since the 9902 interrupt signal comes in on interrupt level 4, it is convenient to enable it at this same time. The loop is such that it loops 4 times. Each loop is controlled by the interval timer of the TMS9901. The TMS9901 timer is set and started when loaded with the value at the label COUNT. The clock decrements until it hits zero and then it gives a level 3 interrupt. The interrupt service routine begins at $FF88_{16}$ as directed by the level 3 vector. It sets R0 to $FFFF_{16}$ and returns to the program. The program will be in a continuous loop (Loop 2) checking R0 for an indication that an interrupt has occured. When the time interval is complete, the I/O bit dictated by R4 is turned off. R4 is incremented and checked to see if it is equal to 8. If not, the I/O bit position of the new R4 is turned on and the sequence restarts. If R4 + 1 = 8, then the program jumps back to BLINKR and starts over causing R4 to be reset to 4 and to restart the sequence.

The software looks like this:

```
INDEX 2   LI   R6,>4          ; OVERALL LOOP COUNT=100ms
TIMER     MOV  @BASE1,R12     ; SET CRU BASE ADDR OF 9901=>100
          CLR  R0             ; INITIALIZE INT3 INDICATOR
          LDCR @COUNT,15      ; LOAD TIMER AND START COUNT
          SBZ  0              ; 9901 TO INTERRUPT MODE
          SBO  3              ; ENABLE INT3 AT 9901
          SBO  4              ; ENABLE 9902 INT AT 9901
LOOP2     CI   R0,>FFFF       ; HAS INT3 OCCURRED?
          JNE  LOOP2          ; IF NO GO TO LOOP2
          DEC  R6             ; R6=R6-1
          JNE  TIMER          ; IF R6=0 GO TO TIMER
          MOV  @BASE2,R12     ; SET 9901 BASE ADDR=>120
          MOV  R4,R5          ; MOV CRU BIT TO R5
          SOC  R1,R5          ; (R5)=SBZ (R5)
          X    R5             ; EXECUTE SBZ SPECIFIED BY (R5)
          INC  R4             ; R4=R4+1
          CI   R4,>9          ; IS R4=9?
          JEQ  BLINKR         ; IF YES RESTART SEQUENCE
          MOV  R4,R5          ; R4=R5
          SOC  R2,R5          ; (R5)=SBO (R5)
          X    R5             ; EXECUTE SBO SPECIFIED BY (R5)
          JMP  INDEX2         ; RESTART TIMING CYCLE AT INDEX 2
```

9902 Interrupt Service Routing

This interrupt service routine is the one resulting from a level 4 interrupt generated by the 9902. It starts at INTREC. As discussed previously, when the interrupt occurs, the program counter points to $FFAC_{16}$, the reserved space, where it finds an instruction directing it to INTREC. This instruction looks like this:

```
ADDRESS   INSTRUCTION
FFAC      B @INTREC         ; GO TO INT4 SERVICE ROUTINE
```

The routine first disables the 9901 timer interrupt level 3, then disables the 9902 interrupt at the 9902 (Set select bit 18=0) and finally loads the address of COMODE into the old PC, so that when an RTWP (return with workspace pointer) is executed, the program returns to the command mode. The software is as follows:

```
INTREC    MOV  @BASE1,R12     ; SET 9901 BASE ADDR=>100
          SBZ  3              ; DISABLE INT3 AT 9901
          SRL  R12, 1         ; SET BASE ADDR=>80 FOR 9902
          SBZ  1B             ; DISABLE 9902 INT
          STOR R7, 0          ; READ 9902 RCV BUFFER (CLEARS)
          LI   R14, COMODE    ; LOAD ADDR OF COMODE INTO PC
          RTWP                ; RETURN TO SMT ROUTINE
```

9◀

9901 Clock Interrupt Service Routine

When the clock decrements to zero it generates a level 3 interrupt. The routine to service this interrupt starts at CLKINT. The level 3 interrupt context switch provides a new PC at $FF88_{16}$ which directs the program to CLKINT. This instruction looks like this:

```
ADDRESS   INSTRUCTION

FF88      B  @CLKINT          ; GO TO INT3 SERVICE ROUTINE
```

Here, after setting the software base address of the 9901 to 0100_{16}, $\overline{INT3}$ is disabled and R0 of the previous workspace is set to $FFFF_{16}$. A RTWP instruction then returns the processor to the interrupted routine.

The software is as follows:

```
CLKINT    LI  R12,>100        ; SET 9901 BASE ADDR
          SBZ  3              ; DISABLE INT3 AT 9901
          SETO  *R13          ; SET PREVIOUS R0=>FFFF
          RTWP                ; RETURN TO INTERRUPTED ROUTINE
```

Message Routines

The remaining routines that must be included in the program are the messages at MSG1 and MSG2. In order to program the message, a $ sign is used at the beginning of each line and each message is terminated with a zero byte. The ASCII code for a carriage return — line feed is $0D0A_{16}$ and is included in the instruction format.

Each character must be coded with the appropriate ASCII code and placed into bytes of memory. A typical example is shown; however, the individual character codes have not been listed. This can be seen on the LBLA listing.

```
MSG1      $5MT I/O DEMONSTRATION ROUTINE
          +>000A
          $MODE 1 — INPUTS 0-3 SWITCH OUTPUTS
          $4-7 RESPECTIVELY
          +>0D0A
          $MODE 2 — OUTPUTS 4-7 ARE SWITCHED SEQUENTIALLY
          +>0D0A
          $A Q RETURNS CONTROL TO THE TIBUG MONITOR
          +>0D0A
          $A CARRIAGE RETURN DURING MODE 1 OR 2
          $OPERATION RETURNS THE USER TO THE
          +>000A
          $CONTROL MODE
          +>0D0A
          +>0000
          +>0D0A
          $SELECT MODE 1, 2 or Q
          +>0D0A
          +>0000
```

SYSTEM OPERATION

With program in hand, it is time to connect the hardware to prove out the complete program. Refer to the block diagram of *Figure 27.*

The terminal and its cable have been previously connected to P2 of the microcomputer module. P1 has the same power supply connections as for Chapter 3 supplying −12V, +12V, +5V and ground. The full connections will be added to P1 to interface with P1 on the TM990/310 I/O expansion board. However, for now, operations will be only with the microcomputer and the 5MT I/O modules. Connection to the modules is made through the cable of *Figure 4* and P4 on the microcomputer and P1 on the 5MT43 module base. There is a separate wire from the J1 connector to provide +8 volts to the 5MT modules. This +8 volts must supply 0.6A worst case if all the positions in the 5MT43 base are populated. *This supply ground must be common with the microcomputer module ground and isolated from the +12V industrial control voltage supply ground.*

The +12V for the industrial control level voltages must supply 200mA. *This must have a minus terminal free of chassis ground, otherwise its case will be at ac line voltage when the 5MT I/O module ac power cord is connected.*

Light bulbs that are rated at 80 mA at 14 Vdc are used for the dc loads. Standard 110 Vac light bulbs and sockets are used for the ac loads. A separate ac power cord is connected to the 5MT43 base for the ac power. The industrial level power *(both dc and ac) is and must be isolated from the dc power for the microcomputer module and low-level logic +8V power source of the 5MT interface modules.*

A summary of the parts list and power supply requirements follows:

SYSTEM PARTS LIST

- TM990/100M-1 board
- TM990/310 48 I/O board (optional)
- 5MT43 base*
- 2 — 5MT11-A05L AC input modules*
- 2 — 5MT12-40AL AC output modules*
- 2 — 5MT13-D03L DC input modules*
- 2 — 5MT14-30CL DC output modules*
- 5MT interface cable-TM990/507
- 743 KSR terminal
- TM 990/503 cable assembly for Terminal
- 4 — TMS 4042-2 (or 2111-1) 256 x 4 RAM's

*In case your local distributor does not have these parts, the address from which they can be ordered is:

Industrial Controls Order Entry
M/S 12-38
34 Forrest St.
Attleboro, Mass. 02703
Phone: (617) 222-2800

9◂

- Line-by-line assembler TM990/402-1 (in two TMS 2708 EPROM's)
- Power supplies for Microcomputer and I/O Expansion (TM990/518)

Voltage	*REG*	*/100M Current*	*w/310 Module Current*
+5V	±3%	1.3A	2.1A
+12V	±3%	0.1A	0.1A
−12V	±3%	0.2A	0.2A

- Industrial Control Level Power Supplies

Voltage	*REG*	*Current*
+8Vdc	±5%	0.6A
+12Vdc	±5%	0.2A
110Vac		1A

- 4 Toggle switches, SPST
- 2 dc lamps and sockets (14V — 80mA)
- 2 ac lamps and sockets (130V — 30 W)
- Power cord
- 14 and 18 AWG insulated stranded wire

Equipment Hookup

Follow these steps in making the system interconnections;

Step 1 — Verify that the power supply connections to P1 are correct for −12V, +12V and +5V. Refer to *Figure 3-11* or to the TM990/100M user's guide *Figure 2-1.* Don't turn on any power supplies. It may be desirable to make all the connections from P1 of the TM990/100M to P1 of the TM990/310 at this time. Refer to *Table 6* for these connections. Some reprogramming because of power shutdown will be required if this is not done.

Step 2 — Verify that the 743 KSR terminal is connected to P2 with the TM990/503 cable. AC power is supplied to the terminal with a separate cord.

Step 3 — Special connections must now be made at the jumpers on the TM990/100M microcomputer. The jumper positions are shown in Chapter 3, *Figures 12* and *13.* Make sure of the following jumper connections.

JUMPERS	*INTERCONNECTION*	*COMMENT*
J15	Disconnected	Power for TM990/301
J14	Disconnected	Microterminal, not
J13	Disconnected	required for 743 KSR
J12	N.A.	For multiple boards
J11	Disconnected	For ASR 745
J10,9,8	N.A.	For multiple boards
J7	EIA position	
J6,5	N.A.	For multiple boards
J4,3,2	In 08, or 2708 Position	For 2708 EPROMS
J1	9902	This will likely need to be positioned

Step 4 — As mentioned previously, the RAM on the TM990/100M should be fully populated for this example. Make sure that 4-TMS 4042-2's have been inserted in U33, U35, U37 and U39 with the #1 pin towards the TMS 9900. The LBLA which is in two TMS 2708 EPROM's should have also been inserted in U43 and U45 with the #1 pin towards the TMS 9900. The higher order byte (bits 0-7) must be in U45. It is quite difficult to insert these packages in the sockets the first time so it must be done carefully. Rocking the packages will help.

Step 5 — Install the 5MT modules in the 5MT43 base as shown in *Figure 33.* Be sure modules are in the proper order. This arrangement will show dc input controlling dc output, dc input—ac output, ac input—dc output and ac input — ac output. Connect the wiring as shown. Be sure to use heavy gage (14 AWG) insulated wire for the ac connections. 18 AWG can be used for dc power connections. *NOTE THAT AC LINE IS CONNECTED TO DC COMMON.* Two screw connections on the base are available for each module as shown in *Figure 33.* All connections to the 5MT modules are to the right-hand leads when facing the terminals and P1 is on the left. Be sure to screw down the locking screw to ensure good connections.

Step 6 — Connect J1 of the cable of *Figure 4* to the 5MT43 base. Connect the +8V lead to the power supply and its ground to the common ground lead on J4 of the cable of *Figure 4. DO NOT CONNECT THIS GROUND TO THE DC COMMON OF THE INDUSTRIAL CONTROL LEVEL POWER SUPPLY OF FIGURE 33.*

Step 7 — Connect the +12Vdc industrial power supply. Don't plug in the 110Vac power cord.

Step 8 — Turn on the +8V and +12V supply and verify that the dc input and output 5MT modules are connected correctly. Use J4 for test voltages.

Step 9 — Plug-in the ac power cord for the 5MT modules and verify that the ac input and output modules are interconnected correctly. The LED's on the modules will be useful for this.

Step 10 — Unplug ac cord, turn off +12V and +8V supplies.

Step 11 — Connect J4 of the cable from the 5MT43 base to P4 on the TM990/100M module.

Step 12 — Turn on the power supplies for the microcomputer *in this order:* −12V, +12V, +5V.

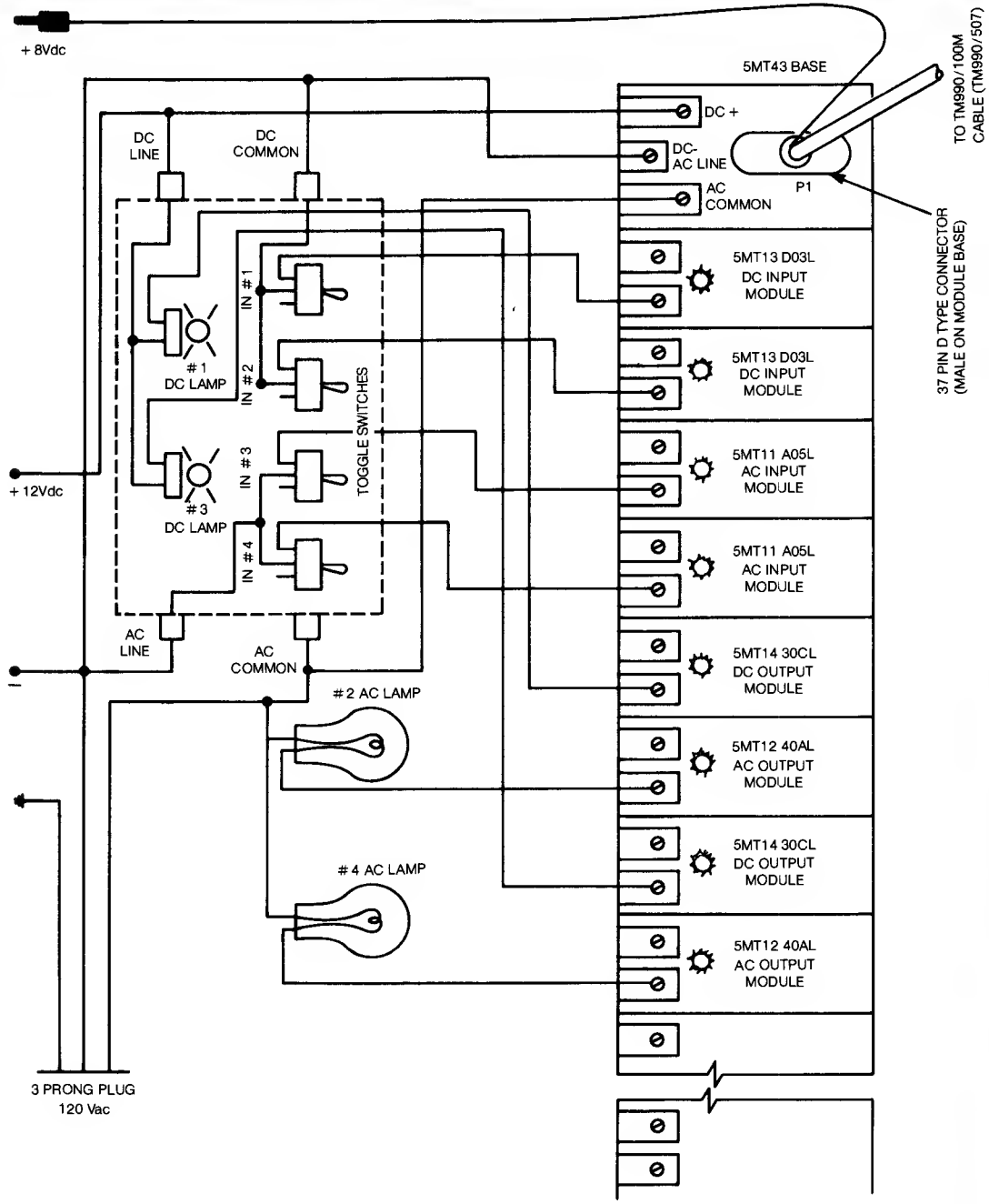


Figure 33. 5MT I/O Module Wiring

Step 13 — Turn on the terminal. Make sure it is "ON LINE."

Step 14 — Turn on the +8V supply for the 5MT modules; then the industrial level +12Vdc and then plug in the power cord for the 110Vac.

Step 15 — Press the RESET switch on the microcomputer. All the light bulbs will be lit since a RESET latches I/O pins on the microcomputer in the "1" state.

The microcomputer system is now ready to be programmed.

Loading the Program

The program as it was developed will now be loaded into RAM in the microcomputer. Instead of assembling the program by hand, the line-by-line assembler contained in EPROM will be used. It works with the EIA terminal and the TIBUG monitor.

The LBLA is a stand alone program that assembles into object code the 69 instructions used by the TM 990/100M microcomputer. To initialize the LBLA, the TIBUG monitor must first be brought up. This is done by switching the reset switch on the TM 990/100M module and pressing the carriage return (CR) on the terminal. The terminal will respond with:

```
TIBUG REV. A.
?
```

The question mark is the TIBUG prompt.

Now an R is typed to inspect/change the WP, PC and ST registers. The LBLA program begins at location $09E6_{16}$* so this is the value that is to be loaded into the PC. After typing an R the terminal prints out the value of the WP. This can be changed by typing the new value and a space or it can be left alone by typing just a space. The terminal will then print the value of the PC. The same procedure as for the WP applies except that ST is printed if a space is typed. A CR after the WP or PC value will cause the TIBUG prompt to be printed, or a space or CR after the ST is printed will do the same.

Loading $09E6_{16}$ into the PC looks like this:

```
?R                    (CR)
W=FFC6                (SP)
P=01A6     09E6       (CR)
?
```

Once the PC has been loaded, executing the program will initialize the LBLA. Pressing the E key accomplishes this. The LBLA responds with an address. That address can be changed to the starting address of the program by typing a slash (/) and the new address and a CR.

```
?E
FE00                  /FC00          (CR)
FC00
```

*This value may change depending on the version of LBLA. Early versions had $09E8_{16}$ as entry point.

The program can then be entered using the machine instructions. The LBLA accepts assembly language inputs from a terminal. As each instruction is input, the assembler interprets it, places the resulting machine code in an absolute address, and prints the machine code (in hexadecimal) next to its absolute address as shown in Figure 34.

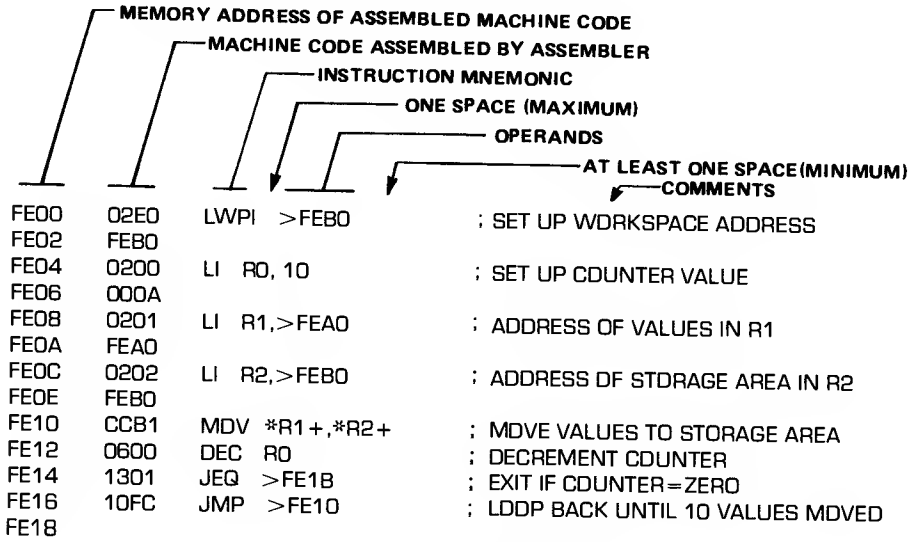


Figure 34. LBLA Format

Only one space is used between the mnemonic and the operand. If comments are used, use at least one space between the operand and the start of the comment. If no comment is used complete the instruction *with a space and a carriage return.* If a comment is used, only a carriage return is required.

Note that to load a hex value directly into a memory location a (+) is used. (see Start of Program, *Table 4.*) Also a string of characters is preceded by a dollar sign ($) and *terminated with two carriage returns*—CR (Example shown under—Message Routines). To change the address location being loaded, type a slash (/) and the address desired. To exit from the LBLA and return to the TIBUG monitor, press the ESC key on the terminal. The terminal will then give the TIBUG prompt—a question mark.

Labels cannot be used with the LBLA. However, in the program of *Table 4,* the left side is the assembled program with LBLA and the right side is for a comparison to the labels and the comments that were previously used on each of the pieces of the program as it was developed on the preceding pages.

Remember to press the ESC when the last program address location is reached. This returns control to the TIBUG monitor.

Table 4. Final Program

```
                LBLA                    Labels          Comments
?R
W=FFB0
P=016B  09E6
?E
F000            /FC00
FC00    0929    +>929                   COUNT       ; SET UP 9901 CLOCK
FC02    0100    +>100                   BASE 1      ; SET UP 9901 CRU BASE
FC04    0120    +>120                   BASE 2      ; SET UP 9901 I/O BASE
FC06    02E0    LWPI  >FF20             START       ; SET WP AT FF20
FC0B    FF20
FC0A    0201    LI  R1,>1E00                        ; SBZ OP CODE TO R1
FC0C    1E00
FC0E    0202    LI  R2,>1000                        ; SBO OP CODE TO R2
FC10    1D00
FC12    0203    LI  R3,>1F00                        ; TB OP CODE TO R3
FC14    1F00
FC16    2FA0    XOP  @>FCF2,14                      ; PRINT HEADER @MSG1
FC1B    FCF2
FC1A    2FA0    XOP  @>FE00,14          COMODE      ; ASK FOR MODE WITH MSG2
FC1C    FE00
FC1E    2EC7    XOP  R7,11                          ; READ CHAR FROM TER TO R7
FC20    02B7    CI  R7,>3100                        ; IS CHAR A 1?
FC22    3100
FC24    130B    JEQ  >FC36                          ; IF YES GO TO MODE 1
FC26    02B7    CI R7,>3200                         ; IS CHAR A 2?
FC2B    3200
FC2A    1325    JEQ  >FC76                          ; IF YES GO TO MODE 2
FC2C    02B7    CI  R7,>5100                        ; IS CHAR A Q?
FC2E    5100
FC30    16F4    JNE  >FC1A                          ; IF NO KEEP LOOPING
FC32    0460    B  @>00B0                           ; IF YES GO TO TIBUG
FC34    00B0
FC36    0360    RSET                    INPUT1      ; PUT 9901 INTO INPUT MODE
FC3B    0300    LIMI  4                             ; ENABLE 9900 INT1-INT4
FC3A    0004
FC3C    020C    LI  R12,>00B0                       ; LOAD R12 W/9902 BASE ADDR
FC3E    00B0
FC40    3407    STCR  R7,0                          ; CLEAR 9902 RCV BUFFER
FC42    1012    SBO  18                             ; ENABLE 9902 RCV INT
FC44    C320    MOV  @>FC02,R12                     ; SET 9901 BASE ADDR TO >100
FC46    FC02
FC4B    1004    SBO  4                              ; ENABLE 9902 INT AT 9901
FC4A    C320    MOV  @>FC04,R12                     ; SET 9901 BASE ADDR TO >120
FC4C    FC04
FC4E    04C4    CLR  R4                 INIT1       ; R4 CONTAINS CRU BIT TO BE TESTED
FC50    C144    MOV  R4,R5              INDEX1      ; MOVE CRU BIT TO R5
FC52    E103    SOC  R3,R4                          ; R4 CONTAINS TB INST [R3]
FC54    04B4    X  R4                               ; EXECUTE TB SPECIFIED BY R4
FC56    130A    JEQ  >FC6C                          ; IF CRU BIT=1 GO TO HIGH
FC5B    C105    MOV  R5,R4              LOW         ; RELOAD CRU BIT INTO R4
FC5A    0225    AI  R5,>4                           ; SHIFT CRU BIT OVER BY 4
FC5C    0004
FC5E    E141    SOC  R1,R5                          ; R5 CONTAINS SBZ OP CODE [R1]
FC60    04B5    X  R5                   XECUTE      ; EXECUTE OP CODE SPECIFIED BY R5
FC62    05B4    INC  R4                             ; INCREMENT TO NEXT CRU BIT
FC64    02B4    CI  R4,>3                           ; IS CRU BIT >3?
FC66    0003
```

```
FC6B  15F2  JGT  >FC4E                  ; IF YES REINITIALIZE
FC6A  10F2  JMP  >FC50                  ; START TESTING NEXT CRU BIT
FC6C  C105  MOV  R5,R4        HIGH      ; RELDAD CRU BIT INTD R4
FC6E  0225  AI   R5,>4                  ; SHIFT CRU BIT OVER 4
FC70  0004
FC72  E142  SDC  R2,R5                  ; R5 CDNTAINS SBD DP CDDE [R2]
FC74  10F5  JMP  >FC60                  ; GD EXECUTE SBD INST
FC76  0360  RSET              BLINKR    ; SET 9901 TO INPUT MODE
FC7B  0300  LIMI 4                      ; ENABLE 9900 INT1-INT4
FC7A  0004
FC7C  020C  LI   R12,>B0                ; SET UP 9902 BASE ADDR
FC7E  00B0
FCB0  3407  STCR R7,0                   ; CLEAR 9902 RCV BUFFER
FC82  1D12  SBD  1B                     ; ENABLE 9902 RCV INT
FCB4  C320  MDV  @>FC04,R12             ; SET 9901 BASE ADDR=>120
FCB6  FC04
FC88  0204  LI   R4,>5        INT2      ; R4 CDNTAINS CRU BIT POS 5
FCBA  0005
FCBC  C144  MDV  R4,R5        LDDP1     ; MDV POS 5 TD R5
FC8E  E141  SDC  R1,R5                  ; R5 CDNTAINS SBZ DP CDDE [R1]
FC90  04B5  X    R5                     ; EXECUTE SBZ SPECIFIED BY [R5]
FC92  05B4  INC  R4                     ; R4=R4+1
FC94  02B4  CI   R4,>8                  ; HAS CRU BIT 7 BEEN SET=0?
FC96  000B
FC98  16F9  JNE  >FC8C                  ; IF ND GD TD LDDP1
FC9A  0204  LI   R4,>4                  ; SET DUTPUT BASE BIT
FC9C  0004
FC9E  0206  LI   R6,>4        INDEX2    ; OVERALL LDDP CDUNT=100MS
FCA0  0004
FCA2  C320  MDV  @>FC02,R12   TIMER     ; SET CRU BASE ADDR DF 9901=>100
FCA4  FC02
FCA6  04C0  CLR  R0                     ; INITIALIZE INT3 INDICATDR
FCA8  33E0  LDCR @>FC00,15              ; LDAD TIMER AND START CDUNT
FCAA  FC00
FCAC  1E00  SBZ  0                      ; 9901 TD INTERRUPT MDDE
FCAE  1D03  SBD  3                      ; ENABLE INT3 AT 9901
FCB0  1D04  SBD  4                      ; ENABLE 9902 INT AT 9901
FCB2  0280  CI   R0,>FFFF     LOOP2     ; HAS INT3 DCCURRED?
FCB4  FFFF
FCB6  16FD  JNE  >FCB2                  ; IF NO GO TD LODP2
FCB8  0606  DEC  R6                     ; R6=R6-1
FCBA  16F3  JNE  >FCA2                  ; IF R6=0 GD TD TIMER
FCBC  C320  MDV  @>FC04,R12             ; SET 9901 BASE ADDR=>120
FCBE  FC04
FCC0  C144  MDV  R4,R5                  ; MDV CRU BIT TD R5
FCC2  E141  SOC  R1,R5                  ; [R5]=SBZ [R5]
FCC4  04B5  X    R5                     ; EXECUTE SBZ SPECIFIED BY [R5]
FCC6  05B4  INC  R4                     ; R4=R4+1
FCC8  0284  CI   R4,>9                  ; IS R4=9?
FCCA  0009
FCCC  13D4  JEQ  >FC76                  ; IF YES RESTART SEQUENCE
FCCE  C144  MDV  R4,R5                  ; R4=R5
FCD0  E142  SDC  R2,R5                  ; [R5]=SBD [R5]
FCD2  0485  X    R5                     ; EXECUTE SBO SPECIFIED BY [R5]
FCD4  10E4  JMP  >FC9E                  ; RESTART TIMING CYCLE AT INDEX2
FCD6  C320  MDV  @>FC02,R12   INTREC    ; SET 9901 BASE ADDR=>100
FCDB  FC02
FCDA  1E03  SBZ  3                      ; DISABLE INT3 AT 9901
FCDC  091C  SRL  R12,1                  ; SET BASE ADDR=>B0 FDR 9902
FCOE  1E12  SBZ  1B                     ; DISABLE 9902 INT
```

▸9

```
FCE0   3407   STCR  R7,0                          ; READ 9902 RCV BUFFER [CLEARS]
FCE2   020E   LI   R14,>FC1A                      ; LOAD ADDR OF COMMODE INTO PC
FCE4   FC1A
FCE6   0380   RTWP                                ; RETURN TO SMT ROUTINE
FCE8   020C   LI   R12,>100          CLKINT       ; SFT 9901 BASE ADDR
FCEA   0100
FCEC   1E03   SBZ  3                              ; DISABLE INT3 AT 9901
FCEE   071D   SETO  *R13                          ; SET PREVIOUS RO=>FFFF
FCF0   0380   RTWP                                ; RETURN TO INTERRUPTED ROUTINE
FCF2          /FF88
FF88   0460   B  @>FCE8                           ; GO TO INT3 SERVICE ROUTINE @CLKINT
FF8A   FCE8
FF8C          /FFAC
FFAC   0460   B  @>FCD6                           ; GO TO INT4 SERVICE ROUTINE @INTREC
FFAE   FCD6
FF8D          /FCF2
FCF2   354D   $SMT I/O DEMONSTRATION ROUTINE
FCF4   5420
FCF6   492F
FCF8   4F20
FCFA   4445
FCFC   4D4F
FCFE   4E53
FD00   5452
FD02   4154
FD04   494F
FD06   4E20
FD08   524F
FD0A   5554
FD0C   494E
FD0E   4520
FD10   0D0A   +>0D0A
FD12   4D4F   $MODE 1 — INPUTS 0-3 SWITCH OUTPUTS
FD14   4445
FD16   2031
FD18   202D
FD1A   2049
FD1C   4E50
FD1E   5554
FD20   5320
FD22   302D
FD24   3320
FD26   5357
FD28   4954
FD2A   4348
FD2C   204F
FD2E   5554
FD30   5055
FD32   5453
FD34   2020
FD36   342D   $4-7 RESPECTIVELY.
FD38   3720
FD3A   5245
FD3C   5350
FD3E   4543
FD40   5449
FD42   5645
FD44   4C59
FD46   2E20
FD48   0D0A   +>0D0A
```

```
FD4A  4D4F  $MODE 2 — OUTPUTS 4-7 ARE SWITCHED SEQUENTIALLY.
FD4C  4445
FD4E  2032
FD50  202D
FD52  204F
FD54  5554
FD56  5055
FD58  5453
FD5A  2034
FD5C  2D37
FD5E  2041
FD60  5245
FD62  2053
FD64  5749
FD66  5443
FD68  4845
FD6A  4420
FD6C  5345
FD6E  5155
FD70  454E
FD72  5449
FD74  414C
FD76  4C59
FD78  2E20
FD7A  0D0A  +>0D0A
FD7C  4120  $A Q RETURNS CONTROL TO THE TI8UG MONITOR
FD7E  5120
FD80  5245
FD82  5455
FD84  524E
FD86  5320
FD88  434F
FD8A  4E54
FD8C  524F
FD8E  4C20
FD90  544F
FD92  2054
FD94  4845
FD96  2054
FD98  4942
FD9A  5547
FD9C  204D
FD9E  4F4E
FDA0  4954
FDA2  4F52
FDA4  0D0A  +>0D0A
FDA6  4120  $A CARRIAGE RETURN DURING MODE 1 OR 2 OPERATION
FDA8  4341
FDAA  5252
FDAC  4941
FDAE  4745
FDB0  2052
FDB2  4554
FDB4  5552
FDB6  4E20
FDB8  4455
FDBA  5249
FDBC  4E47
FDBE  204D
```

```
FDC0  4F44
FDC2  4520
FDC4  3120
FDC6  4F52
FDC8  2032
FDCA  204F
FDCC  5045
FDCE  5241
FDD0  5449
FDD2  4F4E
FDD4  5245  $RETURNS THE USER TD THE
FDD6  5455
FDD8  524E
FDDA  5320
FDDC  5448
FDDE  4520
FDE0  5553
FDE2  4552
FDE4  2054
FDE6  4F20
FDE8  5448
FDEA  4520
FDEC  0D0A  +>0D0A
FDEE  434F  $CDNTRDL MDDE.
FDF0  4E54
FDF2  524F
FDF4  4C20
FDF6  4D4F
FDF8  4445
FDFA  2E20
FDFC  0D0A  +>0D0A,
FDFE  0000  +>0000
FE00  0D0A  +>0D0A
FE02  5345  $SELECT MDDE 1, 2 DR Q
FE04  4C45
FE06  4354
FE08  204D
FE0A  4F44
FE0C  4520
FE0E  312C
FE10  2032
FE12  204F
FE14  5220
FE16  5120
FE18  0D0A  +>0D0A
FE1A  0000  +>0000
FE1C
```

Running the Program

To execute the program, the PC needs to be set to the starting address. This is done by typing an R to enter the inspect/change mode of TIBUG. The WP will be printed. A space will give the PC and here the new PC should be entered. A CR will return to TIBUG and the prompt will be given. Typing an E will cause the program to begin executing. The following is an example of this:

```
?R
W=FFFE  (SP)
P=006C  FC00  (CR)
?E
```

The program will begin by requesting a mode of operation from the user. Typing a "1" will get mode 1 and the state of outputs can be changed by changing the input toggle switches. Pressing a key will cause a return to the command mode. Pressing a 2, switches to mode 2 and the light sequence. Pressing a key returns to the command mode. Pressing a Q on the terminal returns the system to the TIBUG and specific address locations could be inspected for contents, etc.

Debugging

Because of the hard copy given by the terminal, looking for mistakes is made easier. If the program is stuck in a loop, the reset switch on the TM990/100M board can be switched. When in the LBLA use a slash (/) and a new address to change the address. When in TIBUG use the memory inspect/change (M) command to change the address. The TM990/100M user's guide gives the TIBUG commands and the TM990/402 LBLA user's guide gives the LBLA commands. These are also given in Chapter 7 and on the reference cards in the appendix.

I/O EXPANSION WITH THE TM990/310

What remains now is to show the I/O expansion through the use of the TM990/310 module. As shown in *Figure 35,* there are three additional 9901's on the /310 module. The 9901's signals are connected to edge connections P2, P3, and P4, respectively, and are shown in *Table 5.*

All of the pins on the connector to P1 on the 900/100M-1 microcomputer module must now be connected to P1 on the TM990/310 module (if not made previously). These are shown in *Table 6.* Such a power down requires the program to be re-entered.

Table 5. 9901 Pin-Outs on TM990/310.

P2, P3, P4 Pin Number	*Signature*
20	P0
22	P1
14	P2
16	P3
18	P4
10	P5
12	P6
24	INT15/P7
26	INT14/P8
28	INT13/P9
30	INT12/P10
32	INT11/P11
34	INT10/P12
36	INT9/P13
38	INT8/P14
40	INT7/P15
6	Neg. Edge Triggered INT5
8	Pos. Edge Triggered INT6
1	+12V
2	−12V
3	+5V
4	Spare
All remaining pins	Ground

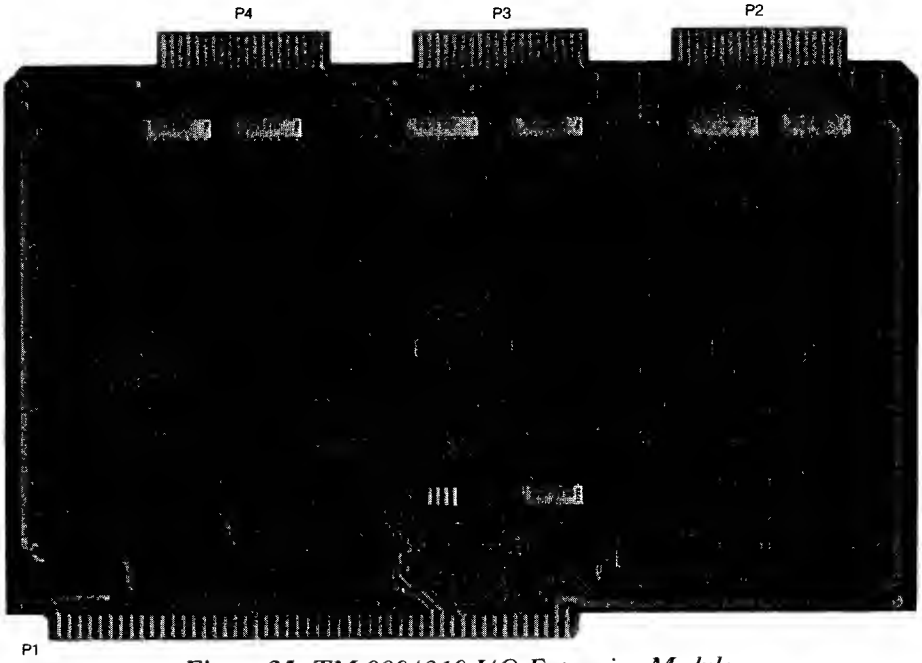


Figure 35. TM 990/310 I/O Expansion Module

Table 6. P1 Connections

P1 PIN	SIGNAL	P1 PIN	SIGNAL	P1 PIN	SIGNAL
33	D0	71	A14	12	$\overline{INT13}$
34	D1	72	A15	11	$\overline{INT14}$
35	D2	22	$\overline{\phi 1}$	14	$\overline{INT15}$
36	D3	24	$\overline{\phi 3}$	28	EXTCLK
37	D4	92	$\overline{HOLD}$	3	+5V
38	D5	86	HOLDA	4	+5V
39	D6	82	$\overline{DBIN}$	97	+5V
40	D7	26	$\overline{CLK}$	98	+5V
41	D8	80	$\overline{MEMEN}$	75	+12V
42	D9	84	$\overline{MEMCYC}$	76	+12V
43	D10	78	$\overline{WE}$	73	−12V
44	D11	90	$\overline{READY}$	74	−12V
45	D12	87	$\overline{CRUCLK}$	1	GND
46	D13	30	CRUOUT	2	GND
47	D14	29	CRUIN	21	GND
48	D15	19	$\overline{IAQ}$	23	GND
57	A0	94	$\overline{PRES}$	25	GND
58	A1	88	$\overline{IORST}$	27	GND
59	A2	16	$\overline{INT1}$	31	GND
60	A3	13	$\overline{INT2}$	77	GND
61	A4	15	$\overline{INT3}$	79	GND
62	A5	18	$\overline{INT4}$	81	GND
63	A6	17	$\overline{INT5}$	83	GNG
64	A7	20	$\overline{INT6}$	85	GND
65	A8	6	$\overline{INT7}$	89	GND
66	A9	5	$\overline{INT8}$	91	GND
67	A10	8	$\overline{INT9}$	99	GND
68	A11	7	$\overline{INT10}$	100	GND
69	A12	10	$\overline{INT11}$	93	$\overline{RESTART}$
70	A13	9	$\overline{INT12}$		

Using the TM990/310 Board

The TMS 9901s on the TM990/310 board are accessed in the same manner as the TMS9901 on the TM990/100M board except the CRU base addresses differ. These hardware base addresses are user selectable by a DIP switch that is on the TM990/310 board. The position of the switch and the corresponding addresses are given in *Figure 36.* The first column of addresses are the actual CRU hardware addresses and the second column is the software address that is to be loaded into workspace register 12 to access the appropriate TMS 9901. The addresses shown correspond to the first TMS 9901 on the TM990/310 board and the positions on the DIP switch. The addresses to be loaded into workspace register 12 for the second TMS9901 are obtained by adding 80_{16} to the addresses of the first TMS 9901. The addresses for the third TMS 9901 are obtained by adding 80_{16} to the addresses of the second TMS 9901 (or 100_{16} to the addresses of the first TMS 9901). For example, if S1 was set to binary 4, workspace register 12 would be loaded with: 0800_{16} to access the first TMS 9901, 0880_{16} to access the second TMS 9901, or 0900_{16} to access the third TMS 9901. The first TMS 9901 corresponds to the P2 pins, the second to the P3 pins, and the third to the P4 pins.

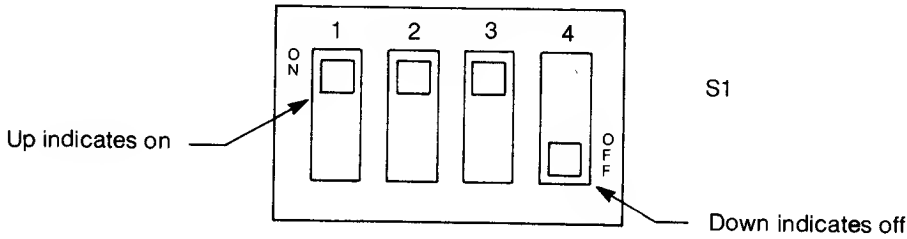
Switch all S1 positions on so the hardware base address 0100 is used for the /310 to correspond to the example in *Figure 11.* The third 9901 will be used so the software base address to be loaded in the program will be 0300_{16} and the I/O software base address will be 0320_{16}. The connection to the 5MT I/O modules will be thru P4 on the TM990/310 as shown in *Figures 27* and *35.* This connection should be made at this time.

Changing the Program

To change the program, the software address at the labels BASE 1 and BASE 2 needs to be changed. In the assembled program, these are at FC02 and FC04. The TIBUG monitor mode is obtained. A memory inspect/change (M) command to address FC02 will allow a change of the contents at that address to 0300_{16}. A space obtains address FC04 and its contents can be changed to 0320_{16}. However, when this change is made, the 9901 in the TM990/100M module no longer is enabled to receive the keyboard interrupt from the 9902 and, thus, the mode operation cannot be interrupted. Additional program changes must be made at $FC44_{16}$, $FCA2_{16}$, and $FCD6_{16}$ to continue to enable the 9901 INT4 in the module.

More sophisticated program changes could be made but one pattern that can be used for such changes is as follows:

```
1. (CR)                      ; CARRIAGE RETURN TO MONITOR
2. R                         ; OBTAIN WORKSPACE POINTER
3. (SP)                      ; OBTAIN PROGRAM COUNTER
4. 09E6 (CR)                 ; SET PC FOR LBLA
5. E                         ; EXECUTE LBLA
6. /FC44 (SP) (CR)           ; GO TO FC44
7. LI R12,>0100  (SP) (CR)   ; LOAD SOFTWARE BASE ADDRESS FOR
                               9901 ON MODULE
```



S1 Switch Settings 1	2	3	4	Binary Equal	TM990/310 Module CRU Base Address (Hex)	Register 12 Contents (Hex)
ON	ON	ON	ON	0	0100	0200
ON	ON	ON	OFF	1	01C0	0380
ON	ON	OFF	ON	2	0280	0500
ON	ON	OFF	OFF	3	0340	0680
ON	OFF	ON	ON	4	0400	0800
ON	OFF	ON	OFF	5	04C0	0980
ON	OFF	OFF	ON	6	0580	0B00
ON	OFF	OFF	OFF	7	0640	0C80
OFF	ON	ON	ON	8	0700	0E00
OFF	ON	ON	OFF	9	07C0	0F80
OFF	ON	OFF	ON	A	0880	1100
OFF	ON	OFF	OFF	B	0940	1280
OFF	OFF	ON	ON	C	0A00	1400
OFF	OFF	ON	OFF	D	0AC0	1580
OFF	OFF	OFF	ON	E	NOT USED	NOT USED
OFF	OFF	OFF	OFF	F	NOT USED	NOT USED

Figure 36. Programming Base Address of TM 990/310 Module

On the terminal the routine looks like this:

```
Q
?R
W=FF20
P=FC1A 09E6
?E
FE00    /FC44
FC44    020C    LI  R12,>100
FC46    0100
FC48
```

The same program change must be made at FCA2 and FCD6. When these are made, return to TIBUG by pressing the ESC key. The memory location just changed can be checked with the M command and the memory location.

To run the program, press the R key (it gives the WP) then (SP) to get the PC. Change the PC to $FC00_{16}$ and execute the program by pressing (CR) and the E key.

Incidentally, after these program changes, the only thing that needs to be done to change the 5MT I/O to the microcomputer module connector P4 is to change the original software base addresses at FC02=>0100 and FC04=>0120. No other changes need be made.

FUTURE EXTENSIONS

Now that the system is available there are endless variations that can be accomplished. Here are some that come to mind immediately:

1. Change the time interval on Mode 2 by:
   a. Changing the value in R6
   b. Changing the value loaded into the clock register
2. Add more modules to the 5MT43 and program a different input-output relationship.
3. Reprogram so that the program itself shifts the 5MT I/O to the /310 module if a /310 is present. Otherwise, the interface would remain on P4 of the microcomputer module.
4. Expand to more modules thru the TM990/310 modules.
5. Investigate how interrupts come through the TM990/310 module to the processor. There are some special linkages that must be connected on the /310 module to choose the interrupts that will come through the /310 to the processor.

CONCLUSION

It has been quite an experience starting at the first encounter and proceeding to the point where a microcomputer system is up and running and capable of being programmed to sense and control real-world industrial level energy. Components are available to easily apply the systems to many varieties of problem solutions.

Continue the learning process by finding real things to do with the system. Build on it to use it to its full capability and then add to it or replace it with a larger system to expand the applications. And remember, all the software that has been learned will be applicable to the new system applications, to different 9900 family members, and to new family members to be added in the future. Common compatible software is a real advantage. It's built into the 9900 family, so build on it. Good Luck.

A Low Cost Data Terminal

9

INTRODUCTION/ MICROCOMPUTER ARCHITECTURES

ABSTRACT

The architecture of the TMS 9940 Microcomputer is briefly reviewed. The microcomputer portion of a data terminal which currently employs the TMS 8080A Microprocessor is described. An equivalent design, which significantly reduces the chip count by using the TMS 9940 Microcomputer, is discussed in detail. Software comparisons between the two systems are made. A cost analysis of the two designs is discussed.

INTRODUCTION

As the complexity of LSI (large scale integration) electronics continues to increase, the system designer gains more and more freedom in designing low cost systems. One example of this capability is the Texas Instruments (TI) Model 745 Electronic Data Terminal, first introduced by TI in 1975. The Model 745 is a self-contained compact, telecommunications terminal which uses the thermal printing technique to achieve silent operation. The Model 745 features a 58 key, TTY33-compatible modular keyboard with integral numeric keypad, carrier detect indicator, two-key rollover, and key debounce circuitry. The Model 745 is capable of operating in full or half duplex modes at 10 or 30 characters per second, using a character set and code compatible with the American Standard Code for Information Interchange (ASCII).

The particular design of the Model 745 Data Terminal was made possible by the use of a microcomputer system as its controller. The Model 745 incorporates a TMS 8080A Microprocessor as the CPU of the Microcomputer. The purpose of this paper is to show how the Model 745 Terminal could be simplified even further by utilizing the newest addition to the 990/9900 Computer family: the TMS 9940 Microcomputer.

MICROCOMPUTER ARCHITECTURES

TMS 8080A Microprocessor

The TMS 8080A is an eight-bit general purpose Microprocessor *(Figure 1)*. The TMS 8080A chip contains seven registers and has a 78-instruction repertoire. The chip requires three power supplies (+12, ±5Vdc) and accepts a two-phase high-level clock input. The TMS 8080A features 64K byte addressing of off-chip memory, and is packaged in a 40-pin package.

9

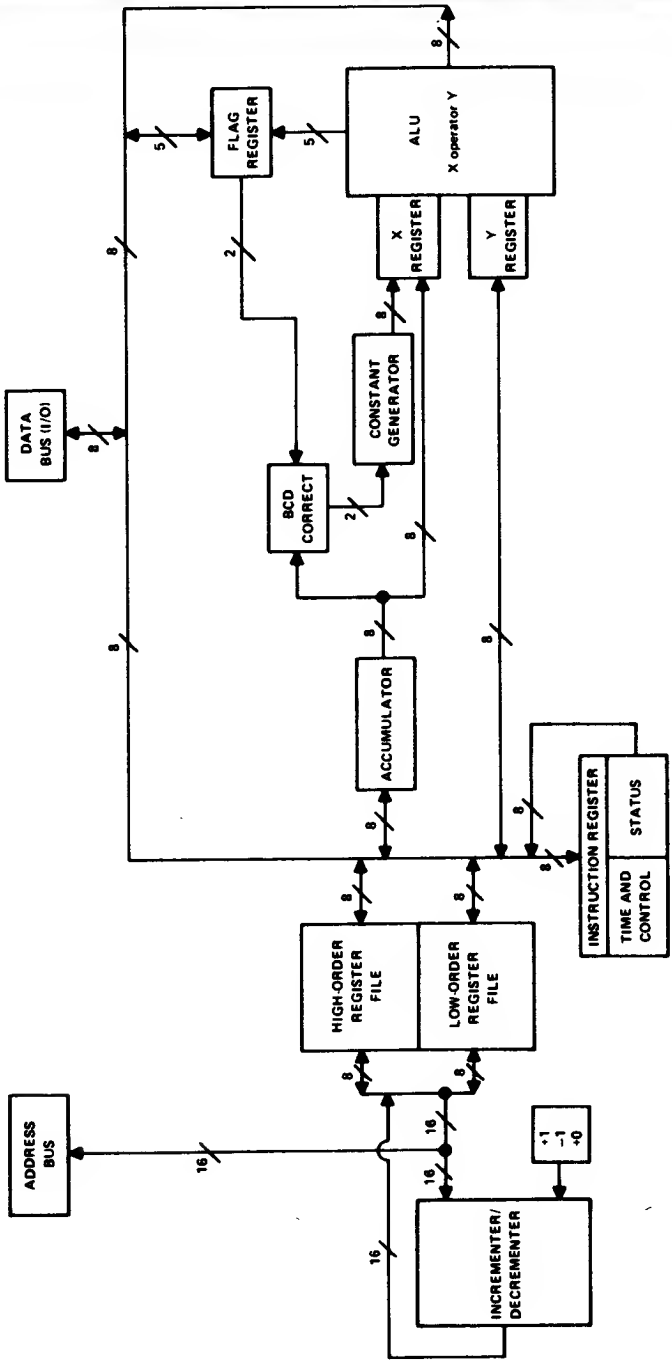


Figure 1. TMS 8080A Functional Block Diagram

TMS 9940 Microcomputer

The TMS 9940 is a 16-bit general purpose, single-chip microcomputer *(Figure 2)*. The TMS 9940 contains 2K bytes of ROM (or EPROM) and 128 bytes of RAM, along with a programmable timer/event counter. The 9940 is software-compatible with the 990/9900 family of microprocessors/minicomputers, and executes 68 instructions. The TMS 9940 requires a single 5-volt power supply and incorporates an (external) crystal-controlled oscillator on the chip. The circuit has 32 bits of general purpose I/O (expandable to 256 bits), and is housed in a 40-pin package.

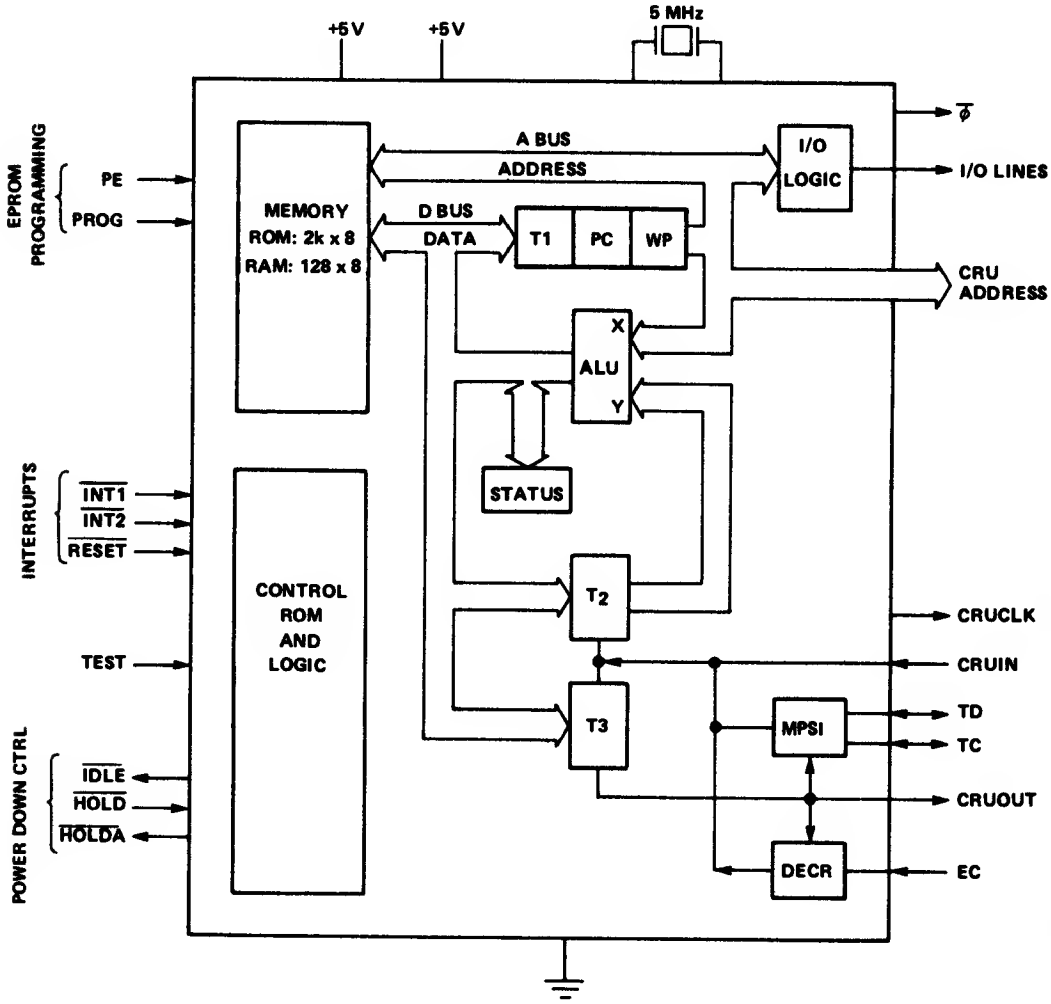


Figure 2. TMS 9940 Functional Block Diagram

HARDWARE DESIGN

A functional block diagram of the Model 745 Data Terminal is shown in *Figure 3.* The control electronics monitor all terminal inputs and generate all necessary timing and control signals to effect data transfers, cause printhead and paper motion, and create printable characters through the thermal printhead. Each block of the diagram is discussed separately below.

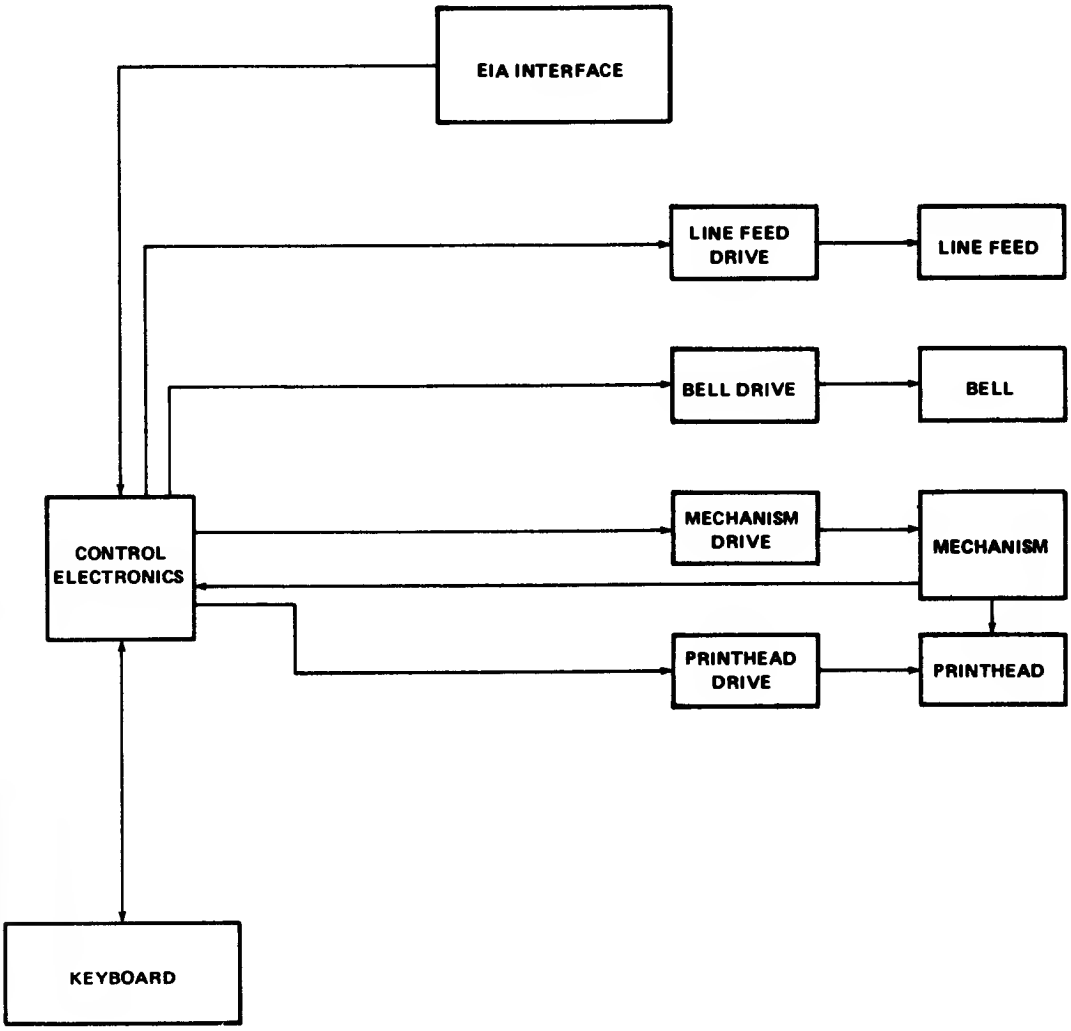


Figure 3. Model 745 Data Terminal Functional Block Diagram

Keyboard

The Model 745 keyboard is a TTY33-compatible, alphanumeric keyboard with an integral numeric keypad. The keyboard is equipped with 54 single-action keys, four alternate action switches, and an indicator lamp which signals that the data carrier signal is being received by the terminal. The control electronics must generate control signals to scan the keyboard and debounce key switch depressions. When a key depression is detected during a scan, the character is encoded and the appropriate action is taken by the terminal. Each scan is total so as to detect possible multiple key depressions. When simultaneous depressions are detected during a scan, neither key is acted upon. This scanning/debounce technique effects a two-key rollover with lockout.

Printhead

The printhead consists of a five by seven dot matrix of 35 heating elements *(Figure 4)* mounted on a monolithic chip. The chip is mounted on a heatsink, and is connected to the printhead drive electronics through a flexible ribbon cable. Upon receipt of a character from the keyboard or the communications line, the control electronics must generate the appropriate control signals to form the selected character utilizing the five by seven dot matrix format. The PRINT signal is switched on; then the matrix data is transferred to the printhead one column at a time. Each of the 35 heating elements on the printhead contains an SCR which controls the heating current. When both X and Y inputs are positive to a given element, the SCR energizes and reamins on (approximately 10 msec) until PRINT is switched off.

The X and Y address drivers are implemented on two SN98614 linear integrated circuits, each of which consists of six driver circuits. Each driver circuit has a low power TTL-AND input stage and a totem-pole, power transistor output stage. The drivers are enabled by the signal LDPRHD.

Printhead Lift

The printhead is lifted to relieve pressure upon the paper during line feed and carriage return operations. The control electronics must generate a signal (LFTHD) to control the solenoid which lifts the printhead.

Mechanism

Horizontal movement of the printhead is controlled by a three-phase 15-degree stepping motor. An optical sensor is mounted on the motor shaft to provide feedback for the control of stepping motion during printing and slew motion during carriage return. The print/step cycle operates synchronously up to 35 characters per second. The control electronics must output five signals to control the motor. The STEP and FAST signals are used to control the current in the motor windings; and PHA, PHB, and PHC are drive signals for the three motor phases. The mechanism drive electronics converts these TTL logic level signals into the closed loop controller dc current required by the motor.

The optical sensor provides data on motor position so that the control electronics "know" when to apply braking to change phases, or to make other decisions concerning motion of the printhead carriage. The sensor consists of a 24-position slotted wheel which interrupts a light path between an IR emitting diode and a photosensitive transistor. The sensor issues pulses to the control electronics as the slots interrupt the light path.

Bell

A buzzer (a piezoelectric disc) produces an audible signal at a nominal frequency of 3.2 kHz. Upon receipt of the BEL character from the keyboard or communications line, the control electronics must generate a timed signal (250 ± 25 msec) to produce the sound.

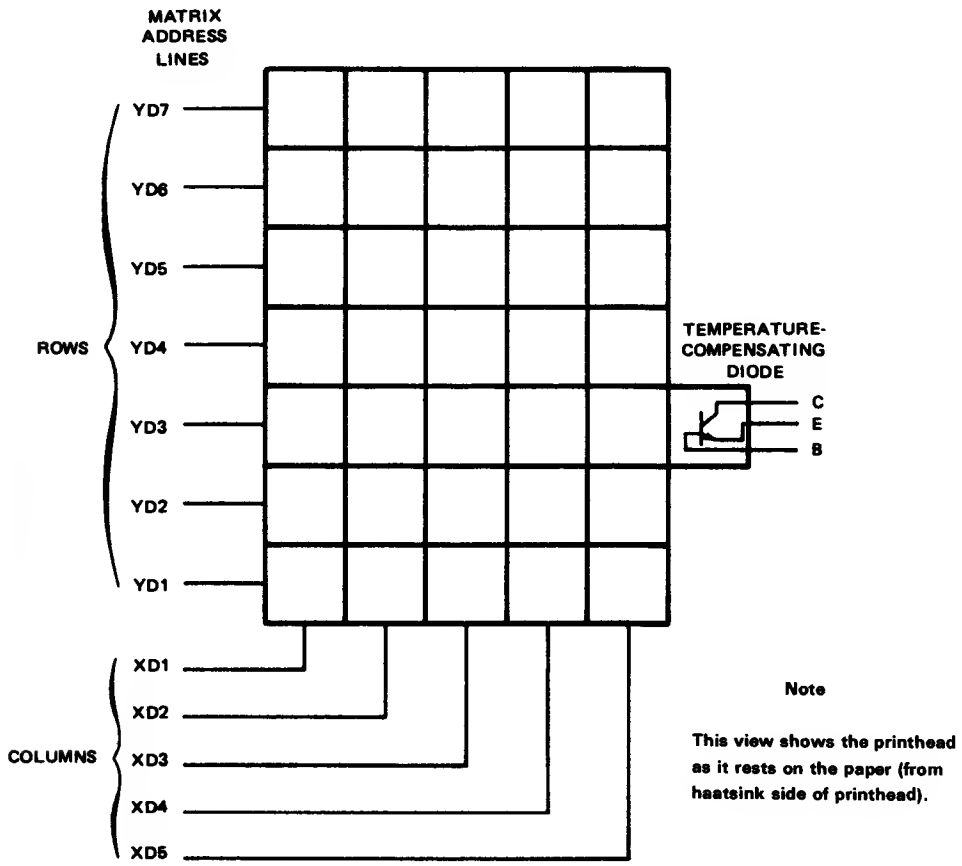


Figure 4. Printhead Matrix Address Lines

Line Feed

Vertical movement of the paper is controlled by the line feed solenoid which is mechanically coupled to a rachet mechanism. To advance the paper one line, the control electronics must lift the printhead and output a timed signal (15 msec) followed by an off period of 16.8 msec to the line feed solenoid.

EIA Interface

The control electronics must transmit and receive asynchronous serial data in accord with *ANSI Standard for Character Structure and Parity Sense,* X3.16-1966 and *ANSI Standard for Bit Sequence,* X3.15-1967. The TTL-level signals RCVD and XD are converted to standard EIA RS-232-C levels in the EIA interface.

Control Electronics

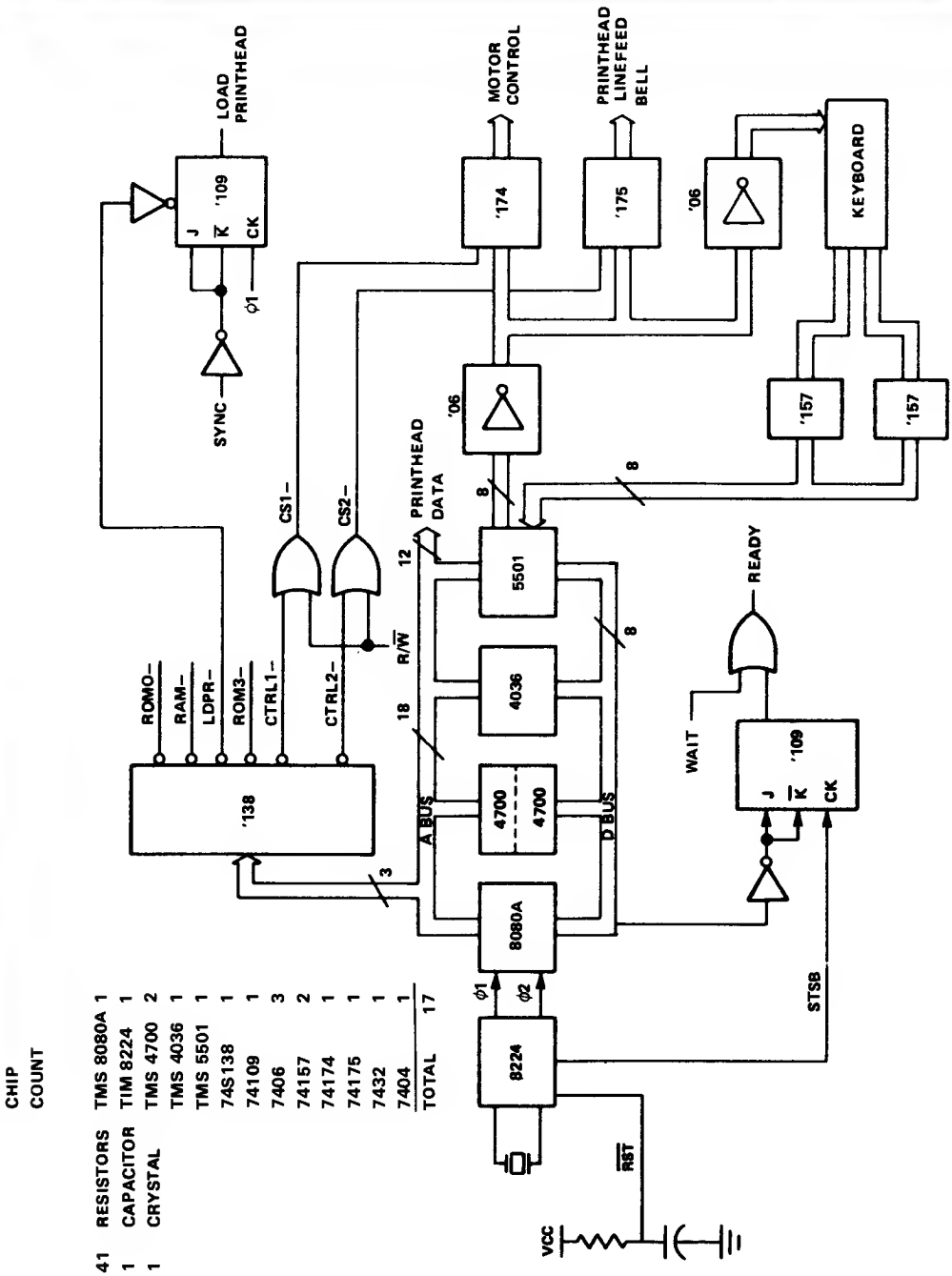
The control electronics function is performed by an interrupt driven, stored program microcomputer. As aforementioned the system requirements for the microcomputer I/O consist of:

Keyboard:	Matrix scan lines
Printhead:	Print data (12),LDPRHD,PRINT,LFTHD
Mechanism:	Step,FAST,PHA,PHB,PHC,SENSOR
Bell:	BELL
Linefeed:	LNFD
EIA Interface:	RCVD,XD

The microcomputer must generate these signals in the specified times and sequences to control the system.

TMS 8080A Microcomputer System

A schematic of the microcomputer design using the TMS 8080A Microprocessor is shown in *Figure 5.* The complete design requires 17 integrated circuits, 41 resistors, one crystal, and one capacitor. The memory consists of 2K bytes of ROM (two TMS 4700's) and 64 bytes of RAM (one TMS 4036). The TMS 5501 is an 8080A peripheral I/O controller which contains a universal asynchronous receiver/transmitter, programmable timers, interrupt prioritization and control, an eight-bit input port, and an eight-bit output port. The eight-bit output port is expanded by using TTL components (7406, 74174, 74175) to provide the necessary number of direct outputs for the keyboard and latched outputs for the static outputs. The input port is expanded using 2-to-1 multiplexers (74157) to permit elimination of diodes from the keyboard matrix. Data is sent to the printhead over 12 bits of the address bus by loading the data into the HL registers, and then executing a dummy MOVM instruction while the 74109 JK flip-flop outputs the LDPRHD strobe signal. The 74S138, 3-to-8 decoder generates the required chip selects for the various components. The SENSOR input feeds into the TMS 5501 interrupt logic to interface to the TMS 8080A.



CHIP COUNT		
	TMS 8080A	1
	TIM 8224	1
	TMS 4700	2
	TMS 4036	1
	TMS 5501	1
	74S138	1
	74109	1
	7406	3
	74157	2
	74174	1
	74175	1
	7432	1
	7404	1
	TOTAL	17
41	RESISTORS	
1	CAPACITOR	
1	CRYSTAL	

Figure 5. TMS 8080A Microcomputer System

TMS 9940 Microcomputer System

A schematic of the microcomputer design using the TMS 9940 Microcomputer is shown in *Figure 6.* The complete design requires two integrated circuits, 18 resistors, one crystal, one capacitor and 16 diodes. The internal memory of the TMS 9940 provides 2K bytes of ROM and 128 bytes of RAM. The TMS 9902 Asynchronous Communications Controller is a TMS 9900-family peripheral which contains a universal asynchronous receiver/transmitter and a programmable timer. The 32 I/O lines provided by the TMS 9940 interface to all the I/O functions with 10 lines software-multiplexed between the keyboard scan, TMS 9902 control, and printhead data. When P14 through P20 are in the input mode, the keyboard is scanned by sequentially raising P1 through P10 *high* (with the others being held *low*) while switching P14 through P20 to the output mode and outputting *high* signals, isolates P1 through P10 so that they can be used for other purposes. The LDPRHD signal is divided into two signals (LDPRHD1 and LDPRHD2) to obtain an output current sink needed for the SN98614's. The two interrupt inputs are used by the SENSOR input (highest priority) and the $\overline{\text{INT}}$ output from the TMS 9902.

FIRMWARE DESIGN

A block diagram of the Model 745 firmware, *Figure 7,* shows that the system firmware can be divided into three major sections: (1) keyboard scanning and encoding, (2) printhead control, and (3) internal data control. The keyboard and printhead routines represent the major portion of the system: the data control routine is used to direct character processing between the keyboard, the printhead, and the EIA interface.

Keyboard Routine

The keyboard is viewed by the control electronics as a matrix of key switches, with all keyboard scanning, debouncing, and encoding done by the microcomputer. The keyboard is scanned once each 4.3 msec. When a key depression is detected, the character is encoded by the addition of a constant number to the row/column number of the key to provide the ASCII code, and the appropriate action is taken by the terminal. (*Note:* In the numeric mode a look-up table is used to provide the ASCII code).

After a depression is detected, 12 msec are allowed for all contact-make bounce to settle out and then scanning resumes at 4.3-msec intervals. No other key depressions are processed by the terminal until the first depression is released. When this occurs, 12 msec are allowed for contact-break bounce, then the keyboard scan again resumes at 4.3-msec intervals. Each scan is a complete scan so that multiple key depressions may be detected. When simultaneous depressions are detected, neither key is acted upon, thus effecting a two-key-rollover-with-lockout operation.

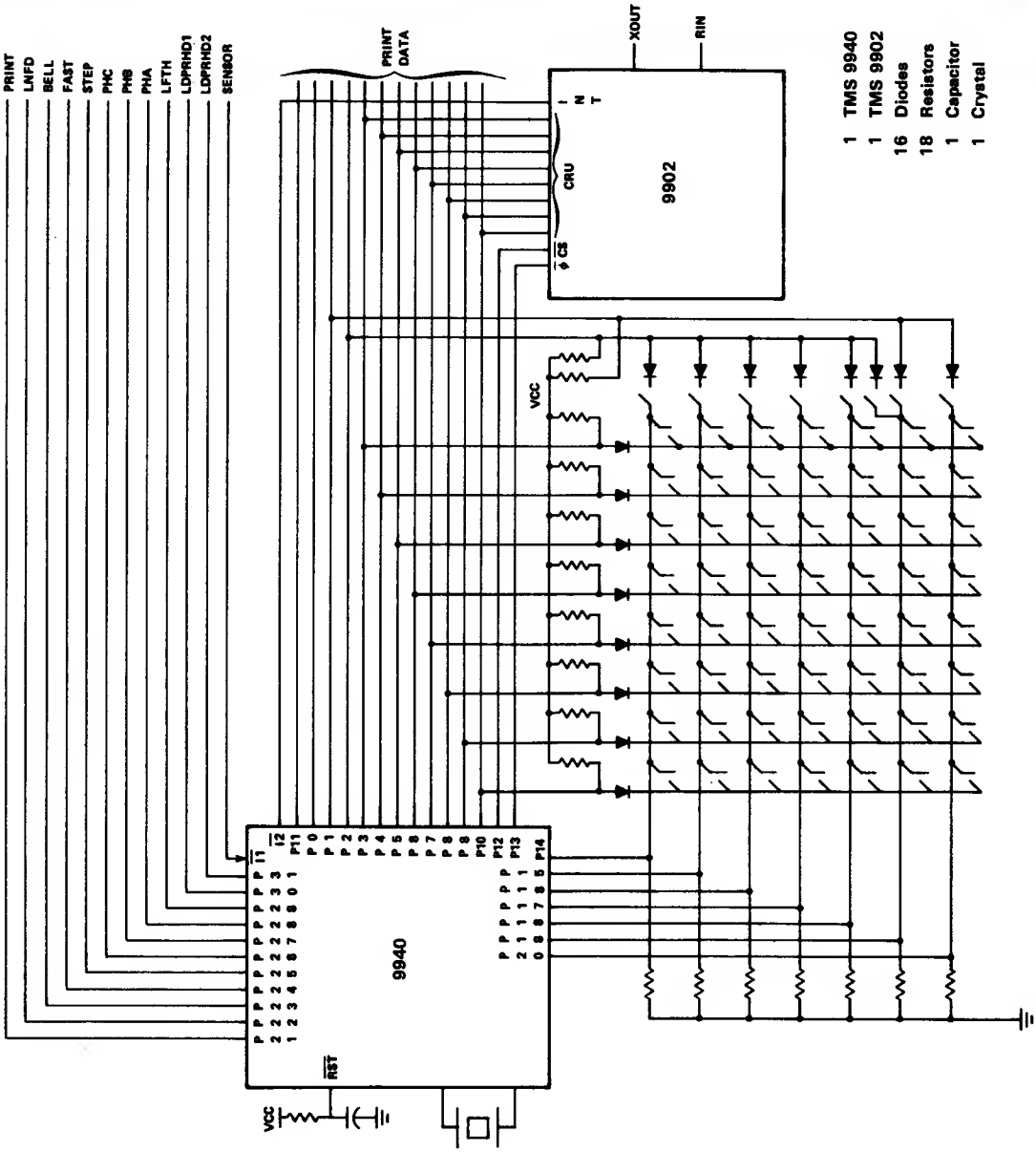


Figure 6. TMS 9940 Microcomputer System

Printhead Control

The microcomputer positions the printhead horizontally by timing different levels of current through the phase windings of the stepping motor. The print/step cycle operates asynchronously up to 35 CPS, with the cycle time divided into three basic segments: settle (11.3 msec), print (10 msec), and step (7.2 msec). Slew time for a full 80 columns is a maximum of 195 msec with backspace operations performed in one character-time. An automatic carriage return/line feed is executed upon receipt of the 81st character in a line. Upon applying power the printhead is backspaced to the left margin.

Fault detection methods are used by the microcomputer to prevent damage during power cycling conditions, obstruction of printhead motion, or loss of optical sensor signal. During the print segment, the microcomputer energizes the printhead voltage (PRINT), indexes into the dot matrix table (part of the 2K of ROM) by the ASCII character value, chooses the appropriate dot pattern, and loads the printhead one column at a time. The printhead is loaded during the first 200 μsec of PRINT; the PRINT signal remains on for 10 msec to allow the thermal sensitive paper to convert.

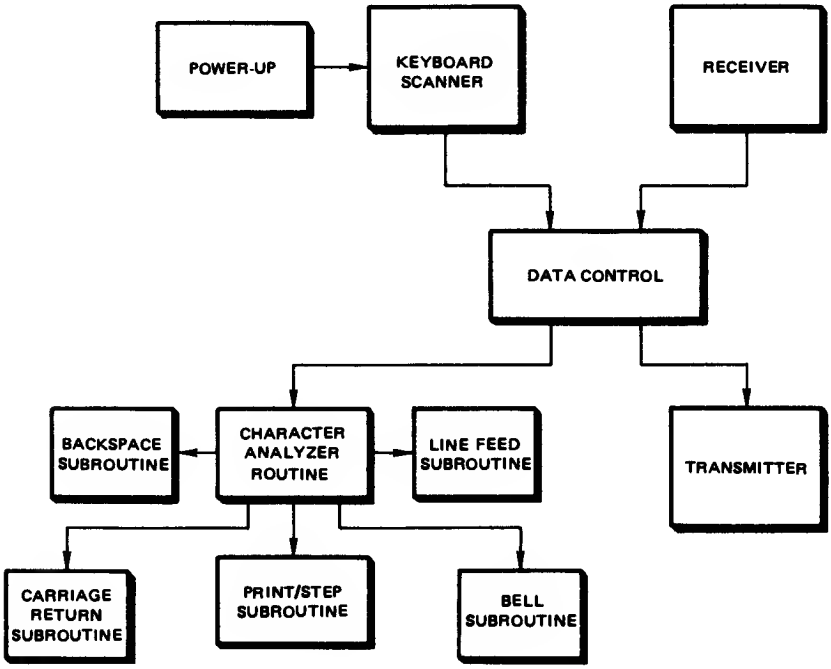


Figure 7. Model 745 Firmware Structure

The step segment steps the printhead one column by using two timers and the sensor. One timer is used to control pulse widths for the FAST and STEP pulses. These pulses control the amount of current in both the leading and lagging winding of the stepper motor, thus controlling the torque generated by the motor. The sensor signals the beginning of braking. The second timer is used to time the total step and is divided into two segments: The first verifies that the sensor occurred, and the second segment defines the end of the step. The use of the second timer makes the step time independent of when the sensor interrupt occurs so that the microcomputer can compensate for varying friction loads on the printhead.

The carriage return operation will slew the head to column one under control of the microcomputer using two timers and the sensor input. The step current remains on during the entire carriage return to develop high torques in the motor. One timer is used to control the fast pulse, thus controlling the current in the lagging phase of the stepper motor. The second timer is used as a reference to which to compare the sensor information, and this comparison results in the microcomputer accelerating or decelerating the motor to maintain control of printhead speed.

Firmware Implementation

Table 1 lists the number of instructions and memory bytes required to implement the system firmware for both the TMS 8080A and the TMS 9940. The three major sections [(1) keyboard routine, (2) printhead control, and (3) data control] are listed separately, along with the dot pattern table for the five by seven printhead matrix. The number of memory bytes required for each system is 2048 (the number available) and the number of instructions required is 867 for the TMS 8080A and 584 for the TMS 9940.

Table 1. *System Firmware Implementation*

Routine	TMS 8080A Microprocessor		TMS 9940 Microcomputer	
	Number of Instructions	Bytes	Number of Instructions	Bytes
Keyboard	260	486	178	472
Printhead	411	855	291	884
Control	196	367	115	352
Dot Pattern	–	340	–	340
TOTAL	867	2048	584	2048

COST ANALYSIS

Table 2 illustrates the component cost for the two microcomputer systems, assuming a production level of 10,000 units. The component cost of the TMS 8080A System is $48.81, and the cost of the TMS 9940 System is $22.78. In addition, other cost reductions will be realized from savings in incoming test (17 IC's versus two IC's), PC board area (approximately 45 square inches versus 6 square inches), and associated assembly labor and overhead. In total a significant overall cost savings will be realized in the recurring cost of the end product.

***Table 2.** Component Cost Analysis*

TMS 8080A System	$48.81
TMS 9940 System	$22.78

TMS 9900
Floppy Disk Controller

9

TABLE OF CONTENTS

SECTION	TITLE	PAGE
I.	INTRODUCTION	9-93
II.	SYSTEM DESCRIPTION	9-94
2.1	Data Terminal	9-94
2.2	Floppy Disk Drive	9-96
2.2.1	Floppy Disk	9-96
2.2.2	Physical Data Structure	9-98
2.2.3	Encoding Technique	9-98
2.2.4	Track Format	9-98
2.2.5	Cyclic Redundancy Check Character	9-98
2.2.6	Reading Data	9-101
2.2.7	Writing Data	9-101
2.2.8	Track Formatting	9-101
2.2.9	Floppy Disk Timing	9-102
III.	HARDWARE DESCRIPTION	9-103
3.1	Clock Generation and Reset	9-103
3.2	CPU	9-104
3.3	Memory Control	9-104
3.4	Disk Read/Write Select	9-105
3.5	Storage Memory	9-105
3.6	Program Memory	9-107
3.7	Control I/O	9-108
3.8	Floppy Disk Drive Interface	9-108
3.9	Index Pulse Synchronization	9-109
3.10	Read Pulse Synchronization	9-110
3.11	Bit Detector	9-110
3.12	Bit Counter	9-110
3.13	Write Control and Data	9-111
3.14	Data Shift Register	9-111
3.15	Clock Shift Register	9-112
IV.	DISKETTE DATA TRANSFER	9-113
4.1	Disk Write Operations	9-113
4.2	Disk Read Operations	9-116
4.2.1	Clock and Data Bit Detection	9-116
4.2.2	Clock/Data Separation	9-120
4.2.3	Byte Synchronization	9-120
4.2.4	Reading Disk Data	9-121
4.3	Read/Write Logic Combination	9-121
V.	SOFTWARE	9-126
5.1	Software Interface Summary	9-126
5.2	Control Software	9-126
5.2.1	Floppy Disk Control Program	9-128
5.2.2	Operator Commands	9-128
VI.	SUMMARY	9-130

LIST OF ILLUSTRATIONS

FIGURE	TITLE	PAGE
1.	TMS 9900 Floppy Disk Controller System	9-94
2.	TI 733 KSR Terminal	9-94
3.	Data Transmission Format	9-95
4.	Terminal Interface	9-95
5.	Floppy Disk Drive	9-97
6.	Diskette Envelope and Diskett	9-97
7.	F M Data Pattern .1011	9-98
8.	Track Recording Format	9-99
9.	Hardware CRC Generation	9-100
10.	Clock Generation and Reset	9-103
11.	TMS 9900 CPU	9-104
12.	Memory Control	9-104
13.	Disk Read/Write Select	9-105
14.	Storage Memory	9-106
15.	Program Memory	9-107
16.	Control I/O	9-108
17.	Floppy Disk Drive Interface	9-109
18.	Index Pulse Synchronization	9-109
19.	INDSYN Timing	9-110
20.	Read Pulse Synchronization	9-110
21.	Bit Detector	9-111
22.	Bit Counter	9-111
23.	Write Control and Data	9-112
24.	Data Shift Register	9-112
25.	Clock Shift Register	9-112
26.	Write Timing	9-115
27.	Bit Shifting	9-116
28.	Bit Detection Timing and Logic	9-118
29.	Clock/Data Separation Timing	9-121
30.	Disk Read Timing	9-122
31.	Memory Address Assignments	9-127
32.	Floppy Disk Control Program	9-131

9◀

LIST OF TABLES

TABLES	TITLE	PAGE
1.	RS-232C Signal Levels	9-96
2.	Memory Address Assignments	9-105
3.	Write Clock Patterns	9-114
4.	Bit Shift Direction	9-117
5.	Worst-Case Pattern Load Values	9-119
6.	Data Mask	9-119
7.	Bit Detector Counter Load Values	9-120
8.	CRU Address Assignments	9-126
9.	Operator Commands	9-128
10.	Command Entry Parameters	9-128
11.	Command Summary	9-129

SECTION I

INTRODUCTION

This application report describes a TMS 9900 microprocessor system which controls a floppy disk drive and interfaces to an RS-232C type terminal. In addition to providing useful information for the design of a similar system, this application report also shows many of the design considerations for any TMS 9900 microprocessor system design.

The floppy disk is rapidly becoming the most widely accepted bulk storage medium for microprocessor systems. Using standard encoding techniques, a single floppy disk will contain in excess of 400K bytes of unformatted data. Access time to a random record of data is vastly superior to serial media such as cassettes and cartridges, and the medium is both non-volatile and removable.

The use of a microprocessor in the floppy-disk controller or "formatter" is desirable for a number of reasons. The number and cost of components is reduced: this design contains 24 integrated circuits, while random-logic designs typically contain more than 100. The commands from the user interface (in this case, the terminal) to the controller may be more sophisticated, relying on the microprocessor to intrepret the commands. The microprocessor also enables the controller to perform diagnostic functions, both on the controller itself and on its associated drives, not available with a random-logic system.

The Texas Instruments TMS 9900 microprocessor is particularly well-suited to this application. The TMS 9900 is a 16-bit microprocessor capable of performing operations on single bits, bytes, and words. The CRU provides an economical port for bit-oriented input/output, while the parallel memory bus is available for high-speed data. The speed of operation of the TMS 9900 minimizes additional hardware requirements. The powerful memory-to-memory instruction set and large number of available registers simplify software, both in terms of number of assembly language statements and total program memory requirements.

SECTION II

SYSTEM DESCRIPTION

Figure 1 illustrates the relationship of the system elements. Commands are entered by the user at the terminal. These commands are serially transmitted to the controller. The controller interprets the commands and performs the operations specified, such as stepping the read/write head of the drive to a particular track, and reading or writing selected data.

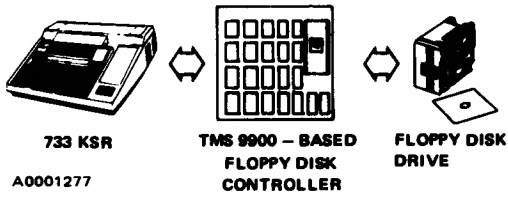
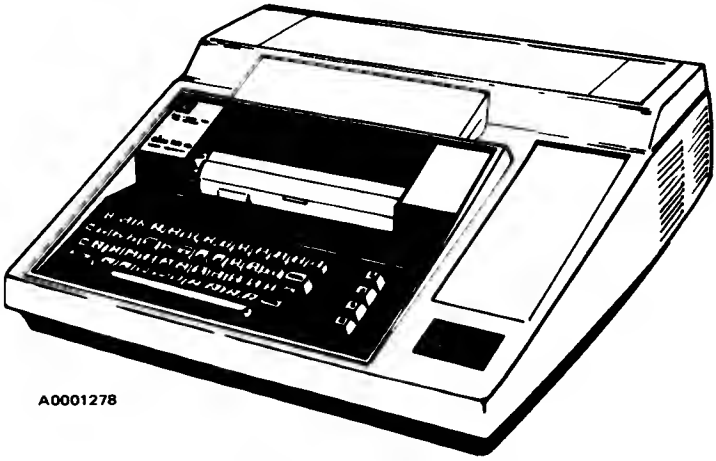


Figure 1. TMS 9900 Floppy Disk Controller System

2.1 DATA TERMINAL

The terminal used in this design is the Texas Instruments 733 KSR Silent Electronic Data Terminal (see Figure 2). Slight modifications to the software will allow the use of virtually any RS-232 terminal.



▶9

Figure 2. TI 733 KSR Terminal

The 733 KSR consists of a keyboard, printer, and a serial-communication line to the controller. The keyboard enables the operator to enter control commands and data for storage on floppy disc. The printer is used for echoing operator entries, data printout, and reporting of operational errors. The serial interface is full duplex, allowing data transmission both to and from the data terminal simultaneously.

Characters entered on the keyboard are transmitted to the controller in 7-bit ASCII code using asynchronous format, and characters to be printed are sent from the controller to the terminal in the same way. Transmission speed is 300 baud. The format for data transmission is shown in Figure 3.

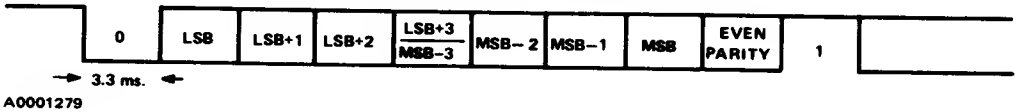


Figure 3. Data Transmission Format

The line idle condition is represented by a logic one. When a character is to be transmitted, the ASCII character is preceded by a zero bit, followed by the 7-bit ASCII code, even parity bit, and the logic-one stop bit. Any amount of idle time may separate consecutive characters by maintaining the logic-one level. Reading data is accomplished by continuously monitoring the line for the one-to-zero transition at the beginning of the start bit. After delaying one-half bit time (1.67 ms) the line is again sampled to ensure that the start bit is valid. If so, the line is sampled each bit time (3.33 ms) until all of the bits of the character have been sampled. The initial one-half bit delay causes subsequent samples to be taken at the theoretical center of each bit, thus providing a margin for distortion due to time base differences between the transmitter and receiver.

The control signals for the terminal are shown in Figure 4.

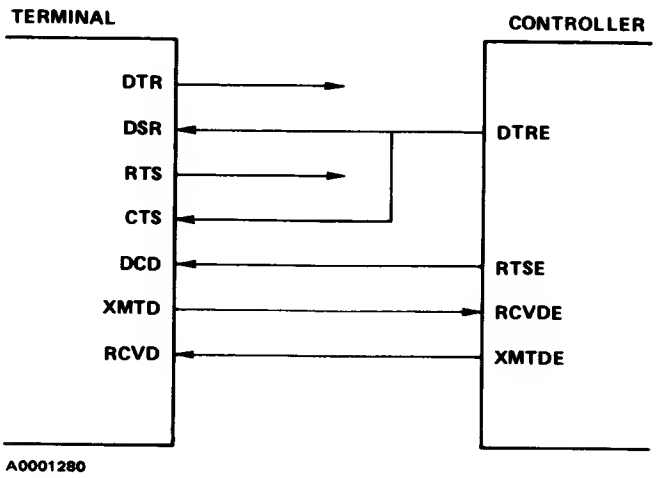


Figure 4. Terminal Interface

Detailed description of the signals is provided in *Electronics Industriès Association Standard RS-232C.* The signals used in this design are briefly described below.

DTRE – Data Terminal Ready is always on when power is applied to the controller, enabling operation of the serial interface by the terminal.

RTSE – Request to Send is on when a character is transmitted from the controller to the terminal.

XMTDE – Transmitted Data from the controller to the terminal.

RCVDE – Received Data from the terminal to the controller.

Signal levels conform to EIA Standard RS-232C, as shown in Table 1.

Table 1. RS-232C Signal Levels

Voltage Level	Data (XMTDE,RCVDE)	Control (DTRE,RTSE)
–25 to –3 VDC	1	OFF
+3 to +25 VDC	0	ON

The other important parameter for interfacing to the terminal is the amount of time required for a carriage return by the printer, which is 200 ms maximum for the 733 KSR.

2.2 FLOPPY-DISK DRIVE

The floppy-disk drive (Figure 5) is the electromechanical unit in which the recording medium, the floppy disk is inserted. The drive contains the electronics which control the rotation of the floppy disk, the reading and writing of data, and the positioning of the read/write head to select a particular track on the diskette.

2.2.1 Floppy Disk

The floppy disk, or diskette, is the recording medium (see Figure 6). It is enclosed in a plastic protective envelope which keeps foreign particles away from the recording surface. The inner material of the envelope is specially treated to minimize friction and static electricity discharge. The read/write head opening enables the head to come in contact with the recording surface. The index-access hole enables detection of the index hole.

▶9

When the index hole in the diskette becomes aligned with the index-access hole, an optical sensor generates the index pulse, providing a reference point for the beginning of each track. There are 77 concentric tracks for recording data. A particular track is accessed by moving the read/write head radially until the desired track is located.

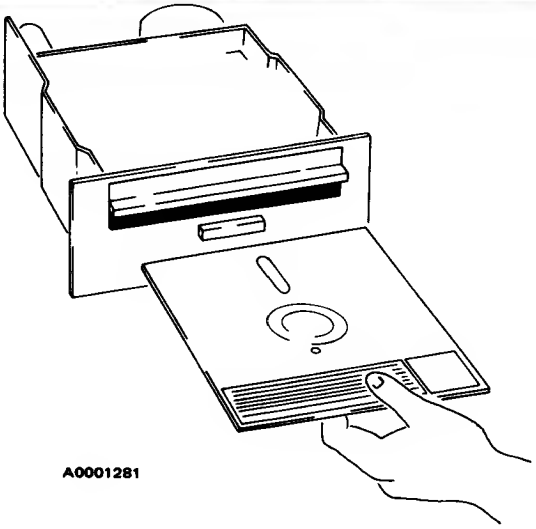


Figure 5. Floppy Disk Drive

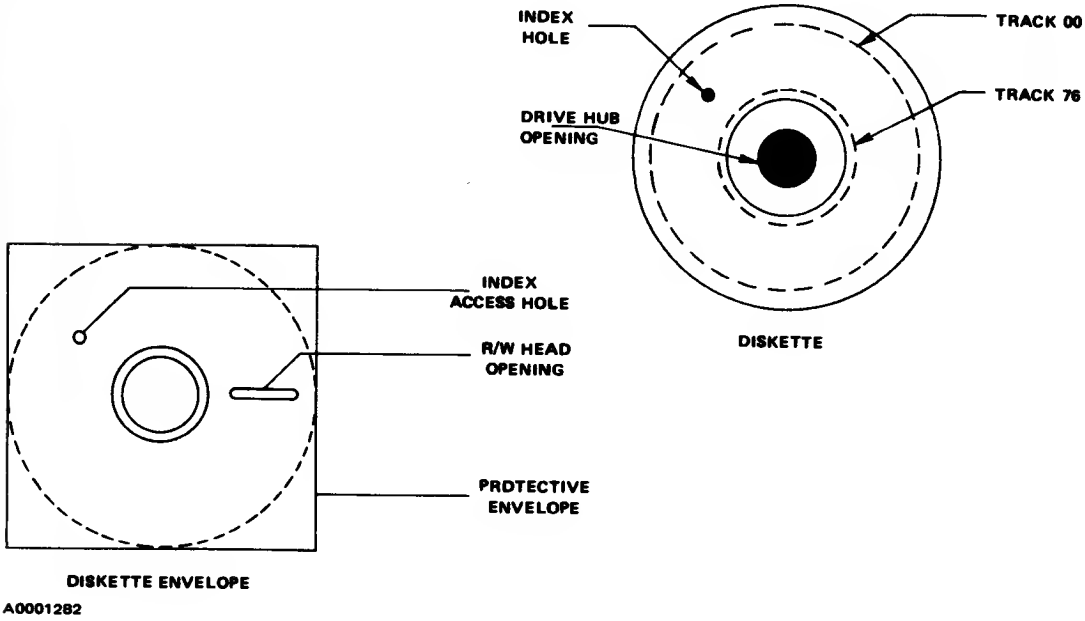


Figure 6. Diskette Envelope and Diskette

2.2.2 Physical Data Structure

The 77 tracks on a diskette are numbered from 00 (outermost) to 76 (innermost). Each track is subdivided into 26 sectors, or records, numbered sequentially from 1 to 26. Each sector consists of two fields: the ID field, which contains sector identification (track and sector number) and the data field, which contains 128 bytes of data.

2.2.3 Encoding Technique

The encoding technique used for representation of data on the diskette is a form of frequency modulation (FM), as shown in Figure 7. Each bit period is 4 microseconds long, resulting in a data-transfer rate of 250K bits per second. A pulse occurs at the beginning of each normal bit period. This pulse is called the clock pulse. If the data bit is a one, a pulse will occur also in the middle of the bit period, 2 μs after the clock bit. If the data bit is a zero, no pulse occurs in the middle of the bit period.

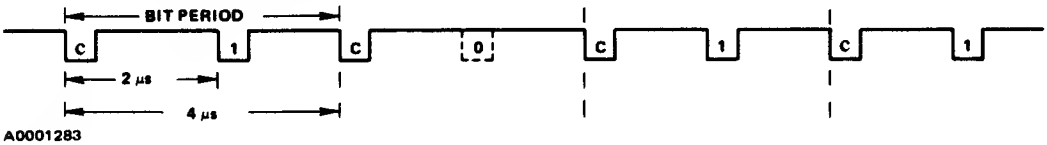


Figure 7. FM Data Pattern · 1011

Selected clock bits are deleted in special characters called marks. The absence of the clock bits results in unique sequences, used for synchronization at the beginning of fields.

2.2.4 Track Format

Each track is formatted to provide 26 "soft" sectors. The term soft sectoring means that the beginning of each sector is encoded on the medium through a unique bit sequence. Each of the sectors is separated by a gap of dummy data. Each of the two fields (ID and data) in each sector are also separated by a gap. The first byte of each field is a mark in which the clock pattern for the byte is $C7_{16}$ rather than FF_{16}. The organization of data and clock bits on each track is shown in Figure 8.

2.2.5 Cyclic Redundancy Check Character

The last two bytes at the end of each ID and data field comprise the 16-bit cyclic redundancy check character (CRC). The CRC is generated by performing modulo-2 division on the data portion of the entire field (including the mark) by the polynomial $X^{16} + X^{12} + X^5 + 1$. Before generation of the CRC begins, ▸9 the initial value is $FFFF_{16}$.

The analogous hardware operation is illustrated in Figure 9. All flip-flops are initially set to one. Each data bit in the field, beginning with the MSB of the mark byte, is shifted into the logic at DATAIN. The previous

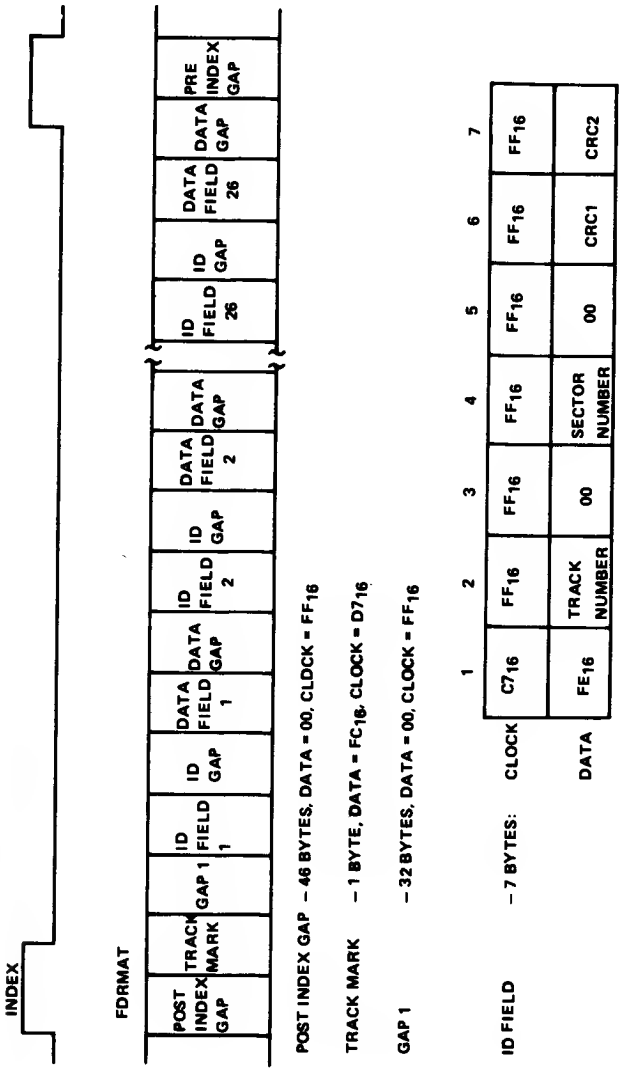


Figure 8. Track Recording Format

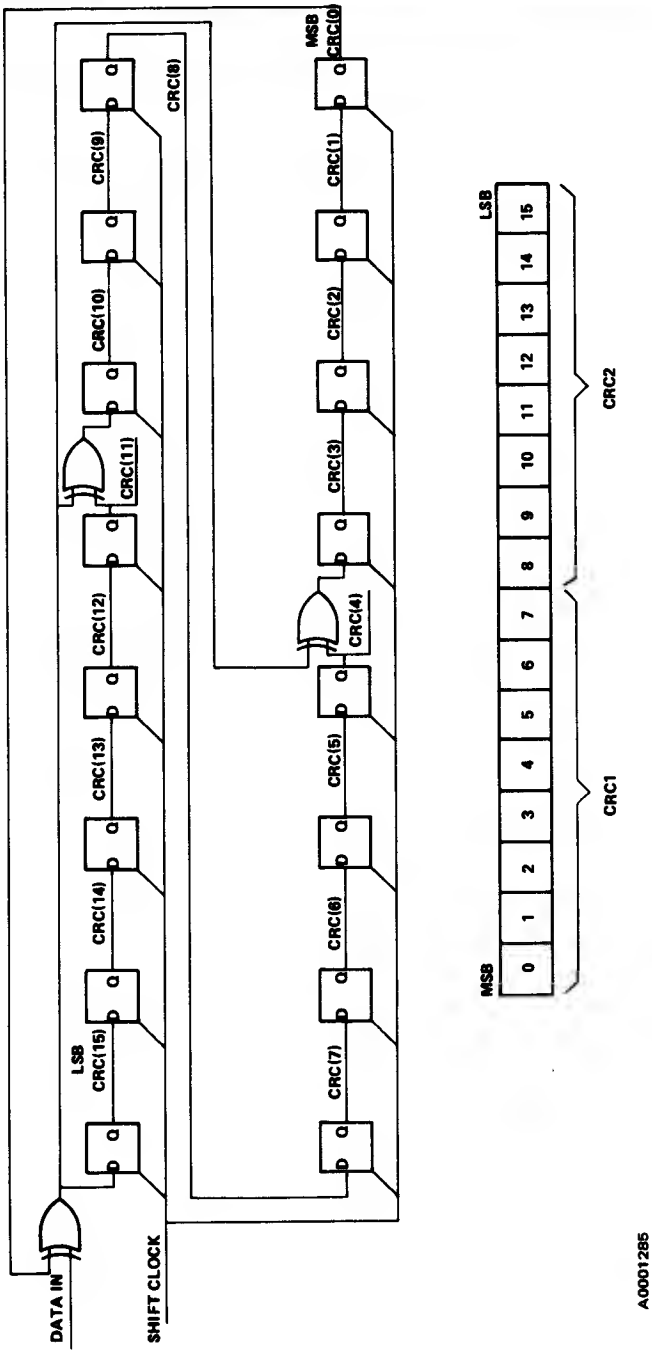


Figure 9. Hardware CRC Generation

MSB is exclusive ORed with the new input bit to generate a feedback term. This feedback term is stored in the LSB of the register, and is also exclusive ORed with other terms of the CRC. After all data bits of the field have been shifted in, the value in the register is the CRC. The most-significant byte is CRC1 and the least-significant byte is CRC2.

When reading the field, the identical operation is performed, presetting all flip-flops and shifting in all data bits. When reading, it is convenient to also shift in the CRC, causing the resultant value in the register to finally become all zeroes.

In this design, the CRC is calculated by software; however, the algorithm is identical.

2.2.6 Reading Data

The procedure for reading diskette data is as follows:

1. Search the serial-bit string for the ID mark (clock = $C7_{16}$, data = FE_{16}).
2. Read the next four bytes to determine if the desired sector has been located. If not, return to 1.
3. Read the CRC for the ID field and compare it to the expected value. If incorrect, report error and/or return to 1.
4. Search the serial-bit string for either the data mark (clock = $C7_{16}$, data = FB_{16}) or the deleted-data mark (clock = $C7_{16}$, data = $F8_{16}$).
5. Read the next 128 bytes and save.
6. Read the CRC for the data field and compare it to the expected value. If incorrect, report error and/or return to 1.

Normally, if the process is not completed before two index pulses are detected, indicating a complete diskette revolution, the try has failed. Either a retry will be performed, or an error is reported.

2.2.7 Writing Data

When writing data, the sector is located as in steps 1 through 3 above. Then, the ID gap, the data field complete with CRC, and a pad byte (data = 0, clock = FF_{16}) are written.

2.2.8 Track Formatting

9◀

The formatting process consists of writing all of the gaps, track mark, ID fields, and data fields, putting dummy data into the data bytes of the data field. After a track is formatted, only the ID gap, data field,

and the first byte of the data gap are altered when updating sectors. The number of bytes in the pre-index gap will possibly vary slightly, due to variations in the speed of revolution of the diskette.

2.2.9 Floppy-Disk Timing

Several important timing parameters pertain to the operation of the disk drive:

Bit transfer rate	250,000 bits/second
Track-to-track stepping time	10 milliseconds
Settling time (before read/write)	10 milliseconds
Rotational speed	360 RPM ±2%
Head load time (before read/write)	35 milliseconds

Thus, data is transferred at a rate of 250K bits/second, or 31.25K bytes/second ±2%. Stepping the head each track position requires 10 ms. An additional 10 ms delay must be observed after the final step before reliable data may be written or read. A delay of 35 ms must occur after the head is loaded ($\overline{RDY}$ = 0) before reliable data may be written or read.

SECTION III

HARDWARE DESCRIPTION

A complete logic diagram of the system is contained in the center of this report. The operation of each section is described separately.

3.1 CLOCK GENERATION AND RESET

The TIM 9904 is used to generate the 4-phase MOS clocks for the TMS 9900 (see Figure 10). Ten ohm resistors are connected in series to the clock lines for damping. The TIM 9904 should always be located physically close to the TMS 9900 to minimize the length of the conductor run for the MOS clocks. The $\overline{\phi 3}$ TTL-level output is used in the synchronous disk read/write control logic.

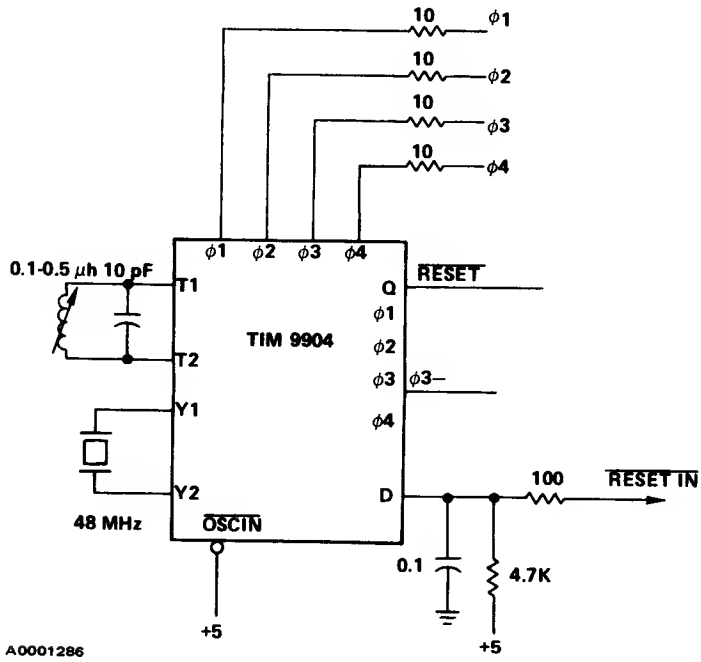


Figure 10. Clock Generation and Reset

A 48 MHz, third overtone crystal causes the clock frequency to be 3 MHz. The inductor of the LC tank circuit need not be variable; however, in wire-wrap prototypes the capacitance due to interconnect is difficult to predict. The $\overline{\text{OSCIN}}$ input is held high to disable the external clock input.

The RC input to the Schmitt-D input provides power-on detection. The $\overline{\text{RESETIN}}$ input is connected to an external pushbutton. The 100 ohm series resistor reduces contact arcing, thereby extending switch life.

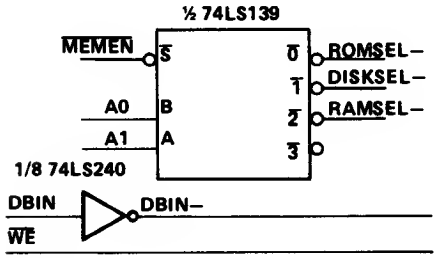
3.2 CPU

The TMS 9900 requires a minimum of external logic. Note that both the data and address buses are connected directly to the memory and disk read/write control logic without buffering as shown in Figure 11. This is due to the ability of the TMS 9900 outputs to sink up to 3.2 mA with 200 pF capacitive load.

The READY input is used to synchronize data transfers to and from the disk read/write control logic, eliminating the need for buffer registers. The $\overline{\text{HOLD}}$, $\overline{\text{LOAD}}$, and interrupt functions are not used in this design and are tied to their inactive (high) level.

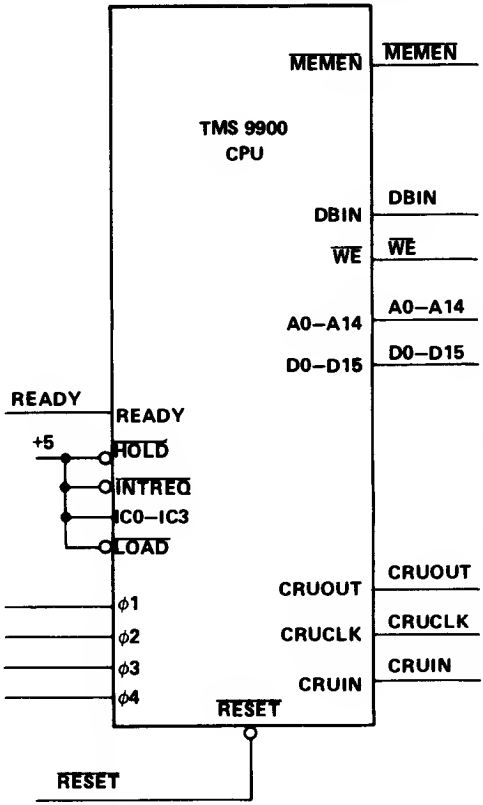
3.3 MEMORY CONTROL

Memory control logic, shown in Figure 12, consists of a simple decode of the high-order address lines, enabled by $\overline{\text{MEMEN}}$. Memory enabling signals are generated for EPROM (ROMSEL–), RAM (RAMSEL–), and the disk interface (DISKSEL–). Table 2 shows the memory address assignments.



A0001287

Figure 12. Memory Control



A0001288

Figure 11. TMS 9900 CPU

Table 2. Memory Address Assignments

Signal	A0	A1	Address Space	Function	Actually Used
ROMSEL–	0	0	000-3FFF	EPROM	000-07FF
DISKSEL–	0	1	4000-7FFF	Disk	7F8E-7FFE
RAMSEL–	1	0	8000-BFFF	RAM	8000-81FF
	1	1	C000-FFFF	Not Used	

Each of the enabling signals will be active when a memory cycle is being performed ($\overline{MEMEN} = 0$) accessing its address space.

3.4 DISK READ/WRITE SELECT

The DISKSEL signal is further decoded to generate separate select lines for disk read (DISKRD–) and disk write (DISKWT–) operations.

$$DISKRD\text{--} = \overline{(DISKSEL)\ (DBIN)\ (A14\text{--})}, \text{ and}$$
$$DISKWT\text{--} = \overline{(DISKSEL)\ (DBIN\text{--})\ (A14)}.$$

Disk read and write operations are specified by different addresses, and are selected only when the DBIN signal is at the proper level for the direction of transfer (see Figure 13). This is required because of the sequence of machine cycles performed by the TMS 9900 when performing a memory-write operation. In the MOV instruction, the CPU first fetches the contents of the memory location to be altered, then replaces this value with the source operand. In this design, the disk read and write operations are controlled by the READY line to synchronize data transfers. If read and write signals were not generated separately, there would be ambiguity with respect to the type of operation desired.

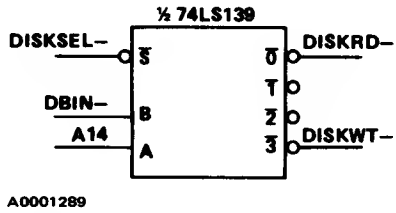


Figure 13. Disk Read/Write Select

This applies to all memory-mapped interfaces in TMS 9900 systems, i.e., the MOV instruction will cause a read operation to precede the write operation to the specified destination address.

3.5 STORAGE MEMORY

Storage memory, shown in Figure 14, is used for implementing workspace registers, maintenance of software pointers and counters, and buffering of a full sector of data.

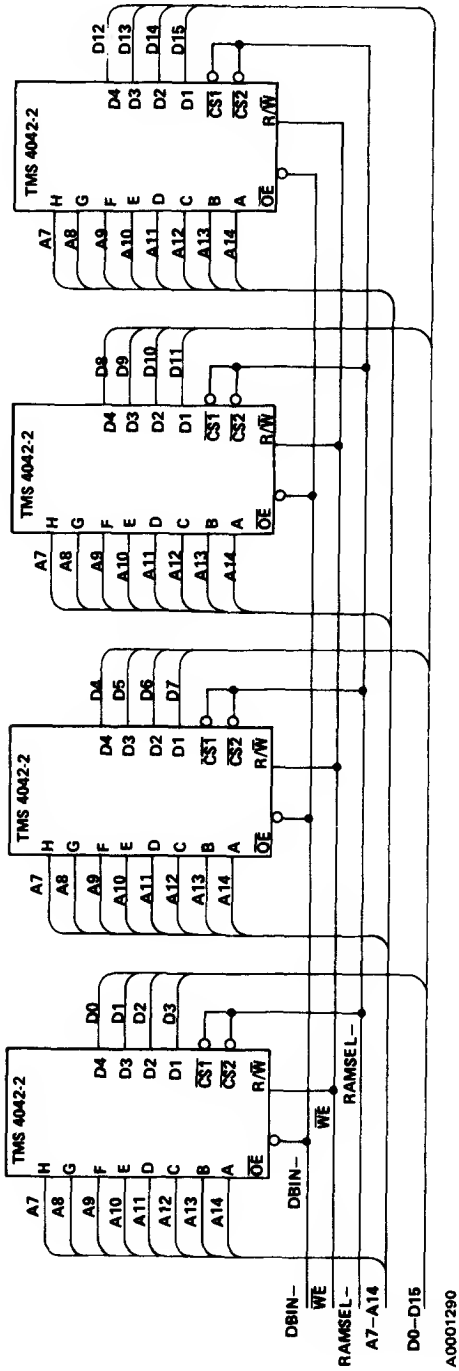


Figure 14. Storage Memory

This design utilizes four TMS 4042-2 RAMs, resulting in a 256-word array of RAM for temporary storage. This 256-word array may be addressed at locations 8000-BFFF, causing each memory location to be multiply defined (e.g., memory address 8000 selects the same word as memory address 8200). For simplificity, RAM will be referred to only as locations 8000-81FF.

Access times for the TMS 4042-2 are sufficiently fast to allow the TMS 9900 to access RAM without any wait states, thus READY will always be true when RAM is addressed. The output enable ($\overline{OE}$) inputs require that the DBIN output from the TMS 9900 be inverted to gate RAM onto the data bus. The $\overline{WE}$ output from the TMS 9900 is directly compatible with the R/$\overline{W}$ input. Data and address lines are connected directly to the CPU.

3.6 PROGRAM MEMORY

Program memory (Figure 15) is used for storage of the machine code program to be executed by the TMS 9900. Also, constants, the RESET vector and XOP vectors are contained in this space.

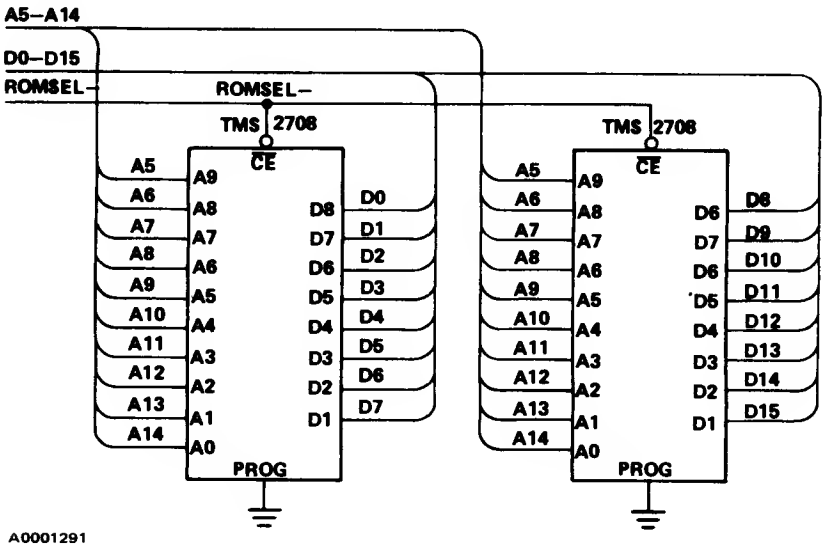


Figure 15. Program Memory

Two TMS 2708 erasable programmable read-only memories (EPROMs) comprise the program memory for this design, resulting in 1024 words of EPROM. EPROM is addressed at memory locations 0000-3FFF. Since these addresses are multiply defined, EPROM will be described only as memory addresses 0000-07FF. Access times for the TMS 2708 are such that no wait states are required.

3.7 CONTROL I/O

All of the control and status signals which require individual testing, setting, or resetting are implemented on the CRU, the bit addressable I/O port for the TMS 9900.

The benefits of using the CRU for these functions is twofold. First, eight bits of input and eight bits of output can be implemented with two 16-pin devices, which are substantially smaller and lower in cost than if these functions were implemented on the parallel-data bus. The second benefit is increased software efficiency. Control and status testing operations can be performed with single one-word instructions, rather than the ORing, ANDing, and maintenance of software images necessary when performing single-bit I/O on the memory bus.

Eight bits of output are implemented with the TIM 9906 8-bit addressable latch. The CRUCLK line must be inverted for input to the TIM 9906. The eight input bits are implemented using the TIM 9905 8-to-1 multiplexer. Individual I/O bits are selected using the three least-significant address lines, A12–A14. The control I/O is illustrated in Figure 16.

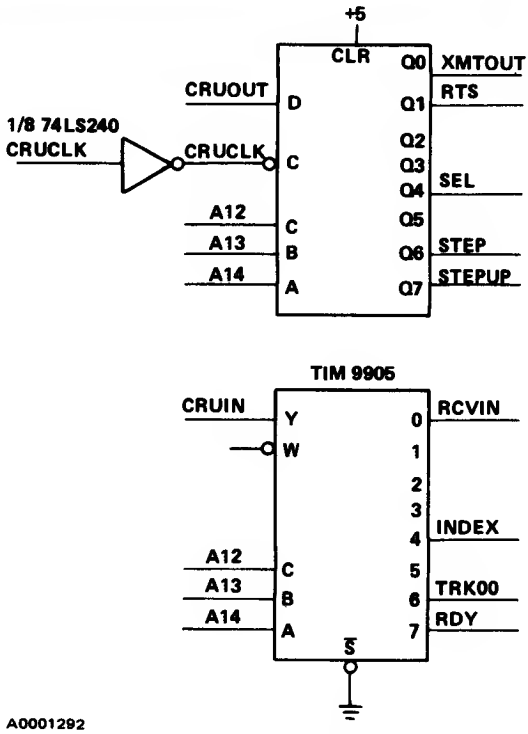


Figure 16. Control I/O

3.8 FLOPPY-DISK-DRIVE INTERFACE

All outputs to the drive are 7406 open-collector, high-voltage and current drivers. Pullups for the output signals are provided in the drive electronics. All inputs are terminated by 150 ohm pullup resistors to +5 volts, and are buffered and inverted. All input and output signals are active low.

$\overline{\text{SEL}}$ – Active when a stepping operation or a data transfer is being performed.

$\overline{\text{RDY}}$ – Active when the disk is ready to perform a stepping or transfer operation (i.e., $\overline{\text{SEL}}$ = 0, diskette is in place, door is closed, power is furnished to the drive).

$\overline{\text{STEP}}$ – A minimum 10 μs pulse causes the read/write head to move one track position in the direction selected by $\overline{\text{STEPUP}}$.

$\overline{\text{STEPUP}}$ – When $\overline{\text{STEPUP}}$ = 0, the read/write head moves in one track position. When $\overline{\text{STEPUP}}$ = 1, the head will move out (toward track 00).

$\overline{\text{TRK00}}$ – Active when the read/write head is located on the outermost track (track 00).

$\overline{\text{INDEX}}$ – As the diskette rotates in the drive, the index pulse occurs once per revolution, providing a reference point for the beginning of each track.

$\overline{\text{WRITE ENABLE}}$ – This signal must be active a minimum of 4 μs before a write operation begins, and must be maintained active during the entire write operation.

$\overline{\text{WRITE DATA}}$ – This signal contains a series of pulses representing the data to be written to the disk in the FM format previously described.

$\overline{\text{READ DATA}}$ – This signal contains a series of pulses representing the data to be read from the disk in the FM format previously described.

Figure 17 illustrates the floppy-disk-drive interface.

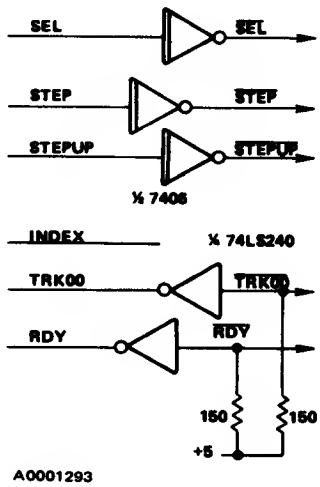


Figure 17. Floppy-Disk Drive Interface

3.9 INDEX PULSE SYNCHRONIZATION

Since the index pulse is a term in some of the expressions that are sampled by the CPU, it must be synchornous to the CPU. The circuit shown in Figure 18 generates a signal one $\phi 3$ clock cycle long at the beginning of each index pulse from the drive. RDY will be inactive when the drive is turned off or the door is open, thus connection of RDY to the preset input of the flip-flop shown causes INDSYN to be active as long as RDY = 0 (see Figure 19). Forcing INDSYN to be one when RDY = 0 prevents the CPU from remaining in a wait state when the drive is disabled during data transfer.

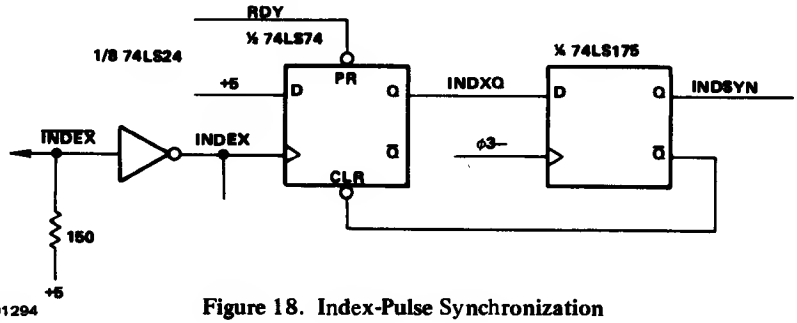


Figure 18. Index-Pulse Synchronization

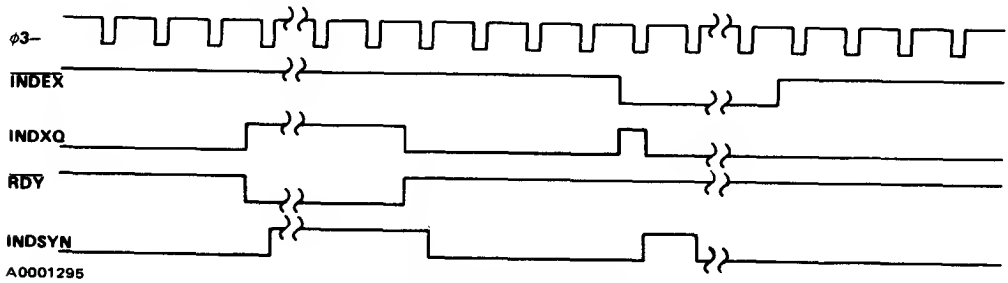


Figure 19. INDSYN Timing

3.10 READ PULSE SYNCHRONIZATION

The read-pulse synchronization logic, Figure 20, generates an active signal, BITIN, one clock cycle long each time a read pulse is detected during read operations. During write operations BITIN is maintained at a logic-one level.

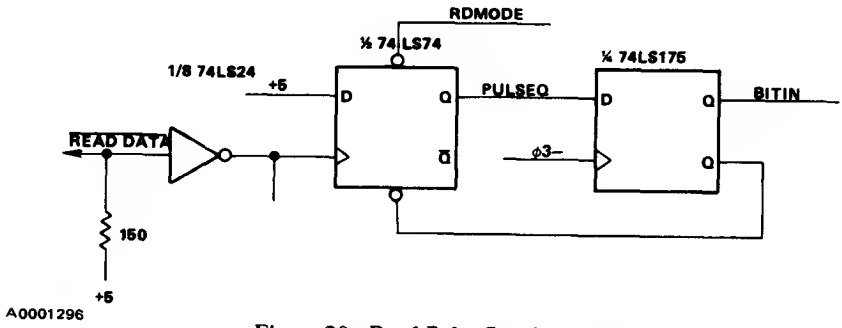


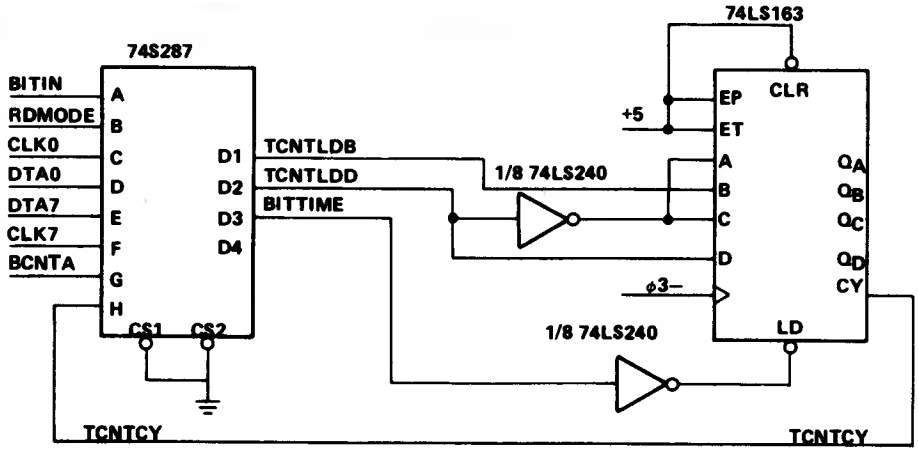
Figure 20. Read-Pulse Synchronization

3.11 BIT DETECTOR

The bit detector, Figure 21, consists of a 74LS163 counter and random logic contained in PROM. During write operations, the counter is used to time the 2 μs spacing between clock bits and data bits. During read operations the bit detector is used to determine the time interval between successive read pulses. The key signal generated by the bit detector is BITTIME, which is active for one clock cycle every 2 μs during disk writing, and which is active each time a one or zero bit is detected during read operations.

3.12 BIT COUNTER

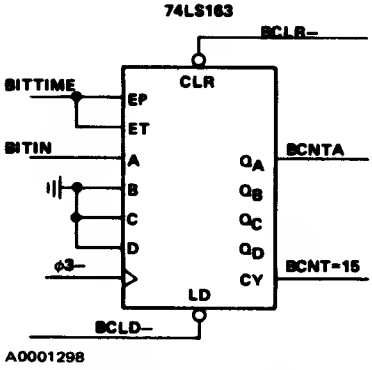
The bit counter, Figure 22, is a 74LS163 used to count the number of bits currently read or written during disk-data transfers. Each time a clock or data bit is detected or written (BITTIME = 1) the bit counter is



A0001297

Figure 21. Bit Detector

incremented. The two key outputs are BCNTA and BCNT = 15. BCNTA is the least-significant bit of the counter and is used to alternately select clock (BCNTA = 0) and data (BCNTA = 1) bits as the counter increments. BCNTA = 15 is active when a complete byte has been read or written. This signal establishes byte boundaries for the data and is used to synchronize the parallel data from the CPU to the serial-bit string and from the disk.



A0001298

Figure 22. Bit Counter

3.13 WRITE CONTROL AND DATA

Writing to the diskette is controlled by $\overline{\text{WRITE ENABLE}}$, which is the inverted and buffered WTMODE signal. WTMODE is active when a write operation has been initiated by the CPU. The $\overline{\text{WRITE DATA}}$ signal is a series of negative pulses representing FM data to be recorded on the diskette. Figure 23 illustrates write control and data.

3.14 DATA SHIFT REGISTER

The data shift register, see Figure 24, is used for accumulation of data bits during read operations and storage of data bits to be shifted out during write operations. Data is transferred to and from the CPU via the eight most-significant data lines (D0–D7). The data shift register is device type 74LS299.

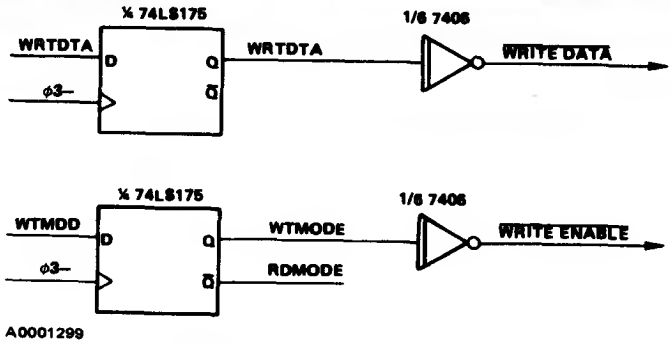


Figure 23. Write Control and Data

3.15 CLOCK SHIFT REGISTER

The clock shift register, Figure 25, is used for accumulation of clock bits during read operations and storage of clock bits to be shifted out during write operations. The clock shift register is device type 74198, which has separate parallel inputs and outputs. Three address lines, A9–A11, are connected to the parallel inputs. As data is loaded into the data shift register during write operations, these three address lines select the clock pattern for that byte (i.e., C7 for ID and data marks, D7 for track mark, FF for normal data). The parallel outputs (CLK0-CLK7) are used to detect mark clock patterns during read operations.

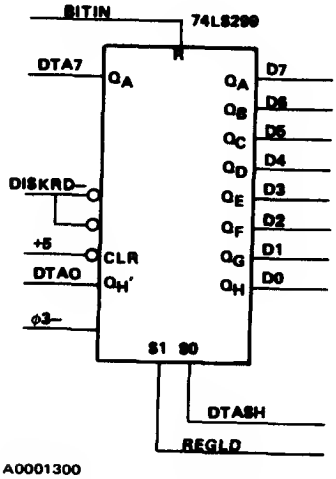


Figure 24. Data Shift Register

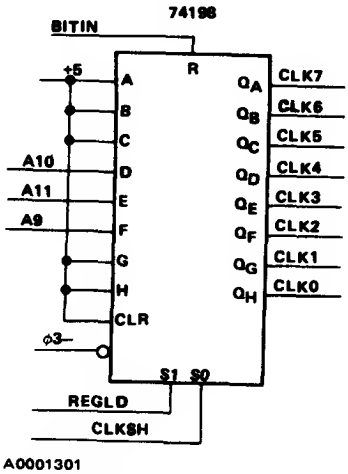


Figure 25. Clock Shift Register

SECTION IV

DISKETTE DATA TRANSFER

The previous section described the various functional blocks in the TMS 9900 floppy-disk controller. However, detailed information was not provided with respect to the logical relationships and timing of the control signal in the read/write control logic.

Most of the read/write control logic varies in function depending on the direction of transfer. This section will describe the operation of the logic separately for read and write operations. After both operations have been completely described, the combined operation will be explained.

4.1 DISK-WRITE OPERATIONS

Disk writing is initiated by executing an instruction which writes data to the data shift register (i.e., when DISKWT– = 0). When this transfer occurs, READY is held low until a byte boundary occurs (BCNT = 15), then READY becomes active, permitting completion of the write cycle. In this way, the data transfers are synchronized to the serial bit string.

To complete the transfer, READY must be active to the CPU, and the CLKSH, DTASH, and REGLD signals to the clock and data shift registers must be active to permit loading. READY = CLKSH = DTASH = REGLD = (DISKWT) (A13) (BCNT = 15) + . . .

The preceding equation indicates that the disk write must be performed with A13 = 1 for data transfer on byte boundaries. When formatting a track, the write operation must be synchronized with the index pulse, and the bit counter must be cleared regardless of its current state. When this type of write operation is to be performed, A13 must be 0.

$$\text{READY} = \text{CLKSH} = \text{DTASH} = \text{REGLD} = (\text{DISKWT})\,(\text{A13})\,(\text{BCNT} = 15) + (\text{DISKWT})\,(\text{A13–})\,(\text{INDSYN}) + \ldots$$

$$\text{BCLR–} = \overline{(\text{DISKWT})\,(\text{A13–})\,(\text{INDSYN}) + \ldots}$$

As the data byte is loaded into the data shift register, address lines A9, A10, and A11 select the clock pattern to be loaded into the clock shift register (see Table 3).

Table 3. Write Clock Patterns

A9	A10	A11	Clock Pattern
0	0	0	C7 (ID and Data Mark)
0	0	1	D7 (Track Mark)
1	1	1	FF (Normal Data)

When the transfer is complete to the clock and data shift registers, the write mode (WTMODE) flip flop is set, causing $\overline{\text{WRITE ENABLE}}$ to become active. If another byte is not written at the next byte boundary, WTMODE is reset, causing the control logic to revert to the read mode (RDMODE = 1). Also, control reverts to read mode and the bit counter is cleared when the index pulse occurs and when no write operation synchronized to the index pulse is being performed. This is useful when formatting a track, since $\overline{\text{WRITE ENABLE}}$ will automatically be turned off when the second index pulse occurs. If an index pulse occurs during a write operation with A13 = 1, the CPU proceeds, but no data transfer takes place.

WTMDD = (WTMODE) (BCNT = 15–) (INDSYN–) + (DISKWT) (A13) (BCNT = 15) + (DISKWT) (A13–) INDSYN)

BCLR– = $\overline{\text{INDSYN}}$ + . . .

READY = (DISKWT) [(A13) (BCNT = 15) + INDSYN)] + . . .

While WTMODE = 1, write data is generated by alternately shifting out bits from the clock and data shift register every two microseconds. Shifting of the clock shift register occurs when CLKSH = 1, and shifting of the data shift register when DTASH = 1. The shift is enabled by BITTIME, which is active for one clock cycle every 2 μs by loading the counter with 10_{10} each time TCNTCY = 1.

BITTIME = (WTMODE) (TCNTCY) + . . .

TCNTLDD = TCNTLDB = WTMODE + . . .

CLKSH = (DISKWT) [(A13) (BCNT = 15) + (A13–) (INDSYN)] + (WTMODE) (BCNTA–) (BITTIME) + . . .

DTASH = (DISKWT) [(A13) (BCNT = 15) + (A13–) (INDSYN)] + (WTMODE) (BCNTA) (BITTIME) + . . .

WRTDTAD = (WTMODE) (BITTIME) [(CLK0) (BCNTA–) + (DTA0) (BCNTA)]

9

On even bit counts (BCNTA = 0) clock bits are shifted, and on odd bits (BCNTA = 1) data bits are shifted, producing the desired interleaving of clock and data bits. (See Figure 26.)

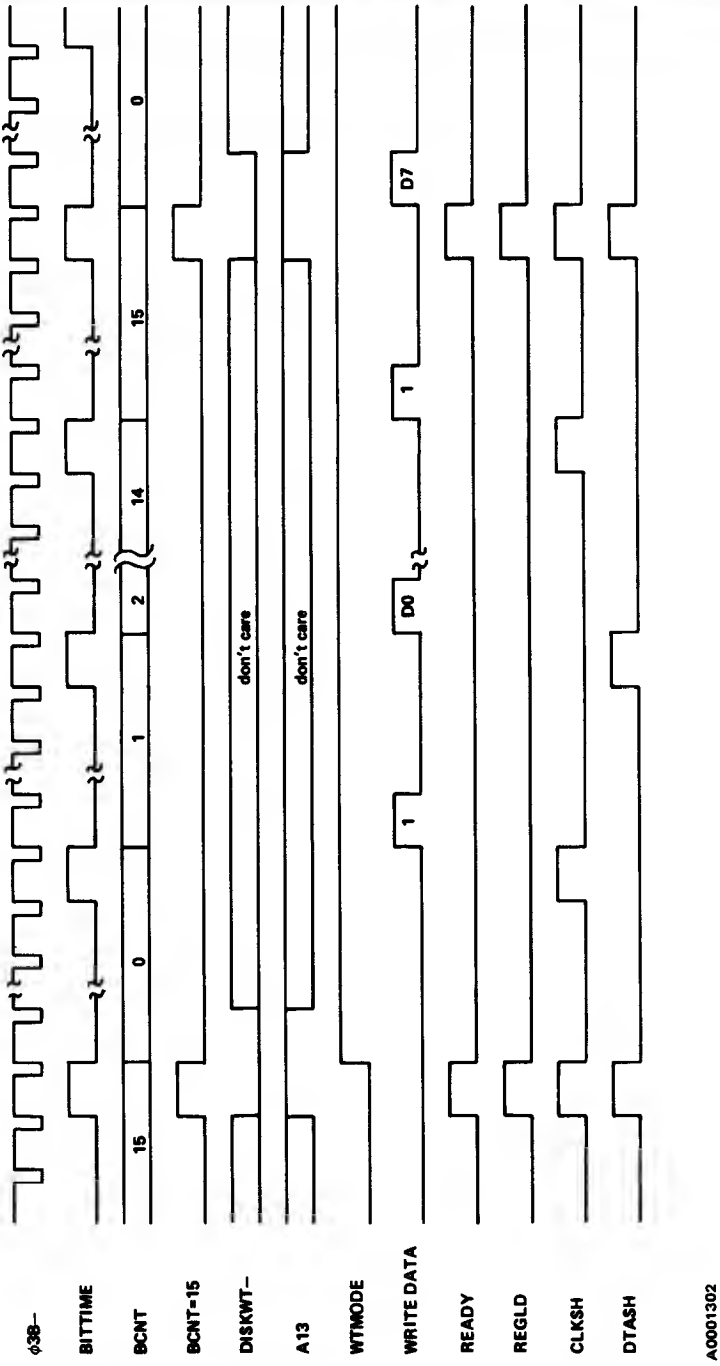


Figure 26. Write Timing

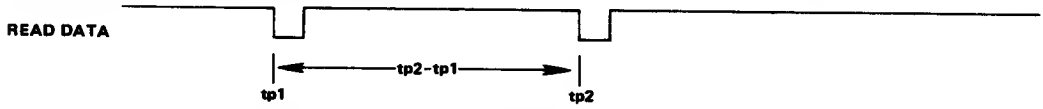
4.2 DISK READ OPERATIONS

Any time disk write operations are not being performed, the read/write control logic defaults to the read mode (RDMODE = 1). The following functions are performed to enable the CPU to read diskette data:

1. Conversion of FM to digital data;
2. Separation of clock and data bits;
3. Byte synchronization of the bit string;
4. Assembly of the seria data into bytes to be ready by CPU.

4.2.1 Clock and Data Bit Detection

Clock and data bits read from the disk are represented as a series of pulses. Each logic one clock or data bit is simply a pulse. Logic zero data and clock bits are indicated by the absence of a pulse between two pulses separated by a full data period (4 μs). Under ideal circumstances, detection of zero bits could be achieved by simply measuring the time between pulses. If $t_{P2}-t_{P1}$ = 2 μs, no zero bit is present; and if $t_{P2}-t_{P1}$ = 4 μs, a zero bit occurs between the two pulses.



Three phenomena make zero-bit detection more complex:

1. Variations in rotational speed of the disk;
2. Uncertainty of measured delays when using synchronous counters;
3. Apparent positional distortion or "bit-shifting" resulting from the tendency of pulses to move away from adjacent pulses.

Disk speed variations are typically specified at ±2% by diskette drive manufacturers. Figure 27 illustrates the bit shifting phenomenon:

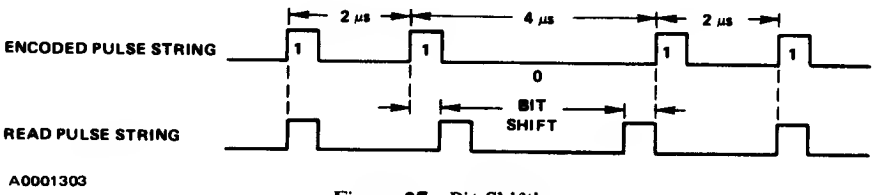


Figure 27. Bit Shifting

Pulses in the string have a tendency to move away from each other, and the closer together the pulses, the stronger the tendency to separate. A zero bit causes contiguous pulses to move toward each other, reducing pulse separation and complicating zero detection.

The bit detector is used to generate the synchronous signal BITTIME, which is active when a one or zero bit has been detected.

BITTIME = (RDMODE) (BITIN) + . . .

Detection of zero bits is accomplished by measuring the time between successive pulses. When TCNTCY = 1 and BITIN = 0, a zero bit is detected.

BITTIME = (RDMODE) (BITIN + TCNTCY) + . . .

Data and clock bits could be detected by measuring the time between read pulses, and if this time is greater than 3 μs, a zero bit is present; otherwise, no zero bit is present. Since the read pulse is asynchronous to the system, the time between pulses can only be measured to an accuracy of 333 ns (±1 clock cycle). For example, if the counter in Figure 28 is loaded with seven, no zero will be detected if the time between pulses ($t_{P2} - t_{P1}$) is less than 3.0 μs, and a zero will always be detected if $t_{P2} - t_{P1} > 3.333$ μs. If 3.0 μs $< t_{P2} - t_{P1} < 3.333$ μs, an ambiguity occurs in that a zero may or may not be detected. Similarly, if the counter is loaded with eight rather than seven, no zero bit will be detected if $t_{P2} - t_{P1} < 2.667$ μs, a zero bit will be detected if $t_{P2} - t_{P1} > 3.0$ μs, and the result is indeterminate if 2.667 μs $< t_{P2} - t_{P1} < 3.0$ μs. Most floppy-disk drive manufacturers specify that the maximum shift for any bit is 500 ns. Thus, two consecutive 1 bits may be separated by nearly 3.0 μs, and two 1 bits separated by a zero bit may shift toward each other to result in a minimum separation of nearly 3.0 μs. The combined distortion of consecutive 1 bits never fully reaches 1 μs, but the 667 ns margin provided by loading the counter with either seven or eight does not provide for reliable, accurate reading of data. (See Figure 28.)

As stated previously, adjacent 1 bits affect the direction of distortion of a particular 1 bit, with the closest pulses having the greatest effect. Empirical observation indicates that only the two bit positions on either side of a pulse have significant effect on a pulse, as shown in Table 4.

Table 4. Bit Shift Direction

Bit n-2	Bit n-1	Bit n	Direction of Distortion For Bit n	Bit n+1	Bit n+2
0	1	1	→	0	1
0	1	1	–	1	0
0	1	1	←	1	1
1	0	1	–	0	1
1	0	1	←	1	0
1	0	1	←	1	1
1	1	1	→	0	1
1	1	1	→	1	0
1	1	1	–	1	1

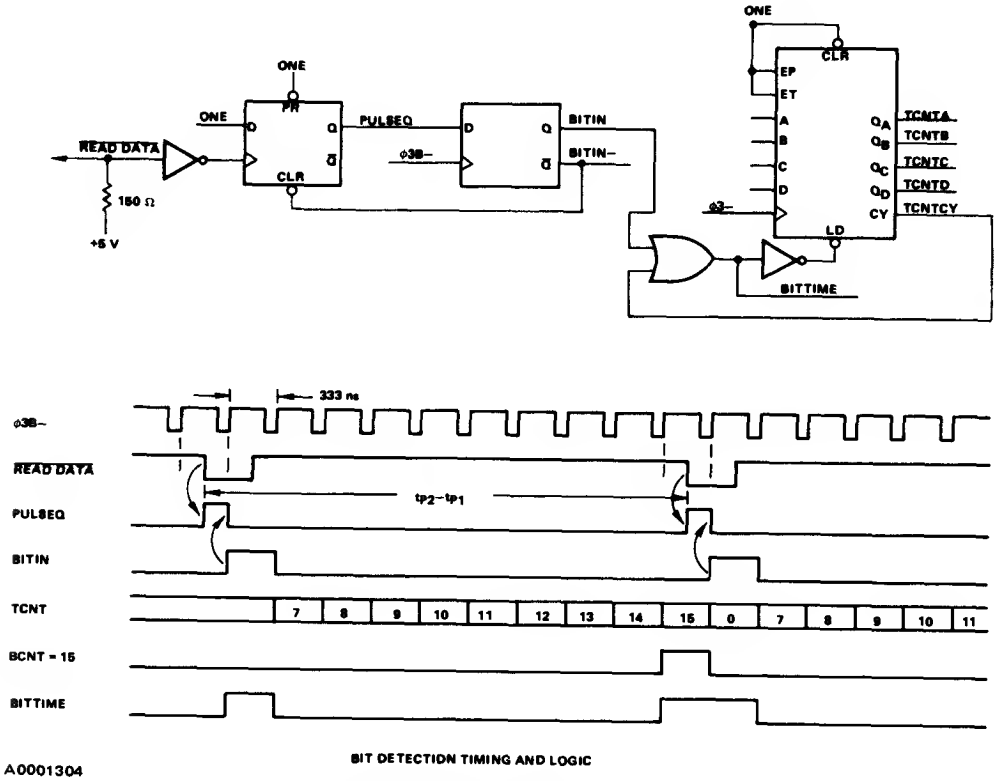


Figure 28. Bit Detection Timing and Logic

The most difficult detection problem is that of differentiating between two contiguous 1 bits which are shifted away from each other (worst case 11) and two 1 bits separated by a zero bit where the 1 bits move toward each other (worst case 101). The worst case 11 occurs in the patterns

			←	→			
Pattern A	0	1	1	1	1	0	, and
Pattern B	1	0	1	1	0	1	.
			→	←			

The worst case 101 occurs in the patterns

			→		←		
Pattern C	0	1	1	0	1	1	, and
Pattern D	1	1	1	0	1	1	.
			→		←		

The timing logic is such that the period of uncertainty does not lie in the area where a severely distorted pulse will occur; that is, when the worst case 11 can occur, and $t_{P2} - t_{P1} < 3.0\ \mu s$, the logic always

indicates that no zero was detected; when the worst case 101 can occur and $t_{P2} - t_{P1} > 3.0\ \mu s$, a zero is always detected. To accomplish this, the value loaded into the counter is shown in Table 5.

Table 5. Worst Case Pattern Load Values

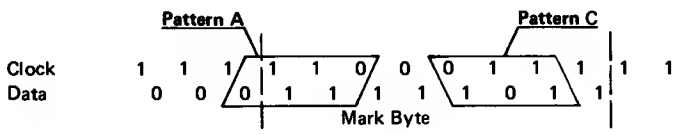
Pattern	Bit n−2	Bit n−1	Bit n	Bit n+1	Bit n+2	Bit n+3	Load Value
A	0	1	1	1	1	0	7
B	1	0	1	1	0	1	7
C	0	1	1	0	1	1	8
D	1	1	1	0	1	1	8

When bit n is detected, the counter is loaded with the value shown, dependent upon the data pattern.

Accommodation of patterns B and D are simple, since bits following that being sampled don't matter. Patterns A and C present the problem that, as the serial pulses are being read, the logic does not know what bits n+1, n+2, and n+3 are going to be.

Further analysis of the data format reveals that patterns A and C occur only when an ID or data mark are being read, see Table 6.

Table 6. Data Mark



Pattern A can only occur at the beginning of an ID, data, or deleted data mark, and pattern C can only occur in a data mark. With pattern A, the first 0 is a data bit, and with pattern C, the first 0 is a clock bit. BCNTA selects whether the current 1 bit is to be shifted into the clock or data shift register. The previous two bits are CLK7 and DTA7, the LSB's of the clock and data shift registers, and the order of these bits is determined by BCNTA. Using this information, the values loaded into the counter are as shown in Table 7.

TCNTLDD = (RDMODE)](CLK7) (DTA7) + (BCNTA−) (DTA7)] + . . .

TCNTLDB = (RDMODE) [(DTA7−) + (BCNTA) (CLK7−)] + . . .

The bit detector will thus adjust its count interval to accommodate the worst-case distortion which can occur for the anticipated data pattern.

Table 7. Bit Detector Counter Load Values

BCNTA	CLK7	DTA7	Load Value
0	0	0	Illegal
0	0	1	8
0	1	1	8
0	1	0	7
1	1	0	7
1	1	1	8
1	0	1	7
1	0	0	Illegal

4.2.2 Clock/Data Separation

Each time BITTIME is active, a new clock or data bit is shifted in. The value of the clock or data bit is BITIN. Since clock and data bits are interleaved, the value of BITIN will be alternately shifted into the clock or data shift register each time BITTIME is active. This is accomplished by incrementing the bit counter each time BITTIME is active, causing BCNTA to toggle. The equations for shifting the clock and data shift registers are:

CLKSH = (BITTIME) (BCNTA–) (RDMODE)+ . . .

DTASH = (BITTIME) (BCNTA) (RDMODE) + . . .

When four consecutive zeroes are detected in the clock shift register, the order in which bits go to the clock and data shift registers is reversed, since four consecutive zero clock bits never occur in the recording format used. This is accomplished by the control signal:

$$BCLD- = \overline{(CLK4-)\ (CLK5-)\ (CLK6-)\ (CLK7-)}.$$

When this signals becomes active, the bit counter is cleared to zero, and remains cleared until the next 1 bit is detected. This 1 bit is directed to the clock shift register, causing BCLD– to become inactive and normal operation is resumed. Synchronization is thus assured at the beginning of each ID and data field because each field is preceded by several bytes with all zero data bits and all one clock bits.

The timing for clock/data separation is shown in Figure 29.

4.2.3 Byte Synchronization

9 Initial byte synchronization is achieved when reading an ID or data field by detecting the unique clock pattern of $C7_{16}$ which occurs only in ID and data marks. The mark detect signal is expressed by the equation:

MRKDT = (CLK0) (CLK1) (CLK2–) (CLK3–) (CLK4–) (CLK5) (CLK6) (CLK7)

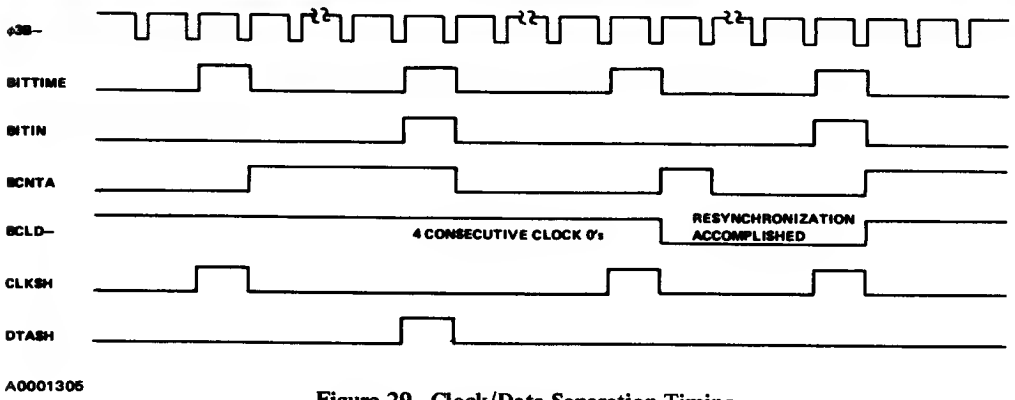


Figure 29. Clock/Data Separation Timing

After the mark is detected, one additional BITTIME must occur, allowing the data bit to be shifted into the data shift register.

4.2.4 Reading Disk Data

Two types of disk reads may be performed. When reading an ID or data field, the first byte read is always the ID or data mark. This is accomplished by performing a disk read with A13 = 0. The READY input signal will not become active until MRKDT = 1 and BITTIME = 1. After the mark is read, byte synchronization is established and subsequent disk reads are performed with A13 = 1. In this case, READY becomes true at each byte boundary when BCNT = 15.

READY = (DSKRD) [(BCNTA) (MRKDT) (BITTIME) (A13–) + (BCNT = 15) (A13) + INDSYN] + . . .

The addresses for the two types of disk reads are $7FF8_{16}$ for reading marks, and $7FFC_{16}$ for reading normal data. The INDSYN term of the above equation causes the read operation to be completed any time the index pulse is detected or when the disk becomes not ready. (See Figure 30.)

4.3 READ/WRITE LOGIC COMBINATION

This subsection summarizes the equations for the control lines resulting from the combination of the read and write control functions.

BCLD–

$$\text{BCLD–} = \overline{(\text{CLK4–})\,(\text{CLK5–})\,(\text{CLK6–})\,(\text{CLK7–})}$$

BCLR–

$$\text{BCLR–} = \overline{(\text{RDMODE})\,(\text{MRKDT})\,(\text{BCNTA})\,(\text{BITTIME}) + (\text{INDSYN})}$$

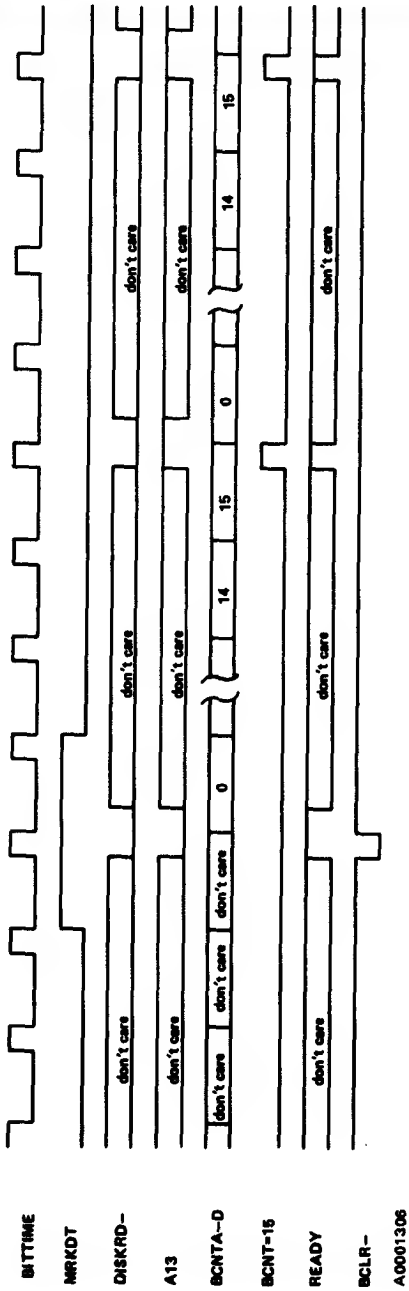
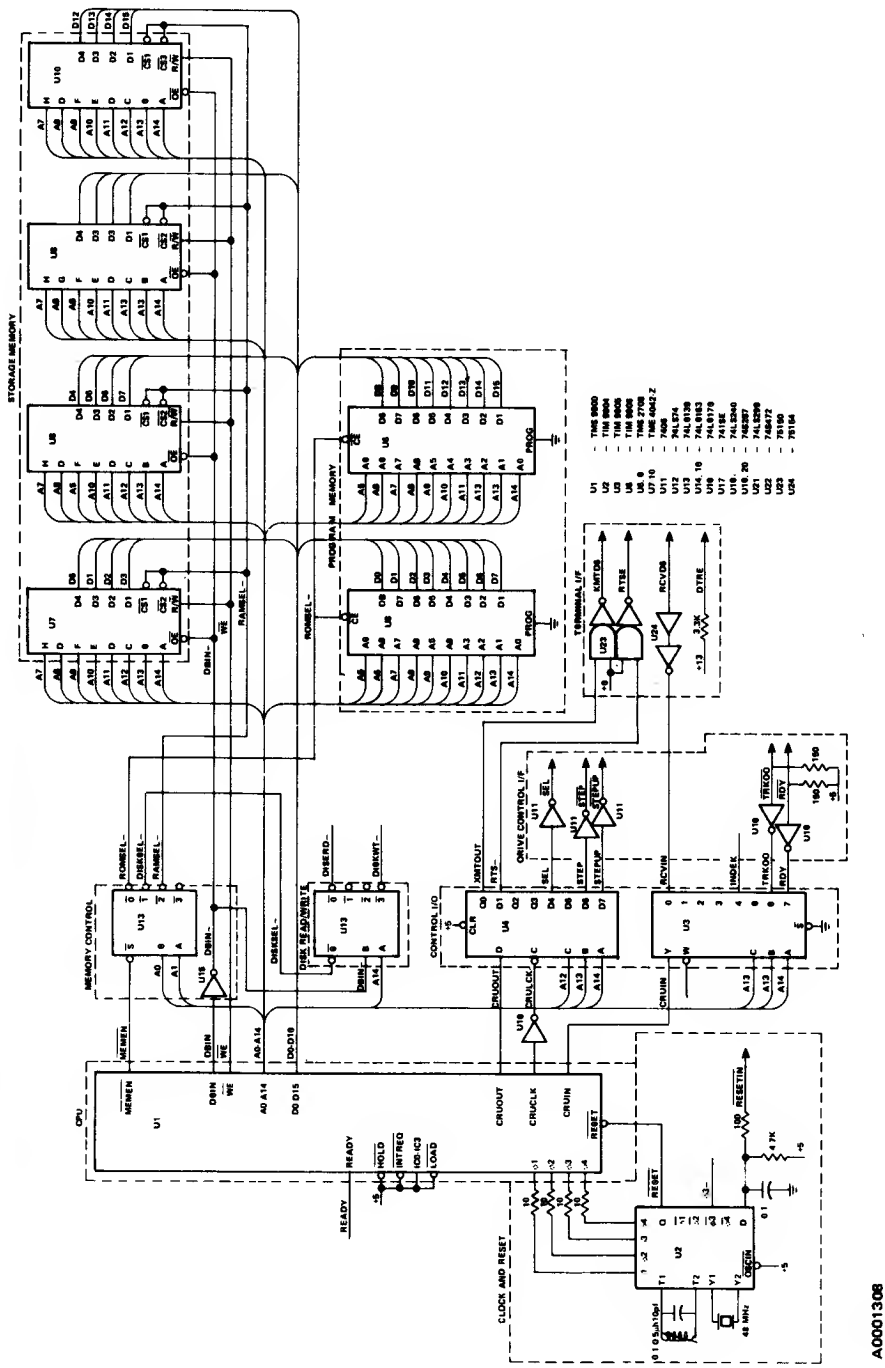
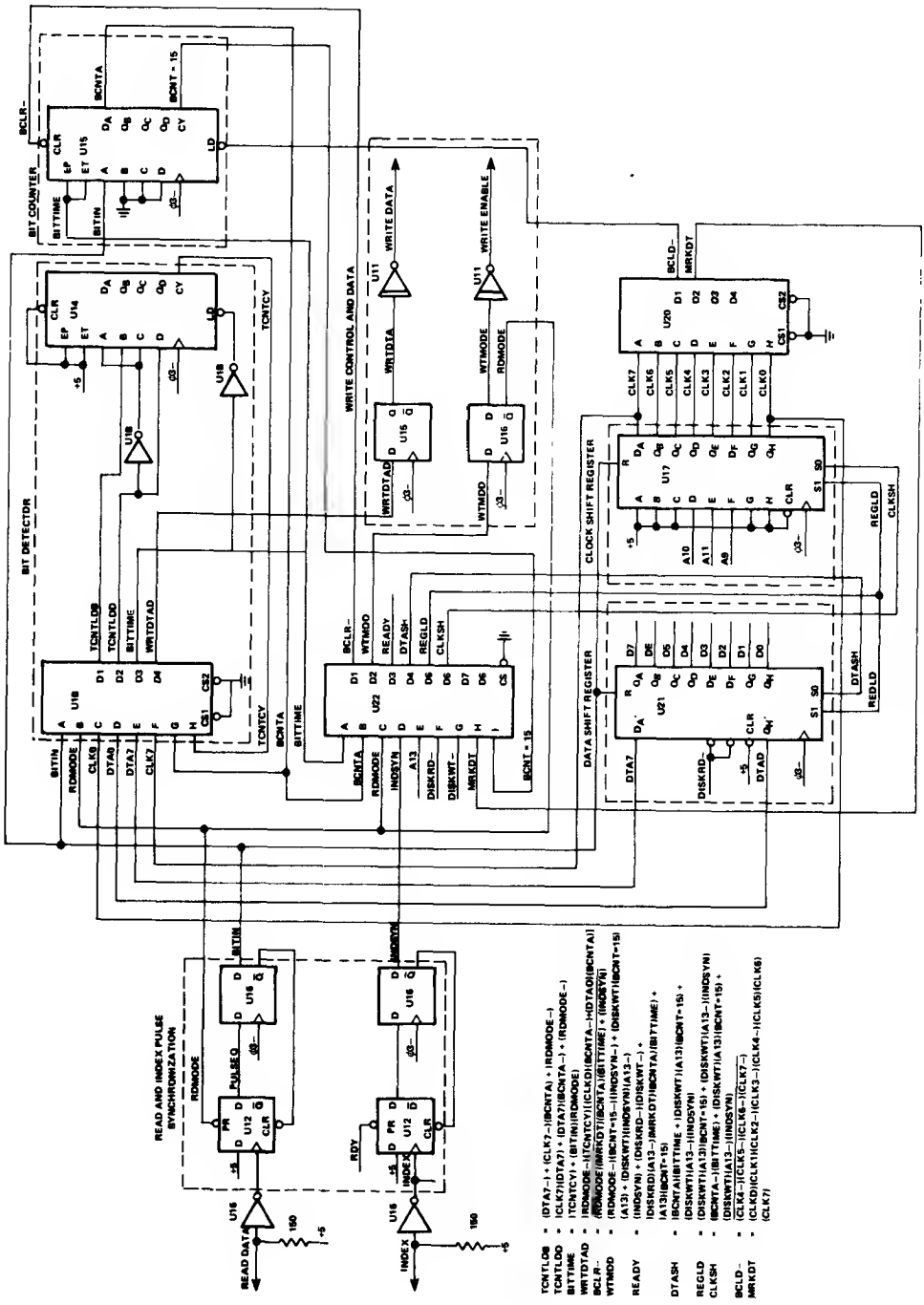


Figure 30. Disk Read Timing



Logic Diagram, TMS 9900 Floppy Disk Controller
(Sheet 1 of 2)



Logic Diagram, TMS 9900 Floppy Disk Controller (Sheet 2 of 2)

BITTIME

BITTIME = (WTMODE) (TCNTCY) + (RDMODE) [(BITIN) + (TCNTCY)]
= (TCNTCY) + (RDMODE) (BITIN)

CLKSH

CLKSH = (DISKWT) [(AI3) (BCNT = 15) + (AI3–) (INDSYN)] + (WTMODE) (BCNTA–) (BITTIME) + (RDMODE) (BCNTA–) (BITTIME)
= (DISKWT) [(AI3) (BCNT = I5) + (AI3–) (INDSYN)] + (BCNTA–) (BITTIME)

DTASH

DTASH = (DISKWT) [(AI3) (BCNT = 15) + (AI3–) (INDSYN)] + (WTMODE) (BCNTA) (BITTIME) + (RDMODE) (BCNTA) (BITTIME)
= (DISKWT) [(A13) (BCNT = 15) + (AI3–) (INDSYN)] + (BCNTA) (BITTIME)

MRKDT

MRKDT = (CLK0) (CLKI) (CLK2–) (CLK3–) (CLK4–) (CLK5) (CLK6) (CLK7)

READY

READY = (DISKWT) [(AI3) (BCNT = 15) + (INDSYN)] + (DISKWT–) (DISKRD–) + (DISKRD) [(A13) (BCNT = 15) + (INDSYN) + (A13–) (MRKDT) (BCNTA) (BITTIME)]
= (DISKWT–) (DISKRD–) + (AI3) (BCNT = I5) + (INDSYN) + (DISKRD) (AI3–) (MRKDT) (BCNTA) (BITTIME)

REGLD

REGLD = (DISKWT) [(AI3) (BCNT = 15) + (AI3–) (INDSYN)]

TCNTLDB

TCNTLDB = (WTMODE) + (RDMODE) [(DTA7–) + (BCNTA) (CLK7–)]
= (WTMODE) + (DTA7–) + (BCNTA) (CLK7–)

TCNTLDD

TCNTLDD = (WTMODE) + (RDMODE) [(CLK7) (DTA7) + (BCNTA–) (DTA7)]
= (WTMODE) + (CLK7) (DTA7) + (BCNTA–) (DTA7)

WRTDTAD

WRTDTAD = (WTMODE) (BITTIME) [(CLK0) (BCNTA–) +(DTA0) (BCNTA)]
= (WTMODE) (TCNTCY) [(CLK0) (BCNTA–) + (DTA0) (BCNTA)]

WTMDD

WTMDD = (WTMODE) (BCNT = 15–) (INDSYN–) + (DISKWT) [(AI3) (BCNT = I5) + (A13–) (INDSYN)]

9◂

SECTION V

SOFTWARE

The software design of a microprocessor system is as important as its hardware design. In this system, several functions which are normally performed by hardware are instead done in software in order to reduce device count. Examples of hardware/software tradeoffs include timing, transmit/receive, and CRC calculation.

5.1 SOFTWARE INTERFACE SUMMARY

The memory map in Figure 31 shows the memory address assignments for program memory, storage memory and the floppy-disk interface.

The CRU bit address assignments are summarized in Table 8 below.

Table 8. CRU Address Assignments

Bit Address	Output	Input
0	XMTOUT	RCVIN
1	RTS–	
2		
3		
4	SEL	INDEX
5		
6	STEP	TRK00
7	STEPUP	RDY

5.2 CONTROL SOFTWARE

Rather than providing individual examples of each individual control and data transfer function, all of the functions are combined to demonstrate complete system operation. The control software is modular, and the various subroutines may easily be adapted to different configurations of a TMS 9900 floppy-disk controller.

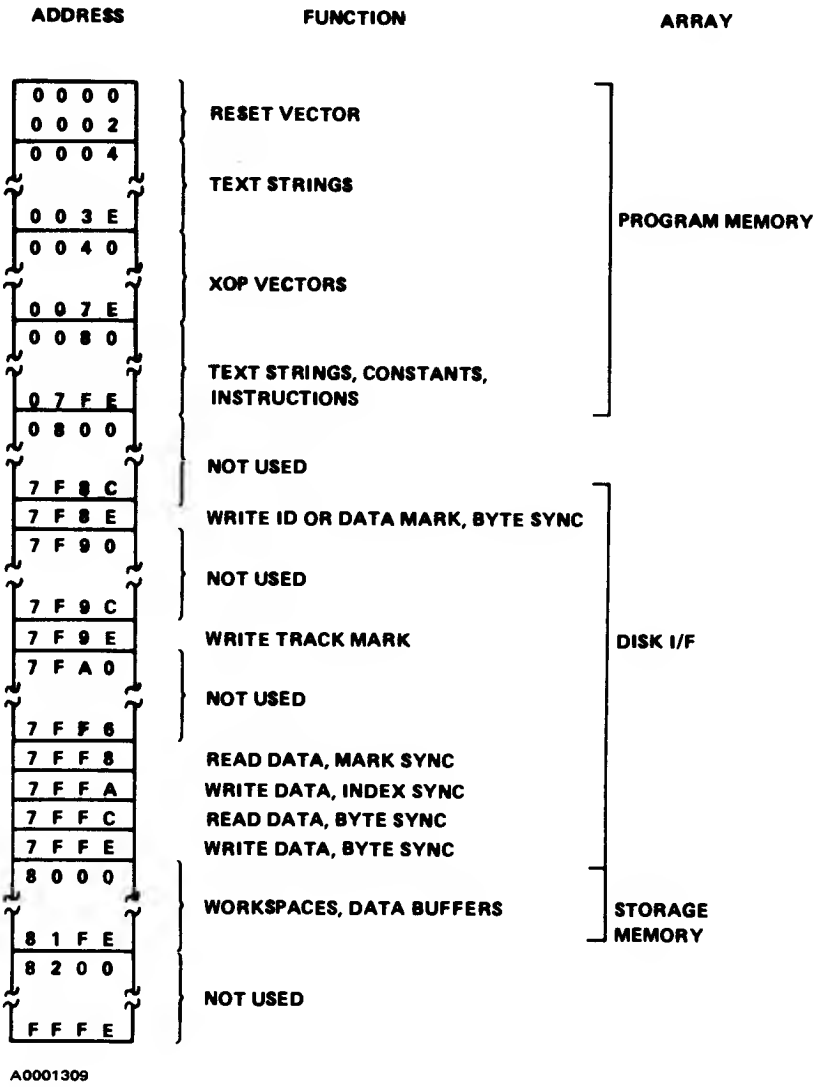


Figure 31. Memory Address Assignments

5.2.1 Floppy-Disk Control Program

This program contains the complete software for interfacing the TMS 9900 floppy-disk controller to both the RS-232 terminal and the floppy-disk drive.

5.2.2 Operator Commands

The commands listed in Table 9 are available to the terminal operator. These commands enable the user to write and read data to and from the diskette, format tracks, display and enter data from memory, and execute from a selected address. The user is able to load and execute diagnostics in addition to performing normal data transfer operations. When errors are encountered, error information is reported at the terminal.

Table 9. Operator Commands

?WA	TRACK = ct st,	SECTOR = cs ss,	NUMBER = sn
?WH	TRACK = ct st,	SECTOR = cs ss,	NUMBER = sn
?WD	TRACK = ct st,	SECTOR = cs ss,	NUMBER = sn
?RA	TRACK = ct st,	SECTOR = cs ss,	NUMBER = sn
?RH	TRACK = ct st,	SECTOR = cs ss,	NUMBER = sn
?FM	TRACK = ct st	END TRACK = st et	
?MD	sadd eadd		
?ME	sadd		
?MX	sadd		

Underscored characters are entered by the user. All others are supplied by the controller. The lower case fields are hexadecimal values. If the users enters a blank into these fields, the default value is used by the controller. Entry of any non-printable character (e.g., Carriage Return, ESCape) during command entry causes the command to be aborted. Entry of a non-hexadecimal value in hexadecimal fields causes the command to be aborted.

Table 10 lists the command entry parameters and Table 11 gives a summary of the commands.

Table 10. Command Entry Parameters

Parameter	Definition	Default Value	Range
ct	Current track number	–	$00 \leq ct \leq 4C$ (76_{10})
st	Starting track number	ct	$00 \leq st \leq 4C$
cs	Current sector number	–	$01 \leq cs \leq 1A$ (26_{10})
ss	Starting sector number	cs	$01 \leq ss \leq 1A$
sn	Number of sectors	01	$01 \leq sn \leq FF$ (255_{10})
et	Ending track number	st	$st \leq et \leq 4C$
sadd	Starting address	8000	$0 \leq sadd \leq FFFF$
eadd	Ending address	sadd	$0 \leq eadd \leq FFFF$

Table 11. Command Summary

Command	Description
WA	Write ASCII. The ASCII character strings entered by the user are written sequentially onto the diskette. Each sector may be terminated, filling remaining bytes with 00, by entry of any non-printable character. (ASCII code $< 20_{16}$) other than ESCape. Entry of ESCape aborts the command.
WH	Write Hexadecimal. Hexadecimal bytes entered by the user are written sequentially onto the diskette. Sector termination and abort are performed in the same way as for the WA command.
WD	Write Deleted Data. Same as WH command, except the Deleted Data Mark (Clock = $C7_{16}$ Data = $F8_{16}$) rather than the Data Mark (Clock = $C7_{16}$, Data = FB_{16}) is written at the beginning of the Data Field.
RA	Read ASCII. The specified sectors are read and printed out as ASCII character strings. Each sector is printed beginning at a new line, and printing continues until the end of the sector, or until a non-printable ASCII character is encountered. When more than 80 characters are printed, the controller prints the eighty-first character in the first position of the next line. The command may be aborted at the end of any sector by depressing the 8REAK key before the last character of the sector is printed. If a Deleted Data field is encountered, it is reported, and normal operation continues.
RH	Read Hexadecimal. The specified sectors are read and printed out as hexadecimal bytes, 16 bytes per line. The command may be aborted by depressing the 8REAK key before the last character of any line is printed. If a Deleted Data field is encountered, it is reported and normal operation continues.
FM	Format Track. The specified tracks are completely rewritten with gaps, Track Marks, ID fields, and Data fields. All zero data is written into the 128 bytes of the data field.
MD	Memory Display. The contents of the specified memory addresses are printed out in hexadecimal byte format. The address of the first word of each line is printed, followed by 16 bytes. The command may be aborted by depressing the 8REAK key before the last character of any line is printed.
ME	Memory Enter. Beginning with the selected location, the memory address and contents are printed. If it is to be modified, the user enters a hexadecimal byte value which will be stored at that address. If the value is not to be changed, the user enters a blank character (SPACE bar). The address is then incremented and the process is repeated until a non-hex character is entered, terminating the command.
MX	The CPU begins execution at the selected memory location.

Figure 32 shows the control software for the system described in this application report.

9◀

SECTION VI

SUMMARY

This application report has provided a thorough discussion of the TMS 9900 floppy-disk controller hardware and software system design. The economy of the CRU and the high throughput capability of the memory bus result in an economical, powerful system. The memory-to-memory architecture of the TMS 9900, along with its powerful instruction set and addressing capability, make the TMS 9900 ideally suited for applications where large amounts of data manipulation are necessary. Also, software development time is optimized by the minimization of lines of code resulting from the memory-to-memory instructions and large number of working registers.

It is likely that the designer using this application report will have requirements that are not addressed in this design. Variations in the sector length are accommodated with slight software modification. Higher density recording formats such as MFM and M^2FM require changes in the bit detector and data-separation logic. Higher throughput can be achieved by using an LSI terminal interface such as the TMS 9902 asynchronous communication controller and hardware CRC generation. Controlling multiple disks requires only the addition of drive select control lines. In short, variations on this design are easily implemented through slight hardware and software modifications.

```
FLOPPY DISK CONTROL PROGRAM                                    PAGE 0001

                        IDT   'FDCTRL'
                ****************************************************
                *
                *            FLOPPY DISK CONTROL PROGRAM
                *
                *            DECEMBER 21, 1976
                *
                *       THIS PROGRAM CONTAINS THE CONTROL SOFTWARE FOR THE
                *       SYSTEM DESCRIBED IN THE "TMS 9900 FLOPPY DISK
                *       CONTROL SYSTEM" APPLICATION REPORT.  THE PROGRAM
                *       ALLOWS THE USER TO READ, WRITE, AND FORMAT DATA ON
                *       FLOPPY DISK.  ADDITIONALLY, THE USER MAY ENTER,
                *       DISPLAY, AND INITIATE EXECUTION FROM
                *       ANY LOCATION IN MEMORY.  IT IS ASSUMED THAT THE
                *       CONSOLE USED FOR COMMAND ENTRY AND DATA DISPLAY IS
                *       A 300 BAUD, RS-232C TYPE TERMINAL.  THE COMMANDS
                *       USED IN INTERFACING THE TERMINAL OPERATOR
                *       INTERFACE TO THE CONTROLLER ARE FULLY
                *       DESCRIBED IN SECTION 5.3 OF THE "TMS 9900
                *       FLOPPY DISK CONTROL SYSTEM" APPLICATION
                *       REPORT.
                *
                ****************************************************
                *
                *       DISK TRANSFER ADDRESSES
                *
        7F8E    MRKWT   EQU   >7F8E            DATA MARK WRITE
        7F9E    TKMWT   EQU   >7F9E            TRACK MARK WRITE
        7FFA    INDXWT  EQU   >7FFA            INDEX SYNC WRITE
        7FFE    DTAWT   EQU   >7FFE            BYTE SYNC WRITE
        7FF8    MRKRD   EQU   >7FF8            MARK SYNC READ
        7FFC    DTARD   EQU   >7FFC            BYTE SYNC READ
                ****************************************************
                *
                *       RAM EQUATES
                *
        80C0    SECBUF  EQU   >80C0            CRC BUFFER FOR FORMATTING
        80F7    IDFLD   EQU   >80F7            ID FIELD IMAGE
        80F8    TKNUM   EQU   >80F8            TRACK NUMBER
        80FA    SECNUM  EQU   >80FA            SECTOR NUMBER
        80FC    IDCRC   EQU   >80FC            CRC FOR ID FIELD
        80FF    DTAFLD  EQU   >80FF            DATA FIELD IMAGE
        8100    DTABUF  EQU   >8100            128 BYTE DATA BUFFER
        8180    DTACRC  EQU   >8180            CRC FOR DATA FIELD
        8170    FDWP5   EQU   >8170            WORKSPACE 5
        8190    FDWP4   EQU   >8190            WORKSPACE 4
        81A0    FDWP3   EQU   >81A0            WORKSPACE 3
        81C0    FDWP2   EQU   >81C0            WORKSPACE 2
        81E0    FDWP1   EQU   >81E0            WORKSPACE 1
```

9◀

Figure 32. Floppy Disk Control Program (Sheet 1 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0002

                    ♦♦♦
```

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0003

 0101                 ◆◆◆
```

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0004

                    ********************************************************
                    *
                    *       ASCII VALUES
                    *
0080    41  ASCIIA TEXT 'A'
0081    46  ASCIIF TEXT 'F'
0082    4D  ASCIIM TEXT 'M'
0083    1B  ESC    BYTE >1B
0084    20  BLANK  BYTE >20
0085    3F  QUEST  BYTE >3F
0086    07  BELL   BYTE >07
0087    08  BACKSP BYTE >08
0088    0D  CARRET BYTE >0D
0089    0A  LINEFD BYTE >0A
                    ********************************************************
                    *
                    *       ADDITIONAL TEXT MESSAGES
                    *
008A    20  TKMSG  TEXT '  TRACK ='
0093    00         BYTE 0
0094    20  ENDMSG TEXT '  END'
0099    00         BYTE 0
009A    2C  SCTMSG TEXT ', SECTOR ='
00A4    00         BYTE 0
00A5    2C  NUMMSG TEXT ', NUMBER ='
00AF    00         BYTE 0
00B0    20  ADDMSG TEXT '  ADDRESS ='
00BB    00         BYTE 0
00BC    44  DLDMSG TEXT 'DELETED DATA FIELD'
00CE    00         BYTE 0
                    ********************************************************
                    *
                    *       DISK MARK CONSTANTS
                    *
00CF    F8  DLDMRK BYTE >F8
00D0    FE  IDMRK  BYTE >FE
00D1    FB  DTMRK  BYTE >FB
00D2    FC  TKMRK  BYTE >FC
                       EVEN
                    ********************************************************
                    *
                    *       SUBROUTINE:  DLAY
                    *
                    *       CALLING SEQUENCE:  DLAY @COUNT
                    *
                    *       A SOFTWARE LOOP WILL BE EXECUTED THE NUMBER
                    *       OF TIMES SPECIFIED BY THE CALLING PROGRAM.
                    *       EACH ITERATION OF THE LOOP RESULTS IN A
                    *       DELAY OF 6.67 MICROSECONDS.
                    *
00D4  060B  DLAYPC DEC  R11                 DECREMENT COUNT
007E**00D4'
00D6  16FE         JNE  DLAYPC              LOOP IF NOT 0
00D8  0380         RTWP                     RETURN
```

Figure 32. Floppy Disk Control Program (Sheet 4 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0005

                    ◆◆◆
```

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0006

                    ********************************************************
                    *
                    *       SUBROUTINE:  XMIT
                    *
                    *       CALLING SEQUENCE:  XMIT @LOCATN
                    *
                    *       AN ASCII CHARACTER WITH CORRECT FORMATTING
                    *       AND EVEN PARITY IS TRANSMITTED AT 300 BAUD.
                    *       THE LOCATION OF THE CHARACTER TO BE TRANS-
                    *       MITTED IS SPECIFIED AS THE CALLING PARAMETER.
                    *
0106  04CC  XMITPC CLR  R12                 INITIALIZE CRU BASE
007A**0106'
0108  020A          LI   R10,3              INITIALIZE ACCUMULATOR
010A  0003
010C  0209          LI   R9,>8000           INITIALIZE PARITY MASK
010E  8000
0110  D29B          MOVB *R11,R10           FETCH CHARACTER
0112  1C--          JOP  PARADJ             IF ODD PARITY INVERT MSB
0114  04C9          CLR  R9                 ELSE, CLEAR PARITY MASK
0116  2A89  PARADJ XOR  R9,R10              XOR PARITY MASK
0112**1C01
                    *                       WITH CHARACTER
0118  1E01          SBZ RTS                 TURN ON RTS
011A  0B8A          SRC  R10,8              ROTATE CHARACTER
011C  18--  XMTLP1 JOC  XOUTON              TEST TRANSMIT BIT
011E  1E00          SBZ  XOUT               IF 0, RESET XOUT
0120  10--          JMP  XFBDLY             AND SKIP
0122  1D00  XOUTON SBO  XOUT                ELSE, SET XOUT
011C**1802
0124  2FE0  XFBDLY DLAY @FBDLY              DELAY FULL BIT TIME
0126  01F4
0120**1001
0128  091A          SRL  R10,1              SHIFT ACCUMULATOR 1 BIT
012A  16F8          JNE XMTLP1              IF NOT ZERO, TRANSMIT NEXT BIT
012C  1D01          SBO  RTS                TURN OFF RTS
012E  0380          RTWP                    RETURN
```

Figure 32. Floppy Disk Control Program (Sheet 6 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0007

 0250                 ♦♦♦
```

```
FLOPPY DISK CONTROL PROGRAM                                    PAGE 0008

         0152  0083'
 0297    0154  16--          JNE  NOTESC          IF NOT,CONTINUE
 0298    0156  0420          BLWP @RTRNVC         ELSE, ABORT COMMAND
         0158  0142'
 0299    015A  9809  NOTESC  CB   R9,@BLANK       COMPARE TO BLANK
         015C  0084'
         0154**1602
 0300    015E  13--          JEQ  HRC2RT          IF = BLANK, RETURN
 0301                 *                           ELSE, CONVERT TO HEXADECIMAL
 0302    0160  0229          AI   R9,->3000       SUBTRACT ASCII BIAS
         0162  D000
 0303    0164  11--          JLT  HRC2AB          IF LESS THAN >30, ABORT
 0304    0166  0289          CI   R9,>A00         TEST FOR NUMERIC
         0168  0A00
 0305    016A  11--          JLT  NOHAJ           IF NUMERIC, SKIP
 0306    016C  0229          AI   R9,->700        ELSE, SUBTRACT ALPHA BIAS
         016E  F900
 0307    0170  0289          CI   R9,>A00         IF LESS THAN >41, ABORT
         0172  0A00
 0308    0174  11--          JLT  HRC2AB
 0309    0176  0289          CI   R9,>FFF         COMPARE TO ASCII F
         0178  0FFF
 0310    017A  15--          JGT  HRC2AB          IF GREATER THAN, ABORT
 0311    017C  F289  NOHAJ   SOCB R9,R10          STORE HEX VALUE IN
         016A**1108
 0312                 *                           ACCUMULATOR
 0313    017E  0588          INC  R8              INCREMENT CHARACTER COUNT
 0314    0180  16--          JNE  HRC2ND          IF NOT 0, SKIP
 0315    0182  0A4A          SLA  R10,4           SHIFT HEX ACCUMULATOR
 0316    0184  10E4          JMP  HRC2LP          FETCH SECOND CHARACTER
 0317    0186  D6CA  HRC2ND  MOVB R10,*R11        STORE HEX VALUE
         0180**1602
 0318                 *                           AT SPECIFIED LOCATION
 0319    0188  0380  HRC2RT  RTWP                 RETURN
         015E**1314
 0320    018A  C3AD  HRC2AB  MOV  @20(R13),R14    MODIFY RETURN PC
         018C  0014
         0164**1112
         0174**110A
         017A**1507
 0321    018E  10FC          JMP  HRC2RT          RETURN
```

▶9

Figure 32. Floppy Disk Control Program (Sheet 8 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                    PAGE 0009

0323                  ♦♦♦
```

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0010

                  ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
                  ◆
                  ◆       SUBROUTINE:   IDRD
                  ◆
                  ◆       CALLING SEQUENCE:   IDRD 0
                  ◆
                  ◆       EACH ID FIELD OF THE CURRENT DISKETTE TRACK
                  ◆       IS READ UNTIL THE ID FIELD WITH THE CORRECT
                  ◆       TRACK, SECTOR, AND CRC IS FOUND, AT WHICH
                  ◆       TIME THE ROUTINE IS EXITED.  IF THE CORRECT
                  ◆       FIELD IS NOT FOUND WITHIN A COMPLETE DISK
                  ◆       REVOLUTION (IE BEFORE 2 INDEX PULSES ARE
                  ◆       DETECTED), THE OPERATION IS ABORTED AND AN
                  ◆       ERROR MESSAGE IS REPORTED.
                  ◆
  01C6  2DC0  IDRDPC CRCI 0                    UPDATE ID FIELD IMAGE CRC
  004A◆◆01C6
  01C8  2D40         DSON 0                    TURN ON DRIVE
  01CA  0209         LI   R9,2                 INITIALIZE INDEX PULSE COUNT
  01CC  0002
  01CE  020A  IDMRD  LI R10,IDFLD              SET POINTER TO ID FIELD
  01D0  80F7
  01D2  9EA0         CB   @MPKRD,◆R10+         COMPARE DISK BYTE TO
  01D4  7FF8
                  ◆                            MARK CHARACTER
  01D6  13--         JEQ  MRKFND               IF MARK, CONTINUE
  01D8  1F04         TB   INDEX                ELSE, TEST FOR INDEX SIGNAL
  01DA  16F9         JNE  IDMRD                IF NO INDEX, REREAD DISK
  01DC  0609         DEC  R9                   IF INDEX,DECREMENT INDEX COUNT
  01DE  16F7         JNE  IDMRD                IF NOT 0, REREAD DISK
  01E0  2C60         ERPT @NIDMSG              ELSE, REPORT ID READ ERROR
  01E2  0004
  01E4  0209  MRKFND LI   R9,6                 LOAD BYTE COUNT
  01E6  0006
  01D6◆◆1306
  01E8  983A  IDRDLP CB   ◆R10+,@DTAWD         COMPARE DISK DATA
  01EA  7FFC
                  ◆                            TO ID FIELD IMAGE
  01EC  16F0         JNE  IDMRD                IF NOT EQUAL, START OVER
  01EE  0609         DEC R9                    DECREMENT BYTE COUNT
  01F0  16FB         JNE IDRDLP                IF NOT 0, READ NEXT BYTE
  01F2  0380         RTWP                      ELSE, ID FOUND, RETURN
```

▶9

Figure 32. Floppy Disk Control Program (Sheet 10 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                    PAGE 0011

 0396                 ****************************************************************
 0397                 *
 0398                 *       SUBROUTINE:  ERPT
 0399                 *
 0400                 *       CALLING SEQUENCE:  ERPT @MESSAGE
 0401                 *
 0402                 *       THE MESSAGE WHOSE ADDRESS IS CONTAINED IN R11
 0403                 *       WHEN THE ROUTINE IS ENTERED IS PRINTED,
 0404                 *       FOLLOWED BY THE CURRENT TRACK AND SECTOR
 0405                 *       NUMBER.  THE DRIVE IS TURNED OFF AND CONTROL
 0406                 *       IS RETURNED TO THE COMMAND ENTRY PROGRAM.
 0407                 *
 0408   01F4  2F00   ERPTPC NLIN 0                    NEW LINE
        0046**01F4'
 0409   01F6  2D9B          AXMT *R11                 PRINT SELECTED MESSAGE
 0410   01F8  2DA0          AXMT @TKMSG               PRINT TRACK MESSAGE
        01FA  008A'
 0411   01FC  2EE0          HXM2 @TKNUM               PRINT TRACK NUMBER
        01FE  80F8
 0412   0200  2DA0          AXMT @SCTMSG              PRINT SECTOR MESSAGE
        0202  009A'
 0413   0204  2EE0          HXM2 @SECNUM              PRINT SECTOR NUMBER
        0206  80FA
 0414   0208  1E04          SBZ  SEL                  TURN OFF DISK DRIVE
 0415   020A  0420          BLWP @RTPNVC              RETURN TO COMMAND
        020C  0142'
 0416                 *                               ENTRY PROGRAM
```

Figure 32. Floppy Disk Control Program (Sheet 11 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0012

 0418                 ♦♦♦
```

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0013

 0445                 ◆◆◆
```

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0014

                    ◆◆◆
```

```
FLOPPY DISK CONTROL PROGRAM                                    PAGE 0015

                  ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
                  ◆
                  ◆       SUBROUTINE:  SINC
                  ◆
                  ◆       CALLING SEQUENCE:  SINC 0
                  ◆
                  ◆       THE SECTOR NUMBER IS INCREMENTED BY 1.
                  ◆       IF THE NEW VALUE IS GREATER THAN 26,
                  ◆       THE SECTOR NUMBER IS SET TO 1 AND
                  ◆       THE TRACK NUMBER IS INCREMENTED,
                  ◆       AND THE HEAD IS STEPPED TO THE NEXT TRACK.
                  ◆
  0270  D2A0  SINCPC MOVB @SECNUM,R10        FETCH SECTOR NUMBER
  0272  80FA
  0052◆◆0270'
  0274  022A          AI   R10,>100          ADD 1 TO SECTOR NUMBER
  0276  0100
  0278  028A          CI   R10,27◆>100       COMPARE TO 27
  027A  1B00
  027C  14--          JHE  SECNXT            IF HIGH OR EQUAL,
                  ◆                          INCREMENT TRACK
  027E  D80A  SECXIT MOVB R10,@SECNUM        RESTORE SECTOR NUMBER
  0280  80FA
  0282  0380          RTWP                   RETURN
  0284  2E40  SECNXT TINC 0                  INCREMENT TRACK NUMBER
  027C◆◆1403
  0286  020A          LI   R10,>100          LOAD NEW SECTOR NUMBER
  0288  0100
  028A  10F9          JMP  SECXIT            STORE SECTOR NUMBER
                  ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
                  ◆
                  ◆       SUBROUTINE:  TKST
                  ◆
                  ◆       CALLING SEQUENCE:  TKST @TRACK
                  ◆
                  ◆       THE READ/WRITE HEAD OF THE DISK DRIVE IS
                  ◆       STEPPED TO THE TRACK NUMBER SPECIFIED BY THE
                  ◆       LEFT BYTE OF R11, UNLESS THE DISK IS NOT
                  ◆       READY, IN WHICH CASE THE OPERATION IS ABORTED.
                  ◆       IF THE SPECIFIED TRACK IS OUT OF RANGE (IE
                  ◆       GREATER THAN 76, THE HEAD IS STEPPED TO TRACK
                  ◆       0.  THE NEW TRACK NUMBER REPLACES THE OLD
                  ◆       TRACK NUMBER IN MEMORY.  IF THE NEW TRACK
                  ◆       NUMBER IS TO BE 0, THE TRACK IS STEPPED
                  ◆       UNTIL THE TRK00 STATUS SIGNAL IS DETECTED.
                  ◆       IF THE OLD TRACK NUMBER WAS 0, THE HEAD IS
                  ◆       STEPPED TO TRACK 0 BEFORE THE NEW STEPPING
                  ◆       OPERATION BEGINS.
                  ◆
  028C  04CC  TKSTPC CLR  R12                INITIALIZE CRU BASE
  004E◆◆028C'
  028E  1D04          SBO  SEL               SELECT DRIVE
  0290  2FE0          DLAY @HDLDLY           HEAD LOAD DELAY
  0292  1482
```

9◀

Figure 32. Floppy Disk Control Program (Sheet 15 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0016

  0556  0294  1F07          TB   RDY                 CHECK DRIVE STATUS
  0557  0296  13--          JEQ  TKCNTU              IF READY, CONTINUE
  0558  0298  2C60          ERPT @NRDYMS             ELSE, REPORT ERROR
        029A  0025'
  0559  029C  C24B  TKCNTU  MOV  R11,R9              SAVE NEW TRACK NUMBER
        0296**1302
  0560  029E  0989          SRL  R9,8                TO RIGHT BYTE OF R9
  0561  02A0  13--          JEQ  TKT00               IF 0, CLEAR TRACK
  0562  02A2  0289          CI   R9,76               NEW TRACK NUMBER IN RANGE?
        02A4  004C
  0563  02A6  12--          JLE  TKNZRO              IS SO, SKIP
  0564  02A8  04C9          CLR  R9                  ELSE, CLEAR NEW TRACK NUMBER
  0565  02AA  06A0  TKT00   BL   @TKCLR              STEP TO TRACK 00
        02AC  ----
        02A0**1304
  0566  02AE  10--          JMP  TKSTRT              RETURN
  0567  02B0  D2A0  TKNZRO  MOVB @TKNUM,R10          FETCH OLD TRACK NUMBER
        02B2  80F8
        02A6**1204
  0568  02B4  098A          SRL  R10,8               MOVE TO RIGHT BYTE
  0569  02B6  16--          JNE  TKNZR1              IF NOT 00, CONTINUE
  0570  02B8  06A0          BL   @TKCLR              ELSE, STEP TO TRACK 00
        02BA  ----
  0571  02BC  8289  TKNZR1  C    R9,R10               COMPARE NEW TRACK
        02B6**1602
  0572              *                                 TO OLD TRACK NUMBER
  0573  02BE  11--          JLT  STPOUT              IF LESS THAN, STEP OUT 1 TRACK
  0574  02C0  13--          JEQ  TKSTRT              IF EQUAL, RETURN
  0575  02C2  1D07          SBO  STEPUP              ELSE, STEP IN 1 TRACK
  0576  02C4  058A          INC  R10                 INCREMENT OLD TRACK
  0577  02C6  10--          JMP  TKGO                STEP HEAD
  0578  02C8  1E07  STPOUT  SBZ  STEPUP              SELECT STEP OUT
        02BE**1104
  0579  02CA  060A          DEC  R10                 DECREMENT OLD TRACK
  0580  02CC  06A0  TKGO    BL   @TKSTEP             STEP HEAD
        02CE  ----
        02C6**1002
  0581  02D0  10F5          JMP  TKNZR1              REPEAT FOR NEXT STEP
  0582  02D2  06C9  TKSTRT  SWPB R9                  MOVE NEW TRACK NUMBER
        02AE**1011
        02C0**1308
  0583              *                                 TO LEFT BYTE
  0584  02D4  D809          MOVB R9,@TKNUM           UPDATE TRACK NUMBER
        02D6  80F8
  0585  02D8  0380          RTWP                     RETURN
```

▸9

Figure 32. Floppy Disk Control Program (Sheet 16 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                    PAGE 0017

                    ♦♦♦
```

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0018

                    ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
                    ◆
                    ◆       SUBROUTINE:  TINC
                    ◆
                    ◆       CALLING SEQUENCE:  TINC 0
                    ◆
                    ◆       THE HEAD IS MOVED TO THE NEXT CONSECUTIVE
                    ◆       POSITION.  IF ON THE INNERMOST TRACK (76),
                    ◆       THE HEAD IS MOVED TO TRACK 00.
                    ◆
0304  D2E0          TINCPC MOVB @TKNUM,R11          FETCH TRACK NUMBER
0306  80F8
0066◆◆0304'
0308  022B                 AI   R11,>100            ADD 1 TO TRACK NUMBER
030A  0100
030C  2CDB                 TKST ◆R11                MOVE HEAD
030E  0380                 RTWP                     RETURN
                    ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
                    ◆
                    ◆       COMMAND CHARACTER LIST
                    ◆
0310    57          CMDLST TEXT 'WAWHWDRARHFMMDMEMX'
                    ◆
                    ◆       COMMAND ENTRY POINT TABLE
                    ◆
0322  ----          CMDENT DATA WTASCI,WRTHEX,WRTDEL,RDASCI,RDHEX
0324  ----
0326  ----
0328  ----
032A  ----
032C  ----                 DATA FORMAT,DUMP,ENTER,EXECUT
032E  ----
0330  ----
0332  ----
```

Figure 32. Floppy Disk Control Program (Sheet 18 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                    PAGE 0019

 0650                 ****************************************************
 0651                 *
 0652                 *       POWER-ON RESET ENTRY POINT
 0653                 *
 0654   0334   04CC  START  CLR  R12                 INITIALIZE CRU BASE
        0002**0334'
 0655   0336   020B         LI   R11,>300            LOAD CRU INITIALIZATION VALUE
        0338   0300
 0656   033A   320B         LDCR R11,8               AND OUTPUT TO CRU
 0657   033C   020B         LI   R11,IDFLD           SET ID FIELD IMAGE POINTER
        033E   80F7
 0658   0340   020A         LI   R10,>100            SET INITIAL SECTOR VALUE
        0342   0100
 0659   0344   DEE0         MOVB @IDMRK,*R11+        ID MARK DATA PATTERN TO
        0346   00D0'
 0660                 *                              FIRST BYTE OF ID FIELD IMAGE
 0661   0348   DECC         MOVB R12,*R11+           0 TO SECOND BYTE
 0662                 *                              (TRACK NUMBER)
 0663   034A   DECC         MOVB R12,*R11+           0 TO THIRD BYTE
 0664   034C   DECA         MOVB R10,*R11+           01 TO FOURTH BYTE
 0665                 *                              (SECTOR NUMBER)
 0666   034E   D6CC         MOVB R12,*R11            0 TO FIFTH BYTE
 0667   0350   2CDC         TKST *R12                SET READ/WRITE HEAD TO TRACK 0
 0668   0352   1E04         SBZ  SEL                 TURN OFF DRIVE
 0669                 ****************************************************
 0670                 *
 0671                 *       OPERATOR COMMAND REQUEST ENTRY POINT
 0672                 *
 0673   0354   2F00  TOP    NLIN 0                   NEW LINE
        0144**0354'
 0674   0356   2FA0         XMIT @QUEST              PRINT PROMPTING MESSAGE
        0358   0085'
 0675   035A   2FA0         XMIT @BELL               (QUESTION MARK, BELL)
        035C   0086'
 0676   035E   2F4A         RECV R10                 READ FIRST CHARACTER
 0677                 *                              OF COMMAND
 0678   0360   06CA         SWPB R10                 SAVE IN RIGHT BYTE
 0679   0362   2F4A         RECV R10                 READ SECOND CHARACTER
 0680                 *                              OF COMMAND
 0681   0364   06CA         SWPB R10                 REVERSE CHARACTERS IN R10
 0682   0366   0208         LI   R8,CMDLST           SET COMMAND LIST POINTER
        0368   0310'
 0683   036A   0209         LI   R9,CMDENT-2         SET COMMAND ENTRY POINTER
        036C   0320'
 0684   036E   C1F8  CMDLP  MOV *R8+,R7               FETCH COMMAND IN LIST
 0685   0370   13F1         JEQ  TOP                 IF LIST VALUE = 0, NOT
 0686                 *                              A LEGAL COMMAND
 0687   0372   05C9         INCT R9                  INCREMENT ENTRY POINTER
 0688   0374   81CA         C    R10,R7              COMPARE ENTERED COMMAND
 0689                 *                              TO LIST
 0690   0376   16FB         JNE  CMDLP               IF NOT EQUAL, REPEAT
 0691   0378   C259         MOV  *R9,R9              ELSE, COMMAND FOUND,
 0692                 *                              FETCH ENTRY POINT
 0693   037A   020A         LI   R10,TOP             COMMAND PROGRAM RETURN
```

9◀

Figure 32. Floppy Disk Control Program (Sheet 19 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0020

        037C  0354'
 0694                 *                                     ADDRESS
 0695   037E  9807          CB    R7,@ASCIIM                TEST FOR MD,ME, OR
        0380  0082'
 0696                 *                                     MX COMMANDS
 0697   0382  13--          JEQ   ADDFCH                    IF SO, FETCH ADDRESS ENTRY
 0698   0384  2DA0          AXMT  @TKMSG                    PRINT TRACK MESSAGE
        0386  008A'
 0699   0388  D220          MOVB  @TKNUM,R8                 FETCH CURRENT TRACK
        038A  80F8
 0700                 *                                     NUMBER
 0701   038C  2EC8          HXM2  R8                        PRINT TRACK NUMBER
 0702   038E  2E88          HRC2  R8                        READ NEW TRACK NUMBER
 0703   0390  0288          CI    R8,77*256                 NEW TRACK NUMBER LEGAL?
        0392  4D00
 0704   0394  14DF          JHE   TOP                       IF NOT, ABORT
 0705   0396  2CD8          TKST  *R8                       STEP HEAD TO NEW TRACK
 0706   0398  1E04          SBZ   SEL                       TURN OFF DRIVE
 0707   039A  9807          CB    R7,@ASCIIF                FORMAT COMMAND?
        039C  0081'
 0708   039E  16--          JNE   SECFCH                    IF NOT, CONTINUE
 0709   03A0  0459          B     *R9                       ELSE, EXECUTE COMMAND
 0710   03A2  2DA0  SECFCH  AXMT  @SCTMSG                    PRINT SECTOR MESSAGE
        03A4  009A'
        039E**1601
 0711   03A6  D1A0          MOVB  @SECNUM,R6                FETCH CURRENT SECTOR
        03A8  80FA
 0712   03AA  2EC6          HXM2  R6                        PRINT CURRENT SECTOR
 0713   03AC  2E86          HRC2  R6                        READ NEW SECTOR NUMBER
 0714   03AE  0286          CI    R6,>100                   LESS THAN 1
        03B0  0100
 0715   03B2  11D0          JLT   TOP                       IF SO, ABORT
 0716   03B4  0286          CI    R6,27*256                 GREATER THAN 26?
        03B6  1B00
 0717   03B8  14CD          JHE   TOP                       IF SO, ABORT
 0718   03BA  D806          MOVB  R6,@SECNUM                UPDATE SECTOR NUMBER
        03BC  80FA
 0719   03BE  2DA0          AXMT  @NUMMSG                   PRINT NUMBER MESSAGE
        03C0  00A5'
 0720   03C2  0205          LI    R5,>100                   LOAD DEFAULT NUMBER
        03C4  0100
 0721   03C6  2E85          HRC2  R5                        READ NUMBER
 0722   03C8  0985          SRL   R5,8                      MOVE TO RIGHT BYTE
 0723   03CA  13C4          JEQ   TOP                       IF NUMBER = 0, ABORT
 0724   03CC  0459          B     *R9                       EXECUTE COMMAND
 0725   03CE  0208  ADDFCH LI    R8,>8000                  LOAD DEFAULT ADDRESS
        03D0  8000
        0382**1325
 0726   03D2  2E88          HRC2  R8                        READ FIRST BYTE OF ADDRESS
 0727   03D4  06C8          SWPB  R8                        SAVE IN RIGHT BYTE
 0728   03D6  2FA0          XMIT  @BACKSP                   BACKSPACE PRINTER
        03D8  0087'
 0729   03DA  2E88          HRC2  R8                        READ SECOND BYTE OF ADDRESS
 0730   03DC  06C8          SWPB  R8                        CORRECT ADDRESS BYTES
 0731   03DE  0459          B     *R9                       EXECUTE COMMAND
```

Figure 32. Floppy Disk Control Program (Sheet 20 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                    PAGE 0021

                    ◆◆◆
```

```
FLOPPY DISK CONTROL PROGRAM                                    PAGE 0022

         0414   00BC'
  0775   0416   C220   DMRKOK MOV   @DTACRC,R8        FETCH READ CRC
         0418   8180
         0402**1309
  0776   041A   2E00          CRCD  0                 RECALCULATE CRC
  0777   041C   8220          C     @DTACRC,R8        CRC CORRECT?
         041E   8180
  0778   0420   13--          JEQ   RDPRT             IF SO CONTINUE
  0779   0422   2C60          ERPT  @CRCMSG           ELSE, REPORT ERROR
         0424   0035'
  0780   0426   2F00   RDPRT  NLIN  0                 NEW LINE
         0420**1302
  0781   0428   04E0          CLR   @DTACRC           CLEAR END OF DATA
         042A   8180
  0782                 *                              FIELD IMAGE
  0783   042C   0206          LI    R6,DTABUF         LOAD FIELD IMAGE
         042E   8100
  0784                 *                              POINTER
  0785   0430   9807          CB    R7,@ASCIIA        RA COMMAND?
         0432   0080'
  0786   0434   13--          JEQ   ASCIRD            IF SO, PRINT IN ASCII
  0787                 *                              FORMAT
  0788   0436   0209          LI    R9,8              LOAD LINE COUNT
         0438   0008
  0789   043A   0208   HXPTLP LI    R8,16             LOAD BYTE COUNT
         043C   0010
  0790   043E   2F00          NLIN  0                 NEW LINE
  0791   0440   2ED6   HXPLP1 HXMT  *R6                PRINT DATA BYTE
  0792   0442   0586          INC   R6                INCREMENT DATA POINTER
  0793   0444   0608          DEC   R8                DECREMENT BYTE COUNT
  0794   0446   16FC          JNE   HXPLP1            IF NOT 0, PRINT NEXT BYTE
  0795   0448   1F00          TB    RIN               OPERATOR INTERRUPT?
  0796   044A   16--          JNE   READRT            IF SO, ABORT
  0797   044C   0609          DEC   R9                DECREMENT LINE COUNT
  0798   044E   16F5          JNE   HXPTLP            IF NOT 0, PRINT NEXT LINE
  0799   0450   10--          JMP   NXTSCT            CONTINUE
  0800   0452   2D96   ASCIRD AXMT  *R6                PRINT DATA FIELD
         0434**130E
  0801                 *                              IN ASCII
  0802   0454   2D00   NXTSCT SINC  0                 UPDATE SECTOR NUMBER
         0450**1001
  0803   0456   1F00          TB    RIN               OPERATOR INTERRUPT?
  0804   0458   16--          JNE   READRT            IF SO, ABORT
  0805   045A   0605          DEC   R5                DECREMENT SECTOR COUNT
  0806   045C   16C2          JNE   READ              IF NOT 0, READ NEXT SECTOR
  0807   045E   1E04   READRT SBZ   SEL               TURN OFF DRIVE
         044A**1609
         0458**1602
  0808   0460   045A          B     *R10              RETURN
```

▶9

Figure 32. Floppy Disk Control Program (Sheet 22 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                    PAGE 0023

 0810                 ♦♦♦
```

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0024

  0849  0498  2F00          NLIN 0                  NEW LINE
  0850  049A  2E98  WTHLP2  HRC2 *R8                READ BYTE
  0851  049C  0588          INC  R8                 INCREMENT BUFFER POINTER
  0852  049E  0606          DEC  R6                 DECREMENT BYTE COUNT
  0853  04A0  16FC          JNE  WTHLP2             IF NOT 0, READ NEXT BYTE
  0854  04A2  0609          DEC  R9                 DECREMENT LINE COUNT
  0855  04A4  16F7          JNE  WTHLP1             IF NOT 0, READ NEXT LINE
  0856  04A6  C284  WTBRDY  MOV  R4,R10             RESTORE RETURN ADDRESS
        048E**04A6'
  0857  04A8  10--          JMP  WTCRCD             CONTINUE
  0858  04AA  0206  WRTASC  LI   R6,128             LOAD CHARACTER COUNT
        04AC  0080
        0488**1310
  0859  04AE  2F00          NLIN 0                  NEW LINE
  0860  04B0  2F58  WTASLP  RECV *R8                READ CHARACTER
  0861  04B2  9818          CB   *R8,@ESC           ESCAPE CHARACTER?
        04B4  0083'
  0862  04B6  13--          JEQ  WRITRT             IF SO, RETURN
  0863  04B8  9838          CB   *R8+,@BLANK        NON-PRINTABLE?
        04BA  0084'
  0864  04BC  11--          JLT  WTCRCD             IF SO, END OF SECTOR
  0865  04BE  0606          DEC  R6                 DECREMENT CHARACTER COUNT
  0866  04C0  16F7          JNE  WTASLP             IF NOT 0, READ NEXT CHAR
  0867  04C2  2E00  WTCRCD  CRCD 0                  GENERATE DATA FIELD CRC
        04A8**100C
        04BC**1102
  0868  04C4  0209          LI   R9,DTAWT           DISK DATA WRITE ADDRESS
        04C6  7FFE
  0869  04C8  0208          LI   R8,DTAFLD          DATA FIELD IMAGE POINTER
        04CA  80FF
  0870  04CC  2C80          IDRD 0                  READ ID FIELD
  0871  04CE  0200          LI   R0,16              REPEAT 16 TIMES
        04D0  0010
  0872  04D2  04D9  WTLPL2  CLR  *R9                WRITE LAST 16 BYTES OF
  0873                *                             ID GAP (FIRST BYTE SKIPPED FOR
  0874                *                             BYTE SYNCHRONIZATION)
  0875  04D4  0600          DEC  R0
  0876  04D6  16FD          JNE  WTLPL2
  0877  04D8  D838          MOVB *R8+,@MRKWT        WRITE DATA MARK
        04DA  7F8E
  0878  04DC  0200          LI   R0,130             REPEAT 130 TIMES
        04DE  0082
  0879  04E0  D678  WTLPL3  MOVB *R8+,*R9           WRITE DATA FIELD
  0880  04E2  0600          DEC  R0
  0881  04E4  16FD          JNE  WTLPL3
  0882  04E6  04D9          CLR  *R9                REWRITE FIRST BYTE OF
  0883                *                             DATA GAP
  0884  04E8  2D00          SINC 0                  UPDATE SECTOR NUMBER
  0885  04EA  1E04          SBZ  SEL                TURN OFF DRIVE
  0886  04EC  0605          DEC  R5                 DECREMENT SECTOR COUNT
  0887  04EE  16C1          JNE  WRITLP             IF NOT 0, WRITE NEXT SECTOR
  0888  04F0  045A  WRITRT  B    *R10                ELSE, RETURN
        04B6**131C
```

▶9

Figure 32. Floppy Disk Control Program (Sheet 24 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0025

                    ********************************************************
                    *
                    *       COMMAND CONTROL PROGRAM:  FORMAT
                    *
                    *       THIS COMMAND ENABLES THE OPERATOR TO FORMAT
                    *       A NUMBER OF TRACKS BEGINNING AT THE CURRENT TRACK
                    *       AND SPECIFYING THE LAST TRACK.  ALL GAPS,
                    *       TRACK, ID, AND DATA MARKS, AND TRACK AND
                    *       SECTOR NUMBERS ARE WRITTEN.  THE DATA IN THE
                    *       DATA FIELDS IS ALL ZEROES.
                    *
                    *       ENTRY PARAMETERS:       R10 = RETURN ADDRESS
                    *
  04F2  2DA0  FORMAT AXMT @ENDMSG                   PRINT END MESSAGE
  04F4  0094'
  032C**04F2'
  04F6  2DA0        AXMT @TKMSG                     PRINT TRACK MESSAGE
  04F8  008A'
  04FA  D260        MOVB @TKNUM,R9                  FETCH TRACK NUMBER
  04FC  80F8
  04FE  2EC9        HXM2 R9                         PRINT TRACK NUMBER
  0500  2E89        HRC2 R9                         READ LAST TRACK NUMBER
  0502  0289        CI   R9,77*256                  LEGAL VALUE?
  0504  4D00
  0506  14--        JHE  FRMTRT                     IF NOT, RETURN
  0508  0208        LI   R8,DTAFLD                  LOAD DATA FIELD POINTER
  050A  80FF
  050C  DE20        MOVB @DTMRK,*R8+                LOAD DATA MARK
  050E  00D1'
  0510  0200        LI   R0,64                      REPEAT 64 TIMES
  0512  0040
  0514  04F8  FFLPL1 CLR  *R8+                      CLEAR DATA BUFFER
  0516  0600        DEC  R0
  0518  16FD        JNE  FFLPL1
  051A  2E00        CRCD 0                          CALCULATE THE CRC FOR THE
                    *                               DATA FIELD
  051C  9809  FRMTLP CB   R9,@TKNUM                 LAST TRACK LESS
  051E  80F8
                    *                               THAN CURRENT TRACK?
  0520  11--        JLT  FRMTRT                     IF SO, RETURN
  0522  0208  FRMT1 LI   R8,>100                    LOAD INITIAL SECTOR
  0524  0100
                    *                               VALUE
  0526  0207        LI   R7,SECBUF                  LOAD SECTOR BUFFER
  0528  80C0
                    *                               POINTER
  052A  D808  FRIDBL MOVB R8,@SECNUM                UPDATE SECTOR
  052C  80FA
                    *                               NUMBER
  052E  2DC0        CRCI 0                          CALCULATE CRC FOR ID FIELD
  0530  CDE0        MOV  @IDCRC,*R7+                SAVE CRC IN BUFFER
  0532  80FC
  0534  0228        AI   R8,>100                    INCREMENT SECTOR NUMBER
  0536  0100
```

9◀

Figure 32. Floppy Disk Control Program (Sheet 25 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0026

 0930  0538  0288          CI    R8,27*256          LAST SECTOR?
       053A  1B00
 0931  053C  16F6          JNE   FRIDBL             IF NOT, REPEAT FOR
 0932                *                                NEXT SECTOR
 0933  053E  0207          LI    R7,SECBUF          LOAD SECTOR BUFFER
       0540  80C0
 0934                *                                POINTER
 0935  0542  0208          LI    R8,>100            LOAD INITIAL SECTOR
       0544  0100
 0936                *                                NUMBER
 0937  0546  2D40          DSON  0                  TURN ON DRIVE
 0938  0548  0206  FMINDX  LI    R6,DTAWT           DISK DATA WRITE
       054A  7FFE
 0939  054C  04E0          CLR   @INDXWT            WRITE 0 AT INDEX PULSE
       054E  7FFA
 0940  0550  0200          LI    R0,45              REPEAT 45 TIMES
       0552  002D
 0941  0554  04D6  FFLPL2  CLR   *R6                 WRITE REST OF POST-INDEX
 0942                *                                GAP
 0943  0556  0600          DEC   R0
 0944  0558  16FD          JNE   FFLPL2
 0945  055A  D820          MOVB  @TKMRK,@TKMWT      WRITE TRACK MARK
       055C  00D2'
       055E  7F9E
 0946  0560  0200  SECTLP  LI    R0,32              REPEAT 32 TIMES
       0562  0020
 0947  0564  04D6  FFLPL3  CLR   *R6                 WRITE 32 BYTE GAP
 0948  0566  0600          DEC   R0
 0949  0568  16FD          JNE   FFLPL3
 0950  056A  D820          MOVB  @IDMRK,@MRKWT      WRITE ID MARK
       056C  00D0'
       056E  7F8E
 0951  0570  D5A0          MOVB  @TKNUM,*R6         WRITE TRACK NUMBER
       0572  80F8
 0952  0574  D58C          MOVB  R12,*R6            WRITE SECOND BYTE
 0953  0576  D588          MOVB  R8,*R6             WRITE SECTOR NUMBER
 0954  0578  0228          AI    R8,>100            INCREMENT SECTOR NUMBER
       057A  0100
 0955  057C  D58C          MOVB  R12,*R6            WRITE FOURTH BYTE
 0956  057E  D5B7          MOVB  *R7+,*R6           WRITE CRC1
 0957  0580  D5B7          MOVB  *R7+,*R6           WRITE CRC2
 0958  0582  0200          LI    R0,17              REPEAT 17 TIMES
       0584  0011
 0959  0586  04D6  FFLPL4  CLR   *R6                 WRITE ID GAP
 0960  0588  0600          DEC   R0
 0961  058A  16FD          JNE   FFLPL4
 0962  058C  0204          LI    R4,DTAFLD          LOAD DATA FIELD
       058E  80FF
 0963                *                                IMAGE POINTER
 0964  0590  D834          MOVB  *R4+,@MRKWT        WRITE DATA MARK
       0592  7F8E
 0965  0594  0200          LI    R0,130             REPEAT 130 TIMES
       0596  0082
 0966  0598  D5B4  FFLPL5  MOVB  *R4+,*R6           WRITE DATA AND CRC
 0967  059A  0600          DEC   R0
```

Figure 32. Floppy Disk Control Program (Sheet 26 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0027

 0968  059C  16FD          JNE  FFLPL5
 0969  059E  04D6          CLR  *R6                 WRITE PAD BYTE
 0970  05A0  0288          CI   R8,>27*256          LAST BYTE?
       05A2  2700
 0971  05A4  16DD          JNE  SECTLP              IF NOT, FORMAT NEXT SECTOR
 0972  05A6  04D6  PREILP  CLR  *R6                 WRITE PRE-INDEX GAP
 0973  05A8  1F04          TB   INDEX               UNTIL INDEX
 0974  05AA  16FD          JNE  PREILP              PULSE OCCURS
 0975  05AC  2E40          TINC 0                   STEP HEAD TO NEXT TRACK
 0976  05AE  10B6          JMP  FRMTLP              FORMAT NEXT TRACK
 0977  05B0  1E04  FRMTRT  SBZ  SEL                  TURN OFF DRIVE
       0506**1454
       0520**1147
 0978  05B2  045A          B    *R10                RETURN
 0979                 ***************************************************
 0980                 *
 0981                 *      COMMAND CONTROL PROGRAM:  EXECUT
 0982                 *
 0983                 *      THIS COMMAND ENABLES THE OPERATOR TO BEGIN
 0984                 *      EXECUTION OF A PROGRAM AT ANY LOCATION
 0985                 *      IN MEMORY.
 0986                 *
 0987                 *      ENTRY PARAMETERS:        R8 = ENTRY POINT
 0988                 *
 0989  05B4  0458  EXECUT B    *R8                  BRANCH TO ENTRY POINT
       0332**05B4'
 0990                 ***************************************************
 0991                 *
 0992                 *      COMMAND CONTROL PROGRAM:  ENTER
 0993                 *
 0994                 *      THIS COMMAND ENABLES THE OPERATOR TO ENTER
 0995                 *      DATA INTO SEQUENTIAL MEMORY LOCATIONS.
 0996                 *
 0997                 *      CALLING PARAMETERS:      R8 = BEGINNING MEMORY
 0998                 *                                    LOCATION
 0999                 *
 1000  05B6  0209  ENTER  LI   R9,8                 LOAD BYTE COUNT
       05B8  0008
       0330**05B6'
 1001  05BA  2F00          NLIN 0                   NEW LINE
 1002  05BC  2EC8          HXM2 R8                  PRINT FIRST BYTE OF ADDRESS
 1003  05BE  06C8          SWPB R8                  REVERSE BYTES
 1004  05C0  2FA0          XMIT @BACKSP             BACKSPACE
       05C2  0087'
 1005  05C4  2EC8          HXM2 R8                  PRINT SECOND BYTE OF ADDRESS
 1006  05C6  06C8          SWPB R8                  RESTORE BYTES
 1007  05C8  2ED8  ENTLP   HXM2 *R8                  PRINT MEMORY CONTENTS
 1008  05CA  2E98          HRC2 *R8                 READ AND STORE NEW VALUE
 1009  05CC  0588          INC  R8                  UPDATE ADDRESS POINTER
 1010  05CE  0609          DEC  R9                  DECREMENT BYTE COUNT
 1011  05D0  13F2          JEQ  ENTER               IF 0, NEW LINE
 1012  05D2  10FA          JMP  ENTLP               ELSE, FETCH NEXT BYTE
```

Figure 32. Floppy Disk Control Program (Sheet 27 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0028

  1014                  ♦♦♦
```

Appendix and Glossary

GLOSSARY

absolute address: 1. An address that is permanently assigned by the machine designer to a storage location. 2. A pattern of characters that identifies a unique storage location without further modification. 3. Synonymous with machine address, specific address.

access time: The time interval between the request for information and the instant this information is available.

accumulator: A device which stores a number and which, on receipt of another number, adds the two and stores the sum.

address: An expression, usually numerical, which designates a specific location in a storage or memory device.

address format: 1. The arrangement of the address parts of an instruction. The expression "plus-one" is frequently used to indicate that one of the addresses specifies the location of the next instruction to be executed, such as one-plus-one, two-plus-one, three-plus-one, four-plus-one. 2. The arrangement of the parts of a single address, such as those required for identifying channel, module, track, etc., in a disc system.

address register: A register in which an address is stored.

ALGOL: ALGOrithmic Language. A language primarily used to express computer programs by algorithms.

algorithm: A term used by mathematicians to describe a set of procedures by which a given result is obtained.

alphanumeric: Pertaining to a character set that contains letters, digits, and usually other characters such as punctuation marks.

ALU: Arithmetic Logic Unit, a computational subsystem which performs the mathematical operations of a digital system.

analog: Electric analog information is information represented by a variable property of electricity, such as voltage, current, amplitude of waves or pulses, or frequency of waves or pulses. Analog circuitry, also called "linear" circuitry, is circuitry that varies certain properties of electricity continuously and smoothly over a certain range, rather than switching suddenly between certain levels.

AND: A logic operator having the property that if P is a statement, Q is a statement, R is a statement . . . , then the AND of P, Q, R . . . is true if all statements are true, false if any statement is false. P AND Q is often represented by P•Q or PQ. Synonymous with logical multiply.

arithmetic shift: 1. A shift that does not affect the sign position. 2. A shift that is equivalent to the multiplication of a number by a positive or negative integral power of the radix.

ASCII: (American National Standard Code for Information Interchange, 1968) The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. Synonymous with USASCII.

assemble: To prepare a machine language program from a symbolic language program by substituting absolute operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

assembler: A computer program that assembles.

asynchronous device: A device in which the speed of operation is not related to any frequency in th system to which it is connected.

base: 1. a reference value. 2. A number that is multiplied by itself as many times as indicated by an exponent. 3. Same as radix.

base address: A given address from which an absolute address is derived by combination with a relative address.

baud: A unit of signaling speed equal to the number of discrete conditions or signal events per second. For example, one baud equals one-half dot cycle per second in Morse code, one bit per second in a train of binary signals, and one 3-bit value per second in a train of signals each of which can assume one of eight different states.

G

BCD: Binary coded decimal notation.

benchmark problem: A problem used to evaluate the performance of hardware or software or both.

binary: 1. Pertaining to a characteristic or property involving a selection, choice, or condition in which there are two possibilities. 2. Pertaining to the number representation system with a radix of two.

binary coded decimal (BCD): A binary numbering system for coding decimal numbers in groups of 4 bits. The binary value of these 4-bit groups ranges from 0000 to 1001, and codes the decimal digits "0" through "9". To count to 9 takes 4 bits; to count to 99 takes two groups of 4 bits; to count to 999 takes three groups of 4 bits, etc.

block diagram: A diagram of a system, instrument, or computer in which the principal parts are represented by suitable associated geometrical figures to show both the basic functions and the functional relationships among the parts.

block transfer: The process of transmitting one or more blocks of data where the data are organized in such blocks.

bootstrap: A technique or device designed to bring itself into a desired state by means of its own action, e.g., a machine routine whose first few instructions are sufficient to bring the rest of itself into the computer from an input device.

borrow: An arithmetically negative carry.

branch: 1. A set of instructions that is executed between two successive decision instructions. 2. To select a branch as in definition 1. 3. A direct path joining two nodes of a network or graph. 4. Loosely, a conditional jump.

branching: A method of selecting, on the basis of results, the next operation to execute while the program is in progress.

breakpoint: A place in a routine specified by an instruction, instruction digit, or other condition, where the routine may be interrupted by external intervention or by a monitor routine.

buffer: An isolating circuit used to avoid reaction of a driven circuit on the corresponding driver circuit. Also, a storage device used to compensate for a difference in the rate of flow of information or the time of occurrence of events when transmitting information from one device to another.

bus: One or more conductors used for transmitting signals or power.

byte: A sequence of adjacent binary digits operated upon as a unit and usually shorter than a computer word. Usually 8 bits.

carry: One or more digits, produced in connection with an arithmetic operation on one digit place of two or more numerals in positional notation, that are forwarded to another digit place for processing there.

CCD: Charge-coupled device. A means for very dense serial-access storage of bits as tiny packets of electric charge moving along the surface of a semiconductor chip.

central processor unit (CPU): Part of a computer system which contains the main storage, arithmetic unit, and special register groups. It performs arithmetic operations, controls instruction processing, and provides timing signals and other housekeeping operations.

character: A letter, digit, or other symbol that is used as part of the organization, control, or representation of data. A character is often in the form of a spatial arrangement of adjacent or connected strokes.

character check: A check that verifies the observance of rules for the formation of characters.

check bit: A binary check digit, e.g., a parity bit.

check digit: A digit used for purpose of performing a check.

checkpoint: A place in a routine where a check or a recording of data for restart purposes, is performed.

chip-enable input: A control input that when active permits operation of the integrated circuit for input, internal transfer, manipulation, refreshing, and/or output of data and when inactive causes the integrated circuit to be in a reduced-power standby mode.

circulating register: A shift register in which data moved out of one end of the register are reentered into the other end as in a closed loop.

clock: 1. A device that generates periodic signals used for synchronization. 2. A register whose content changes at regular intervals in such a way as to measure time.

COBOL: (Common Business Oriented Language) A business data processing language.

code: 1. A set of unambiguous rules specifying the way in which data may be represented, e.g., the set of correspondences in the standard code for information interchange. Synonymous with coding scheme. 2. In telecommunications, a system of rules and conventions according to which the signals representing data can be formed, transmitted, received, and processed. 3. In data processing, to represent data or a computer program in a symbolic form that can be accepted by a data processor.

communication link: The physical means of connecting one location to another for the purpose of transmitting and receiving data.

compile: To prepare a machine language program from a computer program written in another programming language by making use of the overall logic structure of the program, or generating more than one machine instruction for each symbolic statement, or both, as well as performing the function of an assembler.

compiler: A program that compiles.

complement: A number that can be derived from a specified number by subtracting it from a second specified number. For example, in radix notation, the second specified number may be a given power of the radix or one less than a given power of the radix. The negative of a number is often represented by its complement.

computer: A data processor that can perform substantial computation, including numerous arithmetic or logic operations, without intervention by a human operator during the run.

conditional jump: A jump that occurs if specified criteria are met.

controller: Digital subsystem responsible for implementing "how" a system is to function. Not to be confused with "timing" as timing tells the system "when" to perform its function.

counter: A circuit which counts input pulses and will give an output pulse after receiving a predetermined number of input pulses.

CRU: Communications Register Unit: a command-driven bit addressable I/O interface. The processor instruction can set, reset, or test any bit in the CRU array or move data between the memory and CRU data fields.

cycle: 1. An interval of space or time in which one set of events or phenomena is completed. 2. Any set of operations that is repeated regularly in the same sequence. The operations may be subject to variations on each repetition.

data: 1. A representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by humans or automatic means. 2. Any representations such as characters or analog quantities to which meaning is or might be assigned.

data bus: One method of input-output for a system where data are moved into or out of the digital system by way of a common bus connected to several subsystems.

data processing: The execution of a systematic sequence of operations performed upon data. Synonymous with information processing.

data selector: A combinational building-block that routes data from one of several inputs to a single output, according to control signals. Also called "multiplexer." Two or more such one-bit selectors operating in parallel would be called a "two-bit data selector," etc.

debug: To detect, locate, and remove mistakes from a routine or malfunctions from a computer. Synonymous with troubleshoot.

decimal: 1. Pertaining to a characteristic or property involving a selection, choice, or condition in which there are ten possibilities. 2. Pertaining to the number representation system with a radix of ten.

decimal digit: In decimal notation, one of the characters 0 through 9.

decoder: A conversion circuit that accepts digital input information — in the memory case, binary address information — that appears as a small number of lines and selects and activates one line of a large number of output lines.

digital: 1. Pertaining to data in the form of digits. 2. Contrast with analog. 3. Information in discrete or quantized form; not continuous.

direct access: Pertaining to the process of obtaining data from, or placing data into, storage where the time required for such access is independent of the location of the data most recently obtained or placed in storage.

direct addressing: Method of programming that has the address pointing to the location of data or the instruction that is to be used.

direct memory access channel (DMA): A method of input-output for a system that uses a small processor whose sole task is that of controlling input-output. With DMA, data are moved into or out of the system without program intervention.

double precision: Pertaining to the use of two computer words to represent a number

dump: 1. To copy the contents of all or part of a storage, usually from an internal storage into an external storage. 2. A process as in definition 1 above. 3. The data resulting from the process as in definition 1 above.

duplex: In communications, pertaining to a simultaneous two-way independent transmission in both directions. Contrast with half duplex. Synonymous with full duplex.

edge triggering: Activation of a circuit at the edge of the pulse as it begins its change. Circuits then trigger at the edge of the input pulse rather than sensing a level change.

edit: To modify the form or format of data, e.g., to insert or delete characters such as page numbers or decimal points.

effective address: The address that is derived by applying any specified indexing or indirect addressing results to the specified address and that is actually used to identify the current operand.

emulate: To imitate one system with another such that the imitating system accepts the same data, executes the same programs, and achieves the same results as the imitated system.

encode: To apply a set of unambiguous rules specifying the way in which data may be represented such that a subsequent decoding is possible. Synonymous with code.

entry point: In a routine, any place to which control can be passed.

EPROM: Eraseable and programmable read-only memory. An IC memory chip whose stored data can be read at random. The data can be erased and new data can be stored, but only by a special system other than the one in which the memory is used.

erase: To obliterate information from a storage medium, e.g., to clear, to overwrite.

error: Any discrepancy between a computed, observed, or measured quantity and the true, specified, or theoretically correct value or condition.

exclusive-OR function: A modified form of the OR function which has a logic equation equal to the sum output of the half-adder.

execute: That portion of a computer cycle during which a selected control word or instruction is accomplished.

exponent: In a floating-point representation, the numeral, of a pair of numerals representing a number, that indicates the power to which the base is raised.

family: A family of digital integrated circuits is a group of ICs that use the same general design style for all gates, and processed during manufacture in much the same way, and whose input and output signals are all "compatible" with one another so that one can transmit to another.

fetch: That portion of a computer cycle during which the next instruction is retrieved from memory.

field: In a record, a specified area used for a particular category of data, e.g., a group of card columns used to represent a wage rate, a set of bit locations in a computer word used to express the address of the operand.

first-in first-out (FIFO) memory: A memory from which data bytes or words can be read in the same order, but not necessarily at the same rate, as that of the data entry.

fixed-point representation: A positional representation in which each number is represented by a single set of digits, the position of the radix point being fixed with respect to one end of the set, according to some convention.

flag: 1. Any of various types of indicators used for identification, e.g., a wordmark. 2. A character that signals the occurrence of some condition, such as the end of a word. 3. Synonymous with mark, sentinel, tag.

flip-flop (storage element): A circuit having two stable states and the capability of changing from one state to another with the application of a control signal and remaining in that state after removal of signals.

flow chart: A graphical representation for definition, analysis, or solution of a problem, in which symbols are used to represent operations, data, flow, equipment, etc.

format: The arrangement of data.

FORTRAN: (FORmula TRANslating system) A language primarily used to express computer programs by arithmetic formulas.

function: 1. A specific purpose of an entity, or its characteristic action. 2. In communications, a machine action such as a carriage return or line feed.

gate: 1. A device having one output channel and one or more input channels, such that the output channel state is completely determined by the input channel states, except during switching transients. 2. A combinational logic element having at least one input channel. 3. An AND gate. 4. An OR gate.

general-purpose computer: A computer that is designed to handle a wide variety of problems.

generate: To produce a program by selection of subsets from a set of skeletal coding under the control of parameters.

half duplex: In communications, pertaining to an alternate, one way at a time, independent transmission. Contrast with duplex.

hardware: Physical equipment, as opposed to the computer program or method of use, e.g., mechanical, magnetic, electrical, or electronic devices.

hold time: Hold time, t_h. The interval during which a signal is retained at a specified input terminal after an active transition occurs at another specified input terminal.

immediate address: Pertaining to an instruction in which an address part contains the value of an operand rather than its address. Synonymous with zero-level address.

indexed address: An address that is modified by the content of an index register prior to or during the execution of a computer instruction.

indexing: In computers, a method of address modification that is implemented by means of index registers.

index register: A register whose content may be added to or subtracted from the operand address prior to or during the execution of a computer instruction.

indirect addressing: Programming method that has the initial address being the storage location of a word that contains another address. This indirect address is then used to obtain the data to be operated upon.

input/output devices (I/O): Computer hardware by which data is entered into a digital system or by which data are recorded for immediate or future use.

instruction: A statement that specifies an operation and the values or locations of its operands.

instruction cycle: The period of time during which a programmed system obeys a particular instruction.

instruction register: A register that stores an instruction for execution.

interface: A shared boundary. An interface might be a hardware component to link two devices or it might be a portion of storage or registers accessed by two or more computer programs.

interrupt: To stop a process in such a way that it can be resumed.

jump: A departure from the normal sequence of executing instructions in a computer.

jump conditions: Conditions defined in a transition table that determine the changes of flip-flops from one state to another state.

label: One or more characters used to identify a statement or an item of data in a computer program.

language: A set of representations, conventions, and rules used to convey information.

large scale integration (LSI): The simultaneous realization of large area chips and optimum component packing density, resulting in cost reduction by maximizing the number of system connections done at the chip level. Circuit complexity above 100 gates.

level: The degree of subordination in a hierarchy.

linkage: In programming, coding that connects two separately coded routines.

load: In programming, to enter data into storage or working registers.

location: Any place in which data may be stored.

logic diagram: A diagram that represents a logic design and sometimes the hardware implementation.

logic shift: A shift that affects all positions.

logic symbol: 1. A symbol used to represent a logic element graphically. 2. A symbol used to represent a logic operator.

loop: A sequence of instructions that is executed repeatedly until a terminal condition prevails.

LSB: Least significant bit.

machine code: An operation code that a machine is designed to recognize. Usually expressed in ones and zeros.

macroinstruction: An instruction in a source language that is equivalent to a specified sequence of machine instructions.

macroprogramming: Programming with macroinstructions.

magnetic bubble: A tiny moveable magnetized region formed under certain conditions in a thin film of magnetic garnet crystal fabricated similar to an IC. Such bubbles provide very dense serial-access storage of bits.

magnetic drum: A right circular cylinder with a magnetic surface on which data can be stored by selective magnetization of portions of the curved surface.

main storage: The general-purpose storage of a computer. Usually, main storage can be accessed directly by the operating registers. Contrast with auxiliary storage.

mask: 1. A pattern of characters that is used to control the retention or elimination of portions of another pattern of characters. 2. A filter.

microprocessor: An IC (or set of a few ICs) that can be programmed with stored instructions to perform a wide variety of functions, consisting at least of a controller, some registers, and some sort of ALU (that is, the basic parts of a simple CPU.)

mnemonic symbol: A symbol chosen to assist the human memory, e.g., an abbreviation such as "mpy" for "multiply".

modem: (MOdulator – DEModulator) A device that modulates and demodulates signals transmitted over communication facilities.

MSB: Most significant bit.

multiplex: To interleave or simultaneously transmit two or more messages on a single channel.

multiprocessing: 1. Pertaining to the simultaneous execution of two or more computer programs or sequences of instructions by a computer or computer network. 2. Loosely, parallel processing.

multiprocessor: A computer employing two or more processing units under integrated control.

multiprogramming: Pertaining to the concurrent execution of two or more programs by a computer.

MUX: Multiplexer.

NAND: A logic operator having the property that if P is a statement, Q is a statement, R is a statement, . . . , then the NAND of P, Q, R, . . . is true if at least one statement is false, false if all statements are true. Synonymous with NOT-AND, Sheffer stroke.

nest: To imbed subroutines or data in other subroutines or data at a different hierarchical level such that the different levels of routines or data can be executed or accessed recursively.

noise: Any signal that isn't supposed to be there. Electrical noise may be caused by small, irregular sparks when a switch is opened or closed. Or it may be caused by radio waves or by electric or magnetic fields generated by one wire and picked up by another.

NOR: A logic operator having the property that if P is a statement, Q is a statement, R is a statement, . . . , then the NOR of P, Q, R, . . . is true if all statements are false, false if at least one statement is true. P NOR Q is often represented by a combination of "OR" and "NOT" symbols, such as (P + Q). P NOR Q is also called "neither P nor Q". Synonymous with NOT-or.

NOT: A logic operator having the property that if P is a statement, then the NOT of P is true if P is false, false if P is true. The NOT of P is often represented by P.

object code: Output from a compiler or assembler which is itself executable machine code or is suitable for processing to produce executable machine code.

object language: The language to which a statement is translated.

operand: That which is operated upon. An operand is usually identified by an address part of an instruction.

operating system: Software which controls the execution of computer programs and which may provide scheduling, debugging, input/output control, accounting, compilation, storage assignment, data management, and related services.

operation: 1. A defined action, namely, the act of obtaining a result from one or more operands in accordance with a rule that completely specifies the result for any permissible combination of operands. 2. The set of such acts specified by such a rule, or the rule itself. 3. The act specified by a single computer instruction. 4. A program step undertaken or executed by a computer, e.g., addition, multiplication, extraction, comparison, shift, transfer. The operation is usually specified by the operator part of an instruction. 5. The specific action performed by a logic element.

pack: To compress data in a storage medium by taking advantage of known characteristics of the data, in such a way that the original data can be recovered, e.g., to compress data in a storage medium by making use of bit or byte locations that would otherwise go unused.

parallel operation: The organization of data manipulating within circuitry wherein all the digits of a word are transmitted simultaneously or separate lines in order to speed up operation.

parity check: A check that tests whether the number of ones (or zeros) in an array of binary digits is odd or even. Synonymous with odd-even check.
PC: Program counter.

peripheral equipment: Units which work in conjunction with a computer but are not part of it.

phase: The time interval for each clock "cycle" in a system may be divided into two or more "phases". The phases are defined by pulses in a separate network of wires for each phase. During a particular phase, the signal in that clock network is in the state defined as "active". The clock cycles are repeated over and over again, phase by phase. The phases provide a method of making several things happen in the proper order during one clock cycle.

PLA (programmable logic array): An integrated circuit that employs ROM matrices to combine sum and product terms of logic networks.

positive logic: Logic in which the more-positive voltage represents the "1" state; the less positive voltage represents the "0" state.

priority interrupt: Designation given to method of providing some commands to have precedence over others thus giving one condition of operation priority over another.

problem oriented language: A programming language designed for the convenient expression of a given class of problems.

processor: 1. In hardware, a data processor. 2. In software, a computer program that includes the compiling, assembling, translating, and related functions for a specific programming language, COBOL processor, or FORTRAN processor.

program: 1. A series of actions proposed in order to achieve a certain result. 2. Loosely, a routine. 3. To design, write, and test a program as in definition 1 above. 4. Loosely, to write a routine.

programmable read only memory (PROM): A fixed program, read only, semiconductor memory storage element that can be programmed after packaging.

PROM: Programmable read only memory.

propagation delay: The time required for a change in logic level to be transmitted through an element or a chain of elements.

pulse width: Pulse width, t_w The time interval between specified reference points on the leading and trailing edges of the pulse waveform.

pushdown list: A list that is constructed and maintained so that the item to be retrieved is the most recently stored item in the list, i.e., last in, first out.

pushdown stack: A set of registers which implement a pushdown list.

RAM: Random access memory.

random access memory (RAM): A memory from which all information can be obtained at the output with approximately the same time delay by choosing an address randomly and without first searching through a vast amount of irrelevant data.

read only memory (ROM) A fixed program semiconductor storage element that has been preprogrammed at the factory with a permanent program.

real time: 1. Pertaining to the actual time during which a physical process transpires. 2. Pertaining to the performance of a computation during the actual time that the related physical process transpires, in order that results of the computation can be used in guiding the physical process.

recovery time: Sense Recovery time, t_{SR} The time interval needed to switch a memory from a write mode to a read mode and to obtain valid data signals at the output.

refresh: Method which restores charge on capacitance which deteriorates because of leakage.

register: Temporary storage for digital data.

Relative address: The number that specifies the difference between the absolute address and the base address.

relocate: In computer programming, to move a routine from one portion of storage to another and to adjust the necessary address references so that the routine, in its new location, can be executed.

ROM: Read only memory.

routine: An ordered set of instructions that may have some general or frequent use.

sequencing: Control method used to cause a set of steps to occur in a particular order.

sequential logic systems: Digital system utilizing memory elements.

serial: 1. Pertaining to the sequential or consecutive occurrence of two or more related activities in a single device or channel. 2. Pertaining to the sequencing of two or more processes. 3. Pertaining to the sequential processing of the individual parts of a whole such as the bits of a character or the characters of a word, using the same facilities of successive parts. 4. Contrast with parallel.

serial operation: The organization of data manipulation within circuitry wherein the digits of a word are transmitted one at a time along a single line. The serial mode of operation is slower than parallel operation, but utilizes less complex circuitry.

set-up time: The minimum amount of time that data must be present at an input to ensure data acceptance when the device is clocked.

shift: A movement of data to the right or left.

shift register: A register in which the stored data can be moved to the right or left.

sign-and magnitude notation: A system of notation where binary numbers are represented by a sign-bit and one or more number bits:

significant digit: A digit that is needed for a certain purpose, particularly one that must be kept to preserve a specific accuracy or presicion.

sign position: A position, normally located at one end of a number, that contains an indication of the algebraic sign of the number.

simulate: 1. To represent certain features of the behavior of a physical or abstract system by the behavior of another system. 2. To represent the functioning of a device, system, computer program by another, e.g., to represent the functioning of one computer by another, to represent the behavior of a physical system by the execution of a computer program, to represent a biological system by a mathematical model.

simulator: A device, system, or computer program that represents certain features of the behavior of a physical or abstract system.

software: A set of computer programs, procedures, and possibly associated documentation concerned with the operation of a data processing system, e.g., compilers, library routines, manuals, circuit diagrams.

source language: The language from which a statement is translated.

source program: A computer program written in a source language.

state: The condition of an input or output of a circuit as to whether it is a logic "1" or a logic "0". The state of a circuit (gate or flip-flop) refers to its output. A flip-flop is said to be in the "1" state when its Q output is "1". A gate is in the "1" state when its output is "1".

static storage elements: Storage elements which contain storage cells that retain their information as long as power is applied unless the information is altered by external excitation.

G **stored program:** A set of instructions in memory specifying the operations to be performed.

stored program computer: A computer controlled by internally stored instructions that can synthesize, store, and in some cases alter instruction as though they were data and that can subsequently execute these instructions.

subroutine: A routine that can be part of another routine.

synchronous: Refers to two or more things made to happen in a system at the same time, by means of a common clock signal.

temporary storage: In programming, storage locations reserved for intermediate results. Synonymous with working storage.

terminal: A point in a system or communication network at which data can either enter or leave.

transmit: To send data from one location and to receive the data at another location. Synonymous with transfer definition 2, move.

TTL: Bipolar semiconductor transistor-transistor coupled logic circuits.

USASCII: United States of America Standard Code for Information Interchange. The standard code used by the United State for transmission of data. Sometimes simply referred to as the "as'ki" code.

variable: A quantity that can assume any of a given set of values.

volatile storage: A storage device in which stored data are lost when the applied power is removed.

word: A character string or a bit string considered as an entity

working storage: Same as temporary storage.

WR: Working register.

workspace: In the 9900, a set of 16 consecutive words of memory referred to by many of the instructions.

write: To record data in a storage device or a data medium. The recording need not be permanent, such as the writing on a cathode ray tube display device.

Table I. Hexadecimal Arithmetic

ADDITION TABLE

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

MULTIPLICATION TABLE

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

A

Table 2. Table of Powers of 16_{10}

16^n	n	16^{-n}
1	0	0.10000 00000 00000 00000 x 10
16	1	0.62500 00000 00000 00000 x 10^{-1}
256	2	0.39062 50000 00000 00000 x 10^{-2}
4 096	3	0.24414 06250 00000 00000 x 10^{-3}
65 536	4	0.15258 78906 25000 00000 x 10^{-4}
1 048 576	5	0.95367 43164 06250 00000 x 10^{-6}
16 777 216	6	0.59604 64477 53906 25000 x 10^{-7}
268 435 456	7	0.37252 90298 46191 40625 x 10^{-8}
4 294 967 296	8	0.23283 06436 53869 62891 x 10^{-9}
68 719 476 736	9	0.14551 91522 83668 51807 x 10^{-10}
1 099 511 627 776	10	0.90949 47017 72928 23792 x 10^{-12}
17 592 186 044 416	11	0.56843 41886 08080 14870 x 10^{-13}
281 474 976 710 656	12	0.35527 13678 80050 09294 x 10^{-14}
4 503 599 627 370 496	13	0.22204 46049 25031 30808 x 10^{-15}
72 057 594 037 927 936	14	0.13877 78780 78144 56755 x 10^{-16}
1 152 921 504 606 846 976	15	0.86736 17379 88403 54721 x 10^{-18}

Table 3. Table of Powers of 10_{16}

10^n	n	10^{-n}
1	0	1.0000 0000 0000 0000
A	1	0.1999 9999 9999 999A
64	2	0.28F5 C28F 5C28 F5C3 x 16^{-1}
3E8	3	0.4189 374B C6A7 EF9E x 16^{-2}
2710	4	0.68DB 8BAC 710C B296 x 16^{-3}
1 86A0	5	0.A7C5 AC47 1B47 8423 x 16^{-4}
F 4240	6	0.10C6 F7A0 B5ED 8D37 x 16^{-4}
98 9680	7	0.1AD7 F29A BCAF 4858 x 16^{-5}
5F5 E100	8	0.2AF3 1DC4 6118 73BF x 16^{-6}
3B9A CA00	9	0.44B8 2FA0 9B5A 52CC x 16^{-7}
2 540B E400	10	0.6DF3 7F67 5EF6 EADF x 16^{-8}
17 4876 E800	11	0.AFEB FF0B CB24 AAFF x 16^{-9}
E8 D4A5 1000	12	0.1197 9981 2DEA 1119 x 16^{-9}
918 4E72 A000	13	0.1C25 C268 4976 81C2 x 16^{-10}
5AF3 107A 4000	14	0.2D09 370D 4257 3604 x 16^{-11}
3 8D7E A4C6 8000	15	0.480E BE7B 9D58 566D x 16^{-12}
23 86F2 6FC1 0000	16	0.734A CA5F 6226 F0AE x 16^{-13}
163 4578 5D8A 0000	17	0.B877 AA32 36A4 B449 x 16^{-14}
DE0 B6B3 A764 0000	18	0.1272 5DD1 D243 ABA1 x 16^{-14}
8AC7 2304 89E8 0000	19	0.1D83 C94F B6D2 AC35 x 16^{-15}

A

Table 4. Table of Powers of Two

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5

▶A

Table 5. Hexadecimal–Decimal Integer Conversion Table

The table appearing on the following pages provides a means for direct conversion of decimal integers in the range of 0 to 4095 and for hexadecimal integers in the range of 0 to FFF.

To convert numbers above those ranges, add table values to the figures below:

Hexadecimal	Decimal	Hexadecimal	Decimal
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

Table 5. Hexadecimal–Decimal Integer Conversion Table (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
180	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
200	0512	0513	0514	0515	0516	0517	0518	0519	0529	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767

A

Table 5. Hexadecimal–Decimal Integer Conversion Table (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
400	1024	1025	0126	0127	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1291	1293	1294	1295
510	1296	1297	1298	1299	1399	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1329	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1367	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1429	1421	1422	1423
590	1324	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
3B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1515	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

A

Table 5. Hexadecimal–Decimal Integer Conversion Table (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1592	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	17231	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	8102	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1818	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1909	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303

A

Table 5. Hexadecimal–Decimal Integer Conversion Table (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	3496	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	24351	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2479	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2626	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

A◄

Table 5. Hexadecimal–Decimal Integer Conversion Table (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E00	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775

▶A

Table 5. Hexadecimal–Decimal Integer Conversion Table (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F40	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

A◀

Table 6. Hexadecimal–Decimal Fraction Conversion Table

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.40 00 00 00	.25000 00000	.80 00 00 00	.50000 00000	.C0 00 00 00	.75000 00000
.01 00 00 00	.00390 62500	.41 00 00 00	.25390 62500	.81 00 00 00	.50390 62500	.C1 00 00 00	.75390 62500
.02 00 00 00	.00781 25000	.42 00 00 00	.25781 25000	.82 00 00 00	.50781 25000	.C2 00 00 00	.75781 25000
.03 00 00 00	.01171 87500	.43 00 00 00	.26171 87500	.83 00 00 00	.51171 87500	.C3 00 00 00	.76171 87500
.04 00 00 00	.01562 50000	.44 00 00 00	.26562 50000	.84 00 00 00	.51562 50000	.C4 00 00 00	.76562 50000
.05 00 00 00	.01953 12500	.45 00 00 00	.26953 12500	.85 00 00 00	.51953 12500	.C5 00 00 00	.76953 12500
.06 00 00 00	.02343 75000	.46 00 00 00	.27343 75000	.86 00 00 00	.52343 75000	.C6 00 00 00	.77343 75000
.07 00 00 00	.02734 37500	.47 00 00 00	.27734 37500	.87 00 00 00	.52734 37500	.C7 00 00 00	.77734 37500
.08 00 00 00	.03125 00000	.48 00 00 00	.28125 00000	.88 00 00 00	.53125 00000	.C8 00 00 00	.78125 00000
.09 00 00 00	.03515 62500	.49 00 00 00	.28515 62500	.89 00 00 00	.53515 62500	.C9 00 00 00	.78515 62500
.0A 00 00 00	.03906 25000	.4A 00 00 00	.28906 25000	.8A 00 00 00	.53906 25000	.CA 00 00 00	.78906 25000
.0B 00 00 00	.04296 87500	.4B 00 00 00	.29296 87500	.8B 00 00 00	.54296 87500	.CB 00 00 00	.79296 87500
.0C 00 00 00	.04687 50000	.4C 00 00 00	.29687 50000	.8C 00 00 00	.54687 50000	.CC 00 00 00	.79687 50000
.0D 00 00 00	.05078 12500	.4D 00 00 00	.30078 12500	.8D 00 00 00	.55078 12500	.CD 00 00 00	.80078 12500
.0E 00 00 00	.05468 75000	.4E 00 00 00	.30468 75000	.8E 00 00 00	.55468 75000	.CE 00 00 00	.80468 75000
.0F 00 00 00	.05859 37500	.4F 00 00 00	.30859 37500	.8F 00 00 00	.55859 37500	.CF 00 00 00	.80859 37500
.10 00 00 00	.06250 00000	.50 00 00 00	.31250 00000	.90 00 00 00	.56250 00000	.D0 00 00 00	.81250 00000
.11 00 00 00	.06640 62500	.51 00 00 00	.31640 62500	.91 00 00 00	.56640 62500	.D1 00 00 00	.81640 62500
.12 00 00 00	.07031 25000	.52 00 00 00	.32031 25000	.92 00 00 00	.57031 25000	.D2 00 00 00	.82031 25000
.13 00 00 00	.07421 87500	.53 00 00 00	.32421 87500	.93 00 00 00	.57421 87500	.D3 00 00 00	.82421 87500
.14 00 00 00	.07812 50000	.54 00 00 00	.32812 50000	.94 00 00 00	.57812 50000	.D4 00 00 00	.82812 50000
.15 00 00 00	.08203 12500	.55 00 00 00	.33203 12500	.95 00 00 00	.58203 12500	.D5 00 00 00	.83203 12500
.16 00 00 00	.08593 75000	.56 00 00 00	.33593 75000	.96 00 00 00	.58593 75000	.D6 00 00 00	.83593 75000
.17 00 00 00	.08984 37500	.57 00 00 00	.33984 37500	.97 00 00 00	.58984 37500	.D7 00 00 00	.83984 37500
.18 00 00 00	.09375 00000	.58 00 00 00	.34375 00000	.98 00 00 00	.59375 00000	.D8 00 00 00	.84375 00000
.19 00 00 00	.09765 62500	.59 00 00 00	.34765 62500	.99 00 00 00	.59765 62500	.D9 00 00 00	.84765 62500
.1A 00 00 00	.10156 25000	.5A 00 00 00	.35156 25000	.9A 00 00 00	.60156 25000	.DA 00 00 00	.85156 25000
.1B 00 00 00	.10546 87500	.5B 00 00 00	.35546 87500	.9B 00 00 00	.60546 87500	.DB 00 00 00	.85546 87500
.1C 00 00 00	.10937 50000	.5C 00 00 00	.35937 50000	.9C 00 00 00	.60937 50000	.DC 00 00 00	.85937 50000
.1D 00 00 00	.11328 12500	.5D 00 00 00	.36328 12500	.9D 00 00 00	.61328 12500	.DD 00 00 00	.86328 12500
.1E 00 00 00	.11718 75000	.5E 00 00 00	.36718 75000	.9E 00 00 00	.61718 75000	.DE 00 00 00	.86718 75000
.1F 00 00 00	.12109 37500	.5F 00 00 00	.37109 37500	.9F 00 00 00	.62109 37500	.DF 00 00 00	.87109 37500
.20 00 00 00	.12500 00000	.60 00 00 00	.37500 00000	.A0 00 00 00	.62500 00000	.E0 00 00 00	.87500 00000
.21 00 00 00	.12890 62500	.61 00 00 00	.37890 62500	.A1 00 00 00	.62890 62500	.E1 00 00 00	.87890 62500
.22 00 00 00	.13281 25000	.62 00 00 00	.38281 25000	.A2 00 00 00	.63281 25000	.E2 00 00 00	.88281 25000
.23 00 00 00	.13671 87500	.63 00 00 00	.38671 87500	.A3 00 00 00	.63671 87500	.E3 00 00 00	.88671 87500
.24 00 00 00	.14062 50000	.64 00 00 00	.39062 50000	.A4 00 00 00	.64062 50000	.E4 00 00 00	.89062 50000
.25 00 00 00	.14453 12500	.65 00 00 00	.39453 12500	.A5 00 00 00	.64453 12500	.E5 00 00 00	.89453 12500
.26 00 00 00	.14843 75000	.66 00 00 00	.39843 75000	.A6 00 00 00	.64843 75000	.E6 00 00 00	.89843 75000
.27 00 00 00	.15234 37500	.67 00 00 00	.40234 37500	.A7 00 00 00	.65234 37500	.E7 00 00 00	.90234 37500
.28 00 00 00	.15625 00000	.68 00 00 00	.40625 00000	.A8 00 00 00	.65625 00000	.E8 00 00 00	.90625 00000
.29 00 00 00	.16015 62500	.69 00 00 00	.41015 62500	.A9 00 00 00	.66015 62500	.E9 00 00 00	.91015 62500
.2A 00 00 00	.16406 25000	.6A 00 00 00	.41406 25000	.AA 00 00 00	.66406 25000	.EA 00 00 00	.91406 25000
.2B 00 00 00	.16796 87500	.6B 00 00 00	.41796 87500	.AB 00 00 00	.66796 87500	.EB 00 00 00	.91796 87500
.2C 00 00 00	.17187 50000	.6C 00 00 00	.42187 50000	.AC 00 00 00	.67187 50000	.EC 00 00 00	.92187 50000
.2D 00 00 00	.17578 12500	.6D 00 00 00	.42578 12500	.AD 00 00 00	.67578 12500	.ED 00 00 00	.92578 12500
.2E 00 00 00	.17968 75000	.6E 00 00 00	.42968 75000	.AE 00 00 00	.67968 75000	.EE 00 00 00	.92968 75000
.2F 00 00 00	.18359 37500	.6F 00 00 00	.43359 37500	.AF 00 00 00	.68359 37500	.EF 00 00 00	.93359 37500
.30 00 00 00	.18750 00000	.70 00 00 00	.43750 00000	.B0 00 00 00	.68750 00000	.F0 00 00 00	.93750 00000
.31 00 00 00	.19140 62500	.71 00 00 00	.44140 62500	.B1 00 00 00	.69140 62500	.F1 00 00 00	.94140 62500
.32 00 00 00	.19531 25000	.72 00 00 00	.44531 25000	.B2 00 00 00	.69531 25000	.F2 00 00 00	.94531 25000
.33 00 00 00	.19921 87500	.73 00 00 00	.44921 87500	.B3 00 00 00	.69921 87500	.F3 00 00 00	.94921 87500
.34 00 00 00	.20312 50000	.74 00 00 00	.45312 50000	.B4 00 00 00	.70312 50000	.F4 00 00 00	.95312 50000
.35 00 00 00	.20703 12500	.75 00 00 00	.45703 12500	.B5 00 00 00	.70703 12500	.F5 00 00 00	.95703 12500
.36 00 00 00	.21093 75000	.76 00 00 00	.46093 75000	.B6 00 00 00	.71093 75000	.F6 00 00 00	.96093 75000
.37 00 00 00	.21484 37500	.77 00 00 00	.46484 37500	.B7 00 00 00	.71484 37500	.F7 00 00 00	.96484 37500
.38 00 00 00	.21875 00000	.78 00 00 00	.46875 00000	.B8 00 00 00	.71875 00000	.F8 00 00 00	.96875 00000
.39 00 00 00	.22265 62500	.79 00 00 00	.47265 62500	.B9 00 00 00	.72265 62500	.F9 00 00 00	.97265 62500
.3A 00 00 00	.22656 25000	.7A 00 00 00	.47656 25000	.BA 00 00 00	.72656 25000	.FA 00 00 00	.97656 25000
.3B 00 00 00	.23046 87500	.7B 00 00 00	.48046 87500	.BB 00 00 00	.73046 87500	.FB 00 00 00	.98046 87500
.3C 00 00 00	.23437 50000	.7C 00 00 00	.48437 50000	.BC 00 00 00	.73437 50000	.FC 00 00 00	.98437 50000
.3D 00 00 00	.23828 12500	.7D 00 00 00	.48828 12500	.BD 00 00 00	.73828 12500	.FD 00 00 00	.98828 12500
.3E 00 00 00	.24218 75000	.7E 00 00 00	.49218 75000	.BE 00 00 00	.74218 75000	.FE 00 00 00	.99218 75000
.3F 00 00 00	.24609 37500	.7F 00 00 00	.49609 37500	.BF 00 00 00	.74609 37500	.FF 00 00 00	.99609 37500

A

Table 6. Hexadecimal–Decimal Fraction Conversion Table (Cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 40 00 00	.00097 65625	.00 80 00 00	.00195 31250	.00 C0 00 00	.00292 96875
.00 01 00 00	.00001 52587	.00 41 00 00	.00099 18212	.00 81 00 00	.00196 83837	.00 C1 00 00	.00294 49462
.00 02 00 00	.00003 05175	.00 42 00 00	.00100 70800	.00 82 00 00	.00198 36425	.00 C2 00 00	.00296 02050
.00 03 00 00	.00004 57763	.00 43 00 00	.00102 23388	.00 83 00 00	.00199 89013	.00 C3 00 00	.00297 54638
.00 04 00 00	.00006 10351	.00 44 00 00	.00103 75976	.00 84 00 00	.00201 41601	.00 C4 00 00	.00299 07226
.00 05 00 00	.00007 62939	.00 45 00 00	.00105 28564	.00 85 00 00	.00202 94189	.00 C5 00 00	.00300 59814
.00 06 00 00	.00009 15527	.00 46 00 00	.00106 81152	.00 86 00 00	.00204 46777	.00 C6 00 00	.00302 12402
.00 07 00 00	.00010 68115	.00 47 00 00	.00108 33740	.00 87 00 00	.00205 99365	.00 C7 00 00	.00303 64990
.00 08 00 00	.00012 20703	.00 48 00 00	.00109 86328	.00 88 00 00	.00207 51953	.00 C8 00 00	.00305 17578
.00 09 00 00	.00013 73291	.00 49 00 00	.00111 38916	.00 89 00 00	.00209 04541	.00 C9 00 00	.00306 70166
.00 0A 00 00	.00015 25878	.00 4A 00 00	.00112 91503	.00 8A 00 00	.00210 57128	.00 CA 00 00	.00308 22753
.00 0B 00 00	.00016 78466	.00 4B 00 00	.00114 44091	.00 8B 00 00	.00212 09716	.00 CB 00 00	.00309 75341
.00 0C 00 00	.00018 31054	.00 4C 00 00	.00115 96679	.00 8C 00 00	.00213 62304	.00 CC 00 00	.00311 27929
.00 0D 00 00	.00019 83642	.00 4D 00 00	.00117 49267	.00 8D 00 00	.00215 14892	.00 CD 00 00	.00312 80517
.00 0E 00 00	.00021 36230	.00 4E 00 00	.00119 01855	.00 8E 00 00	.00216 67480	.00 CE 00 00	.00314 33105
.00 0F 00 00	.00022 88818	.00 4F 00 00	.00120 54443	.00 8F 00 00	.00218 20068	.00 CF 00 00	.00315 85693
.00 10 00 00	.00024 41406	.00 50 00 00	.00122 07031	.00 90 00 00	.00219 72656	.00 D0 00 00	.00317 38281
.00 11 00 00	.00025 93994	.00 51 00 00	.00123 59619	.00 91 00 00	.00221 25244	.00 D1 00 00	.00318 90869
.00 12 00 00	.00027 46582	.00 52 00 00	.00125 12207	.00 92 00 00	.00222 77832	.00 D2 00 00	.00320 43457
.00 13 00 00	.00028 99169	.00 53 00 00	.00126 64794	.00 93 00 00	.00224 30419	.00 D3 00 00	.00321 96044
.00 14 00 00	.00030 51757	.00 54 00 00	.00128 17382	.00 94 00 00	.00225 83007	.00 D4 00 00	.00323 48632
.00 15 00 00	.00032 04345	.00 55 00 00	.00129 69970	.00 95 00 00	.00227 35595	.00 D5 00 00	.00325 01220
.00 16 00 00	.00033 56933	.00 56 00 00	.00131 22558	.00 96 00 00	.00228 88183	.00 D6 00 00	.00326 53808
.00 17 00 00	.00035 09521	.00 57 00 00	.00132 75146	.00 97 00 00	.00230 40771	.00 D7 00 00	.00328 06396
.00 18 00 00	.00036 62109	.00 58 00 00	.00134 27734	.00 98 00 00	.00231 93359	.00 D8 00 00	.00329 58984
.00 19 00 00	.00038 14697	.00 59 00 00	.00135 80322	.00 99 00 00	.00233 45947	.00 D9 00 00	.00331 11572
.00 1A 00 00	.00039 67285	.00 5A 00 00	.00137 32910	.00 9A 00 00	.00234 98535	.00 DA 00 00	.00332 64160
.00 1B 00 00	.00041 19873	.00 5B 00 00	.00138 85498	.00 9B 00 00	.00236 51123	.00 DB 00 00	.00334 16748
.00 1C 00 00	.00042 72460	.00 5C 00 00	.00140 38085	.00 9C 00 00	.00238 03710	.00 DC 00 00	.00335 69335
.00 1D 00 00	.00044 25048	.00 5D 00 00	.00141 90673	.00 9D 00 00	.00239 56298	.00 DD 00 00	.00337 21923
.00 1E 00 00	.00045 77636	.00 5E 00 00	.00143 43261	.00 9E 00 00	.00241 08886	.00 DE 00 00	.00338 74511
.00 1F 00 00	.00047 30224	.00 5F 00 00	.00144 95849	.00 9F 00 00	.00242 61474	.00 DF 00 00	.00340 27099
.00 20 00 00	.00048 82812	.00 60 00 00	.00146 48437	.00 A0 00 00	.00244 14062	.00 E0 00 00	.00341 79687
.00 21 00 00	.00050 35400	.00 61 00 00	.00148 01025	.00 A1 00 00	.00245 66650	.00 E1 00 00	.00343 32275
.00 22 00 00	.00051 87988	.00 62 00 00	.00149 53613	.00 A2 00 00	.00247 19238	.00 E2 00 00	.00344 84863
.00 23 00 00	.00053 40576	.00 63 00 00	.00151 06201	.00 A3 00 00	.00248 71826	.00 E3 00 00	.00346 37451
.00 24 00 00	.00054 93164	.00 64 00 00	.00152 58789	.00 A4 00 00	.00250 24414	.00 E4 00 00	.00347 90039
.00 25 00 00	.00056 45751	.00 65 00 00	.00154 11376	.00 A5 00 00	.00251 77001	.00 E5 00 00	.00349 42626
.00 26 00 00	.00057 98339	.00 66 00 00	.00155 63964	.00 A6 00 00	.00253 29589	.00 E6 00 00	.00350 95214
.00 27 00 00	.00059 50927	.00 67 00 00	.00157 16552	.00 A7 00 00	.00254 82177	.00 E7 00 00	.00352 47802
.00 28 00 00	.00061 03515	.00 68 00 00	.00158 69140	.00 A8 00 00	.00256 34765	.00 E8 00 00	.00354 00390
.00 29 00 00	.00062 56103	.00 69 00 00	.00160 21728	.00 A9 00 00	.00257 87353	.00 E9 00 00	.00355 52978
.00 2A 00 00	.00064 08691	.00 6A 00 00	.00161 74316	.00 AA 00 00	.00259 39941	.00 EA 00 00	.00357 05566
.00 2B 00 00	.00065 61279	.00 6B 00 00	.00163 26904	.00 AB 00 00	.00260 92529	.00 EB 00 00	.00358 58154
.00 2C 00 00	.00067 13867	.00 6C 00 00	.00164 79492	.00 AC 00 00	.00262 45117	.00 EC 00 00	.00360 10742
.00 2D 00 00	.00068 66455	.00 6D 00 00	.00166 32080	.00 AD 00 00	.00263 97705	.00 ED 00 00	.00361 63330
.00 2E 00 00	.00070 19042	.00 6E 00 00	.00167 84667	.00 AE 00 00	.00265 50292	.00 EE 00 00	.00363 15917
.00 2F 00 00	.00071 71630	.00 6F 00 00	.00169 37255	.00 AF 00 00	.00267 02880	.00 EF 00 00	.00364 68505
.00 30 00 00	.00073 24218	.00 70 00 00	.00170 89843	.00 B0 00 00	.00268 55468	.00 F0 00 00	.00366 21093
.00 31 00 00	.00074 76806	.00 71 00 00	.00172 42421	.00 B1 00 00	.00270 08056	.00 F1 00 00	.00367 73681
.00 32 00 00	.00076 29394	.00 72 00 00	.00173 95019	.00 B2 00 00	.00271 60644	.00 F2 00 00	.00369 26269
.00 33 00 00	.00077 81982	.00 73 00 00	.00175 47607	.00 B3 00 00	.00273 13232	.00 F3 00 00	.00370 78857
.00 34 00 00	.00079 34570	.00 74 00 00	.00177 00195	.00 B4 00 00	.00274 65820	.00 F4 00 00	.00372 31445
.00 35 00 00	.00080 87158	.00 75 00 00	.00178 52783	.00 B5 00 00	.00276 18408	.00 F5 00 00	.00373 84033
.00 36 00 00	.00082 39746	.00 76 00 00	.00180 05371	.00 B6 00 00	.00277 70996	.00 F6 00 00	.00375 36621
.00 37 00 00	.00083 92333	.00 77 00 00	.00181 57958	.00 B7 00 00	.00279 23583	.00 F7 00 00	.00376 89208
.00 38 00 00	.00085 44921	.00 78 00 00	.00183 10546	.00 B8 00 00	.00280 76171	.00 F8 00 00	.00378 41796
.00 39 00 00	.00086 97509	.00 79 00 00	.00184 63134	.00 B9 00 00	.00282 28759	.00 F9 00 00	.00379 94384
.00 3A 00 00	.00088 50097	.00 7A 00 00	.00186 15722	.00 BA 00 00	.00283 81347	.00 FA 00 00	.00381 46972
.00 3B 00 00	.00090 02685	.00 7B 00 00	.00187 68310	.00 BB 00 00	.00285 33935	.00 FB 00 00	.00382 99560
.00 3C 00 00	.00091 55273	.00 7C 00 00	.00189 20898	.00 BC 00 00	.00286 86523	.00 FC 00 00	.00384 52148
.00 3D 00 00	.00093 07861	.00 7D 00 00	.00190 73486	.00 BD 00 00	.00288 39111	.00 FD 00 00	.00386 04736
.00 3E 00 00	.00094 60449	.00 7E 00 00	.00192 26074	.00 BE 00 00	.00289 91699	.00 FE 00 00	.00387 57324
.00 3F 00 00	.00096 13037	.00 7F 00 00	.00193 78662	.00 BF 00 00	.00291 44287	.00 FF 00 00	.00389 09912

A

Table 6. Hexadecimal–Decimal Fraction Conversion Table (Cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 00 40 00	.00000 38146	.00 00 80 00	.00000 76293	.00 00 C0 00	.00001 14440
.00 00 01 00	.00000 00596	.00 00 41 00	.00000 38743	.00 00 81 00	.00000 76889	.00 00 C1 00	.00001 15036
.00 00 02 00	.00000 01192	.00 00 42 00	.00000 39339	.00 00 82 00	.00000 77486	.00 00 C2 00	.00001 15633
.00 00 03 00	.00000 01788	.00 00 43 00	.00000 39935	.00 00 83 00	.00000 78082	.00 00 C3 00	.00001 16229
.00 00 04 00	.00000 02384	.00 00 44 00	.00000 40531	.00 00 84 00	.00000 78678	.00 00 C4 00	.00001 16825
.00 00 05 00	.00000 02980	.00 00 45 00	.00000 41127	.00 00 85 00	.00000 79274	.00 00 C5 00	.00001 17421
.00 00 06 00	.00000 03576	.00 00 46 00	.00000 41723	.00 00 86 00	.00000 79870	.00 00 C6 00	.00001 18017
.00 00 07 00	.00000 04172	.00 00 47 00	.00000 42319	.00 00 87 00	.00000 80466	.00 00 C7 00	.00001 18613
.00 00 08 00	.00000 04768	.00 00 48 00	.00000 42915	.00 00 88 00	.00000 81062	.00 00 C8 00	.00001 19209
.00 00 09 00	.00000 05364	00 00 49 00	.00000 43511	.00 00 89 00	.00000 81658	.00 00 C9 00	.00001 19805
.00 00 0A 00	.00000 05960	.00 00 4A 00	.00000 44107	.00 00 8A 00	.00000 82254	.00 00 CA 00	.00001 20401
.00 00 0B 00	.00000 06556	.00 00 4B 00	.00000 44703	.00 00 8B 00	.00000 82850	.00 00 CB 00	.00001 20997
.00 00 0C 00	.00000 07152	.00 00 4C 00	.00000 45299	.00 00 8C 00	.00000 83446	.00 00 CC 00	.00001 21593
.00 00 0D 00	.00000 07748	.00 00 4D 00	.00000 45895	.00 00 8D 00	.00000 84042	.00 00 CD 00	.00001 22189
.00 00 0E 00	.00000 08344	.00 00 4E 00	.00000 46491	.00 00 8E 00	.00000 84638	.00 00 CE 00	.00001 22785
.00 00 0F 00	.00000 08940	.00 00 4F 00	.00000 47087	.00 00 8F 00	.00000 85234	.00 00 CF 00	.00001 23381
.00 00 10 00	.00000 09536	.00 00 50 00	.00000 47683	.00 00 90 00	.00000 85830	.00 00 D0 00	.00001 23977
.00 00 11 00	.00000 10132	.00 00 51 00	.00000 48279	.00 00 91 00	.00000 86426	.00 00 D1 00	.00001 24573
.00 00 12 00	.00000 10728	.00 00 52 00	.00000 48875	.00 00 92 00	.00000 87022	.00 00 D2 00	.00001 25169
.00 00 13 00	.00000 11324	.00 00 53 00	.00000 49471	.00 00 93 00	.00000 87618	.00 00 D3 00	.00001 25765
.00 00 14 00	.00000 11920	.00 00 54 00	.00000 50067	.00 00 94 00	.00000 88214	.00 00 D4 00	.00001 26361
.00 00 15 00	.00000 12516	.00 00 55 00	.00000 50663	.00 00 95 00	.00000 88810	.00 00 D5 00	.00001 26957
.00 00 16 00	.00000 13113	.00 00 56 00	.00000 51259	.00 00 96 00	.00000 89406	.00 00 D6 00	.00001 27553
.00 00 17 00	.00000 13709	.00 00 57 00	.00000 51856	.00 00 97 00	.00000 90003	.00 00 D7 00	.00001 28149
.00 00 18 00	.00000 14305	.00 00 58 00	.00000 52452	.00 00 98 00	.00000 90599	.00 00 D8 00	.00001 28746
.00 00 19 00	.00000 14901	.00 00 59 00	.00000 53048	.00 00 99 00	.00000 91195	.00 00 D9 00	.00001 29342
.00 00 1A 00	.00000 15497	.00 00 5A 00	.00000 53644	.00 00 9A 00	.00000 91791	.00 00 DA00	.00001 29938
.00 00 1B 00	.00000 16093	.00 00 5B 00	.00000 54240	.00 00 9B 00	.00000 92387	.00 00 DB 00	.00001 30534
.00 00 1C 00	.00000 16689	.00 00 5C 00	.00000 54836	.00 00 9C 00	.00000 92983	.00 00 DC 00	.00001 31130
.00 00 1D 00	.00000 17285	.00 00 5D 00	.00000 55432	.00 00 9D 00	.00000 93579	.00 00 DD 00	.00001 31726
.00 00 1E 00	.00000 17881	.00 00 5E 00	.00000 56028	.00 00 9E 00	.00000 94175	.00 00 DE 00	.00001 32322
.00 00 1F 00	.00000 18477	.00 00 5F 00	.00000 56624	.00 00 9F 00	.00000 94771	.00 00 DF 00	.00001 32918
.00 00 20 00	.00000 19073	.00 00 60 00	.00000 57220	.00 00 A0 00	.00000 95367	.00 00 E0 00	.00001 33514
.00 00 21 00	.00000 19669	.00 00 61 00	.00000 57816	.00 00 A1 00	.00000 95963	.00 00 E1 00	.00001 34110
.00 00 22 00	.00000 20265	.00 00 62 00	.00000 58412	.00 00 A2 00	.00000 96559	.00 00 E2 00	.00001 34706
.00 00 23 00	.00000 20861	.00 00 63 00	.00000 59008	.00 00 A3 00	.00000 97155	.00 00 E3 00	.00001 35302
.00 00 24 00	.00000 21457	.00 00 64 00	.00000 59604	.00 00 A4 00	.00000 97751	.00 00 E4 00	.00001 35898
.00 00 25 00	.00000 22053	.00 00 65 00	.00000 60200	.00 00 A5 00	.00000 98347	.00 00 E5 00	.00001 36494
.00 00 26 00	.00000 22649	.00 00 66 00	.00000 60796	.00 00 A6 00	.00000 98943	.00 00 E6 00	.00001 37090
.00 00 27 00	.00000 23245	.00 00 67 00	.00000 61392	.00 00 A7 00	.00000 99539	.00 00 E7 00	.00001 37686
.00 00 28 00	.00000 23841	.00 00 68 00	.00000 61988	.00 00 A8 00	.00001 00135	.00 00 E8 00	.00001 38282
.00 00 29 00	.00000 24437	.00 00 69 00	.00000 62584	.00 00 A9 00	.00001 00731	.00 00 E9 00	.00001 38878
.00 00 2A 00	.00000 25033	.00 00 6A 00	.00000 63180	.00 00 AA 00	.00001 01327	.00 00 EA 00	.00001 39474
.00 00 2B 00	.00000 25629	.00 00 6B 00	.00000 63776	.00 00 AB 00	.00001 01923	.00 00 EB 00	.00001 40070
.00 00 2C 00	.00000 26226	.00 00 6C 00	.00000 64373	.00 00 AC 00	.00001 02519	.00 00 EC 00	.00001 40666
.00 00 2D 00	.00000 26822	.00 00 6D 00	.00000 64969	.00 00 AD 00	.00001 03116	.00 00 ED 00	.00001 41263
.00 00 2E 00	.00000 27418	.00 00 6E 00	.00000 65565	.00 00 AE 00	.00001 03712	.00 00 EE 00	.00001 41859
.00 00 2F 00	.00000 28014	.00 00 6F 00	.00000 61661	.00 00 AF 00	.00001 04308	.00 00 EF 00	.00001 42455
.00 00 30 00	.00000 28610	.00 00 70 00	.00000 66757	.00 00 B0 00	.00001 04904	.00 00 F0 00	.00001 43051
.00 00 31 00	.00000 29206	.00 00 71 00	.00000 67353	.00 00 B1 00	.00001 05500	.00 00 F1 00	.00001 43647
.00 00 32 00	.00000 29802	.00 00 72 00	.00000 67949	.00 00 B2 00	.00001 06096	.00 00 F2 00	.00001 44243
.00 00 33 00	.00000 30398	.00 00 73 00	.00000 68545	.00 00 B3 00	.00001 06692	.00 00 F3 00	.00001 44839
.00 00 34 00	.00000 30994	.00 00 74 00	.00000 69141	.00 00 B4 00	.00001 07228	.00 00 F4 00	.00001 45435
.00 00 35 00	.00000 31590	.00 00 75 00	.00000 69737	.00 00 B5 00	.00001 07884	.00 00 F5 00	.00001 46031
.00 00 36 00	.00000 32186	.00 00 76 00	.00000 70333	.00 00 B6 00	.00001 08480	.00 00 F6 00	.00001 46627
.00 00 37 00	.00000 32782	.00 00 77 00	.00000 70929	.00 00 B7 00	.00001 09076	.00 00 F7 00	.00001 47223
.00 00 38 00	.00000 33378	.00 00 78 00	.00000 71525	.00 00 B8 00	.00001 09672	.00 00 F8 00	.00001 47819
.00 00 39 00	.00000 33974	.00 00 79 00	.00000 75121	.00 00 B9 00	.00001 10268	.00 00 F9 00	.00001 48415
.00 00 3A 00	.00000 34570	.00 00 7A 00	.00000 72717	.00 00 BA 00	.00001 10864	.00 00 FA 00	.00001 49011
.00 00 3B 00	.00000 35166	.00 00 7B 00	.00000 73313	.00 00 BB 00	.00001 11460	.00 00 FB 00	.00001 49607
.00 00 3C 00	.00000 35762	.00 00 7C 00	.00000 73909	.00 00 BC 00	.00001 12056	.00 00 FC 00	.00001 50203
.00 00 3D 00	.00000 36358	.00 00 7D 00	.00000 74505	.00 00 BD 00	.00001 12652	.00 00 ED 00	.00001 50799
.00 00 3E 00	.00000 36954	.00 00 7E 00	.00000 75101	.00 00 BE 00	.00001 13248	.00 00 FE 00	.00001 51395
.00 00 3F 00	.00000 37550	.00 00 7F 00	.00000 75697	.00 00 BF 00	.00001 13844	.00 00 FF 00	.00001 51991

▶ A

Table 6. Hexadecimal–Decimal Fraction Conversion Table (Cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 00 00 40	.00000 00149	.00 00 00 80	.00000 00298	.00 00 00 C0	.00000 00447
.00 00 00 01	.00000 00002	.00 00 00 41	.00000 00151	.00 00 00 81	.00000 00300	.00 00 00 C1	.00000 00449
.00 00 00 02	.00000 00004	.00 00 00 42	.00000 00153	.00 00 00 82	.00000 00302	.00 00 00 C2	.00000 00451
.00 00 00 03	.00000 00006	.00 00 00 43	.00000 00155	.00 00 00 83	.00000 00305	.00 00 00 C3	.00000 00454
.00 00 00 04	.00000 00009	.00 00 00 44	.00000 00158	.00 00 00 84	.00000 00307	.00 00 00 C4	.00000 00456
.00 00 00 05	.00000 00011	.00 00 00 45	.00000 00160	.00 00 00 85	.00000 00309	.00 00 00 C5	.00000 00458
.00 00 00 06	.00000 00013	.00 00 00 46	.00000 00162	.00 00 00 86	.00000 00311	.00 00 00 C6	.00000 00461
.00 00 00 07	.00000 00016	.00 00 00 47	.00000 00165	.00 00 00 87	.00000 00314	.00 00 00 C7	.00000 00463
.00 00 00 08	.00000 00018	.00 00 00 48	.00000 00167	.00 00 00 88	.00000 00316	.00 00 00 C8	.00000 00465
.00 00 00 09	.00000 00020	.00 00 00 49	.00000 00169	.00 00 00 89	.00000 00318	.00 00 00 C9	.00000 00467
.00 00 00 0A	.00000 00023	.00 00 00 4A	.00000 00172	.00 00 00 8A	.00000 00321	.00 00 00 CA	.00000 00470
.00 00 00 0B	.00000 00025	.00 00 00 4B	.00000 00174	.00 00 00 8B	.00000 00323	.00 00 00 CB	.00000 00472
.00 00 00 0C	.00000 00027	.00 00 00 4C	.00000 00176	.00 00 00 8C	.00000 00325	.00 00 00 CC	.00000 00474
.00 00 00 0D	.00000 00030	.00 00 00 4D	.00000 00179	.00 00 00 8D	.00000 00328	.00 00 00 CD	.00000 00477
.00 00 00 0E	.00000 00032	.00 00 00 4E	.00000 00181	.00 00 00 8E	.00000 00330	.00 00 00 CE	.00000 00479
.00 00 00 0F	.00000 00034	.00 00 00 4F	.00000 00183	.00 00 00 8F	.00000 00332	.00 00 00 CF	.00000 00481
.00 00 00 10	.00000 00037	.00 00 00 50	.00000 00186	.00 00 00 90	.00000 00335	.00 00 00 D0	.00000 00484
.00 00 00 11	.00000 00039	.00 00 00 51	.00000 00188	.00 00 00 91	.00000 00337	.00 00 00 D1	.00000 00486
.00 00 00 12	.00000 00041	.00 00 00 52	.00000 00190	.00 00 00 92	.00000 00339	.00 00 00 D2	.00000 00488
.00 00 00 13	.00000 00044	.00 00 00 53	.00000 00193	.00 00 00 93	.00000 00342	.00 00 00 D3	.00000 00491
.00 00 00 14	.00000 00046	.00 00 00 54	.00000 00195	.00 00 00 94	.00000 00344	.00 00 00 D4	.00000 00493
.00 00 00 15	.00000 00048	.00 00 00 55	.00000 00197	.00 00 00 95	.00000 00346	.00 00 00 D5	.00000 00495
.00 00 00 16	.00000 00051	.00 00 00 56	.00000 00200	.00 00 00 96	.00000 00349	.00 00 00 D6	.00000 00498
.00 00 00 17	.00000 00053	.00 00 00 57	.00000 00202	.00 00 00 97	.00000 00351	.00 00 00 D7	.00000 00500
.00 00 00 18	.00000 00055	.00 00 00 58	.00000 00204	.00 00 00 98	.00000 00353	.00 00 00 D8	.00000 00502
.00 00 00 19	.00000 00058	.00 00 00 59	.00000 00207	.00 00 00 99	.00000 00356	.00 00 00 D9	.00000 00505
.00 00 00 1A	.00000 00060	.00 00 00 5A	.00000 00209	.00 00 00 9A	.00000 00358	.00 00 00 DA	.00000 00507
.00 00 00 1B	.00000 00062	.00 00 00 5B	.00000 00211	.00 00 00 9B	.00000 00360	.00 00 00 DB	.00000 00509
.00 00 00 1C	.00000 00065	.00 00 00 5C	.00000 00214	.00 00 00 9C	.00000 00363	.00 00 00 DC	.00000 00512
.00 00 00 1D	.00000 00067	.00 00 00 5D	.00000 00216	.00 00 00 9D	.00000 00365	.00 00 00 DD	.00000 00514
.00 00 00 1E	.00000 00069	.00 00 00 5E	.00000 00218	.00 00 00 9E	.00000 00367	.00 00 00 DE	.00000 00516
.00 00 00 1F	.00000 00072	.00 00 00 5F	.00000 00221	.00 00 00 9F	.00000 00370	.00 00 00 DF	.00000 00519
.00 00 00 20	.00000 00074	.00 00 00 60	.00000 00223	.00 00 00 A0	.00000 00372	.00 00 00 E0	.00000 00521
.00 00 00 21	.00000 00076	.00 00 00 61	.00000 00225	.00 00 00 A1	.00000 00374	.00 00 00 E1	.00000 00523
.00 00 00 22	.00000 00079	.00 00 00 62	.00000 00228	.00 00 00 A2	.00000 00377	.00 00 00 E2	.00000 00526
.00 00 00 23	.00000 00081	.00 00 00 63	.00000 00230	.00 00 00 A3	.00000 00379	.00 00 00 E3	.00000 00528
.00 00 00 24	.00000 00083	.00 00 00 64	.00000 00232	.00 00 00 A4	.00000 00381	.00 00 00 E4	.00000 00530
.00 00 00 25	.00000 00086	.00 00 00 65	.00000 00235	.00 00 00 A5	.00000 00384	.00 00 00 E5	.00000 00533
.00 00 00 26	.00000 00088	.00 00 00 66	.00000 00237	.00 00 00 A6	.00000 00386	.00 00 00 E6	.00000 00535
.00 00 00 27	.00000 00090	.00 00 00 67	.00000 00239	.00 00 00 A7	.00000 00388	.00 00 00 E7	.00000 00537
.00 00 00 28	.00000 00093	.00 00 00 68	.00000 00242	.00 00 00 A8	.00000 00391	.00 00 00 E8	.00000 00540
.00 00 00 29	.00000 00095	.00 00 00 69	.00000 00244	.00 00 00 A9	.00000 00393	.00 00 00 E9	.00000 00542
.00 00 00 2A	.00000 00097	.00 00 00 6A	.00000 00246	.00 00 00 AA	.00000 00395	.00 00 00 EA	.00000 00544
.00 00 00 2B	.00000 00100	.00 00 00 6B	.00000 00249	.00 00 00 AB	.00000 00398	.00 00 00 EB	.00000 00547
.00 00 00 2C	.00000 00102	.00 00 00 6C	.00000 00251	.00 00 00 AC	.00000 00400	.00 00 00 EC	.00000 00549
.00 00 00 2D	.00000 00104	.00 00 00 6D	.00000 00253	.00 00 00 AD	.00000 00402	.00 00 00 ED	.00000 00551
.00 00 00 2E	.00000 00107	.00 00 00 6E	.00000 00256	.00 00 00 AE	.00000 00405	.00 00 00 EE	.00000 00554
.00 00 00 2F	.00000 00109	.00 00 00 6F	.00000 00258	.00 00 00 AF	.00000 00407	.00 00 00 EF	.00000 00556
.00 00 00 30	.00000 00111	.00 00 00 70	.00000 00260	.00 00 00 B0	.00000 00409	.00 00 00 F0	.00000 00558
.00 00 00 31	.00000 00114	.00 00 00 71	.00000 00263	.00 00 00 B1	.00000 00412	.00 00 00 F1	.00000 00561
.00 00 00 32	.00000 00116	.00 00 00 72	.00000 00265	.00 00 00 B2	.00000 00414	.00 00 00 F2	.00000 00563
.00 00 00 33	.00000 00118	.00 00 00 73	.00000 00267	.00 00 00 B3	.00000 00416	.00 00 00 F3	.00000 00565
.00 00 00 34	.00000 00121	.00 00 00 74	.00000 00270	.00 00 00 B4	.00000 00419	.00 00 00 F4	.00000 00568
.00 00 00 35	.00000 00123	.00 00 00 75	.00000 00272	.00 00 00 B5	.00000 00421	.00 00 00 F5	.00000 00570
.00 00 00 36	.00000 00125	.00 00 00 76	.00000 00274	.00 00 00 B6	.00000 00423	.00 00 00 F6	.00000 00572
.00 00 00 37	.00000 00128	.00 00 00 77	.00000 00277	.00 00 00 B7	.00000 00426	.00 00 00 F7	.00000 00575
.00 00 00 38	.00000 00130	.00 00 00 78	.00000 00279	.00 00 00 B8	.00000 00428	.00 00 00 F8	.00000 00577
.00 00 00 39	.00000 00132	.00 00 00 79	.00000 00281	.00 00 00 B9	.00000 00430	.00 00 00 F9	.00000 00579
.00 00 00 3A	.00000 00135	.00 00 00 7A	.00000 00284	.00 00 00 BA	.00000 00433	.00 00 00 FA	.00000 00582
.00 00 00 3B	.00000 00137	.00 00 00 7B	.00000 00286	.00 00 00 BB	.00000 00435	.00 00 00 FB	.00000 00584
.00 00 00 3C	.00000 00139	.00 00 00 7C	.00000 00288	.00 00 00 BC	.00000 00437	.00 00 00 FC	.00000 00586
.00 00 00 3D	.00000 00142	.00 00 00 7D	.00000 00291	.00 00 00 BD	.00000 00440	.00 00 00 FD	.00000 00589
.00 00 00 3E	.00000 00144	.00 00 00 7E	.00000 00293	.00 00 00 BE	.00000 00442	.00 00 00 FE	.00000 00591
.00 00 00 3F	.00000 00146	.00 00 00 7F	.00000 00295	.00 00 00 BF	.00000 00444	.00 00 00 FF	.00000 00593

A

TI WORLDWIDE SALES OFFICES

ALABAMA: Huntsville, 4717 University Dr., Suite 101, Huntsville, AL 35805, (205) 837-7530.

ARIZONA: Phoenix, P.O. Box 35160, 8102 N 23rd Ave., Suite A, Phoenix, AZ 85069, (602) 249-1313

CALIFORNIA: Costa Mesa, 3168J Airway, Costa Mesa, CA 92626, (714) 540-7311. **El Segundo,** 631 S Douglas St., El Segundo, CA 90245, (213) 973-2571; **San Diego,** 4333 View Ridge Ave., Suite B., San Diego, CA 92123, (714) 278-9600. **Sunnyvale,** P.O. Box 9064, 776 Palomar Ave., Sunnyvale, CA 94088, (408) 732-1840

COLORADO: Denver, 9725 E. Hampden St., Suite 301, Denver, CO 80231, (303) 751-1780

CONNECTICUT: Hamden, 2405 Whitney Ave., Hamden, CT 06518, (203) 281-0074

FLORIDA: Clearwater, 2260 U.S. Hwy. 19 N., Suite 232, Clearwater, FL 33515, (813) 725-1861. **Ft. Lauderdale,** 4600 W. Commercial Blvd., Ft. Lauderdale, FL 33319, (305) 733-3300; **Winter Park,** 1850 Lee Rd., Suite 115, Winter Park, FL 32789, (305) 644-3535

GEORGIA: Atlanta, 330 Northeast Expy., Suite 9, Atlanta, GA 30341, (404) 456-3140

ILLINOIS: Arlington Heights, 515 W. Algonquin, Arlington Heights, IL 60005, (312) 640-3000

INDIANA: Ft. Wayne, 2020 Inwood Dr., Ft. Wayne, IN 46605, (219) 424-5174; **Indianapolis,** 5726 Professional Cir., Suite 103, Indianapolis, IN 48241, (317) 248-8555

MASSACHUSETTS: Waltham, 504 Totten Pond Rd., Waltham, MA 02154, (617) 690-7400.

MICHIGAN: Southfield, Central Park Plaza, 26211 Central Park Blvd., Suite 215, Southfield, MI 46076, (313) 353-0830

MINNESOTA: Edina, A.I.C. Bldg., Suite 202, 7615 Metro Blvd., Edina, MN 55435, (612) 835-2900

MISSOURI: Kansas City, 8080 Ward Pkwy., Kansas City, MO 64114, (816) 523-2500. **St. Louis,** 2368 Schuetz Rd., St. Louis, MO 63141, (314) 569-0801

NEW JERSEY: Clark, 1245 Westfield Ave., Clark, NJ 07066, (201) 574-9800

NEW MEXICO: Albuquerque, 5907 Alice N.E., Suite E, Albuquerque, NM 67110, (505) 265-8491

NEW YORK: East Syracuse, 6700 Old Collamer Rd., East Syracuse, NY 13057, (315) 463-9291. **Endicott,** 112 Nanticoke Ave., P.O. Box 618, Endicott, NY 13760, (607) 754-3900. **Melville,** 1 Huntington Quadrangle, Suite 1C01, Melville, NY 11746, (516) 293-2560. **Poughkeepsie,** 201 South Ave., Poughkeepsie, NY 12601, (914) 473-2900. **Rochester,** 1210 Jefferson Rd., Rochester, NY 14623, (716) 461-1800.

NORTH CAROLINA: Charlotte, 1 Woodlawn Green, Woodlawn Rd., Charlotte, NC 28210, (704) 527-0930. **Raleigh,** 4130 C-2 Camelot Dr., Raleigh, NC 27609, (919) 787-9376

OHIO: Cleveland, Belmont Bldg., Suite 120, 28790 Chagrin Blvd., Cleveland, OH 44122, (218) 464-2990. **Dayton,** Kingsley Bldg., 4124 Linden Ave., Dayton, OH 45432, (513) 253-3121

OKLAHOMA: Tulsa, 3105 E. Skelly Dr., Suite 110, Tulsa, OK 74128, (918) 749-9548.

OREGON: Beaverton, 10700 S.W. Beaverton Hwy., Suite 565, Beaverton, OR 97005, (503) 643-6759

PENNSYLVANIA: Ft. Washington, 275 Commerce Dr., Suite 300, Ft. Washington, PA 19034, (215) 643-6450

TENNESSEE: Johnson City, P.O. Drawer 1255, Erwin Hwy., Johnson City, TN 37601, (615) 926-1167

TEXAS: Dallas, 6000 Denton Dr., P.O. Box 225012, M/S 366, Dallas, TX 75265, (214) 238-6805. **Houston,** 9000 Southwest Frwy., Suite 400, Houston, TX 77036, (713) 776-6511

VIRGINIA: Arlington, Crystal Square 4, 1745 Jefferson Davis Hwy., Suite 600, Arlington, VA 22202, (703) 553-2200. **Richmond,** 3930 Beulah Rd., Richmond, VA 23234, (804) 275-8148.

WASHINGTON: Bellevue, 700 112th N.E., Suite 10, Bellevue, WA 98004, (206) 455-3480.

CANADA: St. Laurent, 945 McCallery St., St. Laurent H4T1N3, Quebec, Canada, (514) 341-3232. **Richmond Hill,** 280 Centre St. E., Richmond Hill L4C1B1, Ontario, Canada, (416) 884-9181 A

ARGENTINA, Texas Instruments Argentina S.A.I.C.F. Km 25, 5 Ruta Panamericana Don Torcuato, C.C. 2296, 1000-Correo Central, Buenos Aires, Argentina, 748-1141

ASIA, Texas Instruments Asia Ltd. 902, Asian House, 1, Hennessy Rd., Hong Kong, 05-279041, P.O. Box 2093, 990 Bendemeer Rd., Singapore 1, Republic of Singapore, 65-2581122, Room 507, Chia Hsin Bldg., 96 Chung Shan North Rd., Sec. 2, Taipei, Taiwan, 02-5630731, Aoyama Tower Bldg., 4, 5, & 6F, 24-15 Minami Aoyama, 2-Chome, Minato-Ku, Tokyo, Japan 107, 03-402-6171

AUSTRALIA, Texas Instruments Australia Ltd. Unit 1A, 5 Byfield St., P.O. Box 106, North Ryde, N.S.W. 2113, Sydney, Australia, 02-887-1122

AUSTRIA, Texas Instruments Ges. m.b.H. Rennweg 17, 1030 Vienna, Austria, 0222-724186

BELGIUM, Texas Instruments S/A, Mercurie Centre, Raketstraat 100, Rue De La Fusee, 1130 Brussels, Belgium 02-7208000

BRAZIL, Texas Instrumentos Electronicos do Brasil Ltda. Rua Padre Pereira De Andrade, 591 Cep 05489 Sao Paulo, Brazil, 011-260-6347

DENMARK, Texas Instruments A/S, Marielundvej 46E, 2730 Herlev, Denmark, 02-917400

FINLAND, Texas Instruments Finland OY, Friessenkatu 6, P.L. 917, 00101 Helsinki 10, Finland, 80-408300

FRANCE, Texas Instruments France, La Boursidiere, Bloc C, R.N. 186, 92350 Le Plessis Robinson, France, 01-6303243, 31 Quai Rambaud, 69002 Lyon, France, 078-373585, 1, Av. de la Chartreuse, 36240 Meylan, France, 076-904574, 9, Place de Bretange, 35000 Rennes, France, 099-795481, 100-102 Alle de Barcelone, Residence L'Autay, 31000 Toulouse, France, 061-213032

GERMANY, Texas Instruments Deutschland GmbH Kurtuerstendamm 146, 1000 Berlin 31, Germany, 030-8927063, Lazarettstrasse 19, 4300 Essen, Germany, 0201-233551 Akazienstrasse 22-26, 6230 Frankfurt-Griesheim, Germany, 0611-399061, Haggertystrasse 1, 8050 Freising, Germany, 08161-801, Riethorst 4, 3000 Hanover 51, Germany, 0511 848021, Frankfurter Ring 243, 6000 Munich 40, Germany, 069-325011, Kreiselderstrasse 11-15, 7000 Stuttgart 50 Germany, 0711-547001

ITALY, Texas Instruments Italia Spa, Via Europa 38/44 Cologno Monzese, Milan, Italy, 02-253-2451, Via Salaria 1319, 00138 Rome, Italy, 06-6917127, Via Montebello 27, 10124 Turin, Italy, 011-832276

MEXICO, Texas Instruments de Mexico S.A. Poniente 116 #489, Col. Industrial Vallejo, Mexico City 15, D.F., Mexico, 905-567-9200

NETHERLANDS, Texas Instruments Holland BV Laan Van de Helende Meesters 421 A, P.O. Box 283, Amstelveen, Holland, 020-473391

NORWAY, Texas Instruments A/S, Ryensvingen 15, Oslo 6, Norway, 02-889487

PORTUGAL, Texas Instruments Equipamento Electronico LDA, Rua Eng. Frederico Ulrich, 2650 Moreira Da Maia, Portugal, 948-1003

SPAIN, Texas Instruments Espana S.A. Calle Mallorca 272-276, 12 Barcelona 12, Spain, 03-2152950

SWEDEN, Texas Instruments International Trade Corporation (Sverigeliliaten), Norra Hannvagen 3, Fack S-100 54 Stockholm, Sweden 08-235480

UNITED KINGDOM, Texas Instruments Ltd. Manton Lane, Bedford, England MK41 7PA, 0234-67466 A

▶A

ALABAMA: Huntsville, Hall-Mark/Huntsville (205)837-8700

ARIZONA: Phoenix, Kierulff Electronics (602) 243-4101, R.V. Weatherford (602) 272-7144, **Tempe**, Marshall Industrias (602) 968-6181.

CALIFORNIA: Anaheim, R.V. Waatherford (714) 634-9600 **Canoga Park**, Marshall Industrias (213) 999-5001, **Chatsworth**, JACO (213) 998-2200, **El Monte**, Marshall Industrias (213) 886-0141, **El Segundo**, TI Supply (213) 973-2571, **Glendale**, R.V. Weatherford (213) 849-3451, **Goleta**, RPS (605) 964-6823, **Irvine**, Cramar/Los Angeles (714) 979-3000, (213) 771-8300, Marshall Industrias (714) 558-8400, **Los Angeles**, Kierulff Electronics (213) 685-5511, RPS (213) 748-1271, **Palo Alto**, Kierulff Electronics (415) 968-8292, **Pomona**, R.V. Weatherford (714) 823-1281; **San Diego**, Cramer/San Diego (714) 585-1881, Kierulff Electronics (714) 278-2112, Marshall Industrias (714) 278-8350, R.V. Weatherford (714) 278-7400, **Sunnyvale**, Cramer/San Francisco (408) 739-3011, Marshall Industries (408) 732-1100, TI Supply (408) 732-5555 Unitad Components (408) 737-7474, **Tustin**, Kierulff Electronics (714) 731-5711

COLORADO: Denver, Cramer/Denver (303) 758-2100 Kierulff Electronics (303) 371-6500 **Englewood**, R.V. Weatherford (303) 770-9782.

CONNECTICUT: Hamden, Arrow Electronics (203) 248-3801 TI Supply (203) 281-4669, Wilshire Electronics (203) 281-1186, **North Haven**, Cramer/Connecticut (203) 239-5641, **Orange**, Milgray/Connecticut (203) 795-0714

FLORIDA: Clearwater, Diplomat/Southland (813) 443-4514, **Ft. Lauderdale**, Arrow Electronics (305) 778-7790, Diplomat/Ft. Lauderdale (305) 771-0440, Hall-Mark/Miami (305) 971-9280, **Hollywood**, Cramer/Hollywood (305) 921-7878, **Orlando**, Cramar/Orlando (305) 894-1511, Hall-Mark/Orlando (305) 855-4020, **Palm Bay**, Arrow Electronics (305) 725-1480, **Winter Park**, Milgray Electronics (305) 647-5747.

GEORGIA: Doraville, Arrow Electronics (404) 455-4054, **Norcross**, Cramar/Atlanta (404) 448-9050

ILLINOIS: Arlington Heights, TI Supply (312) 640-2964, **Elk Grove**, Hall-Mark/Chicago (312) 437-6800, Kierulff Electronics (312) 840-0200, **Chicago**, Newark Electronics (312) 638-4411, **Mt. Prospect**, Cramer/Chicago (312) 593-8230 **Schaumburg**, Arrow Electronics (312) 893-9420

INDIANA: Ft. Wayne, Ft. Wayne Electronics (219) 423-3422, **Indianapolis**, Graham Electronics (317) 834-8202

IOWA: Cedar Rapids, Deeco (319) 365-7551

KANSAS: Shawnee Mission, Hall-Mark/Kansas City (913) 888-4747

MASSACHUSETTS: Billerica, Kierulff Electronics (617) 667-8331, **Burlington**, Wilshire Electronics (617) 272-8200, **Newton**, Cramar/Nawton (617) 969-7700 **Waltham**, TI Supply (617) 890-0510 **Woburn**, Arrow Electronics (617) 933-8130

MARYLAND: Baltimore, Arrow Electronics (202) 737-1700, (301) 247-5200 Hall-Mark/Baltimore (301) 796-9300, **Columbia**, Technico (301) 481-2200, **Saltbersberg**, Cramar/Washington (301) 948-0110, **Lanham**, Milgray/Washington (301) 459-2222

MICHIGAN: Ann Arbor, Arrow Electronics (313) 971-8220, **Oak Park**, Newark Electronics (313) 967-0600 **Grand Rapids**, Newark Electronics (618) 241-6681

MINNESOTA: Bloomington, Arrow Electronics (612) 887-8400 **Minneapolis**, Cramer/Minnesota (812) 835-7811

MISSOURI: Earth City, Hall-Mark/St. Louis (314) 291-5350 **Kansas City**, LCOMP/Kansas City (818) 221-2400, **St. Louis**, LCOMP/St. Louis (314) 291-6200

NEW JERSEY: Camden, General Radio Supply (609) 964-8580 **Cherry Hill**, Cramer/Pennsylvania (215) 923-5950, (609) 424-5993, Milgray/Delaware Valley (609) 424-1300, (215) 228-2000, **Clark**, TI Supply (201) 382-6400, **Clifton**, Wilshire Electronics (201) 340-1900 **Fairfield**, Kierulff Electronics (201) 575-6750, **Little Falls**, Cramer/New Jersey (201) 785-4300 **Moorestown**, Arrow Electronics (609) 235-1900, **Saddlebrook**, Arrow Electronics (201) 797-5800

NEW MEXICO: Albuquerque, Cramer/New Mexico (505) 243-4566 International Electronics (505) 265-6453

NEW YORK: East Syracuse, Cramer/Syracuse (315) 437-6671, **Endwell**, Wilshire Electronics (607) 754-1570, **Farmingdale**, Arrow Electronics (516) 894-6800, **Freeport**, Milgray Electronics (516) 546-6000, N.J. (800) 645-3986, **Hauppauge**, Cramer/Long Island (516) 231-5600 JACO (516) 273-5500, **Rochester**, Cramer/Rochestar (718) 275-0300, Rochester Radio Supply (716) 454-7800, Wilshire Electronics (718) 235-7820, **Woodbury**, Diplomat/Long Island (516) 921-7920

NORTH CAROLINA: Kernersville, Arrow Electronics (919) 966-2039, **Raleigh**, Hall-Mark/Raleigh (919) 832-4465 **Winston-Salem**, Cramer/Winston-Salem (919) 725-8711

OHIO: Cleveland, Arrow Electronics (216) 484-2000, Cramer/Cleveland (216) 248-8400, TI Supply (216) 464-2435, **Columbus**, Hall-Mark/Ohio (614) 846-1882, **Dayton**, ESCO Electronics (513) 228-1133, **Kettering**, Arrow Electronics (513) 253-9176

OKLAHOMA: Tulsa, Hall-Mark/Tulsa (918) 835-8458 TI Supply (918) 582-8272

PENNSYLVANIA: Huntingdon Valley, Hall-Mark/Philadelphia (215) 355-7300

TEXAS: Austin, Hall-Mark/Austin (512) 837-2814, **Dallas**, Hall-Mark/Dallas (214) 234-7400, TI Supply (214) 238-6821 **El Paso**, International Electronics (915) 778-9761 **Houston**, Hall-Mark/Houston (713) 781-6100, Harrison Equipment (713) 652-4700, TI Supply (713) 776-8511 R.V. Waatherford (713) 888-7406

UTAH: Salt Lake City, Diplomat/Altaland (801) 488-4134 Kierulff Electronics (801) 973-6913

VIRGINIA: Roanoke, Technico (703) 563-4975

WASHINGTON: Seattle, Almac/Stroum Electronics (208) 783-2300, Cramer/Seattle (206) 575-0907, Kierulff Electronics (206) 575-4420

WISCONSIN: Brookfield, Newark Electronics (414) 781-2450 **Oak Creek**, Arrow Electronics (414) 764-6600 **West Allis**, Hall-Mark/Milwaukee (414) 478-1270

CANADA: Calgary, Cam Gard Supply (403) 287-0520, **Downsview**, CESCO Electronics (418) 661-0220, **Edmonton**, Cam Gard Supply (403) 428-1805, **Halifax**, Cam Gard Supply (902)454-8581, **Kamloops**, Cam Gard Supply (604) 372-3338, **Moncton**, Cam Gard Supply (506) 855-2200, **Montreal**, CESCO Electronics (514) 735-5511, Future Electronics (514) 735-5775 **Ottawa**, CESCO Electronics (813) 729-5118, Future Electronics (613) 820-9471, **Quebec City**, CESCO Electronics (418) 887-4231 **Regina**, Cam Gard Supply (306) 525-1317, **Rexdale**, Future Electronics (418) 663-5583, **Saskatoon**, Cam Gard Supply (306) 852-8424, **Vancouver**, Cam Gard Supply (604) 291-1441 F

TEXAS INSTRUMENTS
INCORPORATED

A

TI PROGRAMMER

HEXADECIMAL AND OCTAL CALCULATOR/CONVERTER FOR COMPUTER PROGRAMMING PROFESSIONALS.

Hexadecimal. Octal. Decimal. Texas Instruments new TI Programmer lets you perform fast, accurate conversions and calculations in any of these number bases . . . portable power you can apply right to the job, right where the job is.

NUMBER BASE CONVERSIONS.

Enter a number in base 8, 10 or 16. Then with a touch of a key, that number is quickly and accurately converted to either of the other number bases. Results appear instantly on the TI Programmer's bright, easy-to-read LED display. And a convenient mode indicator means you always know what number base you're operating in.

NUMBER BASE CALCULATIONS.

The TI Programmer quickly handles arithmetic computations, too — in all three bases. Immediate answers to binary computer problems . . . giving you more time for important programming or troubleshooting tasks.

IDEAL FOR USE WITH ANY SIZE COMPUTER.

TI Programmer gives you 8-digit capacity in all bases . . . capability to handle even IBM 370 problems with ease. And since the TI Programmer uses integer "two's complement" arithmetic in hexadecimal and octal bases, it operates naturally, just like the computer does. Decimal base features signed floating point arithmetic for convenience in day-to-day math. [1/sC] key provides "one's complement" capability in HEX and OCT bases.

For additonal flexibility in minicomputer and microcomputer work, the TI Programmer has the logical functions AND , OR , Exclusive OR [XOR] and Shift [SHF] . These functions operate bit by bit on numbers in HEX and OCT to give you the same capability provided by many computer instruction sets including the latest technology in mini/micro systems . . . and a unique tool for computer repair and digital logic design.

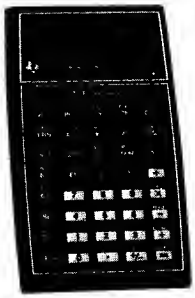
VERSATILE THREE KEY MEMORY.

Three key memory lets you store, recall or sum to memory contents. Parentheses provide the capability to specify the order of operation execution in a problem — with up to 4 pending operations. The TI Programmer even handles mixed number bases and combined logical and arithmetic operations; conversions and operations take place automatically in the order you specify. Constant mode allows constant operations with all arithmetic and logical operations.

TI PROGRAMMER CAN BE A REAL TIME SAVER FOR YOU.

Texas Instruments new TI Programmer does away with conversion tables and tedious longhand methods. Complete with vinyl carrying case, fast charge battery pack and AC adapter/charger, the TI Programmer can multiply the effectiveness of anyone involved with computer programming, program debugging or troubleshooting.

PROGRAMMABLE TI-57



THE KEY PROGRAMMABLE SUPER SLIDE RULE CALCULATOR.
With statistics and decision making capabilities to help solve repetitive problems. Eight multi-use memories provide addressable memory locations for you to store and recall data. Program memory stores 50 fully-merged steps for up to 150 keystrokes.

Once your program is built, it can be executed repeatedly by supplying new sets of variables. The calculator recalls the program for you and executes on command with each set of variables.

Plus, functions of x, logarithmic functions, trigonometric functions, statistical functions, nine levels of parentheses, and up to four pending operations let you handle even complex problems with ease. And TI's unique AOS™ algebraic operating system allows you to enter problems from left-to-right, exactly as they are stated algebraically.

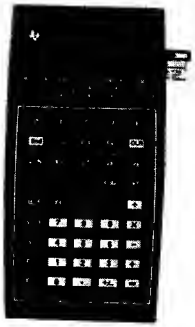
More than just a super slide ruler, TI-57 is the most powerful single-chip calculator ever produced!

PROGRAMMABLE TI-58

A POWERFUL PROGRAMMABLE CALCULATOR WITH PREPROGRAMMED SOLID STATE SOFTWARE™ MODULES.
The TI Programmable 58 is truly computer-like. Up to 480 program steps or 60 memories to work with individually or to integrate with the Master Library module to deliver up to 5000 additional steps. 4 types of display testing with independent test or "t" register. 10 additional test registers directly available for: Looping, Increment, Decrement. 6 levels of subroutines. 72 useful labels. 2 modes of indirect addressing. 10 user flags available: Set, Reset, Test. 10 user defined label keys. Over 170 functions and operations in scientific, engineering and statistical fields.

Complete editing and error correction capabilities. Single-step and back-step keys let you review and revise your program. Insert and delete keys make it simple to add or remove instructions at any point in the program.

PROGRAMMABLE TI-59



A REVOLUTIONARY NEW ADVANCE IN PERSONAL PROGRAMMABLE CALCULATORS.
An extraordinary card programmable calculator with plug-in Solid State Software™ and magnetic card storage. You have all the features of the TI-58 plus . . . up to 960 program steps, or up to 100 memories, and magnetic card read/write capabiity so you can record your own programs and make them a part of your permanent personal library. The Master Library module simply plugs in. It includes 25 different programs in key areas such as math, statistics, finance. Plus, blank magnetic cards let you write and record custom programs.

The TI Programmable 59 is the most powerful handheld programmable in the world — more powerful and more versatile than some computers of the past decade.

With the PC-100C printer/plotter — which allows you to print, list or trace your programs, plot curves and histograms, and print out alpha headings — the TI Programmable 58 and the TI Programmable 59 become even more flexible.

A

MMANDS

	Definition
	List all user symbols, procedures, functions, and arrays
)	Delete user procedure, function, or array
	Save all user defined symbols, functions, and arrays on 'file'.
	Restore user defined symbols, procedures, functions, and arrays from 'file'.
	Delete all user symbols, procedures, functions and arrays.
	List object program labels
	Delete all object program labels
	Exit from AMPL back to operating system.

Year (1976 to 1999)
Julian day (1 to 366)
Hour (0 to 23)
Minute (0 to 59)
Second (0 to 59)

Suspend AMPL for <expr>*50 milliseconds (<expr> = 20 is one second).

IORY COMMANDS

	Emulator memory mapping 9900/9980 map 8K bytes (0->1FFF) 9940 define RAM and ROM sizes
as[,IDT] [+ DEF] [+ REF]]])	Load object program by bias and enter program labels into table
as])	Verify object program, listing differences between object and target memory
w,high[,start])	Dump program from target <low> to <high> in nonrelocatable format

CONTROL COMMANDS

	Definition
{1/0}[,'TM0n']])	Initialize Emulator device, clock 0 = prototype/ 1 = emulator.
	Processor clock
	Processor type: -1 = TMS9940, 0 = SBP9900, 1 = TMS9900, 2 = TMS9980
{,count[,low,high]]}	Trace qualifier, completion break count (OFF-255), address range
[+ ILLA] [,address]...)	Address breakpoint(s) (ILLA only valid for TMS9940).
	Run emulation at PC, WP, ST
	Emulation status (3 LSBits) HOLD, IDLE, Running
	Halt emulation, return status
{MR, MW, IAO}])	Indexed bus signal from buffer (TRUE if expression matches)
	Indexed address from trace buffer
	Emulator Trace buffer limits Oldest, Newest sample indices

TRACE MODULE CONTRDL

Syntax	Definition
TINT ('TM0n')	Initialize trace module
TTRC ([INT] {OFF, [±O0] [±O1][±O2][±O3], [±IAO][±DBIN]} [,count[, {ON, OFF}]])	Qualify data samples, trace completion counter (OFF-255), latch option on D0-D3
TEVT ({OFF, [±D0] [±D1] [±D2] [±D3], [±IAO] [±DBIN], EXT} [,value[,mask]])	Qualify D0-D3 event (or EXTernal), <value> and <mask> for D4-D19
TBRK (count [,<delay>[,INV] [+ EDGE]]])	Set event counter (OFF-FFFF), set delay counter (OFF-244), count INVerted/EDGE events
TRUN	Start Trace module tracing.
TST	Trace module status (3 LSB's), event occurred, trace full, tracing
THLT	Halt trace module, return status
TNE	Number of events since last TRUN
TNCE	Number of event count overflows
TTBH (index[, {±D0} [±D1][±D2] [±D3] {±IAO}[±DBIN] }])	D0-D3 of indexed samples, (TRUE if expression matches).
TTB (<index>)	D4-D19 indexed samples (data bus)
TTBO, TTBN	Trace module trace buffer limits Oldest, Newest sample indices

TRACE MODULE INTERCONNECT TO EMULATOR

O0	Memory address bit 15 (TMS9940 only)
D0	Byte memory cycle (TMS9940 only).
O1,D1, IAO	Instruction Acquisition.
O2,D2,DBIN	DataBusIN = MR(read), MW = -DBIN(write).
O3	Emulator trace qualifier and range (ETRC).
D3, External Event	Emulator address breakpoint (EBRK)
D4-D19	Emulator data bus (bits 0-15).
External Clock	Emulator memory cycle clock
Control Cable	Synchronizes emulation and tracing. Trace module will halt emulator for EINT ('EM0n', clock 'TM0n')

TARGET REGISTERS

PC,WP,ST	Processor registers.
R0-R15	Workspace registers
DR	Display all registers

CRU READ/WRITE

CRUB	CRU interface base address
CRUR (offset,width)	Read target CRU field
CRUW (offset,width,value)	Write <value> into target CRU field

KEYWDRDS

ARG	FORM	THEN	GE
ARRAY	FUNC	TO	GT
BEGIN	IF	UNTIL	HI
BY	LOC	WHILE	HIE
CASE	MOD	AND	LE
DO	NULL	NAND	LO
ELSE	OF	OR	LOE
END	PROC	XOR	LT
ESCAPE	REPEAT	NOT	NE
FOR	RETURN	EQ	

KEYWORD CONSTANTS

D0	EXT	IO	Q2
D1	EXTEND	MA	Q3
D2	GRAPH	MR	REF
D3	IAQ	MW	REL
DBIN	IAQX	N	REWIND
DEF	IDT	OFF	SEQ
DIRECT	ILLA	ON	UNLOAD
EDGE	IN	OUT	VDT
EOF	INT	Q0	Y
ETBN	INV	Q1	

ERROR MESSAGES

0 — ! UNDEFINED ERROR CODE !
1 — I/O ERROR, OS ERROR CODE RETURNED
2 — INSUFFICIENT MEMORY TO CONTINUE
3 — ! SEGMENT VIOLATION !
4 — I/O ERROR: INVALID UNIT ID
5 — I/O ERROR: READ/WRITE VIOLATION
6 — I/O ERROR: INSUFFICIENT MEMORY FOR OPEN
7 — ! DELETE UNIT CONTROL BLOCKS ERROR !
8 — TOO MANY IDT DEF/REF SYMBOLS IN LOAD
9 — EXCEEDED 15 LOAD OPERATIONS SINCE LAST CLR
10 — CANNOT ALLOCATE MEMORY FOR USER SYMBOL TABLE
11 — ! ERROR IN I/O UNIT CHAIN POINTERS !
12 — OVERLAY ERROR
101 — VARIABLE CANNOT BE READ
102 — VARIABLE CANNOT BE WRITTEN
103 — SYMBOL IS UNDEFINED
104 — ! INVALID CODEGEN BRANCH TABLE INDEX !
105 — INSUFFICIENT MEMORY TO COMPILE STATEMENT
106 — SYMBOL IS DEFINED; CANNOT BE REDEFINED
107 — INSUFFICIENT MEMORY TO COMPILE PROC/FUNC
108 — INPUT RECORD CANNOT BE CLASSIFIED
109 — INPUT STRING EXCEEDS MAXIMUM ALLOWED LENGTH
110 — ! INVALID SCANNER BRANCH TABLE INDEX !
111 — UNRECOGNIZABLE INPUT ITEM
112 — ! UNDEFINED OPERATOR !
114 — SYMBOL NOT AN IDT/DEF/REF LOAD SYMBOL
115 — USER SYMBOL TABLE FULL
116 — CONSTANT EXCEEDS 16 BITS
117 — SYNTAX ERROR
118 — ! INVALID KEYWORD STRING LENGTH !
119 — SYNTAX ERROR IN ONE-LINE-ASSEMBLY STATEMENT
120 — INCORRECT NUMBER OF ARRAY SUBSCRIPTS
121 — ESCAPE SPECIFIED OUTSIDE A LOOP CONSTRUCT
122 — ARRAY REDEFINED WITH INCORRECT SUBSCRIPTS

NOTE A hexadecimal number is also printed with some error messages Refer to the AMPL System Operation Guide for complete explanation

ERRDR MESSAGES

201 — SYMBOL NOT FOUND TO DELETE
202 — SYMBOL CANNOT BE DELETED
203 — INVALID DISPLAY FORMAT CHARACTER FOLLOWING
204 — NO LIST DEVICE ASSIGNED
205 — EMULATOR I/O ERROR CODE RETURNED
209 — INVALID INDEX INTO EMULATOR TRACE BUFFER
210 — !CANNOT ALLOCATE FORM CURRENT VALUE SEGMENT!
211 — INSUFFICIENT MEMORY TO SAVE FORM PARAMETERS
214 — INVALID RESTORE FILE
215 — INSUFFICIENT MEMORY TO COMPLETE THE RESTORE
216 — BAD TRACE OR COMPARISON MODE SELECTED
219 — TRACE MODULE I/O ERROR CODE RETURNED
220 — CANNOT EDIT ON THIS DEVICE TYPE
221 — TRACE INTERFACE CHANGE ILLEGAL WHILE TRACING
222 — INVALID INDEX INTO TRACE MODULE BUFFER
223 — INSUFFICIENT ARGUMENTS IN PROC/FUNC CALL
224 — STACK OVERFLOW, DELETE PROC/FUNC/ARRAY
225 — DELETED PROC/FUNC/ARRAY REFERENCED
226 — INSUFFICIENT ARGUMENTS IN FORM FOR PROC/FUNC
227 — ! INVALID FORM SEGMENT ID !
228 — ! INVALID FORM CURRENT VALUE SEGMENT ID !
229 — INVALID CHARACTER IN LOAD FILE
230 — CHECKSUM ERROR IN LOAD FILE
231 — ARITHMETIC OVERFLOW
233 — PROC/FUNC CALL ARGUMENT OUT OF RANGE
234 — INVALID "ARG" OR "LOC" INDEX FOR WRITING
235 — INVALID "ARG" OR "LOC" INDEX FOR READING
237 — ARRAY ALREADY DEFINED
238 — INVALID ARRAY DIMENSION
240 — REFERENCE TO UNDECLARED ARRAY
241 — INVALID ARRAY SUBSCRIPT
242 — ! ERROR ARRAY SEGMENT LENGTH !
243 — DELETED IDT/DEF/REF LOAD SYMBOL REFERENCED
244 — ALL IDT/DEF/REF LOAD SYMBOLS DELETED
245 — INVALID DEVICE TYPE TO "EINT" OR "TINT"

NOTE. Error messages withing exclamation marks (!) are AMPL internal system errors Contact Texas Instruments if problem persists

TEXAS INSTRUMENTS

AMPL Reference Data

Microprocessor Series™

EXPLANATION OF THE NOTATION USED IN THIS CARD

	Notation	Explanation
Optional Items	[item]	Bracketed item may be omitted
	{item 1, item 2}	Exactly one item must be selected from the items in braces
Substitution	expr 'file'	Any expression may be used. File or device name required
Repetition	item . . .	A list of items may be used
Required	<item>	Replace with item.

CHARACTER SET

Type	Characters	Use
Special	RETURN SPACE ! " $ / ( ) ° + , – . / , < = > ? @	Any printable character may be used in a quoted string RETURN terminates line and statement "." may separate statements SPACE separates adjacent numbers and identifiers
Numerals	0 — 9	
Letters	A — Z,a — z	

NOTE All AMPL reserved words use only upper case (UPPER CASE LOCK)

SYMBOL NAMES

Type	Example	Definition
System	RO ETRC	Up to four alphanumeric characters, all system symbols are predefined
User-defined	USRVAR X3 BRKADR GO	Up to six alphanumeric characters, assignment defines a variable ARRAY statement defines an array PROC/FUNC statement defines a procedure/function
Program label	IDT DEF	Up to six alphanumeric characters Period after IDT and before DEF labels, defined by LOAD command.

CONSTANTS

Type	Example	Range
Decimal	10833	1 . 32767
Hexadecimal	02A51 >2A51	>0 . . . >FFFF
Octal	125121	!0 . . !177777
Binary	<101010010100001	<0 <1111111111111111

GENERAL COMMANDS

Syntax	Definition
USYM	List all user symbols, procedures, functions, and arrays
DELE ('name'....)	Delete user procedure, function, or array.
SAVE ('file')	Save all user defined symbols, functions, and arrays on 'file'.
RSTR ('file')	Restore user defined symbols, procedures, functions, and arrays from 'file'
CLR	Delete all user symbols, procedures, functions and arrays.
MSYM	List object program labels
MDEL	Delete all object program labels.
EXIT	Exit from AMPL back to operating system.

TIMING

YR	Year (1976 to 1999)
DAY	Julian day (1 to 366)
HR	Hour (0 to 23)
MIN	Minute (0 to 59)
SEC	Second (0 to 59)
WAIT (expr)	Suspend AMPL for <expr>*50 milliseconds (<expr> = 20 is one second).

TARGET MEMORY COMMANDS

EMEM	Emulator memory mapping: 9900/9980 map 8K bytes (0->1FFF) 9940 define RAM and ROM sizes
LOAD ('file'[,bias[,IDT] [+DEF] [+REF]])	Load object program by bias and enter program labels into table.
VRFY ('file' [,bias])	Verify object program, listing differences between object and target memory.
DUMP ('file',low,high[,start])	Dump program from target <low> to <high> in nonrelocatable format

EMULATOR CONTROL COMMANDS

Syntax	Definition
EINT ('EM0n' [,{1 / 0}[,'TM0n']])	Initialize Emulator device, clock 0 = prototype/ 1 = emulator.
ECLK	Processor clock
ETYP	Processor type: -1 = TMS9940, 0 = SBP9900, 1 = TMS9900, 2 = TMS9980.
ETRC ({MA / IAOX / IAO} [,count[,low,high]])	Trace qualifier, completion break count (OFF-255), address range
EBRK ({MA / IAO / MR / MW} [+ILLA] [,address]. .)	Address breakpoint(s) (ILLA only valid for TMS9940).
ERUN	Run emulation at PC, WP, ST
EST	Emulation status (3 LSBits) HOLD, IDLE, Running
EHLT	Halt emulation, return status
ETBH (index[,{MR / MW / IAO}])	Indexed bus signal from buffer ,(TRUE if expression matches).
ETB (index)	Indexed address from trace buffer
ETBO, ETBN	Emulator Trace buffer limits Oldest, Newest sample indices

TRACE MODULE CONTROL

Syntax	Definition
TINT ('TM0n')	Initialize trace module.
TTRC ([INT] {OFF / [±D0] [±D1] [±D2] [±D3] / [±IAO][±DBIN]} [count[, {ON / OFF}]])	Qualify data samples, trace completion counter (OFF-255) latch option on D0-D3
TEVT ({OFF / [±D0] [±D1] [±D2] [±D3] / [±IAO] [±DBIN] / EXT} [,value[,mask]])	Qualify D0-D3 event (or EXTernal), <value> and <mask> for D4-D19
TBRK (count [,<delay>[,INV] [+EDGE]]])	Set event counter (OFF-FFFF), set delay counter (OFF-244), count INVerted/EDGE events.
TRUN	Start Trace module tracing
TST	Trace module status (3 LSB's), event occurred trace full, tracing.
THLT	Halt trace module, return status
TNE	Number of events since last TRUN
TNCE	Number of event count overflows
TTBH (index[, {[±D0] [±D1] [±D2] [±D3] / [±IAO][±DBIN]}])	D0-D3 of indexed samples, (TRUE if expression matches).
TTB (<index>)	D4-D19 indexed samples (data bus)
TTBO TTBN	Trace module trace buffer limits Oldest, Newest sample indices

TRACE MODULE INTERCONNECT TO EMULATOR

Q0	Memory address bit 15 (TMS9940 only)
D0	Byte memory cycle (TMS9940 only)
Q1,D1, IAO	Instruction Acquisition
Q2,D2,DBIN	DataBusIN = MR(read), MW = -DBIN(write)
Q3	Emulator trace qualifier and range (ETRC)
D3, External Event	Emulator address breakpoint (EBRK)
D4-D19	Emulator data bus (bits 0-15)
External Clock	Emulator memory cycle clock.
Control Cable	Synchronizes emulation and tracing Trace module will halt emulator for EINT ('EM0n', clock 'TM0n').

TARGET REGISTERS

PC,WP,ST	Processor registers.
R0-R15	Workspace registers
DR	Display all registers

CRU READ/WRITE

CRUB	CRU interface base address
CRUR (offset,width)	Read target CRU field
CRUW (offset,width,value);	Write <value> into target CRU field

KEYWORDS

ARG	FORM	THEN	GE
ARRAY	FUNC	TO	GT
BEGIN	IF	UNTIL	HI
BY	LOC	WHILE	HIE
CASE	MOD	AND	LE
DO	NULL	NAND	LO
ELSE	OF	OR	LOE
END	PROC	XOR	LT
ESCAPE	REPEAT	NOT	NE
FOR	RETURN	EO	

KEYWORD CONSTANTS

D0	EXT	IO	Q2
D1	EXTEND	MA	Q3
D2	GRAPH	MR	REF
D3	IAO	MW	REL
DBIN	IAOX	N	REWIND
DEF	IDT	OFF	SEQ
DIRECT	ILLA	ON	UNLOAD
EDGE	IN	OUT	VDT
EOF	INT	Q0	Y
ETBN	INV	Q1	

ERROR MESSAGES

0 — ! UNDEFINED ERROR CODE !
1 — I/O ERROR, OS ERROR CODE RETURNED
2 — INSUFFICIENT MEMORY TO CONTINUE
3 — ! SEGMENT VIOLATION !
4 — I/O ERROR INVALID UNIT ID
5 — I/O ERROR READ/WRITE VIOLATION
6 — I/O ERROR INSUFFICIENT MEMORY FOR OPEN
7 — ! DELETE UNIT CONTROL BLOCKS ERROR !
8 — TOO MANY IDT DEF/REF SYMBOLS IN LOAD
9 — EXCEEDED 15 LOAD OPERATIONS SINCE LAST CLR
10 — CANNOT ALLOCATE MEMORY FOR USER SYMBOL TABLE
11 — ! ERROR IN I/O UNIT CHAIN POINTERS !
12 — OVERLAY ERROR
101 — VARIABLE CANNOT BE READ
102 — VARIABLE CANNOT BE WRITTEN
103 — SYMBOL IS UNDEFINED
104 — ! INVALID CODEGEN BRANCH TABLE INDEX !
105 — INSUFFICIENT MEMORY TO COMPILE STATEMENT
106 — SYMBOL IS DEFINED, CANNOT BE REDEFINED
107 — INSUFFICIENT MEMORY TO COMPILE PROC/FUNC
108 — INPUT RECORD CANNOT BE CLASSIFIED
109 — INPUT STRING EXCEEDS MAXIMUM ALLOWED LENGTH
110 — ! INVALID SCANNER BRANCH TABLE INDEX !
111 — UNRECOGNIZABLE INPUT ITEM
112 — ! UNDEFINED OPERATOR !
114 — SYMBOL NOT AN IDT/DEF/REF LOAD SYMBOL
115 — USER SYMBOL TABLE FULL
116 — CONSTANT EXCEEDS 16 BITS
117 — SYNTAX ERROR
118 — ! INVALID KEYWORD STRING LENGTH !
119 — SYNTAX ERROR IN ONE-LINE-ASSEMBLY STATEMENT
120 — INCORRECT NUMBER OF ARRAY SUBSCRIPTS
121 — ESCAPE SPECIFIED OUTSIDE A LOOP CONSTRUCT
122 — ARRAY REDEFINED WITH INCORRECT SUBSCRIPTS

NOTE A hexadecimal number is also printed with some error messages. Refer to the AMPL System Operation Guide for complete explanation

ERROR MESSAGES

201 — SYMBOL NOT FOUND TO DELETE
202 — SYMBOL CANNOT BE DELETED
203 — INVALID DISPLAY FORMAT CHARACTER FOLLOWING
204 — NO LIST DEVICE ASSIGNED
205 — EMULATOR I/O ERROR CODE RETURNED
209 — INVALID INDEX INTO EMULATOR TRACE BUFFER
210 — !CANNOT ALLOCATE FORM CURRENT VALUE SEGMENT!
211 — INSUFFICIENT MEMORY TO SAVE FORM PARAMETERS
214 — INVALID RESTORE FILE
215 — INSUFFICIENT MEMORY TO COMPLETE THE RESTORE
216 — BAD TRACE OR COMPARISON MODE SELECTED
219 — TRACE MODULE I/O ERROR CODE RETURNED
220 — CANNOT EDIT ON THIS DEVICE TYPE
221 — TRACE INTERFACE CHANGE ILLEGAL WHILE TRACING
222 — INVALID INDEX INTO TRACE MODULE BUFFER
223 — INSUFFICIENT ARGUMENTS IN PROC/FUNC CALL
224 — STACK OVERFLOW, DELETE PROC/FUNC/ARRAY
225 — DELETED PROC/FUNC/ARRAY REFERENCED
226 — INSUFFICIENT ARGUMENTS IN FORM FOR PROC/FUNC
227 — ! INVALID FORM SEGMENT ID !
228 — ! INVALID FORM CURRENT VALUE SEGMENT ID !
229 — INVALID CHARACTER IN LOAD FILE
230 — CHECKSUM ERROR IN LOAD FILE
231 — ARITHMETIC OVERFLOW
233 — PROC/FUNC CALL ARGUMENT OUT OF RANGE
234 — INVALID "ARG" OR "LOC" INDEX FOR WRITING
235 — INVALID "ARG" OR "LOC" INDEX FOR READING
237 — ARRAY ALREADY DEFINED
238 — INVALID ARRAY DIMENSION
240 — REFERENCE TO UNDECLARED ARRAY
241 — INVALID ARRAY SUBSCRIPT
242 — ! ERROR ARRAY SEGMENT LENGTH !
243 — DELETED IDT/DEF/REF LOAD SYMBOL REFERENCED
244 — ALL IDT/DEF/REF LOAD SYMBOLS DELETED
245 — INVALID DEVICE TYPE TO "EINT" OR "TINT"

NOTE Error messages within exclamation marks (!) are AMPL internal system errors Contact Texas Instruments if problem persists

TEXAS INSTRU

AMPL Reference Data

Microprocesso
Series™

EXPLANATION OF THE NOTATION USED IN THIS CARD

	Notation	Explanation
Optional Items	[item]	Bracketed item ma
	{item 1 / item 2}	Exactly one item m from the items in b
Substitution	expr 'file	Any expression ma File or device name
Repetition	item	A list of items may
Required	<item>	Replace with item.

CHARACTER SET

Type	Characters	Use
Special	RETURN SPACE ! " $ / ( ) " + , – / , < = > ? @	Any printable character may be use string. RETURN terminates line and may separate statements. SPACE s numbers and identifiers
Numerals	0 – 9	
Letters	A – Z,a – z	

NOTE: All AMPL reserved words use only upper case (UPPER CASE LOC

SYMBOL NAMES

Type	Example	Definition
System	RO ETRC	Up to four alphanumeric character symbols are predefined
User-defined	USRVAR X3 BRKADR GO	Up to six alphanumeric characters a variable ARRAY statement defines an array PROC/FUNC statement defines a
Program label	IDT DEF	Up to six alphanumeric characters and before DEF labels, defined by

CONSTANTS

Type	Example	Range
Decimal	10833	1 32767
Hexadecimal	02A51, >2A51	>0 . . . >FFFF
Octal	125121	!0 . . . !177777
Binary	[illegible]	<0 [illegible]

Example (expr)	Definition
+ expr	Value of <expr>.
– expr	Two's complement of <expr>
@addr	<addr> used as word address into emulator or target memory.
ARG expr	Argument in position <expr> of call list, ARG 0 is number of arguments in list
LOC expr	Word <expr> of local variable array, LOC 0 is length of local variable array.
expr1 * expr2	Signed product (warning on overflow).
expr1 / expr2	Signed quotient (warning on divide by zero).
expr1 MOD EXPR2	Signed remainder of division (warning on divide by zero).
expr1 + expr2	Signed sum.
expr1 - expr2	Signed difference.

lational operator is either FALSE (0) or TRUE (-1).

expr1 EQ expr2 expr1 NE expr2	16-bit comparison.
expr1 LT expr2 expr1 LE expr2 expr1 GT expr2 expr1 GE expr2	Signed, 16-bit comparison
expr1 LO expr2 expr1 LOE expr2 expr1 HI expr2 expr1 HIE expr2	Unsigned, 16-bit comparison.
NOT expr	16-bit one's complement.
expr1 AND expr2 expr 1 NAND expr2	16-bit boolean AND. 16-bit boolean not AND
expr1 OR expr2 expr1 XOR expr2	16-bit boolean OR 16-bit boolean exclusive OR

re given in order of precedence, highest to lowest. Solid lines separate groups, within each group, precedence is equal and evaluation is left to ation results in a 16-bit integer value.

IMETIC

	Definition
)	Low-order 16 bits of unsigned product <expr1> * <expr2>, high order 16 in MDR
nd)	Unsigned quotient of 32-bit number (MDR <dividend>) over <divisor>, remainder in MDR.
	High-order 16-bits of MPY product and of DIV dividend; remainder of DIV, unsigned carry of + and-.

ON

1[,expr2]),	User <name> (previously undefined or name of deleted array) is defined as one- or two-dimension array

IENTS

	Value of expression
	Literal string
f] ? [f]	Target memory

n/[f]

A	set default	G	octal	O[i]
B[i]	hexadecimal	H[i]	symbolic	S
D[i]	instruction	I	unsigned	U[i]
E	newline	N[j]	space	X[j]

= 9 field width 'i' digits, then two blanks
= 9 default field width, no trailing blanks
repeat 'j' times
repeat 10 times

/modify mode(?)

RETURN, +	replace contents	<expr>
–	open new address	@<addr>
.	change display	.f f

DST	Destination address
SRC	Source address

ASSIGNMENT STATEMENTS

Type	Example	Definition
Variable	sym = expr	User-defined or writable system symbol or REF program label
Target memory	@addr = expr	Put value of <expr> at target <addr>
Proc/Func argument	ARG n = expr	Local copy of argument in position <n> of call list
Command local	LOC n = expr	Word <n> of local storage array
Array	A[(i1[,i2])] = e	User defined array name, zero, one, or two index expressions

NOTE Precedence of @, ARG and LOC may require parenthesis around following expression.

COMPOUND STATEMENTS

Syntax	Definition
BEGIN statements END	Statements are executed sequentially. Use in place of any single statement syntax.

CONTROL STATEMENTS

IF expr THEN s1 [ELSE s2]	<s1> is executed if <expr> is TRUE (nonzero) Otherwise, <s2> is executed, if included
CASE expr OF expr 1 s1, exprn sn [ELSE s] END	Statement <si> at first label expression <expr> equal to <expr> is executed. If none, statement <s> is executed if included
WHILE expr DO statement	While <expr> is TRUE (nonzero), <statement> is executed
REPEAT statement UNTIL expr	<statement> is executed if <expr> FALSE (zero). <statement> is executed until <expr> is TRUE
FOR var = expr1 TO expr 2[BY expr3] DO statement	Value of <expr1> is assigned to <var> <statement> is executed until <var> is equal to <expr2>, <expr3> is added to <var>, and <statement> repeated. Default value of <exp3> is 1.
ESCAPE	Exit from innermost enclosing WHILE, REPEAT, or FOR statement.

PROCEDURE/FUNCTION/FORM DEFINITION

PROC name [(args[,locs])] statements END FUNC name [(args[,locs])] statements END	User-defined <name> (previously undefined or deleted procedure/function) is bound to <statements> <args> is the required number of arguments. <locs> is the size of local storage array
RETURN [expr]	Pass control back to calling statement. In a procedure, <expr> is ignored. In a function, value of <expr> replaces the function call in the calling expression
FORM name 'prompt' [= [{ constant / 'string' }]],	END
	<name> must be a previously defined procedure or function, semicolon required between prompts

PROCEDURE/FUNCTION CALLS

proc name [(expr,)]	User-defined or system procedure/function with list of argument expressions.
func name [(expr,)]	Command definition determines number of arguments required. Some system commands require quoted strings as arguments

NOTE Procedure/functions with defined FORM when called with no arguments will prompt for arguments using the FORM

example FORM

COMMENTARY ENTRY	comment, not a prompt required argument,
PROMPT 1 = default value PROMPT 2 = PROMPT 3 =	with default value required argument, must enter value default given if value not entered

FORM control function keys

Next prompt	TAB, ↓, →FIELD, SKIP, RETURN
Previous prompt	↓, ←FIELD
First prompt	HOME
Erase value.	ERASE FIELD, ERASE INPUT
Redisplay default.	INSERT LINE
Duplicate previous value	F4

INPUT/OUTPUT COMMANDS

Syntax	Definition
HCRB	Host computer CRU base address
HCRR (offset,width)	Read host computer CRU field
HCRW (offset,width,value)	Write <value> into host CRU field
COPY ({ 'file' / edit id })	AMPL input from 'file' AMPL input from edit buffer
LIST ({ 'file' / OFF / ON / EOF })	Initialize listing device or file. Disable listing output. Enable listing output. Close listing device or file with EOF
NL	Print newline

unit = OPEN [({ 'file' / edit id } [, { 0 / IN / OUT / IO } [, { 0 / REWIND / EXTEND } [, { SEQ / REL }]]])]

no arguments — list all open units and edit buffers
initialize 'file'/<edit id> I/O unit
0 — device IO, file IN only
IN — for input only
OUT — for output only
IO — for input/output
REWIND — position to beginning of file
EXTEND — position to end of file
SEQ — auto-create sequential file
REL — auto-create rel-rec file

event-READ [(unit [, { 0 / DIRECT } [, { 0 / GRAPH }]] [, { VDT / SEQ / REL } , { 0 / f row } [, { 0 / f col } [, s col]] [, rec #]]])]

no arguments — read console
Read record from (unit)
0 — issue read ASCII
DIRECT — issue read direct
GRAPH — read graphics on 922 VDT
VDT — read in cursor positioning mode
f row — field start row
f col — field start column
s col — cursor start column
SEQ — read sequentially
REL — read specified record
rec # — record number to read
<event>/256 = cursor column after read if VDT
<event> AND 255 = event key value if VDT,
else >OD for end of record,
>13 for end of file

value = EVAL [(unit)]	Evaluate expression in <unit>'s buffer; if no <unit>, READ/EVAL the console
DPLY [(unit)]	AMPL display unit for output to <unit>, if no <unit>, to console
okay = MOVE (from unit, to unit)	Move contents of <from unit>'s buffer to <to unit>'s buffer <okay> = 0 if moved = >FFFF if too big and not moved
REW[(unit)]	Rewind (unit) — repositions, file clears console no argument — clears console

Cursor = WRIT (unit [, { 0 / DIRECT } [, { 0 / GRAPH }]] [, { VDT / SEQ / REL } , { 0 / f row } [, [f col]] [, rec #]])]

no arguments — write console
Write record to (unit),
0 — issue write ASCII
DIRECT — issue write direct
GRAPH — write graphics on 911 VDT
VDT — write in cursor positioning mode
f row — field start row
f col — field start column
SEQ — write sequentially
REL — read specified record
rec # — record number to read
<cursor>/256 = cursor column after write if VDT

CLSE (unit [, { EOF / UNLOAD }]) Release I/O <unit>.
EOF — write end-of-file mark
UNLOAD — unload unit

SYSTEM SYMBOLS

V — variable F — function P — procedure

CLR	P — clear	MDEL	P — symbols
CLSE	P — I/O close	MDR	V — arithmetic
COPY	P — copy	MIN	V — minutes
CRUB	V — CRU base	MOVE	F — I/O buffer
CRUR	F — CRU read	MPY	F — multiply
CRUW	P — CRU write	MSYM	P — symbols
DAY	V — day	NL	P — newline
DBUF	P — delete buffer	OPEN	F — I/O open
DELE	P — delete symbol	PC	V — registers
DIV	F — divide	R0-R15	V — registers
DPLY	P — display	READ	F — I/O read
DR	P — registers	REW	P — I/O rewind
DST	V — destination	RSTR	P — restore
DUMP	P — dump	SAVE	P — save
EBRK	P — emulator	SEC	V — seconds
ECLK	V — emulator	SRC	V — source
EDIT	F — edit	ST	V — register
EHLT	F — emulator	TBRK	P — trace module
EINT	P — emulator	TEVT	P — trace module
EMEM	V — emulator	THLT	F — trace module
ERUN	P — emulator	TINT	P — trace module
EST	F — emulator	TNCE	V — trace module
ETB	F — emulator	TNE	V — trace module
ETBH	F — emulator	TRUN	P — trace module
ETBO	V — emulator	TST	F — trace module
ETRC	P — emulator	TTB	F — trace module
ETYP	V — emulator	TTBH	F — trace module
EVAL	F — evaluate	TTBN	V — trace module
EXIT	P — exit AMPL	TTBO	V — trace module
HCRB	V — host CRU	TTRC	P — trace module
HCRR	F — CRU read	USYM	P — user symbols
HCRW	P — CRU write	VRFY	P — verify
HR	V — hour	WAIT	F — delay AMPL
IOR1	V — I/O	WP	V + register
KEEP	P — keep edit	WRIT	P — I/O write
LIST	P — list	YR	V — year
LOAD	P — load object		

EDIT

Syntax	Definition
edit id = EDIT[({ 'file' / edit id } [,record])]	Create edit buffer with 'file'. Edit existing buffer. No argument creates an empty buffer
KEEP (edit id, 'file')	Save edit buffer onto 'file' and delete edit buffer.
DBUF (edit id)	Delete edit buffer

EDIT CONTROL FUNCTION KEYS

Function	911 KEY	913 KEY	CONTROL CHARACTER
edit/compose mode	F7	F7	V
quit edit mode	CMD	HELP	X
roll up	F1	F1	A
roll down	F2	F2	B
set tab	F3	F3	C
clear tab	F4	F4	D
tab	TAB (shift SKIP)	TAB	I
back tab	FIELD	BACK TAB	T
newline	RETURN	NEWLINE	RETURN
insert line	unlabeled gray	INSERT LINE	O
delete line	ERASE INPUT	DELETE LINE	N
erase line	ERASE FIELD	CLEAR	W
truncate line	SKIP	SET	K
insert character	INS CHAR	INSERT CHAR	
delete character	DEL CHAR	DELETE CHAR	
cursor up	.		U
cursor down	.		J
cursor right	.	.	R
cursor left	.	.	H
top of screen	HOME	HOME	

EXPRESSIONS

Type	Example	Definition
Subexpression	(expr)	
Identity	+ expr	Value of <expr>
Negation	– expr	Two's complement of <expr>
Target memory	@addr	<addr> used as word address into emulator or target memory
Proc/Func Argument	ARG expr	Argument in position <expr> of call list; ARG 0 is number of arguments in list
Proc/Func local variable	LOC expr	Word <expr> of local variable array; LOC 0 is length of local variable array
Multiplication	expr1 * expr2	Signed product (warning on overflow)
Division	expr1 / expr2	Signed quotient (warning on divide by zero)
Remainder	expr1 MOD EXPR2	Signed remainder of division (warning on divide by zero)
Addition	expr1 + expr2	Signed sum.
Subtraction	expr1 - expr2	Signed difference

NOTE Result of relational operator is either FALSE (0) or TRUE (-1).

Equality	expr1 EQ expr2 expr1 NE expr2	16-bit comparison
Arithmetic inequality	expr1 LT expr2 expr1 LE expr2 expr1 GT expr2 expr1 GE expr2	Signed, 16-bit comparison
Logical inequality	expr1 LO expr2 expr1 LOE expr2 expr1 HI expr2 expr1 HIE expr2	Unsigned 16-bit comparison
Complement	NOT expr	16-bit one's complement
Conjunction	expr1 AND expr2 expr 1 NAND expr2	16-bit boolean AND 16-bit boolean not AND
Disjunction	expr1 QR expr2 expr1 XOR expr2	16-bit boolean OR 16-bit boolean exclusive OR

NOTE Operators are given in order of precedence, highest to lowest. Solid lines separate precedence groups; within each group, precedence is equal and evaluation is left to right. Evaluation results in a 16-bit integer value

UNSIGNED ARITHMETIC

Syntax	Definition
MPY (expr1, expr2)	Low-order 16 bits of unsigned product <expr1>* <expr2>, high order 16 in MDR.
DIV (divisor, dividend)	Unsigned quotient of 32-bit number (MDR, <dividend>) over <divisor>; remainder in MDR
MDR	High-order 16-bits of MPY product and of DIV dividend; remainder of DIV; unsigned carry of + and-

ARRAY DEFINITION

ARRAY name(expr1[,expr2]).	User <name> (previously undefined or name of deleted array) is defined as one- or two-dimension array

DISPLAY STATEMENTS

expr[:f f]	Value of expression
'LITERAL STRING'	Literal string
add1 [TO addr2] [:f f] ? [:f f]	Target memory

Format specification/[:f f]

ASCII	A	set default	G	octal	O[i]
binary	B[i]	hexadecimal	H[i]	symbolic	S
decimal	D[i]	instruction	I	unsigned	U[i]
name =	E	newline	N[j]	space	X[j]

Note: 1< =i< = 9 — field width i' digits, then two blanks
i = 0 — default field width, no trailing blanks
1< =j< = 9 — repeat 'j' times
j = 0 — repeat 10 times

Response to display/modify mode(?)

forward step	RETURN, +	replace contents	<expr>
back step	–	open new address	@<addr>
exit	.	change display	:f . f

DISASSEMBLER

Instruction	DST	Destination address
operands	SRC	Source address

ASSIGNMENT STATEMENTS

Type	Example	Definition
Variable	sym = expr	User-defined or writable system symbol or REF program label
Target memory	@addr = expr	Put value of <expr> at target <addr>
Proc/Func argument	ARG n = expr	Local copy of argument in position <n> of call list.
Command local	LOC n = expr	Word <n> of local storage array
Array	A[(i1[,i2])] = e	User defined array name; zero, one, or two index expressions

NOTE Precedence of @, ARG, and LOC may require parenthesis around following expression.

COMPOUND STATEMENTS

Syntax	Definition
BEGIN statements END	Statements are executed sequentially. Use in place of any single statement syntax.

CONTROL STATEMENTS

IF expr THEN s1 [ELSE s2]	<s1> is executed if <expr> is TRUE (nonzero). Otherwise, <s2> is executed, if included
CASE expr OF expr 1 s1, ... exprn sn [ELSE s] END	Statement <si> at first label expression <expr> equal to <expr> is executed. If none, statement <s> is executed, if included.
WHILE expr DO statement	While <expr> is TRUE (nonzero), <statement> is executed
REPEAT statement UNTIL expr	<statement> is executed. If <expr> FALSE (zero), <statement> is executed until <expr> is TRUE
FOR var = expr1 TO expr 2[BY expr3] DO statement	Value of <expr1> is assigned to <var>. <statement> is executed until <var> is equal to <expr2>. <expr3> is added to <var>, and <statement> repeated. Default value of <exp3> is 1
ESCAPE	Exit from innermost enclosing WHILE, REPEAT, or FOR statement

PROCEDURE/FUNCTION/FORM DEFINITION

PROC name [(args[,locs])] statements END FUNC name [(args[,locs])] statements END	User-defined <name> (previously undefined or deleted procedure/function) is bound to <statements>. <args> is the required number of arguments. <locs> is the size of local storage array
RETURN [expr]	Pass control back to calling statement. In a procedure, <expr> is ignored. In a function, value of <expr> replaces the function call in the calling expression.
FORM name 'prompt' [= [{ constant / 'string' }]], . . . END	<name> must be a previously defined procedure or function, semicolon required between prompts

PROCEDURE/FUNCTION CALLS

proc name [(expr, . .)]	User-defined or system procedure/function with list of argument expressions
func name [(expr, . .)]	Command definition determines number of arguments required. Some system commands require quoted strings as arguments.

NOTE Procedure/functions with defined FORM when called with no arguments will prompt for arguments using the FORM

example FORM:

COMMENTARY ENTRY	comment, not a prompt required argument,
PROMPT 1 = default value PROMPT 2 = PROMPT 3" =	with default value required argument, must enter value default given if value not entered

FORM control function keys:

Next prompt:	TAB, ↓, →FIELD, SKIP, RETURN
Previous prompt:	↓, ←FIELD
First prompt	HOME
Erase value.	ERASE FIELD, ERASE INPUT
Redisplay default	INSERT LINE
Duplicate previous value:	F4

INPUT/OUTPUT COMMANDS

Syntax	Definition
HCRB	Host computer CRU base address
HCRR (offset,width)	Read host computer CRU field
HCRW (offset,width,value)	Write <value> into host CRU field
COPY ({ 'file' / edit id })	AMPL input from 'file' AMPL input from edit buffer
LIST ({ 'file' / OFF / ON / EOF })	Initialize listing device or file. Disable listing output. Enable listing output. Close listing device or file with EOF
NL	Print newline

unit = OPEN [({ 'file' / edit id }) [, { 0 / IN / OUT / IO } [, { 0 / REWIND / EXTEND } [, { SEQ / REL }]]]]

no arguments — list all open units and edit buffers
initialize 'file'/<edit id> I/O unit
0 — device IO, file IN only
IN — for input only
OUT — for output only
IO — for input/output
REWIND — position to beginning of file
EXTEND — position to end of file
SEQ — auto-create sequential file
REL — auto-create rel-rec file

event-READ [(unit [, { 0 / DIRECT } [, { 0 / GRAPH }]] [, { VDT / SEQ / REL } , { 0 / f row } [, { 0 / f col } [, s col]]] [, rec #]]])

no arguments — read console
Read record from (unit)
0 — issue read ASCII
DIRECT — issue read direct
GRAPH — read graphics on 922 VDT
VDT — read in cursor positioning mode
f row — field start row
f col — field start column
s col — cursor start column
SEQ — read sequentially
REL — read sepecified record
rec # — record number to read
<event>/256 = cursor column after read if VDT
<event> AND 255 = event key value if VDT,
else >0D for end of record,
>13 for end of file

value = EVAL [(unit)]	Evaluate expression in <unit>'s buffer, if no <unit>, READ/EVAL the console
DPLY [(unit)]	AMPL display unit for output to <unit>; if no <unit>, to console
okay = MOVE (from unit to unit)	Move contents of <from unit>'s buffer to <to unit>'s buffer <okay> = 0 if moved = >FFFF if too big and not moved
REW[(unit)]	Rewind (unit) — repositions, file clears console no argument — clears console

Cursor = WRIT (unit [, { 0 / DIRECT } [, { 0 / GRAPH }]] [, { VDT / SEQ / REL } , { 0 / f row } [, [f col]]] [, rec #]]])

no arguments — write console
Write record to (unit),
0 — issue write ASCII
DIRECT — issue write direct
GRAPH — write graphics on 911 VDT
VDT — write in cursor positioning mode
f row — field start row
f col — field start column
SEQ — write sequentially
REL — read specified record
rec # — record number to read
<cursor>/256 = cursor column after write if VDT

CLSE (unit [, { EOF / UNLOAD }]) Release I/O <unit>
EOF — write end-of-file mark
UNLOAD — unload unit

SYSTEM SYMBOLS

V — variable F — function P — procedure

CLR	P — clear	MDEL	P — syn
CLSE	P — I/O close	MDR	V — arit
COPY	P — copy	MIN	V — min
CRUB	V — CRU base	MOVE	F — I/O
CRUR	F — CRU read	MPY	F — mul
CRUW	P — CRU write	MSYM	P — syn
DAY	V — day	NL	P — new
DBUF	P — delete buffer	OPEN	F — I/O
DELE	P — delete symbol	PC	V — reg
DIV	F — divide	R0-R15	V — reg
DPLY	P — display	READ	F — I/O
DR	P — registers	REW	P — I/O
DST	V — destination	RSTR	P — rest
DUMP	P — dump	SAVE	P — sav
EBRK	P — emulator	SEC	V — sec
ECLK	V — emulator	SRC	V — sou
EDIT	F — edit	ST	V — reg
EHLT	F — emulator	TBRK	P — trac
EINT	P — emulator	TEVT	P — trac
EMEM	V — emulator	THLT	F — trac
ERUN	P — emulator	TINT	P — trac
EST	F — emulator	TNCE	V — trac
ETB	F — emulator	TNE	V — trac
ETBH	F — emulator	TRUN	P — trac
ETBO	V — emulator	TST	F — trac
ETRC	P — emulator	TTB	F — trac
ETYP	V — emulator	TTBH	F — trac
EVAL	F — evaluate	TTBN	V — trac
EXIT	P — exit AMPL	TTBO	V — trac
HCRB	V — host CRU	TTRC	P — trac
HCRR	F — CRU read	USYM	P — use
HCRW	P — CRU write	VRFY	P — ver
HR	V — hour	WAIT	F — del
IOR1	V — I/O	WP	V + reg
KEEP	P — keep edit	WRIT	P — I/O
LIST	P — list	YR	V — yea
LOAD	P — load object		

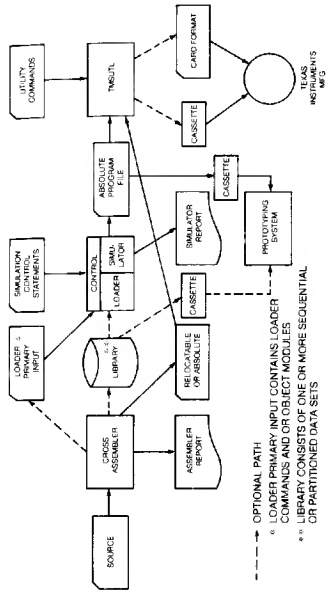
EDIT

Syntax	Definition
edit id = EDIT[({ 'file' / edit id } [,record])]	Create edit buffer with 'file'. Ed No argument creates an empty
KEEP (edit id, 'file')	Save edit buffer onto 'file' and
DBUF (edit id)	Delete edit buffer

EDIT CONTROL FUNCTION KEYS

Function	911 KEY	913 KEY
edit/compose mode	F7	F7
quit edit mode	CMD	HELP
roll up	F1	F1
roll down	F2	F2
set tab	F3	F3
clear tab	F4	F4
tab	TAB (shift SKIP)	TAB
back tab	FIELD	BACK TAB
newline	RETURN	NEWLINE
insert line	unlabeled gray	INSERT LINE
delete line	ERASE INPUT	DELETE LINE
erase line	ERASE FIELD	CLEAR
truncate line	SKIP	SET
insert character	INS CHAR	INSERT CHAR
delete character	DEL CHAR	DELETE CHAR
cursor up	.	.
cursor down	.	.
cursor right	.	.
cursor left	.	←
top of screen	HOME	HOME

neral purpose utility program that accepts as input TI microprocessor
ROM manufacturing formats, or ROM manufacturing formats. This data is
, output options are gathered, the input data converted and an output file



T CONTROL CARD FORMATS

CRIPTION

[addr1 addr2] [WIDTH = x] [PARTITION = y]

— is the format number (integer 1-12)
— is the starting address where input data is to be stored.
— is the maximum address where data is to be stored
— is the bit width of the input words
— is the number of input data set partitions 1 Y 4

m addr1 addr2 WIDTH = x PARTITION = y

— is the format number (integer 1-12)
— is the minimum address to be output.
— is the maximum address to be output
— is the bit width of an output word
—

OMMAND FILE indicator

ORMATS

ORMAT	INPUT	OUTPUT
exadecimal # 1 (PROM)	X	X
exadecimal # 2 (ROM)	X	X
NPF	X	X
71 & 371 ROM/HILO of prototyping System	X	X
MS8080/TMS1000 Absolute Object from M8080/SIM1000 Loader/Simulator	X	X
MS1000 Absolute ROM Object from Assembler	X	X
MS1000 Listed Absolute Object	X	X
MS1000 OPLA Data	X	
MS9900 Standard Absolute Object of Cross upport System (Assembler or Loader/Simulator) & rototyping System	X	X
MS9900 Compressed Absolute Object of rototyping System	X	X
4700 ROM	X	X
4800 ROM	X	X

TMSUTL FORMAT PATHS

Output Format →	1	2	3	4	5	6	7	8	9	10	11	12
1) Hexadecimal # 2 (PROM)	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	YES
2) Hexadecimal # 2 (ROM)	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	YES
3) BNPF	YES	YES	YES	YES	YES	YES	YES	NO	YES	YES	YES	YES
4) 271 & 371 ROM/HILO of Prototyping System	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	YES
5) TMS1000 / TMS8080 Absolute Object from Loader/Simulator	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES
6) TMS1000 Absolute ROM Objects from Assembler for masking	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES
7) TMS1000 Listed Absolute Object	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES
8) TMS1000 OPLA Data	YES	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO
9) TMS9900 Standard Absolute Object of Cross Support System (Assembler or Loader/Simulator) & Prototyping System	YES	YES	YES	YES	NO	NO	NO	NO	YES	YES	YES	YES
10) TMS9900 Compressed Absolute Object of Protoyping System	YES	YES	YES	YES	NO	NO	NO	NO	YES	YES	YES	YES
11) TI4700 ROM	YES	YES	YES	YES	YES	NO	YES	NO	NO	NO	YES	YES
12) TI4800 ROM	YES	YES	YES	YES	YES	NO	YES	NO	NO	NO	YES	YES

DATA DELIMITERS

The following is a table of data delimiters or end-of-module records for Input Data

FORMAT #	TYPES
1. Hex format 1	End of file record (:00)
2. Hex format 2	Trailer record — "END OF TEXT" (hollerith code 12-9-3) character followed by 79 non-blank characters (without asterisks)
3. BNPF	End of file record ($ in column 1)
4. 271/371 ROM and HILO of Prototyping System	End of file record ($END)
5. TMS8080/TMS1000 Absolute Object from Loader/Simulator	End record (+ END)
6. TMS1000 Absolute ROM Object	End of file record ($END)
7. TMS1000 Listed Absolute Object	End of file record ($END)
8 TMS1000 OPLA Data	End of file record ($ENO)
9. TMS9900 Standard Absolute Object	End of module record (:)
10 TMS9900 Binary Compressed Absolute Object	End of file record ($END)
11 TI4700 ROM	End of file record ($END)
12 TI4800 ROM	End of file record ($END)

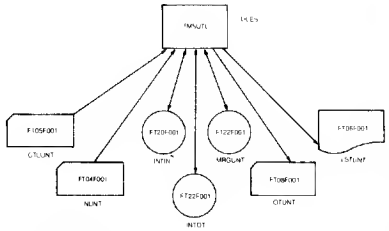
ADDRESS RANGES FOR FORMATS

FORMAT#	FORMAT	ADDRESS RANGE
1	Hexadecimal # 1 (PROM)	$(0\text{-}FFFF)_H$
2	Hexadecimal # 2 (ROM)	None
3	BNPF	None
4	271 & 371 ROM/HILO of Prototyping System	None
5	TMS8080/TMS1000 Absolute Object from Loader/Simulator	(0-255)
6	TMS1000 Absolute ROM Object	$(0\text{-}800)_H$
7	TMS1000 Listed Absolute Object	(0-1 Chapter 0-15 page 0-3F location)$_H$
8	TMS1000 OPLA Data	$(0\text{-}1F)_H$
9	TMS9900 Standard Absolute Object	$(0\text{-}FFFF)_H$
10	TMS9900 Compressed Absolute Object	$(0\text{-}FFFF)_H$
11	TI4700 ROM	$(0\text{-}400)_H$
12	TI4800 ROM	$(0\text{-}400)_H$

INPUT AND OUTPUT WIDTHS FOR FORMATS

FORMAT#	FORMAT	WIDTH (BITS)
1	Hexadecimal # 1 (PROM)	8
2	Hexadecimal # 2 (ROM)	8
3	BNPF	2 or 4 or 8 or 16
4	271 & 371 ROM/HILO of Prototyping System	4 or 8
5	TMS8080/TMS1000 Absolute Object from Loader/Simulator	8
6	TMS1000 Object from Assembler	8
7	TMS1000 Listed Absolute Object	8
8	TMS1000 OPLA Data	8 or 16
9	TMS9900 Standard Absolute Object	16
10	TMS9900 Compressed Absolute Object	16
11	TI4700 ROM	8
12	TI4800 ROM	4 or 8

FILES DEFINITIONS & DESCRIPTIONS



CTLUNT — Input file for control cards
INUNT — Input file for data
INTIN — Intermediate file for storage of input data. It must be a rewindable file with a logical record length of 80 bytes.
INTOT — Intermediate file for storage of internal data. It must be a rewindable file with a logical record length of 80 bytes
OTUNT — Output file for translated data
LSTUNT — Print file for listing of data and error messages.
MRGUNT — Intermediate file for storage of internal data. It must be a rewindable file with a logical record length of 80 bytes.

TEXAS INSTRUMENTS

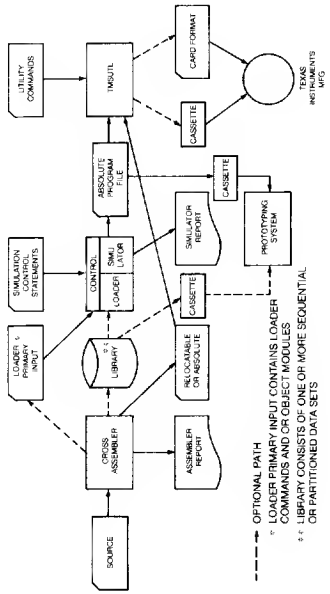
Cross Support Reference Data

Microprocessor Series™

TMSUTL

CONCEPT

TMSUTL is a general purpose utility program that accepts as input TI microprocessor object format, PROM manufacturing formats, or ROM manufacturing formats. This data is syntax checked, output options are gathered, the input data converted and an output file is produced.



INPUT, OUTPUT CONTROL CARD FORMATS

GENERAL DESCRIPTION

INPUT trmt [addr1 addr2] [WIDTH = x] [PARTITION = y]

- trmt — is the format number (integer 1-12).
- addr1 — is the starting address where input data is to be stored
- addr2 — is the maximum address where data is to be stored
- x — is the bit width of the input words
- y — is the number of input data set partitions 1 Y 4

OUTPUT num addr1 addr2 WIDTH = x PARTITION = y

- num — is the format number (integer 1-12)
- addr1 — is the minimum address to be output.
- addr2 — is the maximum address to be output
- x — is the bit width of an output word
- y —

EOF—End of COMMAND FILE indicator

AVAILABLE FORMATS

FORMAT #	FORMAT	INPUT	OUTPUT
1	Hexadecimal #1 (PROM)	X	X
2	Hexadecimal #2 (ROM)	X	X
3	BNPF	X	X
4	271 & 371 ROM/HILO of prototyping System	X	X
5	TMS8080/TMS1000 Absolute Object from SIM8080/SIM1000 Loader/Simulator	X	X
6	TMS1000 Absolute ROM Object from Assembler	X	X
7	TMS1000 Listed Absolute Object	X	X
8	TMS1000 OPLA Data	X	
9	TMS9900 Standard Absolute Object of Cross Support System (Assembler or Loader/Simulator) & Prototyping System	X	X
10	TMS9900 Compressed Absolute Object of Prototyping System	X	X
11	TI4700 ROM	X	X
12	TI4800 ROM	X	X

TMSUTL FORMAT PATHS

Output Format →	1	2	3	4	5	6	7	8	9	10	11	12
1) Hexadecimal #2 (PROM)	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	YES
2) Hexadecimal #2 (ROM)	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	YES
3) BNPF	YES	YES	YES	YES	YES	YES	YES	NO	YES	YES	YES	YES
4) 271 & 371 ROM/HILO of Prototyping System	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	YES
5) TMS1000 / TMS8080 Absolute Object from Loader/Simulator	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES
6) TMS1000 Absolute ROM Objects from Assembler for masking	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES
7) TMS1000 Listed Absolute Object	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES
8) TMS1000 OPLA Data	YES	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO
9) TMS9900 Standard Absolute Object of Cross Support System (Assembler or Loader/Simulator) & Prototyping System	YES	YES	YES	YES	NO	NO	NO	NO	YES	YES	YES	YES
10) TMS9900 Compressed Absolute Object of Protoyping System	YES	YES	YES	YES	NO	NO	NO	NO	YES	YES	YES	YES
11) TI4700 ROM	YES	YES	YES	YES	YES	NO	YES	NO	NO	NO	YES	YES
12) TI4800 ROM	YES	YES	YES	YES	YES	NO	YES	NO	NO	NO	YES	YES

DATA DELIMITERS

The following is a table of data delimiters or end-of-module records for Input Data.

FORMAT #	TYPES
1. Hex format 1	End of file record (:00)
2. Hex format 2	Trailer record — "END OF TEXT" (hollerith code 12-9-3) character followed by 79 non-blank characters (without asterisks)
3. BNPF	End of file record ($ in column 1)
4. 271/371 ROM and HILO of Prototyping System	End of file record ($END)
5. TMS8080/TMS1000 Absolute Object from Loader/Simulator	End record (+ END)
6. TMS1000 Absolute ROM Object	End of file record ($END)
7. TMS1000 Listed Absolute Object	End of file record ($END)
8. TMS1000 OPLA Data	End of file record ($END)
9. TMS9900 Standard Absolute Object	End of module record (.)
10. TMS9900 Binary Compressed Absolute Object	End of file record ($END)
11. TI4700 ROM	End of file record ($END)
12. TI4800 ROM	End of file record ($END)

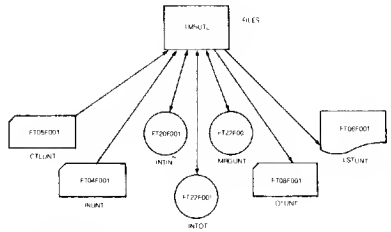
ADDRESS RANGES FOR FORMATS

FORMAT#	FORMAT	ADDRESS RANGE
1	Hexadecimal #1 (PROM)	$(0\text{-}FFFF)_H$
2	Hexadecimal #2 (ROM)	None
3	BNPF	None
4	271 & 371 ROM/HILO of Prototyping System	None
5	TMS8080/TMS1000 Absolute Object from Loader/Simulator	(0-255)
6	TMS1000 Absolute ROM Object	$(0\text{-}800)_H$
7	TMS1000 Listed Absolute Object	(0-1 Chapter 0-15 page 0-3F location)$_H$
8	TMS1000 OPLA Data	$(0\text{-}1F)_H$
9	TMS9900 Standard Absolute Object	$(0\text{-}FFFF)_H$
10	TMS9900 Compressed Absolute Object	$(0\text{-}FFFF)_H$
11	TI4700 ROM	$(0\text{-}400)_H$
12	TI4800 ROM	$(0\text{-}400)_H$

INPUT AND OUTPUT WIDTHS FOR FORMATS

FORMAT#	FORMAT	WIDTH (BITS)
1	Hexadecimal #1 (PROM)	8
2	Hexadecimal #2 (ROM)	8
3	BNPF	2 or 4 or 8 or 16
4	271 & 371 ROM/HILO of Prototyping System	4 or 8
5	TMS8080/TMS1000 Absolute Object from Loader/Simulator	8
6	TMS1000 Object from Assembler	8
7	TMS1000 Listed Absolute Object	8
8	TMS1000 OPLA Data	8 or 16
9	TMS9900 Standard Absolute Object	16
10	TMS9900 Compressed Absolute Object	16
11	TI4700 ROM	8
12	TI4800 ROM	4 or 8

FILES DEFINITIONS & DESCRIPTIONS



- CTLUNT — Input file for control cards.
- INUNT — Input file for data.
- INTIN — Intermediate file for storage of input data. It must be a rewindable file with a logical record length of 80 bytes
- INTOT — Intermediate file for storage of internal data. It must be a rewindable file with a logical record length of 80 bytes
- OTUNT — Output file for translated data
- LSTUNT — Print file for listing of data and error messages
- MRGUNT — Intermediate file for storage of internal data. It must be a rewindable file with a logical record length of 80 bytes.

TEXAS INSTRU

Cross Sup Reference Data

Microprocesso Series™

er data base which is assigned to PUNIT, is read by the FORTRAN
tile at execution time. It is the actual Cross Assembler program written in
t is suggested that it be assigned to a permanent disk file.

DEFAULT UNIT	DEVICE TYPE	RECORD LENGTH	FUNCTION
5	CR,CS MT,DF	80	TMS 9900 Source Input
6	CS,MT	80	Listing Output
7	CS,MT	80	TMS9900 Object Output
10	MT,DF	80	Assembly Scratch
11	CR,CS	80	Data Base INPUT

R CS—CASSETTE TAPE MT—MAGNETIC TAPE DF—DISKFILE CP—
—LINE PRINTER

CROSS ASSEMBLER SYSTEM FILES

xpression value in the location counter, and defines the succeeding
te

AORG

IN

:
label >]ø AORGø <wd-exp>ø . [<comment>]
xpression value in the location counter, and defines the succeeding
able

RORG

RIGIN

:
<label>ø RORGø [<exp>]ø [<comment>]
xpression value in the location counter, and defines the succeeding
ny section. No object code is generated in a dummy section

DORG

:
<label>ø DORGø <exp>ø [<comment>]
e label, if present, and increments the location counter by the value of the

BSS

G WITH SYMBOL

:
<label>ø BSSø <wd-exp>ø [<comment>]
s the location counter by the value of the expression, and then assigns

BES

WITH SYMBOL

:
abel>]ø BESø <wd-exp>ø [<comment>]
embly-time constant to the label

EQU

LY-TIME CONSTANT

:
<label>ø EQUø <exp>ø [<comment>]
he label, if present, and then aligns the location counter on a word
dress)

EVEN

RY

:
[<label>]ø EVENø [<comment>]
pss referencing when XREF is specified, and allows printing of the
SYMT is present.

OPTION

S

:
DNø <keyword>[,<keyword>] ø [<comment>]
to the program, and must precede any code-generating directive or

IDT

TIFIER

:
<label>]ø IDTø <string>ø [<comment>]
ng to be printed at the top of each subsequent source listing page

TITL

:
label>]ø TITLø <string>ø [<comment>]
ng of the source listing

LIST

:
[<label>]ø . LISTø [<comment>]
of the source listing

UNL

T

:
[<label>]ø . UNLø [<comment>]
sembler to continue the source listing on the next page

PAGE

:
[<label>]ø PAGEø . . [<comment>]
ssions in successive bytes, optionally assigning the label the address of

BYTE

:
]ø BYTEø <exp>[,<exp>] . . ø . . [<comment>]

DATA places expressions in successive words, optionally assigning the label the address of the first word.

DATA

INITIALIZE WORD

Syntax Definition:

[<label>]ø . . DATAø . <exp>[,<exp>] ø . . . [<comment>]

TEXT places characters in successive bytes, arithmetically negating the last character, and optionally assigns the label the address of the first character.

TEXT

INITIALIZE TEXT

Syntax Definition:

[<label>]ø TEXTø [–]<string>ø [<comment>]

DEF makes symbols available to other programs as external references.

DEF

EXTERNAL DEFINITION

Syntax Definition:

[<label>]ø . DEFø <symbol>[,<symbol>] ø [<comment>]

REF directs the assembler to look externally for symbols.

REF

EXTERNAL REFERENCE

Syntax Definition:

[<label>]ø REFø <symbol>[,<symbol>] ø [<comment>]

DXOP assigns an extended operation to a symbol.

DXOP

DEFINE EXTENDED OPERATIONS

Syntax Definition:

[<label>]ø . DXOPø . <symbol>,<term>ø [<comment>]

END terminates the assembly.

END

PROGRAM END

Syntax Definition:

[<label>]ø ENDø [<symbol>]ø [<comment>]

NOP places a no-operation code in the object file.

NOP

NO OPERATION

Syntax Definition:

[<label>]ø NOPø . [<comment>]

RT assembles as a return from subroutine by substituting a branch through register 11.

RT

RETURN

Syntax Definition:

[<label>]ø RTø . [<comment>]

SIMULATOR FILES

INTERNAL NAME	DEFAULT UNIT	DEVICE TYPE	RECORD LENGTH	FUNCTION	WHERE USED
INCOPY	4	MT,DF	80	Batch copy file	C
INCOM	5	TE,CR MT,DF	80	Simulation command	C
OUTPRT OUTTRC	6	MT,DF TE,CR	80 or 136	Listing output	L,C,R
INLOD	10	TE,CR MT,DF	80	Linker commands	L
OUTCOM	11	TE,LP	80 or 136	Prompts and error msg for linker output	L
OUTSAV	17	MT,CP DF	80	Absolute object	L,S
INSCR	20	MT,DF	136	Input scratch file	C,R,S
OUTSCR	21	MT,DF	136	Output scratch file	L,C,R

Device type legend
TE—terminal; CR—card reader, MT—magnetic tape DF—disk file, CP—card punch

Where used legend
L—link processor, C—command processor; R—run processor, S-save processor

In addition to the above unit number assignments, the user must also assign unique FORTRAN logical unit numbers to each TMS9900 object code module to be included in the LINK processor.

SIMULATOR DIRECTIVES

ORIGIN COMMAND. The "ORIGIN" command can be used to specify where relocatable code is to be loaded.

ORIGIN hex-number

INCLUDE COMMAND. The "INCLUDE" command directs the loader to load an object module from a data set (e.g., card reader, disc, tape). The data set must be a sequential data set and may contain one or more object modules. At least one "INCLUDE" command should be used in the LINK processor command stream. The format for the command is as follows:

INCLUDE n

ENTRY COMMAND. The "ENTRY" command specifies the program entry point to the loader. The format for the command is as follows:

ENTRY name

SUMMARY OF CONTROL LANGUAGE STATEMENTS

The formats of the control statements for the "COMMAND" processor are shown below, with a brief description following.

[label] {R | RUN} [*] {F | FOR} n [{FR | FROM} n1] [{T | TO}] n2 [,label]	Specifies where to start and stop simulation. Control passes to statement at label operand when a breakpoint occurs
[label] {T | TRACE} [list]	Specifies locations to be traced
[label] {NOT | NOTRACE} [list]	Disables trace for specified locations
[label] {RE | REFER} [list]	Specifies locations for reference breakpoint
[label] {NOR | NOREFER} [list]	Disables reference breakpoint at specified locations.
[label] {A | ALTER} [list]	Specifies locations for alteration breakpoint
[label] {NOA | NOALTER} [list]	Disables alteration breakpoint at specified locations.
[label] {P | PROTECT} [list]	Specifies areas for memory protection
[label] IF (logical expression) label	Conditional transfer of control program
[label] {J | JUMP} label	Unconditional transfer of control program.
[label] {TI | TIME} [n]	Prints the value of the 9900 time and optionally sets a new value.
[label] {D | DISPLAY} [D] {CP | CPU} [register list]	Prints contents of registers
[label] {D | DISPLAY} [D | C] {M | MEMORY} list	Prints contents of memory
[label] {D | DISPLAY} {S | SYMBOL} [$ | symbol | number]	Prints values from symbol table.
[label] {D | DISPLAY} {CR | CRU} {I | INPUT | O | OUTPUT} list	Prints CRU values
[label] {S | SET} {C | CPU} register-value list	Places values into registers
[label] {S | SET} {M | MEMORY} location-value list	Places values into memory.
[label] {S | SET} {I | INT} level, n_1 [,n_2,n_3]	Sets up one or more interrupts
[label] {E | END}	Disables breakpoints and traces, and initializes simulation. Passes control to next control statement
[label] {I | INPUT} {n1 [n TO n2]} [F | FIRST | L | LAST | A | ALL] [data]	Defines input lines and fields, and supplies data for program being simulated
[label] {O | OUTPUT} {n1 [n TO n2]}	Defines output lines and fields, or prints output of program being simulated
[label] {CONN | CONNECT} list	Connects input CRU lines to output CRU lines
[label] {C | CONVERT} expression list	Evaluates and prints values of expressions in decimal and hexadecimal form
{B | BATCH}	Specifies batch mode
[label] {L | LOAD}	Loads Wp and PC from locations $FFFC_{16}$ and $FFFE_{16}$.
[label] {CL | CLOCK} t	Specify clock period
[label] {M | MEMORY} {RA | RAM | RO | ROM} {R | READ} = n1 [{W | WRITE} = n2] list	Define available memory. Default is 32K RAM
[label] {SA | SAVE}	Create absolute object module
[label] {W | WIDTH} n	Specifies number of columns available for printing

MONITOR COMPLETION CODES

The simulator signals completion by executing and writing an appropriate STOP I statement, where I takes on one of the following values:

CODE	MEANING
0	Normal completion
1	Abnormal completion from LNKPRC
2	Premature EOF —If this error occurs it indicates that a premature EOF was encountered while attempting to reposition the BATCH command file.
3	Internal error; invalid label value
4	Roll memory overflow
5	Loader error —If this error occurs it means an attempt was made to load an object file into simulated memory and it failed causing termination of the link processor
8	Abnormal completion from LOADER
9	Abnormal completion from CMDPRC
99	Internal error —Illegal completion from CMDPRC Internal error
999	Internal error —Illegal parameter passed to WRITER

If an error of 99 or 999 results, an internal error has occurred and the error should be reported to TEXAS INSTRUMENTS INC.

LINK PROCESSOR ERRORS

CODE	MESSAGE
L01	Load not completed
L02	Multiply defined external symbol (name)
L03	Empty object file on unit
L04	Attempt to load undefined memory
L05	Tag D follows tag 0
L06	Invalid tag character
L09	Undefined external memory
L13	Empty memory on save
L14	(name) not in external symbol table
L18	Maximum memory size exceeded
L19	Missing end
L21	Checksum error (computed value)
L22	Odd origin value specified—even value used
L24	Ref chain loop
L25	Object module does not start with tag 0
L26	Odd value (value) specified for tag (tag) even value used
L27	Missing F tag in record (number)
L28	Bad REF chain for (name)
L29	Bad object format in object module
L30	Illegal hex digit in field (digit)

COMMAND PROCESSOR ERRORS

CODE NUMBER	NAME	MESSAGE	CODE NUMBER	NAME	MESSAGE
1	BADCHR	Bad character	18	RANGE	Range error
2	BADCMD	Unrecognizable command	19	SYNTAX	Syntax error
3	BADIGT	Bad digit	20	TOOMNY	Too many values
4	BADMOD	Bad module name	21	UNDEF	Undefined symbol
5	BADREG	Bad register mnemonic			
6	BADVAL	Bad value			
7	CRUSPC	CRU specification error			
8	FLDCNT	Too few / many fields			
9	HITEOF	Hit EOF			
10	HITEOL	Hit end-of-line			
11	MEMDEF	Undefined			
12	MISSEQ	Missing equal sign			
13	NODATA	No data found			
14	NOROL	No data rolls available			
15	NOSET	Set not performed			
16	NOTIMP	Command not implemented			
17	ORDER	Command out of order			

RUN PROCESSOR ERRORS

CODE	MESSAGE
1	PC interrupt vector entry in undefined memory
2	WP interrupt vector entry in undefined memory
3	Register out of address space (WP 65502)
4	Registers in undefined memory
5	Registers in ROM
6	PC interrupt vector refer breakpoint
7	WP interrupt vector refer breakpoint
8	Register alter breakpoint
9	Register protect breakpoint
10	Register refer breakpoint
11	Undefined opcode
12	Undefined memory reference
13,14	Unused
15	PC refer breakpoint
16	Unimplemented opcode
17,18,19	Unused

The Cross Assembler data base which is assigned to PUNIT, is read by the FORTRAN program as the first file at execution time It is the actual Cross Assembler program written in internal code, and it is suggested that it be assigned to a permanent disk file

INTERNAL NAME	DEFAULT UNIT	DEVICE TYPE	RECORD LENGTH	FUNCTION
IUNIT	5	CR,CS MT,DF	80	TMS 9900 Source Input
LUNIT	6	CS MT	80	Listing Output
OUNIT	7	CS MT	80	TMS9900 Object Output
SUNIT	10	MT,DF	80	Assembly Scratch
PUNIT	11	CR,CS	80	Data Base INPUT

CR—CARD READER, CS—CASSETTE TAPE, MT—MAGNETIC TAPE, DF—DISKFILE CP—CARD PUNCH LP—LINE PRINTER

CROSS ASSEMBLER SYSTEM FILES

AORG places the expression value in the location counter, and defines the succeeding locations as absolute

ABSOLUTE ORIGIN **AORG**

Syntax Definition

[<label>]ⱱ AORGⱱ <wd-exp>ⱱ [<comment>]

RORG places the expression value in the location counter, and defines the succeeding locations as relocatable

RELOCATABLE ORIGIN **RORG**

Syntax Definition

[<label>]ⱱ RORGⱱ [<exp>]ⱱ [<comment>]

DORG places the expression value in the location counter, and defines the succeeding locations as a dummy section No object code is generated in a dummy section

DUMMY ORIGIN **DORG**

Syntax Definition:

<label>ⱱ DORGⱱ <exp>ⱱ [<comment>]

BSS first assigns the label, if present, and increments the location counter by the value of the expression

BLOCK STARTING WITH SYMBOL **BSS**

Syntax Definition

[<label>]ⱱ BSSⱱ <wd-exp>ⱱ [<comment>]

BSS first increments the location counter by the value of the expression, and then assigns the label, if present

BLOCK ENDING WITH SYMBOL **BES**

Syntax Definition:

[<label>]ⱱ BESⱱ <wd-exp>ⱱ [<comment>]

EQU assigns an assembly-time constant to the label

DEFINE ASSEMBLY-TIME CONSTANT **EQU**

Syntax Definition

<label>ⱱ EQUⱱ <exp>ⱱ [<comment>]

EVEN first assigns the label If present, and then aligns the location counter on a word boundary (even address)

WORD BOUNDARY **EVEN**

Syntax Definition

[<label>]ⱱ EVENⱱ [<comment>]

OPTIONS allows cross referencing when XREF is specified and allows printing of the symbol table when SYMT is present

OUTPUT OPTIONS **OPTION**

Syntax Definition:

ⱱ OPTIONⱱ <keyword>[,<keyword>] ⱱ [<comment>]

IDT assigns a name to the program, and must precede any code-generating directive or instruction.

PROGRAM IDENTIFIER **IDT**

Syntax Definition

[<label>]ⱱ IDTⱱ . <string>ⱱ [<comment>]

TITL supplies a string to be printed at the top of each subsequent source listing page.

PAGE TITLE **TITL**

Syntax Definition

[<label>]ⱱ TITLⱱ <string>ⱱ [<comment>]

LIST restores printing of the source listing

LIST SOURCE **LIST**

Syntax Definition

[<label>]ⱱ . . LISTⱱ . [<comment>]

UNL inhibits printing of the source listing

NO SOURCE LIST **UNL**

Syntax Definition

[<label>]ⱱ . . UNLⱱ . . . [<comment>]

PAGE directs the assembler to continue the source listing on the next page.

PAGE EJECT **PAGE**

Syntax Definition

[<label>]ⱱ PAGEⱱ [<comment>]

BYTE places expressions in successive bytes, optionally assigning the label the address of the first byte.

INITIALIZE BYTE **BYTE**

Syntax Definition

[<label>]ⱱ BYTEⱱ <exp>[,<exp>] ⱱ [<comment>]

DATA places expressions in successive words, optionally assigning the label the address of the first word.

INITIALIZE WORD **DATA**

Syntax Definition

[<label>]ⱱ . . . DATAⱱ <exp>[,<exp>] ⱱ [<comment>]

TEXT places characters in successive bytes, arithmetically negating the last character, and optionally assigns the label the address of the first character

INITIALIZE TEXT **TEXT**

Syntax Definition:

[<label>]ⱱ TEXTⱱ [−]<string>ⱱ . . [<comment>]

DEF makes symbols available to other programs as external references

EXTERNAL DEFINITION **DEF**

Syntax Definition

[<label>]ⱱ DEFⱱ <symbol>[,<symbol>] ⱱ [<comment>]

REF directs the assembler to look externally for symbols

EXTERNAL REFERENCE **REF**

Syntax Definition

[<label>]ⱱ REFⱱ <symbol>[,<symbol>] ⱱ [<comment>]

DXOP assigns an extended operation to a symbol

DEFINE EXTENDED OPERATIONS **DXOP**

Syntax Definition

[<label>]ⱱ DXOPⱱ <symbol>,<term>ⱱ . . . [<comment>]

END terminates the assembly

PROGRAM END **END**

Syntax Definition

[<label>]ⱱ . . ENDⱱ [<symbol>]ⱱ [<comment>]

NOP places a no-operation code in the object file.

NO OPERATION **NOP**

Syntax Definition

[<label>]ⱱ . . . NOPⱱ . . [<comment>]

RT assembles as a return from subroutine by substituting a branch through register 11

RETURN **RT**

Syntax Definition:

[<label>]ⱱ RTⱱ . [<comment>]

SIMULATOR FILES

INTERNAL NAME	DEFAULT UNIT	DEVICE TYPE	RECORD LENGTH	FUNCTION	WHERE USED
INCOPY	4	MT,DF	80	Batch copy file	C
INCOM	5	TE,CR MT,DF	80	Simulation command	C
OUTPRT OUTTRC	6	MT,DF TE,CR	80 or 136	Listing output	L,C,R
INLOD	10	TE,CR MT,DF	80	Linker commands	L
OUTCOM	11	TE,LP	80 or 136	Prompts and error msg for linker output	L
OUTSAV	17	MT,CP DF	80	Absolute object	L,S
INSCR	20	MT,DF	136	Input scratch file	C,R,S
OUTSCR	21	MT,DF	136	Output scratch file	L,C,R

Device type legend
TE—terminal, CR—card reader, MT—magnetic tape; DF—disk file, CP—card punch

Where used legend
L—link processor, C—command processor, R—run processor; S-save processor

In addition to the above unit number assignments, the user must also assign unique FORTRAN logical unit numbers to each TMS9900 object code module to be included in the LINK processor

SIMULATOR DIRECTIVES

ORIGIN COMMAND The "ORIGIN" command can be used to specify where relocatable code is to be loaded

ORIGIN hex-number

INCLUDE COMMAND. The "INCLUDE" command directs the loader to load an object module from a data set (e.g., card reader, disc, tape) The data set must be a sequential data set and may contain one or more object modules. At least one "INCLUDE" command should be used in the LINK processor command stream The format for the command is as follows

INCLUDE n

ENTRY COMMAND The "ENTRY" command specifies the program entry point to the loader The format for the command is as follows:

ENTRY name

SUMMARY OF CONTROL LANGUAGE STATEMENTS

The formats of the control statements for the "COMMAND" processor are shown below, with a brief description following

[label] {R | RUN} [*] {F | FOR} n [{FR | FROM} i1] [{T | TO} i2 [,label]	Specifies where to start and stop simulation Control passes to statement at label operand when a breakpoint occurs
[label] {T | TRACE} [list]	Specifies locations to be traced.
[label] {NOT | NOTRACE} [list]	Disabtes trace for specified locations
[label] {RE | REFER} [list]	Specifies locations for reference breakpoint.
[label] {NOR | NOREFER} [list]	Disables reference breakpoint at specified locations
[label] {A | ALTER} [list]	Specifies locations for alteration breakpoint
[label] {NOA | NOALTER} [list]	Disables alteration breakpoint at specified locations.
[label] {P | PROTECT} [list]	Specifies areas for memory protection.
[label] IF (logical expression) label	Conditional transfer of control program
[label] {J | JUMP} label	Unconditional transfer of control program.
[label] {TI | TIME} [n]	Prints the value of 9900 time and optionally sets a new value
[label] {D | DISPLAY} [D] {CP | CPU} [register list]	Prints contents of registers.
[label] {D | DISPLAY} [D | C] {M | MEMORY} list	Prints contents of memory.
[label] {D | DISPLAY} {S | SYMBOL} [$ | symbol | number]	Prints values from symbol table
[label] {D | DISPLAY} {CR | CRU} {I | INPUT | O | OUTPUT} list	Prints CRU values
[label] {S | SET} {C | CPU} register-value list	Places values into registers
[label] {S | SET} {M | MEMORY} location-value list	Places values into memory.
[label] {S | SET} {I | INT} level, n_1 [,n_2,n_3]	Sets up one or more interrupts
[label] {E | END}	Disables breakpoints and traces, and initializes simulation Passes control to next control statement
[label {I | INPUT} {n_1 | n TO n_2} [F | FIRST | L | LAST | A | ALL] [data]	Defines input lines and fields, and supplies data for program being simulated.
[label] {O | OUTPUT} {n_1 | n TO n_2}	Defines output lines and fields, or prints output of program being simulated
[label] {CONN | CONNECT} list	Connects input CRU lines to output CRU lines
[label] {C | CONVERT} expression list	Evaluates and prints values of expressions in decimal and hexadecimal form
{B | BATCH}	Specifies batch mode
[label] {L | LOAD}	Loads Wp and PC from locations $FFFC_{16}$ and $FFFE_{16}$
[label] {CL | CLOCK} i	Specify clock period
[label] {M | MEMORY} {RA | RAM | RO | ROM} {R | READ} = n_1 [W | WRITE] = n_2 list	Define available memory Default is 32K RAM
[label] {SA | SAVE}	Create absolute object module.
[label] {W | WIDTH} n	Specifies number of columns available for printing

MONITOR COMPLETION CODES

The simulator signals completion by executing and writing an appropriate STC statement, where I takes on one of the following values:

CODE	MEANING
0	Normal completion
1	Abnormal completion from LNKPRC
2	Premature EOF —If this error occurs it indicates that a premature EOF was enco attempting to reposition the BATCH command file
3	Internal error, invalid label value
4	Rotl memory overflow
5	Loader error —If this error occurs it means an attempt was made to load an o simulated memory and it failed causing termination of the link pr
8	Abnormal completion from LOADER
9	Abnormal completion from CMDPRC
99	Internal error —Illegal completion from CMDPRC Internal error
999	Internal error —Illegal parameter passed to WRITER

If an error of 99 or 999 results, an internal error has occurred and the error sh reported to TEXAS INSTRUMENTS INC

LINK PROCESSOR ERRORS

CODE	MESSAGE
L01	Load not completed
L02	Multiply defined external symbol (name)
L03	Empty object file on unit
L04	Attempt to load undefined memory
L05	Tag D follows tag 0
L06	Invalid tag character
L09	Undefined external memory
L13	Empty memory on save
L14	(name) not in external symbol table
L18	Maximum memory size exceeded
L19	Missing end
L21	Checksum error (computed value)
L22	Odd origin value specified—even value used
L24	Ref chain loop
L25	Object module does not start with tag 0
L26	Odd value (value) specified for tag (tag) even value use
L27	Missing F tag in record (number)
L28	Bad REF chain for (name)
L29	Bad object format in object module
L30	Illegal hex digit in field (digit)

COMMAND PROCESSOR ERRORS

CODE NUMBER	NAME	MESSAGE	CODE NUMBER	NAME	MES
1	BADCHR	Bad character	18	RANGE	Ran
2	BADCMD	Unrecognizable command	19	SYNTAX	Synt
3	BADIGT	Bad digit	20	TOOMNY	Too
4	BADMOD	Bad module name	21	UNDEF	Und
5	BADREG	Bad register mnemonic			
6	BADVAL	Bad value			
7	CRUSPC	CRU specification error			
8	FLDCNT	Too few / many fields			
9	HITEOF	Hit EOF			
10	HITEOL	Hit end-of-line			
11	MEMDEF	Undefined			
12	MISSEQ	Missing equal sign			
13	NODATA	No data found			
14	NOROL	No data rolls available			
15	NOSET	Set not performed			
16	NOTIMP	Command not implemented			
17	ORDER	Command out of order			

RUN PROCESSOR ERRORS

CODE	MESSAGE
1	PC interrupt vector entry in undefined memory
2	WP interrupt vector entry in undefined memory
3	Register out of address space (WP 65502)
4	Registers in undefined memory
5	Registers in ROM
6	PC interrupt vector refer breakpoint
7	WP interrupt vector refer breakpoint
8	Register alter breakpoint
9	Register protect breakpoint
10	Register refer breakpoint
11	Undefined opcode
12	Undefined memory reference
13,14	Unused
15	PC refer breakpoint
16	Unimplemented opcode
17,18,19	Unused

USASCII/HOLLERITH CHARACTER CODE

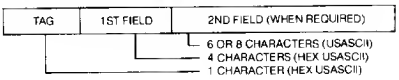
CHAR	USASCII (HEXADECIMAL)	HOLLERITH*	CHAR	USASCII (HEXADECIMAL)	HOLLERITH*
NUL	00		3	33	3
SOH	01		4	34	4
STX	02		5	35	5
ETX	03		6	36	6
EOT	04		7	37	7
ENQ	05		8	38	8
ACK	06		9	39	9
BEL	07			3A	2-8
BS	08		.	3B	11-6-8
HT	09		<	3C	12-4-8
LF	0A		=	3D	6-8
VT	0B		>	3E	0-6-8
FF	0C		?	3F	0-7-8
CR	0D		@	40	4-8
SO	0E		A/a	41/61	12-1
SI	0F		B/b	42/62	12-2
DLE	10		C/c	43/63	12-3
DC1	11		D/d	44/64	12-4
DC2	12		E/e	45/64	12-5
DC3	13		F/f	46/66	12-6
DC4	14		G/g	47/67	12-7
NAK	15		H/h	48/68	12-8
SYN	16		I/i	49/69	12-9
ETB	17		J/j	4A/6A	11-1
CAN	18		K/k	4B/6B	11-2
EM	19		L/l	4C/6C	11-3
SUB	1A		M/m	4D/6D	11-4
ESC	1B		N/n	4E/6E	11-5
FS	1C		O/o	4F/6F	11-6
GS	1D		P/p	50/70	11-7
RS	1E		Q/q	51/71	11-8
US	1F		R/r	52/72	11-9
SPACE	20	BLANK	S/s	53/73	0-2
!	21	11-2-8	T/t	54/74	0-3
"	22	7-8	U/u	55/75	0-4
#	23	3-8	V/v	56/76	0-5
$	24	11-3-8	W/w	57/77	0-6
%	25	0-4-8	X/x	58/78	0-7
&	26	12	Y/y	59/79	0-8
'	27	5-8	Z/z	5A/7A	0-9
(	28	12-5-8	[	5B	12-2-8
)	29	11-5-8		5C	
*	2A	11-4-8	]	5D	12-7-8
+	2B	12-6-8	^	5E	11-7-8
,	2C	0-3-8	_	5F	0-5-8
-	2D	11	`	60	
	2E	12-3-8	{	7B	
/	2F	0-1	>	7C	
0	30	0	}	7D	
1	31	1	~	7E	
2	32	2	DEL	7F	

*PUNCH IN CARD ROWS

HEX-DECIMAL TABLE

EVEN BYTE				ODD BYTE			
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16 384	4	1,024	4	64	4	4
5	20 480	5	1,280	5	80	5	5
6	24 576	6	1,536	6	96	6	6
7	28 672	7	1,792	7	112	7	7
8	32,766	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,066	B	2 816	B	176	B	11
C	49,152	C	3 072	C	192	C	12
D	53,248	D	3 328	D	208	D	13
E	57,344	E	3 584	E	224	E	14
F	61,440	F	3 840	F	240	F	15

OBJECT RECORD FORMAT AND CODE



TAG	FIRST FIELD	SECOND FIELD	MEANING
0	LENGTH OF ALL RELOCATABLE CODE	PROGRAM ID (8-CHARACTER)	PROGRAM START
1	ADDRESS	(NOT USED)	ABSOLUTE ENTRY ADDRESS
2	ADDRESS	(NOT USED)	RELOCATABLE ENTRY ADDRESS
3	LOCATION OF LAST APPEARANCE OF SYMBOL	6 CHARACTER SYMBOL	EXTERNAL REFERENCE LAST USED IN RELOCATABLE CODE
4	LOCATION OF LAST APPEARANCE OF SYMBOL	6 CHARACTER SYMBOL	EXTERNAL REFERENCE LAST USED IN ABSOLUTE CODE
5	LOCATION	6 CHARACTER SYMBOL	RELOCATABLE EXTERNAL DEFINITION
6	LOCATION	6 CHARACTER SYMBOL	ABSOLUTE EXTERNAL DEFINITION
7	CHECKSUM FOR CURRENT RECORD	(NOT USED)	CHECKSUM
8	ANY VALUE	(NOT USED)	IGNORE CHECKSUM VALUE
9	LOAD ADDRESS	(NOT USED)	ABSOLUTE LOAD ADDRESS
A	LOAD SDDRESS	(NOT USED)	RELOCATABLE LOAD ADDRESS
B	DATA	(NOT USED)	ABSOLUTE DATA
C	DATA	(NOT USED)	RELOCATABLE DATA
D	LOAD BIAS	(NOT USED)	LOAD BIAS OR OFFSET (NOT A PART OF ASSEMBLER OUTPUT)
E			ILLEGAL
F	(NOT USED)	(NOT USED)	END OF RECORD

TEXAS INSTRUMENTS

9900 Reference Data

Microprocessor Series™

INSTRUCTION FORMAT

FORMAT (USE)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1 (ARITH)	OP CODE			B	T_D			D			T_S		S			
2 (JUMP)	SIGNED DISPLACEMENT*															
3 (LOGICAL)	OP CODE							D			T_S		S			
4 (CRU)	OP CODE							C			T_S		S			
5 (SHIFT)	OP CODE								C				W			
6 (PROGRAM)	OP CODE										T_S		S			
7 (CONTROL)	OP CODE												NOT USED			
	OP CODE											NU	W			
8 (IMMEDIATE)	IMMEDIATE VALUE															
9 (MPY,DIV XOP)	OP CODE							D			T_S		S			

KEY

B = BYTE INDICATOR
(1 – BYTE, 0 – WORD)
T_D = D ADDR MODIFICATION
D = DESTINATION ADDR
T_S = ADDR MODIFICATION

S = SOURCE ADDR
C = XFR OR SHIFT LENGTH (COUNT)
W = WORKSPACE REGISTER NO
= SIGNED DISPLACEMENT OF – 128 TO + 127 WORDS
NU = NOT USED

T_D/T_S FIELD

CODE		EFFECTIVE ADDRESS	MNEMONIC
00	REGISTER	WP + 2 · [S OR D]	Rn
01	INDIRECT	(WP + 2 · [S OR D])	*Rn
10	INDEXED (S OR D ≠ 0)	(WP + 2 · [S OR D]) + (PC) PC ← PC + 2	NUM (Rn)
10	SYMBOLIC (DIRECT, S OR D = 0)	(PC) PC ← PC + 2	NUM
11	INDIRECT WITH AUTO INCREMENT	(WP + 2 · [S OR D]) INCREMENT EFF ADDR	*Rn+

STATUS REGISTER

0	1	2	3	4	5	6	7	11	12	15
L>	A>	=	C	O	P	X	RESERVED		INTERRUPT MASK	

0 – LOGICAL GREATER THAN
1 – ARITHMETIC GREATER THAN
2 – EQUAL/TB INDICATOR
3 – CARRY FROM MSB
4 – OVERFLOW
5 – PARITY (ODD NO OF BITS SET)
6 – XOP IN PROGRESS

INTERRUPT MASK
F = ALL INTERRUPTS ENABLED
0 = ONLY LEVEL 0 ENABLED

INTERRUPTS

TRAP ADDR	WP
TRAP ADDR + 2	PC

LEVEL	ID	TRAP ADDR	LEVEL	ID	TRAP ADDR
0	RESET	0000	8	EXTERNAL	0020
1	EXTERNAL	0004	9	EXTERNAL	0024
2	EXTERNAL	0008	10	EXTERNAL	0028
3	EXTERNAL	000C	11	EXTERNAL	002C
4	EXTERNAL	0010	12	EXTERNAL	0030
5	EXTERNAL	0014	13	EXTERNAL	0034
6	EXTERNAL	0018	14	EXTERNAL	0038
7	EXTERNAL	001C	15	EXTERNAL	003C

NOTES 1) XOP VECTORS 0–15 OCCUPY MEMORY LOCATIONS 0040-007C
2) LOAD VECTOR OCCUPIES MEMORY LOCATIONS FFFC–FFFF

BLWP TRANSFERS	RTWP TRANSFERS	BL TRANSFER	XOP TRANSFER
WP → NEW W13	CURRENT W13 → WP	PC → W11	EFF ADDR → NEW W11
PC → NEW W14	CURRENT W14 → PC		WP → NEW W13
ST → NEW W15	CURRENT W15 → ST		PC → NEW W14
			ST → NEW W15
			1 → ST6

INSTRUCTIONS BY MNEMONIC

MNEMONIC	OP CODE	FORMAT	RESULT COMPARED TO ZERO	STATUS AFFECTED	INSTRUCTIONS
A	A000	1	Y	0-4	ADD(WORD)
AB	B000	1	Y	0-5	ADD(BYTE)
ABS	0740	6	Y	0-4	ABSOLUTE VALUE
AI	0220	8	Y	0-4	ADD IMMEDIATE
ANDI	0240	8	Y	0-2	AND IMMEDIATE
B	0440	6	N	—	BRANCH
BL	0680	6	N	—	BRANCH AND LINK (W11)
BLWP	0400	6	N	—	BRANCH LOAD WORKSPACE POINTER
C	8000	1	N	0-2	COMPARE (WORD)
CB	9000	1	N	0-2,5	COMPARE (BYTE)
CI	0280	8	N	0-2	COMPARE IMMEDIATE
CKOF	03C0	7	N	—	EXTERNAL CONTROL
CKON	03A0	7	N	—	EXTERNAL CONTROL
CLR	04C0	6	N	—	CLEAR OPERAND
COC	2000	3	N	2	COMPARE ONES CORRESPONDING
CZC	2400	3	N	2	COMPARE ZEROES CORRESPONDING
DEC	0600	6	Y	0-4	DECREMENT (BY ONE)
DECT	0640	6	Y	0-4	DECREMENT (BY TWO)
DIV	3C00	9	N	4	DIVIDE
IDLE	0340	7	N	—	COMPUTER IDLE
INC	0580	6	Y	0-4	INCREMENT (BY ONE)
INCT	05C0	6	Y	0-4	INCREMENT (BY TWO)
INV	0540	6	Y	0-2	INVERT (ONES COMPLEMENT)
JEO	1300	2	N	—	JUMP EQUAL (ST2 = 1)
JGT	1500	2	N	—	JUMP GREATER THAN (ST1 = 1)
JH	1B00	2	N	—	JUMP HIGH (ST0 = 1 AND ST2 = 0)
JHE	1400	2	N	—	JUMP HIGH OR EQUAL (ST0 OR ST2 = 1)
JL	1A00	2	N	—	JUMP LOW (ST0 AND ST2 = 0)
JLE	1200	2	N	—	JUMP LOW OR EQUAL (ST0 = 0 OR ST2 = 1)
JLT	1100	2	N	—	JUMP LESS THAN (ST1 AND ST2 = 0)
JMP	1000	2	N	—	JUMP UNCONDITIONAL
JNC	1700	2	N	—	JUMP NO CARRY (ST3 = 0)
JNE	1600	2	N	—	JUMP NOT EQUAL (ST2 = 0)
JNO	1900	2	N	—	JUMP NO OVERFLOW (ST4 = 0)
JOC	1800	2	N	—	JUMP ON CARRY (ST3 = 1)
JOP	1C00	2	N	—	JUMP ODD PARITY (ST5 = 1)
LDCR	3000	4	Y	0-2,5	LOAD CRU
LI	0200	8	N	0-2	LOAD IMMEDIATE
LIMI	0300	8	N	12-15	LOAD IMMEDIATE TO INTERRUPT MASK
LREX	03E0	7	N	12-15	EXTERNAL CONTROL
LWPI	02E0	8	N	—	LOAD IMMEDIATE TO WORKSPACE POINTER
MOV	C000	1	Y	0-2	MOVE (WORD)
MOVB	D000	1	Y	0-2,5	MOVE (BYTE)
MPY	3800	9	N	—	MULTIPLY
NEG	0500	6	Y	0-4	NEGATE (TWO'S COMPLEMENT)
ORI	0260	8	Y	0-2	OR IMMEDIATE
RSET	0360	7	N	12-15	EXTERNAL CONTROL
RTWP	038O	7	N	0-6 12-15	RETURN WORKSPACE POINTER
S	6000	1	Y	0-4	SUBTRACT (WORD)
SB	7000	1	Y	0-5	SUBTRACT (BYTE)
SBO	1D00	2	N	—	SET CRU BIT TO ONE
SBZ	1E00	2	N	—	SET CRU BIT TO ZERO
SETO	0700	6	N	—	SET ONES
SLA	0A00	5	Y	0-4	SHIFT LEFT (ZERO FILL)
SOC	E000	1	Y	0-2	SET ONES CORRESPONDING (WORD)
SOCB	F000	1	Y	0-2,5	SET ONES CORRESPONDING (BYTE)
SRA	0800	5	Y	0-3	SHIFT RIGHT (MSB EXTENDED)
SRC	0800	5	Y	0-3	SHIFT RIGHT CIRCULAR
SRL	0900	5	Y	0-3	SHIFT RIGHT (LEADING ZERO FILL)
STCR	3400	4	Y	0-2,5	STORE FROM CRU
STST	02C0	8	N	—	STORE STATUS REGISTER
STWP	02A0	8	N	—	STORE WORKSPACE POINTER
SWPB	06C0	6	N	—	SWAP BYTES
SZC	4000	1	Y	0-2	SET ZEROES CORRESPONDING (WORD)
SZCB	5000	1	Y	0-2,5	SET ZEREOS CORRESPONDING (BYTE)
TB	1F00	2	N	2	TEST CRU BIT
X	0480	6	N	—	EXECUTE
XOP	2C00	9	N	6	EXTENDED OPERATION
XOR	2800	3	Y	0-2	EXCLUSIVE OR
DCA	2C00	9	N	0-3 5 7	DECIMAL CORRECT ADD
DCS	2C00	9	N	0-3 5 7	DECIMAL CORRECT SUB
LIIM	2C00	9	N	14 15	LOAD INTERRUPT MASK

ILLEGAL OP CODES 0000-01FF 0320-033F 0780-07FF,0C00-0FFF

INSTRUCTIONS BY OP CODE

OP CODE	MNEMONIC	OP CODE	MNEMONIC
0000-01FF	ILLEGAL	1000	JMP
0200	LI	1100	JLT
0220	AI	1200	JLE
0240	ANDI	1300	JEO
0260	ORI	1400	JHE
0280	LI	1500	JGT
0240	STWP	1600	JNE
02C0	STST	1700	JNC
02E0	LWPI	1800	JOC
0300	LIMI	1900	JND
0320-033F	ILLEGAL	1A00	JL
0340	IDLE	1B00	JH
0350	RSET	1C00	JOP
0350	RTWP	1D00	SBO
03A0	CKON	1E00	SBZ
03C0	CKOF	1F00	TB
03E0	LREX	2000	COC
0400	BWLP	2400	CZC
0440	B	2800	XOR
0480	X	2C00	XOP
04C0	CLR	3000	LDCR
0500	NEG	3400	STCR
0540	INV	3800	MPY
0580	INC	3C00	DIV
05C0	INCT	4000	SZC
0600	DEC	5000	SZCB
0640	DECT	6000	S
0680	BL	7000	SB
06C0	SWPB	8000	C
0700	SETO	9000	CB
0740	ABS	A000	A
0780-07FF	ILLEGAL	B000	AB
0800	SRA	C000	MOV
0900	SRL	D000	MOVB
0A00	SLA	E000	SOC
0B00	SRC	F000	SOCB
0C00	ILLEGAL		

PSEUDO-INSTRUCTIONS

MNEMONIC	PSEUDO-INSTRUCTIONS	CODE GENERATED
NOP	NO OPERATION	1000
RT	RETURN	0458

PIN DESCRIPTIONS

PIN #	FUNCTION	PIN #	FUNCTION	PIN #	FUNCTION
1	V_{SS}	23	A1	44	D3
2	V_{CC}	24	A0	45	D4
3	WAIT	25	$\phi 4$	46	D5
4	LOAD	26	V_{SS}	47	D6
5	HOLDA	27	V_{DD}	48	D7
6	RESET	28	$\phi 3$	49	D8
7	IAQ	29	DBIN	50	D9
8	$\phi 1$	30	CRUOUT	51	D10
9	$\phi 2$	31	CRUIN	52	D11
10	A14	32	INTREQ	53	D12
11	A13	33	IC3	54	D13
12	A12	34	IC2	55	D14
13	A11	35	IC1	56	D15
14	A10	36	IC0	57	NC
15	A9	37	NC	58	NC
16	A8	38	NC	59	NC
17	A7	39	NC	60	CRUCLK
18	A6	40	NC	61	WE
19	A5	41	D0	62	READY
20	A4	42	D1	63	MEMEN
21	A3	43	D2	64	HOLD
22	A2				

ASSEMBLER DIRECTIVES

MNEMONIC	DIRECTIVE
AORG	ABSOLUTE ORIGIN
BES	BLOCK ENDING WITH SYMBOL
BSS	BLOCK STARTING WITH SYMBOL
BYTE	INITIALIZE BYTE
DATA	INITIALIZE WORD
DEF	EXTERNAL DEFINITION
DORG	DUMMY ORIGIN
DXOP	DEFINE EXTENDED OPERATION
END	PROGRAM END
EQU	DEFINITE ASSEMBLY — TIME CONSTANT
EVEN	WORD BOUNDARY
IDT	PROGRAM IDENTIFIER
LIST	LIST SOURCE
PAGE	PAGE EJECT
REF	EXTERNAL REFERENCE
RORG	RELOCATABLE ORIGIN
TEXT	INITIALIZE TEXT
TITL	PAGE TITLE
UNL	NO SOURCE LIST

TIBUG COMMANDS

INPUT	RESULTS
B	Execute under Breakpoint
C	CRU Inspect/Change
D	Dump Memory to Cassette/Paper Tape
E	Execute
F	Find Word/Byte in Memory
H	Hex Arithmetic
L	Load Memory from Cassette/Paper Tape
M	Memory Inspect/Change
R	Inspect/Change User WP, PC, and ST Registers
S	Execute in Step Mode
T	1200 Baud Terminal
W	Inspect/Change Current User Workspace

COMMAND SYNTAX CONVENTIONS

CONVENTION SYMBOL	EXPLANATION
< >	Items to be supplied by the user. The term within the angle brackets is a generic term
[ ]	Optional Item — May be included or omitted at the user's discretion. Items not included in brackets are required
{ }	One of several optional items must be chosen
(CR)	Carriage Return
^	Space Bar
LF	Line Feed
R or Rn	Register (n = 0 to 15)
WP	Current User Workspace Pointer contents
PC	Current User Program Counter contents
ST	Current User Status Register contents

COMMAND	SYNTAX
Execute under Breakpoint (B)	B<address><(CR)>
CRU Inspect/Change (C)	C<base address>{^}<count><(CR)>
Dump Memory to Cassette/Paper Tape (D)	D<start address>{^}<stop address>{^}<entry address>{^}IDT = <name><^> (MONITOR PROMPT)
Execute Command (E)	E
Find Command (F)	F<start address>{^}<stop address>{^}<value>{(CR)}
Hexadecimal Arithmetic (H)	H<number 1>{^}<number 2><(CR)>
Load Memory from Cassette or Paper Tape (L)	L<bias><(CR)>
Memory Inspect/Change Memory Dump (M)	Memory Inspect/Change Syntax M<address><(CR)> Memory Dump Syntax M<start address>{^}<stop address><(CR)>
Inspect/Change User WP,PC, and ST Registers (R)	R<(CR)>
Execute In Single Step Mode (S)	S
TI 733 ASR Baud Rate (T)	T
Inspect/Change User Workspace (W)	W [Register Number] <(CR)>

USER ACCESSIBLE UTILITIES

XOP	FUNCTION
8	Write 1 Hexadecimal Charter to Terminal
9	Read Hexadecimal Word from Terminal
10	Write 4 Hexadecimal Characters to Terminal
11	Echo Character
12	Write 1 Character to Terminal
13	Read 1 Character from Terminal
14	Write Message to Terminal
	NOTE All characters are in ASCII code

TIBUG ERROR MESSAGES

ERROR	CONDITION
0	Invalid tag detected by the loader
1	Checksum error detected by the loader
2	Invalid termination character detected
3	Null input field detected by the dump routine
4	Invalid command entered

TEXAS INSTRUMENTS

TIBUG Monitor TM990/302 Reference Data

Microprocessor Series™

TM990/302
Software Development Board

EPROM's which may be programmed by the '302

2708
2716
2516
2532
9940

SOFTWARE COMPONENTS

	Access Command
Executive	(CR)
Text Editor	TE
Symbolic Assembler	SA
Debug Package	DP
EPROM Programmer	EP
Relocating Loader	RL
EIA Interface	EI
I/O Scheduler/Handler	SR

LUNO ASSIGNMENTS

Device	Logical Unit No
Dummy	0
Terminal (LOG)	1
Audio Cassette 1	2
Audio Cassette 2	3
Second EIA Connector	4
Memory	5

SOFTWARE COMPONENT CALLS

Text Editor	TE␣(input device),(output device)
Symbolic Assembler	SA␣(source device), (object device), (listing device)
Debug Package	DP␣(output device)
EPROM Programmer	EP
Relocating Loader	RL␣(input device)
Set Baud Rate	SR␣(nnnn)
Escape	ESC (return to executive)

TEXT EDITOR COMMANDS

D	Delete lines n thru m
I	Insert at line n with optional auto increment by m
K	Keep buffer and print new top line in the buffer
G	Get buffer and print new bottom line in the buffer
P	Print lines n thru m
Q	Flush the input file until end of input file and return to executive
R	Resequence input to output, n is initialized line # and m is the increment

COMMAND	SYNTAX
Delete Lines n thru m (Rn,m)	D (starting line #)[,(ending line #)]
Insert After Line n with optional auto increment by m (In,m)	I (line number after which new data is entered) [,(auto increment value)]
Get Buffer (G)	G
Keep Buffer (K)	K
Print lines n thru m (Pn,m)	P (first line # to be printed) [,(last line # to be printed)]
Quit Text Editor (Q)	Q
Resequence Output (Rn,m)	R (initial line number) [,(increment value)]

ASSEMBLER DIRECTIVES

AORG	[label]␣AORG␣(value)␣[comment]
BSS	[label]␣BSS␣(value)␣[comment]
BYTE	[label]␣BYTE␣(value),(value),(value), ␣[comment]
DXOP	[label]␣DXOP␣(symbol),(value)␣[comment]
END	[label]␣END␣(symbol)␣[comment]
EQU	[label]␣EQU␣(expression)␣[comment]
DATA	[label]␣DATA␣(exp),(exp), ␣[comment]
EVEN	[label]␣EVEN␣[comment]
IDT	[label]␣IDT␣(string)␣[comment]
TEXT	[label]␣TEXT␣(–),'string'␣[comment]

DEBUG Package

Verb	Command
SB	Set Software Breakpoint and Execute
IM	Inspect/Change Memory
IC	Inspect/Change CRU
IR	Inspect/Change MPU Registers
ST	Set Software Trace
RU	Single Step for 1 or more instructions with or without trace
DM	Dump Memory

DEBUG COMMANDS

Set Breakpoint and Execute	SB␣(address)
Inspect/Change Memory	IM␣(address)
Inspect/Change CRU	IC␣(CRU base addr)(no. of bits)
Inspect/Change MPU registers	IR
Set Software Trace	ST␣(0 or 1)
Run 1 or more Instructions	RU␣(no of instructions in decimal)
Dump Memory	DM␣(starting addr) (ending addr)

EPROM PROGRAMMING CRU ASSIGNMENTS

CRU BASE ADDRESS$_{16}$	INPUT/OUTPUT	FUNCTION
1710	I/O	EPROM DATA BIT 0
1712	I/O	
1714	I/O	
1716	I/O	
1718	I/O	
171A	I/O	
171C	I/O	
171E	I/O	EPROM DATA BIT 7
1720	O	EPROM ADDRESS LSB
1722	O	
1724	O	
1726	O	
1728	O	
172A	O	
172C	O	
172E	O	
1730	O	
1732	O	
1734	O	
1736	O	
1738	O	EPROM ADDRESS MSB
173A	O	EPROM PROGRAM ENABLE
173E	O	EPROM PROGRAMMING PULSE

EPROM PROGRAMMING RESPONSES

PP = Program EPROM
RE = Read EPROM to Memory
CE = Compare EPROM to Memory
Memory Bounds MEM BDS? (start addr),(stop addr)
EPROM Start addr EPROM START? (start addr)
Programming Mode MODE? P(parallel) or I(in line)
Starting Byte. ST byte ? (0 or 1 if P above)

PREDEFINED CRU ADDRESSES FOR I/D DEVICES

Device	CRU Address
Users Terminal (9902)	80_{16}
Timer (9901)	100_{16}
EIA Interface (9902)	180_{16}
Recorder 1 Forward	1700_{16}
Recorder 2 Forward/9940 Flag 1	1702_{16}
Recorder 2 Write Data/9940 Flag 2	1704_{16}
Recorder 1 Read Data/9940 Flag 3	1706_{16}
Personality Card Code Bit 0	1708_{16}
Personality Card Code Bit 1	$170A_{16}$
Personality Card Code Bit 2	$170C_{16}$
Switch Code Bit	$170E_{16}$
EPROM Data	1710_{16} – $171E_{16}$
EPROM Address	1720_{16} – 1738_{16}
EPROM Program Enable	$173A_{16}$
EPROM Programming Pulse	$173C_{16}$

INPUT OPTIONS

string-var	Prompt with colon and input character data. Example: INPUT $A
,	Delimit expressions. Example: A, B
;	Suppress prompting or CR LF if at end of line. Examples: INPUT ,A INPUT A;
# exp	Allow a maximum of exp characters to be entered. Example: INPUT # 1 "Y or N"$1
%exp	*Must enter exactly exp number of characters. Example: INPUT %4 "CODE "C
?<ln>	*Upon an invalid input or entry of a control character, a GOSUB is performed to the line #. SYS(0) will be equal to −1 if there was an invalid input. Otherwise, SYS(0) will equal the decimal equivalent of the control character. Example: INPUT ?100;A

OUTPUT OPTIONS

,	Delimit expressions or suppress CR LF if at end of line. Examples: PRINT A,B PRINT A;
,	Tab to next print field. Example: PRINT A, B
TAB <(exp)>	Tab to exp column. Example: PRINT TAB (50);A
string	Print string or string-var. Example: PRINT "HI";$A(0)
# exp	*Print exp as hexadecimal in free format. Example: PRINT # 123
#,exp	*Print exp as hexadecimal in byte format. Example: PRINT #,50
#,exp	*Print exp as hexadecimal in word format. Example: PRINT #,A
<hex value>	*Direct output of ASCII codes. Example: PRINT "<0D><0A>"
# string	*Print under specified format where: PRINT # "9999"I 9 = digit holder PRINT # "000-00-0000"SS 0 = digit holder or force 0 PRINT # "$$$,$$$.00"DLR $ = digit holder and floats $ PRINT # "SSS 0000"4*ATN1 S = digit holder and floats sign PRINT # "<<<<.00>"I < = digit holder and float on negative >number PRINT # "990.99E"N E = sign holder after decimal PRINT # "990.99"N . = decimal point specifier PRINT # "999,990.99"N , = suppressed if before significant digit PRINT # "999,990^.00"I ^ = translates to decimal point PRINT # "HI = 99"I any other character is printed

ERROR CODES

1 =	SYNTAX ERROR	37 =	ILLEGAL DELIMITER
2 =	UNMATCHED PARENTHESIS	38 =	UNDEFINED FUNCTION
3 =	INVALID LINE NUMBER	39 =	UNDIMENSIONED VARIABLE
4 =	ILLEGAL VARIABLE NAME	40 =	UNDERFINED VARIABLE
5 =	TOO MANY VARIABLES	41 =	EXPANSION EPROM NOT INSTALLED
6 =	ILLEGAL CHARACTER	42 =	INTERRUPT W/O TRAP
7 =	EXPECTING OPERATOR	43 =	INVALID BAUD RATE
8 =	ILLEGAL FUNCTION NAME	44 =	TAPE READ ERROR
9 =	ILLEGAL FUNCTION ARGUMENT	45 =	EPROM VERIFY ERROR
10 =	STORAGE OVERFLOW	46 =	INVALID DEVICE NUMBER
11 =	STACK OVERFLOW		
12 =	STACK UNDERFLOW		
13 =	NO SUCH LINE NUMBER		
14 =	EXPECTING STRING VARIABLE		
15 =	INVALID SCREEN COMMAND		
16 =	EXPECTING DIMENSIONED VARIABLE		
17 =	SUBSCRIPT OUT OF RANGE		
18 =	TWO FEW SUBSCRIPTS		
19 =	TOO MANY SUBSCRIPTS		
20 =	EXPECTING SIMPLE VARIABLE		
21 =	DIGITS OUT OF RANGE (0< # of digits <12)		
22 =	EXPECTING VARIABLE		
23 =	READ OUT OF DATA		
24 =	READ TYPE DIFFERS FROM DATA TYPE		
25 =	SQUARE ROOT OF NEGATIVE NUMBER		
26 =	LOG OF NON-POSITIVE NUMBER		
27 =	EXPRESSION TOO COMPLEX		
28 =	DIVISION BY ZERO		
29 =	FLOATING POINT OVERFLOW		
30 =	FIX ERROR		
31 =	FOR WITHOUT NEXT		
32 =	NEXT WITHOUT FOR		
33 =	EXP FUNCTION HAS INVALID ARGUMENT		
34 =	UNNORMALIZED NUMBER		
35 =	PARAMETER ERROR		
36 =	MISSING ASSIGNMENT OPERATOR		

TEXAS INSTRUMENTS

POWER BASIC Reference Data

Microprocessor Series™

REFERENCE CARD FOR DEVELOPMENT AND EVALUATION BASIC

This card contains a summary of all POWER BASIC† statements and commands for Development and Evaluation BASIC. An explanation preceded by an asterisk (*) indicates the statement or command is not supported by Evaluation BASIC. A * indicates the statement is supported only by the Development BASIC software enhancement package.

†Trademark of Texas Instruments

COMMANDS

CONtinue

*Execution continues from last break.

LIST

LIST the user's POWER BASIC program. In LIST will list from specified line number through end of program or until ESC key is struck.

LOAD

Reads a previously recorded POWER BASIC program from an auxiliary device or configures POWER BASIC to execute a BASIC program in EPROM.
LOAD reads program from 733ASR digital cassette.
LOAD 1 or LOAD 2 * reads program from audio cassette drive No. 1 or No. 2.
LOAD <address>* configures POWER BASIC to execute BASIC program in EPROM at specified address.

NEW

Prepare for entry of NEW POWER BASIC program or set the lower RAM memory bound after auto-sizing.
NEW clears pointers of POWER BASIC and prepares for entry of new program.
NEW <address>* sets the lower RAM memory bound used by POWER BASIC after auto-sizing or power-up.

PROGRAM

Program current POWER BASIC application program into EPROM *

RUN

Begin program execution at the lowest line number.

SAVEn (n is interpreted as in LOADn command)

Record current user program on auxiliary device.

SIZE

Display current program size, variable space allocated, and available memory in bytes.

GENERAL INFORMATION

ARITHMETIC OPERATIONS

A = B	Assignment
A − B	Negation or subtraction
A + B, $A + $B	Addition or string concatenation
A*B	Multiplication
A/B	Division
A^B	Exponentiation
−A	Unary Minus
+A	Unary Plus

LOGICAL OPERATORS

LNOT A	*1's complement of integer.
A LAND B	*Bit wise AND.
A LOR B	*Bit wise OR.
A LXOR B	*Bit wise exclusive OR.

RELATIONAL OPERATORS

1 if TRUE and 0 if FALSE

A = B	TRUE if equal, else FALSE.
A = = B	*TRUE if approximately equal (1E-7), else FALSE
A<B	TRUE if less than, else FALSE
A<=B	TRUE if less than or equal, else FALSE
A>B	TRUE if greater than, else FALSE.
A>=B	TRUE if greater than or equal, else FALSE.
A<>B	TRUE if not equal, else FALSE.
NOT A	*TRUE if zero, else FALSE
A AND B	*TRUE if both non-zero, else FALSE
A OR B	*TRUE if either non-zero, else FALSE.

OPERATOR PRECEDENCE

1	Expressions in parentheses	7	=,>
2	Exponentiation and negation	8	= =, LXOR
3	*,/	9	NOT, LNOT
4	+,−	10	AND, LAND
5	<=,<>	11	OR, LOR
6	>=,<	12	(=)ASSIGNMENT

SPECIAL CHARACTERS

:	Separates statements typed on same line.
!	Tail remark used for comments after program statement.
'	Equivalent to PRINT.

EDITING

The phrase "(ctrl)" indicates that the user holds down the control key while depressing the key corresponding to the character immediately following.

(CR)	Enter edited line.
(ctrl)In	*Insert n blanks
(ctrl)Dn	*Delete n characters
(ctrl)H	Backspace one character.
(ctrl)F	Forward apace one cherac ter.
In(ctrl)E	Display for editing source line indicated by line number (In).
(ctrl)T	Toggle from one partition to the other partition (only in Evaluation BASIC).
(esc)	Cancel input line or break program execution
(Rubout) or (DEL)	Backspace and delete character

STATEMENTS

InBAUD <exp 1,> <exp 2>
*sets baud rate of serial I/O port(s).

InBASE <(exp)>
Sets CRU bese address for subsequent CRU operations

InCALL"Name"<subroutine address>[, <var 1>, <var 2>, <var 3>, <var 4>]
*Transfers to external subroutines. If variable is contained in parentheses, the address will be passed; otherwise, the value will be passed

InDATA {<exp> / <string const>} [, {<exp> / <string const>}] . . .
defines internal data block

In DEF FN<x>[(<arg 1> [, arg 2] [, erg 3])] = <exp>
*Defines user arithmetic function.

InDIM <var (dim[, dim] . . .)> [,]
Allocates user variable space for dimensioned or array variables.

InEND
Terminates program execution and returns to edit mode.

In ERROR<In>
*Specifies a subroutine that will be called via a GOSUB statement when an error occurs.

In ESCAPE
InNOESC
*Enables or disables the excape key to interrupt program execution (always enabled in Evaluation BASIC).

InFOR <sim-var> = <exp> TO <exp> [STEP <exp>]
InNEXT <sim-var>
Open and close program loop. Both identify the same control variable. FOR assigns starting, ending, and optionally stepping values.

InGOSUB<In>
Transfer of control to en internal subroutine beginning at the specified line.

InPOP
*Removal of most previous return address from GOSUB stack without an execution transfer.

InRETURN
Return from internal subroutine

InGOTO<In>
Transfers program execution to specified line number.

InIF<exp>THEN<statement>
InELSE<statement>
Causes conditional execution of the statement following THEN. *ELSE statements execute when IF condition is false.

InIMASK<LEVEL>
*Set interrupt mask of TMS 9900 processor to spacified level.

InTRAP<level>TO<In>
*Assign interrupt level to interrupt subroutine.

InIRTN
*Return from BASIC interrupt service routine.

InINPUT<var> [{<var>} . . . {<var>}]
Accesses numeric constants end strings from the keyboard into variables in the INPUT list.

In [LET] <var> = <exp>
Evaluates and assigns values to variables or array elements

InON {<var> / <exp>} THEN GOTO In [,In] . . .

InON {<var> / <exp>} THEN GOSUB In [,In]
*Transfers execution to the line number specified by the expression or variable.

InPRINT <exp> [,exp]
Print (format free) the evaluated expressions.

InRANDOM [exp]
*Set the seed to the specified expression value

InREAD {<numeric var> / <string var>} [, {<numeric var> / <string var>}] . .
Assigns values from the internal data list to variables or array elements.

InREM [text]
Inserts comments

InRESTOR [exp]
Without an argument, resets pointer to beginning of data sequence; with an argument, resets pointer to line number specified

InSTOP
Terminates program execution and returns to Edit mode

InTIME
Sets, displays, or stores the 24 hour time of day clock.
InTIME <exp>, <exp>, <exp>
Sets and starts clock
InTIME <string-var>
Enables storing clock time into a string variable.
InTIME
Prints clock time as HR MN SD

InUNIT <exp>
*Designates device(s) to receive all printed output.

FUNCTIONS

ABS <(exp)>	*Absolute value of expression.
ASC <(string var)>	*Returns decimal ASCII code for first character of string variable.
ATN <(exp)>	Arctangent of expression in radians.
BIT <(var, exp)>	*Reads or modifies any bit within a variable.
BIT <(var, exp 1)> = <exp 2>	Returns a 1 if bit is set and 0 it not set. Selected bit is set to 1 if assigned velue is non-zero and to zero if the assigned value is zero.
COS >(exp)>	Cosine of the expression in radians.
CRB <(exp)>	Reeds CRU bit as selected by CRU base + exp. Exp is valid for − 127 thru 128.
CRB <(exp 1)> = <(exp 2)>	Sets or resets CRU bit as selected by CRU base + exp 1. If exp 2 is non-zero, the bit will be set, else reset Exp 1 is valid for − 127 thru 128.
CRF <(exp)>	Reads n CRU bits as selected by CRU base where exp evaluates to n. Exp is valid for 0 thru 15. If exp = 0, 16 bits will be read.
CRF <(exp 1)> = <(exp 2)>	Sfores exp 1 bits of exp 2 to CRU lines as selected by CRU BASE. Exp 1 it valid for 0 thru 15. If exp 1 = 0, 16 bits will be stored
EXP <(exp)>	*Raise the constant e to the power of the evaluated expression.
INP <(exp)>	Returns the signed integer portion of the expression.
LOG <(exp)>.	*Returns natural logarithm of the expression
MEM <(exp)>	Reads byte from user memory et address specified by exp. Exp must be in the integer range, (0 to 65535).
MEM <(exp 1)> = <(exp 2)>	Stores byte exp 2 into user memory specified by exp 1. Exp 1 and exp 2 must be in the integer range
MCH <(string 1), (string 2)>	*Returns the number of characters to which the two strings agree.
NYK <(exp)>	Conditionally samples the keyboard in run time mode. If exp <>0, return decimal value of last key struck and clear key register. (0 if no key struck.) If exp = 0, return a 1 if the last key struck has the same decimal velue as the expression. Clear key register if TRUE, else return 0 if FALSE
RND	Returns a random number between 0 and 1
SIN <(exp)>	Sine of the expression in radians.
SOR <(exp)>	Square root of expression.
SRH <(string 1), (string 2)>	*Return the position of string 1 in string 2, 0 if not found
SYS <(exp)>	*Obtains system parameters generated during program execution. Example: SYS(0) = INPUT control character, SYS(1) = Error code number, SYS(2) = error line number.
TIC <(exp)>	Returns the number of time tics less the expression value. One TIC equals 40 milliseconds (1/25 second).

STRINGS

ASCII Character Conversion Function	ASC (string-var) *Convert first character of string to ASCII numeric representation.
Assignment	<string-var> = {<string-var> / <string-constant>} Store string into string-var ending with a null
Character Match Function	MCH (<string 1>, <string 2>) *Return the number of chercters to which the 2 strings agree
Character Search Function	SRH (<string 1>, <string 2>) *Return the position of string 1 in string 2. Zero is returned it not found.
Concatenate	<string-var> = {<string-var> / <string-constant>} + {<string-var> / <string-constant>} [+ {. . .}]
Convert to ASCII	<string-var> = <exp> <string-var> = # <string>, <exp> *Convert exp to ASCII characters ending with a null. # string specifies a formatted conversion
Convert to Binary	<var 1> = <string>, <var 2> *Convert string into binary equivalent. Var 2 receives the delimiting non-numeric character in first byte.
Deletion	<String-var> = / <exp> *Delete exp characters from string-var.
Insertion	<string-var> = / <string> *Pick byte into string-var.
Pick	<string-var> = {<string-var> / <string-constant>}, <exp> Pick number of characters specified by exp into string-var ending with a null.
Replace	<string-var> = {<string-var> / <string-constant>}, <exp> Replace number of characters specified by exp of string-var with string
String Length Function	<var> = LEN <(string-var)> <ver> = LEN "string" *Return the length of string

TEXAS INSTRUMENTS

TXDS Reference Data

Microprocessor Series™

List of Commands and Special Keys/Characters

COMMAND SYNTAX	DESCRIPTION
	SETUP COMMANDS
SL	Start Line Numbers (SL) command causes line numbers to be printed with each line of text
SN	Stop Line Numbers (SN) comman causes line numbers not to be printed
SP	Set Print Margin (SP) command sets the right boundary for print display
SM	Set Margin (SM) for Find command sets the left and right boundaries for the Find command
ST	Set Tabs (ST) command sets up to five tab stops
	PRINTER-MOVEMENT COMMANDS
D	Down (D) command moves the pointer down toward the bottom of the buffer
U	Up (U) command moves the pointer up towards the first line in the buffer
T	Top (T) command moves the pointer to the first line in the buffer
B	Bottom (B) command moves the pointer to the last line in the buffer
	EDIT COMMANDS
C	Change (C) command removes lines from the buffer and inserts new ones in their place. The new lines are input from the terminal.
I	Insert (I) command takes input from the terminal and places the new lines into the buffer
M	Move (M) command moves lines from one place in the buffer to another
R	Remove (R) command deletes lines from the buffer
F	Find string (F) command searches for the first occurrence of a character string in a line and replaces it with another string of characters.
	PRINT COMMANDS
L	Limits (L) command causes the first line and the last line to be displayed
P	Print (P) command displays lines of text
	OUTPUT COMMANDS
K	Keep (K) command takes lines of text out of the buffer and puts them in the output file
O	Quit (Q) command takes lines of text out of the buffer or the input files and puts them in the output file
E	An (E) command terminates without writing an EOF to the output file.
	TERMINATE-SEQUENCE COMMANDS
T or C	Allows the user to make multiple single directional editing passes on a source or object program
	SPECIAL KEYS/CHARACTERS
CTRL-H	Pressing the control key and the H key simultaneously on the hard copy terminal causes the terminal to backspace a character to enable rewriting over an entered character-error.
RUB OUT	The RUB OUT key causes the line just entered to be deleted so that a new line can replace it
CTRL-I	Pressing the control (CTRL) key and the I key simultaneously on a hard-copy terminal causes a tab stop to be entered in the input string, although only one space will be echoed on the terminal.
ESC/RESET	Pressing the ESCape or RESET key on the system console causes a display to be aborted.
position keys	When using a VDT, only the left position key (←) and the right (→) position key are recognized. The up and down position keys cause garbage to be entered into the input string. The left position key causes characters to be deleted from the character string, a right position key causes whatever was under the cursor to be entered
DELETE LINE	DELETE LINE on a VDT acts the same as a RUB OUT on a hardcopy terminal
TAB	A SPACE character is echoed. The TAB is interpreted by the text editor and spaces are inserted to fill the text line to the next TAB setting

TXMIRA Options

OPTION	DESCRIPTION
Mnnnnn	Overrides memory size default, default is 2400 bytes
X	Produce cross-reference
L	Produce assembly listing
T	Expand TEXT code on listing
S	Produce sorted symbol list
C	Produce compressed object output where n is a decimal digit

TXLINK Options

OPTION	DESCRIPTION
Mnnnnn	Override default memory size, default is 11800 bytes
C	Compressed object output
Iaaaaaaaa	IDT for linked object
P	Partial link desired
L	Print load map and symbol list.

Note: n is a decimal digit and a is an alphanumeric character

TXCCAT Options

OPTION	DESCRIPTION
TRnnnn	Truncate record to length nnnn
FLnnnn	Fix records to size nnnn by padding with blanks or by truncation.
SKnnnn	Skip nnnn input records, prior to output
LFnn	List file page length = nn, default = 55
SLnn	Space lines on listing, nn = space count, default = 0
NL	Number lines on listing
RI	Do not rewind input on open
RO	Do not rewind output on open

Note: n is a decimal digit and the maximum field size is given by the number of n's

TXDBUG Keyboard Commands

DEBUG Commands

IC	Inspect Communications Register Unit (CRU)
IM	Inspect Memory
IR	Inspect AU Register (WP, PC, ST)
IS	Inspect Snapshot
IW	Inspect Workspace Registers
MC	Modify Communications Register Unit (CRU)
MM	Modify Memory
MR	Modify Registers
MW	Modify Workspace Registers
SB	Set Breakpoint
SP	Set H/W Write Protect Option
SR	Set Trace Region
SS	Set Snapshot
ST	Set Trace
CB	Clear Breakpoint
CP	Clear H/W Write Protect Option
CR	Clear Trace Region
CS	Clear Snapshot

USER'S RESPONSE SHEET

Manual Title: 9900 Family Systems Design and Data Book LCC4400

Manual Date: November 1978 **Date of This Letter:** _____

User's Name: _____ **Telephone:** _____

Company: _____ **Office/Department:** _____

Street Address: _____

City/State/Zip Code: _____

Please list any discrepancy found in this manual by page, paragraph, figure, or table number in the following space. If there are any other suggestions that you wish to make, feel free to include them. Thank you.

Location in Manual	Comment/Suggestion

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
FOLD ON TWO LINES (LOCATED ON REVERSE SIDE), TAPE AND MAIL

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7284 DALLAS, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

TEXAS INSTRUMENTS INCORPORATED
P.O. BOX 1443 MS 6404
HOUSTON, TEXAS 77001

FOLD

9900 Family Systems Design

and Data Book

MICROPROCESSOR SERIES™

TEXAS INSTRUMENTS
INCORPORATED

LCC4400
97049-118-NI

Printed in U.S.A.
1064 Pages

1st Edition