The Engineering Staff of
TEXAS INSTRUMENTS INCORPORATED
Semiconductor Group

J. D. DiSanto

TM 990/101M
MICROCOMPUTER
USER'S
GUIDE

MARCH 1980

TEXAS INSTRUMENTS
INCORPORATED

```
┌─MANUAL HISTORY─────────────────────────────────────────────┐
│                                                             │
│  This manual contains the following revisions:             │
│                                                             │
│                      Revision Change                        │
│      Date            (From - to)          ECN Number        │
│    02/18/80            C  to  D            454310           │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

IMPORTANT NOTICES

Texas Instruments reserves the right to make changes at any time
in order to improve design and to supply the best product possible.

TI cannot assume any responsibility for any circuits shown or
represent that they are free from patent infringement.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

APPENDICES

INDEX

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

INTRODUCTION

1.1 GENERAL

The Texas Instruments TM 990/101M is a self-contained microcomputer on a single printed-circuit board. The board's component side is shown in Figure 1-1, which also highlights major features and components. Figure 1-2 shows board dimensions. This microcomputer board contains features found on computer systems of much larger size, including a central processing unit (CPU) with hardware multiply and divide, programmable serial and parallel I/O lines, external interrupts, and a debug-monitor to assist the programmer in program development and execution. Other features include:

- TMS 9900 microprocessor based system: the microprocessor with the minicomputer instruction set - software compatible with other members of the 990 family.

- 1K x 16 bits of TMS 4045 random-access memory (RAM) expandable on-board to 2K x 16 bits.

- 1K x 16 bits of TMS 2708 erasable programmable read-only memory (EPROM), expandable on-board to 2K x 16 bits. Simple jumper modifications enable substitution of the larger TMS 2716 EPROM's (16K bits each) for the smaller TMS 2708's (8K bits each). Four TMS 2716's permit EPROM expansion to 4K x 16 bits.

NOTE

Three board configurations are available. The characteristics of each configuration are explained in paragraph 1.3.

- Buffered address, data, and control lines for off-board memory and I/O expansion; full DMA capabilities are provided by the buffer controllers.

- 3 MHz crystal-controlled clock.

- One 16-bit parallel I/O port, each bit is individually programmable.

- Modified EIA RS-232-C serial I/O interface, capable of communication to both EIA-compatible terminals and popular modems (data sets).

- A local serial I/O port, with interfaces for an EIA terminal and either a Teletype (TTY) or a twisted-pair balanced-line multidrop system (interface choices are detailed in paragraph 1.3).

- Three programmable interval timers.

- 17 prioritized interrupts, including RESET and LOAD functions. Interrupt 6 is level triggered (active LOW) and edge-triggered (either polarity) and latched on-board.

- A directly addressable five-position DIP switch and an addressable light emitting diode (LED) for custom system applications.

- PROM memory decoder permits easy reassignment of memory map configuration; see Figure 1-3 for memory map of the standard board.
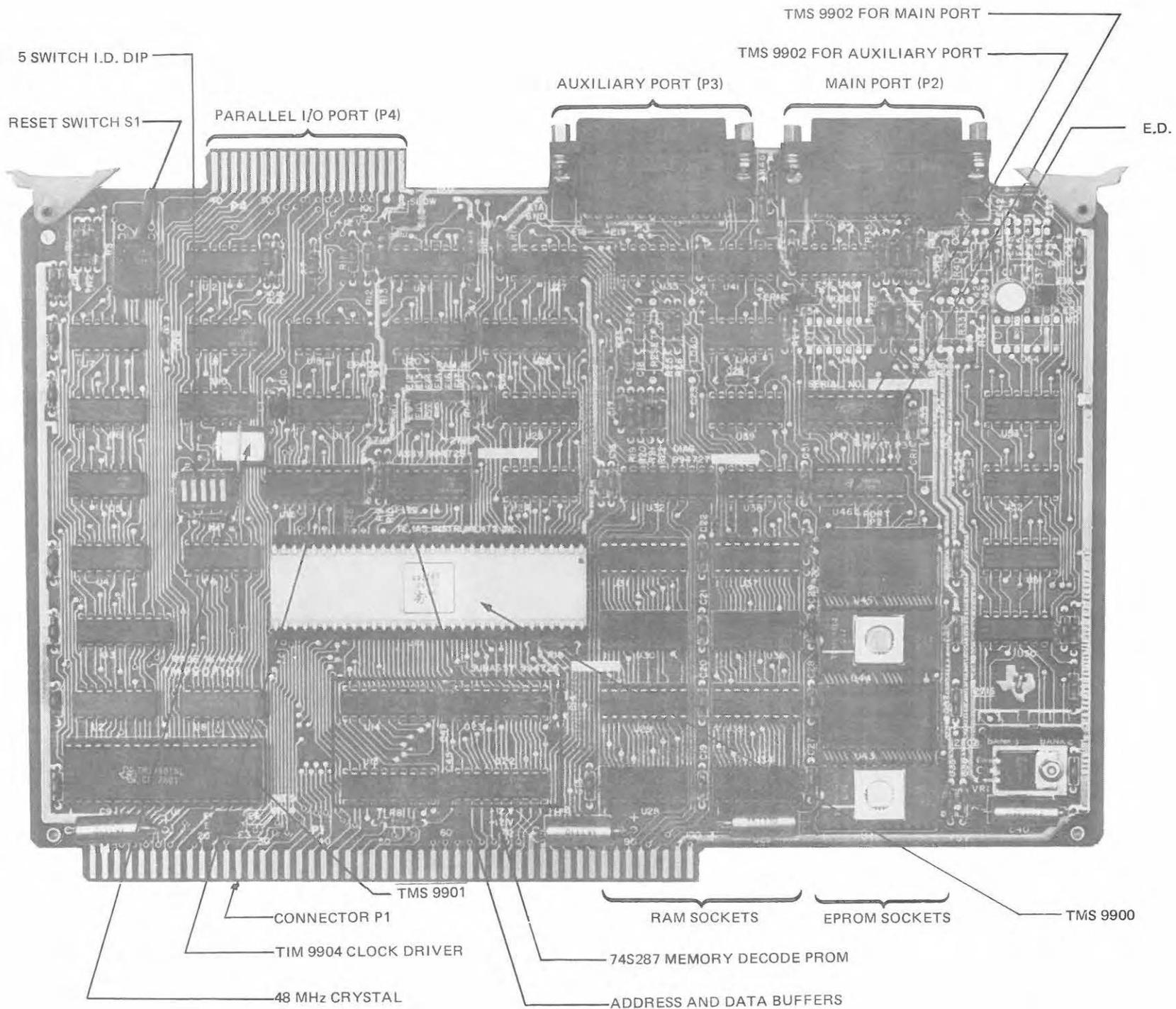
Figure 1-1. TM 990/101M Major Components

5 SWITCH I.D. DIP

RESET SWITCH S1

PARALLEL I/O PORT (P4)

AUXILIARY PORT (P3)

MAIN PORT (P2)

TMS 9902 FOR MAIN PORT

TMS 9902 FOR AUXILIARY PORT

E.D.

TMS 9901

CONNECTOR P1

TIM 9904 CLOCK DRIVER

48 MHz CRYSTAL

RAM SOCKETS

EPROM SOCKETS

TMS 9900

74S287 MEMORY DECODE PROM

ADDRESS AND DATA BUFFERS

1-2
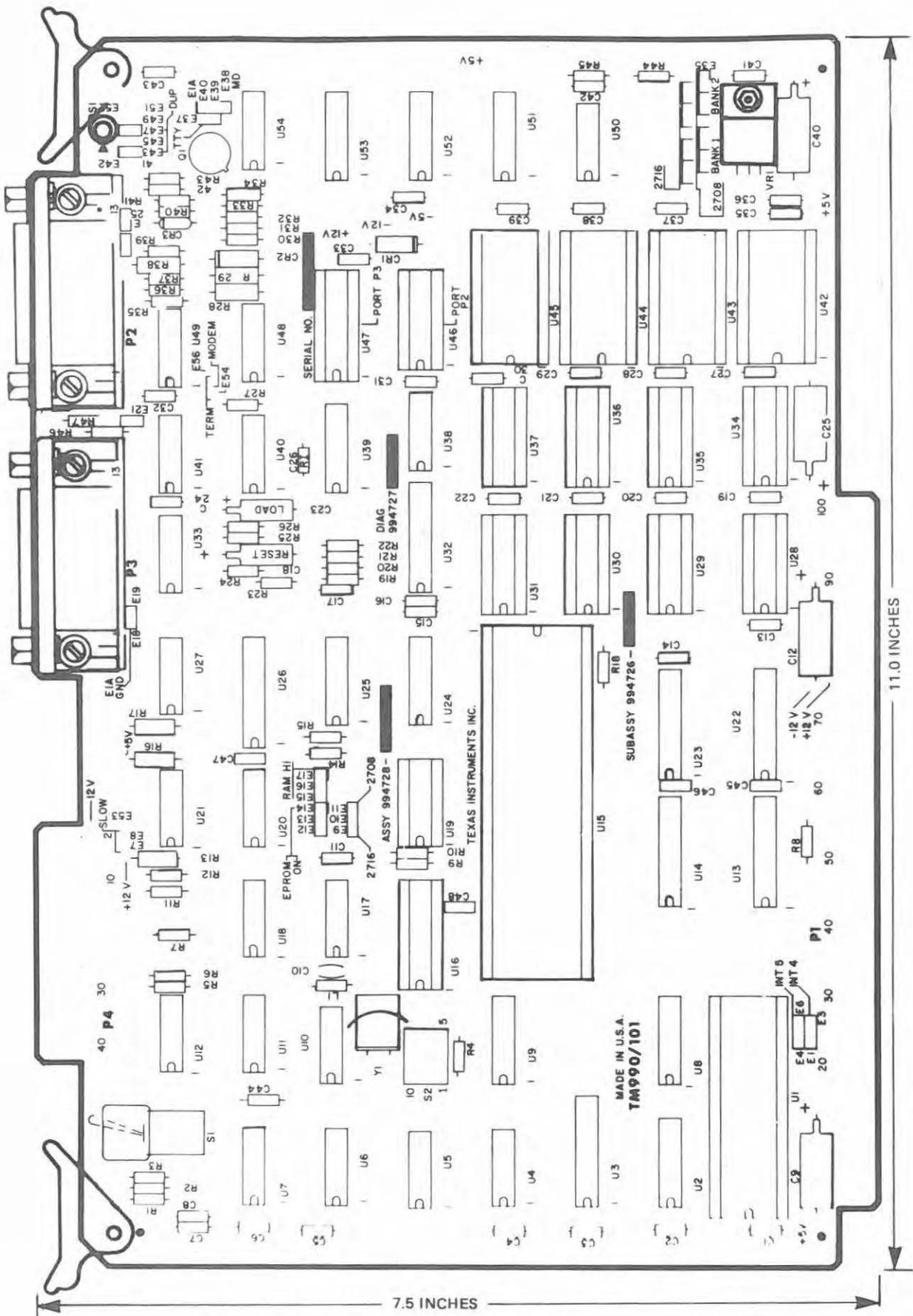
Figure 1-2. TM 990/101M Dimensions and Component Placement

1.2 MANUAL ORGANIZATION

Section 1 covers board specifications and characteristics. A glossary in paragraph 1.7 explains terms used throughout the manual.

Section 2 explains how to install, power-up, and operate the TM 990/101 microcomputer with the addition of a data terminal, power supplies, and appropriate connectors.

Section 3 explains how to communicate with the TM 990/101M using the TIBUG monitor. This versatile monitor, complete with supervisor calls and operator communication commands, facilitates the development and execution of software.

Section 4 describes the instruction set of the TM 990/101M, giving examples of each class of instructions and providing some explanation of the TMS 9900 architecture.

Section 5 explains basic programming procedures for the microcomputer, giving an explanation of the programming environment and hardware-dependent features. Numerous program examples are included for utilizing the various facilities of the TM 990/101M.

Section 6 is a basic theory of operation, explaining the hardware design configuration and circuitry. This section provides explanations of the bus structure, the control logic, and the various subsystems which make up the microcomputer.

Section 7 describes various options available for the microcomputer, both those supplied on-board and those which Texas Instruments offers for off-board expansion of the system.

Section 8 features various hardware applications which can be built using the TM 990/101M.

1.3 PRODUCT INDEX

The TM 990/101M microcomputer is available in three different configurations, which are specified by a "dash number" appended to the product name; e.g., TM 990/101M-1. These configurations are listed in Table 1-1. A memory map is shown in Figure 1-3.

Table 1-1. TM 990/101M Configurations

TM 990/101M Dash No.	EPROM		RAM	Main Serial Port Option (EIA Terminal I/F Stand)
	Socketed	Program		
-1	2 TMS 2708 (1K x 16)	TIBUG Monitor	4 TMS 4045 (1K x 16)	TTY
-2	2 TMS 2716 (2K x 16)	Blank	4 TMS 4045 (1K x 16)	Multidrop
-3	4 TMS 2716 (4K x 16)	Blank	8 TMS 4045 (2K x 16)	TTY

```
MAIN EPROM*

                    0000
                    0800                              07FF
                                                      0FFF
        EXPANSION
          EPROM


        EXPANSION
          RAM

                    F000
                    F800                              F7FF
                                                      FFFF
                              TM 990/101M
        MAIN RAM
```
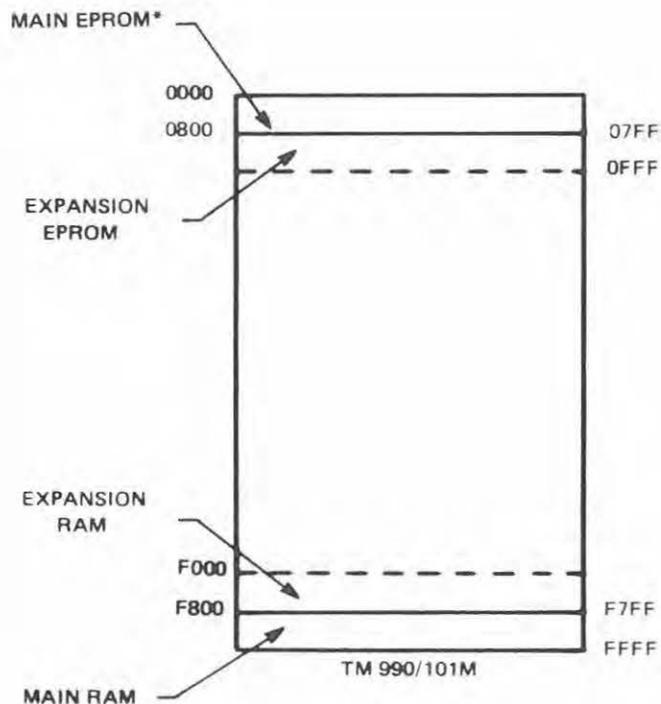
*EPROM's programmed with TIBUG monitor.

Figure 1-3. Main And Expansion EPROM and RAM

1.4 BOARD CHARACTERISTICS

Figure 1-1 shows the major portions and components of the microcomputer. The system bus connector is P1, which is a 100-pin (50 each side) PC board edge connector spaced on 0.125 inch centers. Connector P2 is the main serial port and P3 is the RS-232-C auxiliary serial port. Both connectors are standard 25-position female jacks used in RS-232-C communications. The parallel I/O port is PC board edge connector P4, which has 40 pins (20 each side) spaced on 0.1-inch centers.

Figure 1-2 shows the PC board silkscreen markings which detail the various components on the board; also included are the board dimensions and tolerances.

1.5 GENERAL SPECIFICATIONS

	+5 V		+12 V		-12 V	
Power Consumption	TYP	MAX	TYP	MAX	TYP	MAX
TM 990/101M-1	1.8	2.6	0.30	0.50	0.25	0.40
TM 990/101M-2	1.8	2.6	0.30	0.50	0.25	0.40

Clock Rate: 3 MHz

Baud Rates (set by TIBUG): 110, 300, 600, 1200, 2400, 4800, 9600, 19200

1-5

Memory Size: The microcomputer is shipped with:
       RAM: Four TMS 4045 (1K x 4 bits each)
       EPROM: Two TMS 2708 (1K x 8 bits each), preprogrammed with TIBUG.

Total capacity is:
       RAM: Eight TMS 4045's (1K x 4 bits each)
       EPROM: Four TMS 2708's (1K x 8 bits each)
                                or
                   Four TMS 2716's (2K x 8 bits each)
Board Dimensions: See Figure 1-2

Parallel I/O Port (P4): One 16-bit port, uses TMS 9901 programmable systems interface

Serial I/O Port (P2 and P3): Two asynchronous ports:
       Main port (P2) has two interfaces: RS-232-C answer mode and either a TTY or a
       balanced-line differential multidrop interface.

       Auxiliary port (P3) meets RS-232-C specification interface, capable
       of either originate or answer mode.

       Both serial ports use TMS 9902 asynchronous communication controllers, but the
       Auxiliary Port will readily accept the TMS 9903 synchronous communication
       controller. Simply plug in the TMS 9903 for synchronous systems.

1.6 REFERENCE DOCUMENTS
The following documents provide supplementary information for the TM 990/101M user's manual.

   ● TMS 9900 Microprocessor Data Manual
   ● TMS 9901 Programmable Systems Interface Data Manual
   ● TMS 9902 Asynchronous Communication Controller Data Manual
   ● TMS 9903 Synchronous Communication Controller Data Manual
   ● TMS 990 Computer, TMS 9900 Microprocessor Assembly Language Programmer's
     Guide (P/N 943441-9701)
   ● TM 990/301 Microterminal
   ● TM 990/401 TIBUG Monitor Listing
   ● TM 990/402 Line-by-Line Assembler User's Guide
   ● TM 990/402L Line-by-Line Assembler Listing
   ● TM 990/502 Cable Assembly (RS-232-C)
   ● TM 990/503 Cable Assembly (TI Terminal 743 or 745)
   ● TM 990/504 Cable Assembly (Teletype)
   ● TM 990/506 Cable Assembly (Modem cable for /101 board)
   ● TM 990/510 Card Chassis
   ● TM 990/511 Extender Board User's Guide
   ● TM 990/512 Prototyping Board User's Guide

1.7 GLOSSARY
The following are definitions of terms used with the TM 990/101M. Applicable areas in
this manual are in parentheses.

Absolute Address: The actual memory address in quantity of bytes. Memory addressing is
usually represented in hexadecimal from 0000_{16} to $FFFF_{16}$ for the TM 990/101M.

Alphanumeric Character: Letters, numbers, and associated symbols.

ASCII Code: A seven-bit code used to represent alphanumeric characters and control
(Appendix C).

Assembler: Program that translates assembly language source statements into object code.

Assembly Language: Mnemonics which can be interpreted by an assembler and translated into an object program (paragraph 4.6).

Bit: The smallest part of a word; it has a value of either a 1 or 0.

Breakpoint: Memory address where a program is intentionally halted. This is a program debugging tool.

Byte: Eight bits or half a word.

Carry: A carry occurs when the most-significant bit is carried out in an arithmetic operation (i.e., result cannot be contained in only 16 bits), (paragraph 4.3.3.4).

Central Processing Unit (CPU): The "heart" of the computer: responsibilities include instruction access and interpretation, arithmetic functions, I/O memory access. The TMS 9900 is the CPU of the TM 990/101M.

Chad: Dot-like paper particles resulting from the punching of paper tape.

Command Scanner: A given set of instructions in the TIBUG monitor which takes the user's input from the terminal and searches a table for the proper code to execute.

Context Switch: Change in program execution environment, includes new program counter (PC) value and new workspace area.

CRU (Communications Register Unit): The TMS 9900's general purpose, command-driven input/output interface. The CRU provides up to 4096 directly addressable input and output bits (paragraph 4.8).

Effective Address: Memory address value resulting from interpretation of an instruction operand, required for execution of that instruction.

EPROM: See Read Only Memory.

Hexadecimal: Numerical notation in the base 16 (Appendix D).

Immediate Addressing: An immediate or absolute value (16-bits) is part of the instruction (second word of instruction).

Indexed Addressing: The effective address is the sum of the contents of an index register and an absolute (or symbolic) address (paragraph 4.5.3.5).

Indirect Addressing: The effective address is the contents of a register (paragraph 4.5.3.2).

Interrupt: Context switch in which new workspace pointer (WP) and program counter (PC) values are obtained from one of 16 interrupt traps in memory addresses 0000_{16} to $003E_{16}$ (paragraph 4.9).

I/O: The input/output lines are the signals which connect an external device to the data lines of the TMS 9990.

Least Significant Bit (LSB): Bit having the smallest value (samllest power of base 2); represented by the right-most bit.

Link: The process by which two or more object code modules are combined into one, with cross-referenced label address locations being resolved.

Load: Transfer control to operating system using the equivalent of a BLWP instruction to vectors in upper memory ($FFFC_{16}$ and $FFFE_{16}$). See Reset.

Loader: Program that places one or more absolute or relocatable object programs into memory (Appendix G).

Machine Language: Binary code that can be interpreted by the CPU (Table 4-4).

Monitor: A program that assists in the real-time aspects of program execution such as operator command interpretation and supervisor call execution. Sometimes called supervisor (Section 3).

Most Significant Bit (MSB): Bit having the most value; the left-most bit representing the highest power of base 2. This bit is often used to show sign with a 1 indicating negative and a 0 indicating positive.

Object Program: The hexadecimal interpretations of source code output by an assembler program. This is the code executed when loaded into memory.

One's Complement: Binary representation of a number in which the negative of the number is the complement or inverse of the positive number (all ones become zeroes, vice versa). The MSB is one for negative numbers and zero for positive. Two representations exist for zero: all ones or all zeroes.

Op Code: Binary operation code interpreted by the CPU to execute the instruction (paragraph 4.5.1).

Overflow: An overflow occurs when the result of an arithmetic operation cannot be represented in two's complement (i.e., in 15 bits plus sign bit), (paragraph 4.3.3.5).

Parity: Means for checking validity of a series of bits, usually a byte. Odd parity means an odd number of one bits; even parity means an even number of one bits. A parity bit is set to make all bytes conform to the selected parity. If the parity is not as anticipated, an error flag can be set by software. The parity jump instruction can be used to determine parity (paragraph 4.3.3.6).

PC Board: (Printed Circuit Board) a copper-coated fiberglass or phenolic board on which areas of copper are selectively etched away, leaving conductor paths forming a circuit. Various other processes such as soldermasking and silkscreen markings are added to higher quality PC boards.

Program Counter (PC): Hardware register that points to the next instruction to be executed or next word to be interpreted (paragraph 4.3.1).

PROM: See Read Only Memory.

Random Access Memory (RAM): Memory that can be written to as well as read from (vs. ROM).

Read Only Memory (ROM): Memory that can only be read from (can't change contents). Some can be programmed (PROM) using a PROM burner. Some PROM's can be erased (EPROM's) by exposure to ultraviolet light.

Reset: Transfer control to operating system using the equivalent of a BLWP instruction to vectors in lower memory (0000_{16} and 0002_{16}). See Load.

Source Program: Programs written in mnemonics that can be translated into machine language (by an assembler).

Status Register (ST): Hardware register that reflects the outcome of a previous instruction and the current interrupt mask (paragraph 4.3.3).

Supervisor: See Monitor

Utilities: A unique set of instructions used by differnt parts of the program to perform the same function. In the case of TIBUG, the utilities are the I/O XOP's (paragraph 3.3).

Word: Sixteen bits or two bytes.

Workspace Register Area: Sixteen words, designated registers 0 to 15, located in RAM for use by the executing program (paragraph 4.4).

Workspace Pointer (WP): Hardware register that contains the memory address of the beginning (register 0) of the workspace area (paragraph 4.3.2).

SECTION 2

INSTALLATION AND OPERATION OF TM 990/101M-1

2.1 GENERAL

This section explains procedures for unpacking and setting up the TM 990/101M board
for operation. This section assumes (1) the TIBUG monitor is resident on EPROM's as
initially shipped from the factory, and (2) that a terminal suitable for connection to
the main communications port is used with the proper cable assembly.

CAUTION

Be sure that the correct cable assembly is used with
your data terminal. For teletypewriters (TTY), refer to
Appendix A. For RS-232-C compatible terminals, refer to
Appendix B for the signal configuration used by the
main I/O port. Most RS-232-C compatible terminals, such
as a Lear Siegler ADM-1, will require the TM 990/502
cable, or equivalent. A TI 743 or 745 must use a TM
990/503 cable, or equivalent because of the connector
on the terminal end of the cable. A TI 733 requires the
use of a TM 990/505 cable, or equivalent. Many RS-232-C
compatible terminals come with their own cables, and
therefore will probably work with no problem.

2.2 REQUIRED EQUIPMENT

The basic equipment required, along with appropriate options, is explained in the
following paragraphs.

2.2.1 POWER SUPPLY

A power supply capable of meeting at least the following specification is required:

Voltage	Regulation	Current
+5 V	3%	1.8 A
-12 V	3%	0.3 A
+12 V	3%	0.4 A

A heavier duty supply is recommended, if possible, especially for supplying the +5
voltage.

2.2.2 TERMINALS AND CABLES

A 25-pin RS-232 male plug, type DB25P, is required. Ready made cables are available
from TI: see Appendix A or B.

  •   RS-232-C compatible terminal, including the TI 733 (using its own cable):
      see Appendix B to verify cabling you already have, or for instructions to
      make a custom cable.

  •   TI 743/745: see Appendix B for special cabling required (these terminals
      usually come with the correct cable).

  •   Teletype Model 3320/5JE (for TM 990/101M-1 and -3 microcomputer boards
      only): see Appendix A for required modifications for 20 mA neutral
      current-loop operation and proper cable connections.

2.2.3 POWER CABLE/CHASSIS

Use of a TM 990/510 OEM chassis greatly facilitates operation and setup. Alternately, one of the following 100-pin, 0.125 inch (center-to-center) PCB edge connectors may be used to interface with connector P1, such as with wire-wrap models:

- TI H321150
- Amphenol 225-804-50
- Viking 3VH50/9CND5
- Elco 00-6064-100-061-001.

2.2.4 PARALLEL I/O CONNECTOR

If the P4 parallel I/O port is used, a ribbon cable with a 40-pin, 0.1-inch center spacing PCB edge connector is needed. (The TIBUG monitor does not use the parallel port in its normal processing.) Wire-wrap connector examples are as follows:

- TI H311120
- Viking 3VH20/IJND5.

2.2.5 MISCELLANEOUS EQUIPMENT

- Volt-ohmmeter to measure completed/open connections and to verify power sup-ply voltages and connections.

- If any custom connections are required, a soldering iron (25-45 watt), rosin core solder, and wire are needed. Suggested wire sizes are 18 AWG insulated stranded wire for power connections, 24 AWG insulated stranded wire for I/O connections.

2.3 UNPACKING

Lift the TM 990/101M board from its carton and remove the protective wrapping. Check the board for shipping damage. If any damage is found, notify your TI distributor.

Verify that at least the following items are included:
- TM 990/101 User's Guide (this manual)
- TM 990/401 TIBUG Monitor Listing
- Data Manuals for the TMS 9900, TMS 9901, and TMS 9902

2.4 POWER AND TERMINAL HOOKUP

These procedures assume that the TIBUG monitor is resident in the required address space (0000_{16} to $07FF_{16}$), and that a terminal and cable of the proper type to match the intended serial interface (TTY, EIA, multidrop) is also employed.

Check the board and verify that the jumper configuration is as described in Table 2-1. Table 7-1 (in Section 7, Options) further defines jumper configurations.

Table 2-1. Board Jumper Positions As Shipped

Function	Stake Pins Used	Proper Connection & Description
Interrupt 4 source	E1,E2,E3	E1 to E2 - pin 18, connector P1
Interrupt 5 source	E4,E5,E6	E4 to E5 - pin 17, connector P1
Slow EPROM	E7,E8,E53	E8 to E53 - No WAIT state
2708/2716 Memory Map	E9,E10,E11	E10 to E11 - Use TMS 2708's
EPROM Enable	E12,E13,E14	E13 to E14 - On-board EPROM
HI/LO Memory Map	E15,E16,E17	E16 to E17 - EPROM low, RAM high
EIA Connector Ground	E18,E19	E18 to E19* - pin 1 of P3 grounded*
Microterminal +5 V	E20,E21	Shipped installed on -0001,3 only*
Microterminal +12 V	E22,E23	Shipped installed on -0001,3 only*
Microterminal -12 V	E24,E25	Shipped installed on -0001,3 only*
Main EPROM TYPE	E26 through E30	E27 to E28, E29 to E30 - TMS 2708's
Expansion EPROM type	E31 through E35	E32 to E33, E34 to E35 - TMS 2708's
Teletype	E36,E37	Shipped removed. On -0001,3 only, if using a TTY, borrow a Microterminal jumper plug for use here.
EIA/MD receive select	E38,E39,E40	E39 to E40 - EIA (and TTY) receive
Multidrop Termination	E41 through E52	Shipped installed on -0002 only*
Resistors and Duplex Select P3 Port Terminal/Modem	E54,E55,E56	E54 to E55 - Terminal Use*

*Jumper connection is not relevant for TIBUG operation with an RS-232-C or TTY terminal.

CAUTION
Be very careful to apply correct voltage levels to the
TM 990/101M. Texas Instruments assumes no responsibility
for damage caused by improper wiring or voltage
application by the user.

2.4.1 POWER SUPPLY CONNECTIONS
Figure 2-1 shows how the power supply is connected to the TM 990/101M through connector P1, using a 100-pin edge connector. Be careful to use the correct pins as numbered on the board; these pin numbers may not correspond to the numbers on the particular edge connector used. Check connections with an ohmmeter before applying power if there is any doubt about the quality or location of a connection.

The table in Figure 2-1 shows suggested color coding for the power supply plugs. To prevent incorrect connection, label the top side of the edge connector "TOP" and the bottom "TURN OVER".

Figure 2-2 shows how to correctly place the TM 990/101M in the TM 990/510 card chassis. Slot 1 of the chassis is reserved for the microcomputer because termination resistors for the control bus signals are at the opposite end of the backplane, according to transmission line concepts. Slide the microcomputer into the slot, following the guides. Be sure the P1 connector is correctly aligned in the socket on the backplane, then gently but firmly push the board edge into the edge connector socket.
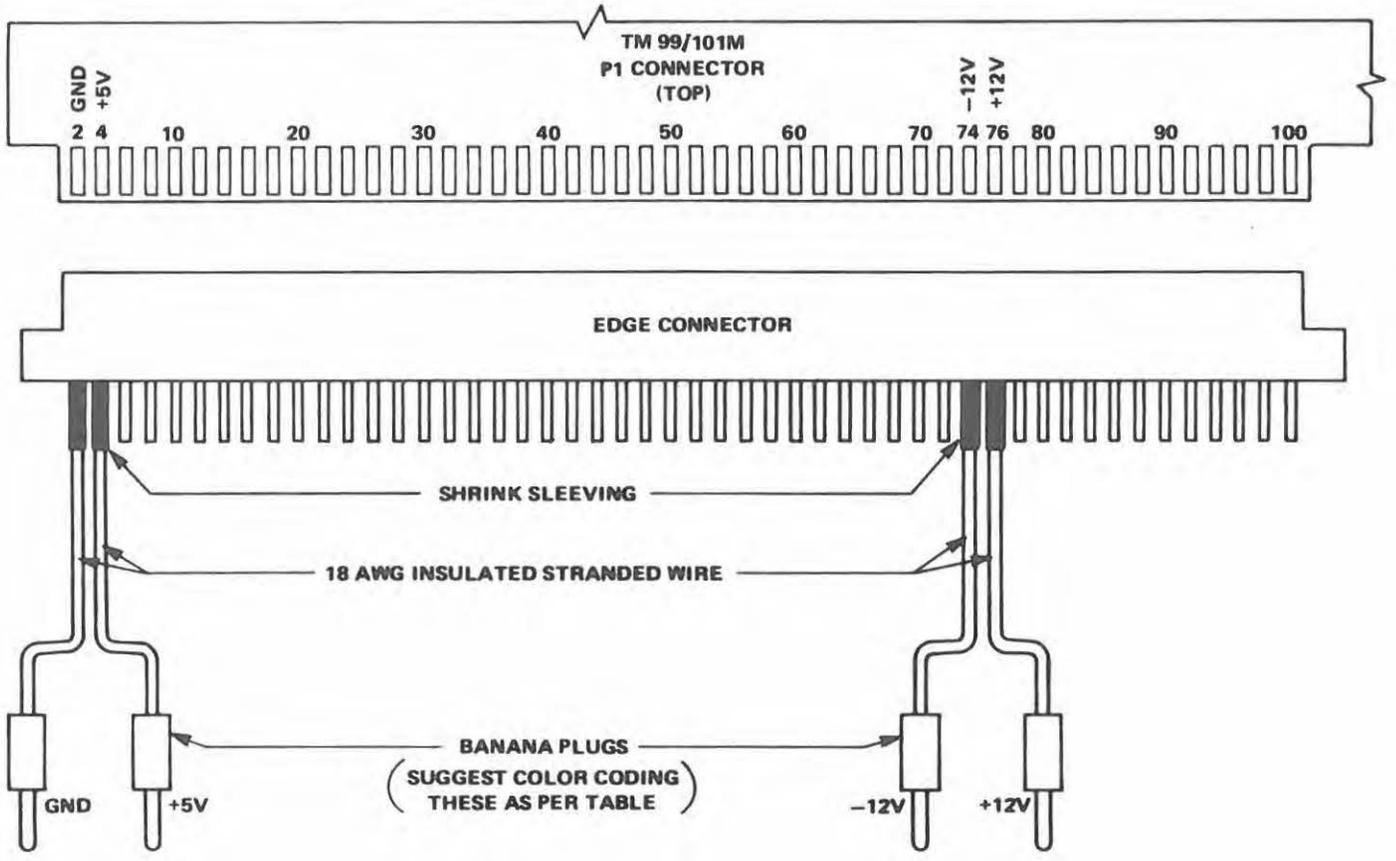
VOLTAGE	P1 PIN*	SUGGESTED PLUG COLORS
+5V	3, 4, 97, 98	RED
+12V	75, 76	BLUE
−12V	73, 74	GREEN
GND	1, 2, 99, 100	BLACK

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

A0001417

Figure 2-1. Power Supply Hookup
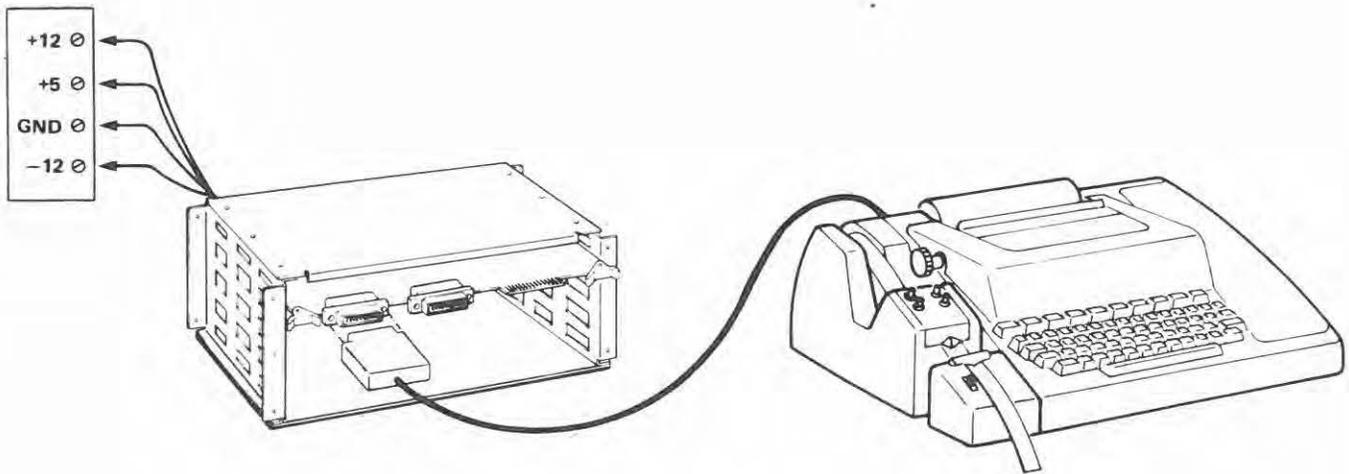
Figure 2-2.   TM 990/101M Board In TM 990/510 Chassis

Looking on the backside of the backplane, find the connections for each of the supply voltages and connect them to the power supply.

CAUTION

BEFORE connecting the power supply to the microcomputer, use a volt-ohmmeter to verify that correct voltages are present at the power supply. After verification, switch the power supply OFF, and then make the connections to the chassis as shown in Figure 2-2.

2.4.2 TERMINAL HOOKUP

Figure 2-3 shows how the TM 990/101M is connected to the TI 743 KSR terminal through connector P2. DE15S connector attaches to the terminal; a DB25P connector attached to P2 on the board. A table of point-to-point connections between the connectors are shown in the figure. Figure 2-4 shows a RS-232 terminal (e.g., TI 733), and Figure 2-5 shows a TTY.

All terminals connected to the microcomputer will have a similar hookup procedure and point-to-point configuration. For the differences between terminal cables, see Appendixes A and B. Terminals for communication directly with TIBUG must be connected to the main communications port (connector P2) at the corner of the board.

DB25P   DE15S

TO P2 ON   TO 743 DATA
TM 990/101M   TERMINAL

4 CONDUCTOR CABLE, 24 AWG
INSULATED STRANDED WIRE
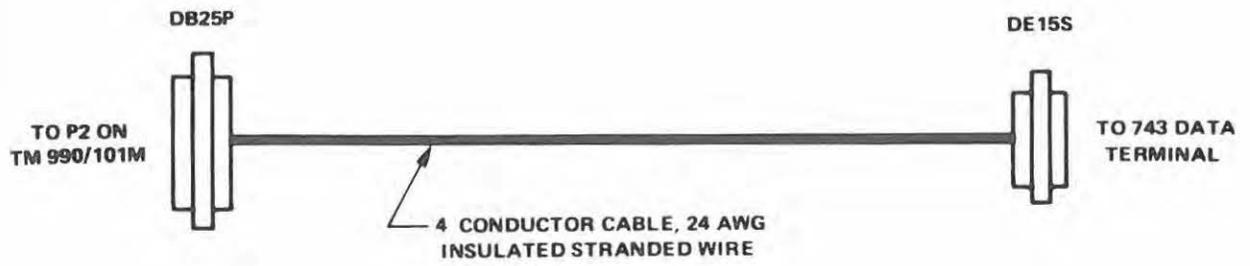
CONNECTIONS		
PIN ON DE15S	PIN ON DB25P	SIGNAL
13	2	XMIT
12	3	RECV
11	8	DCD
1	7	GND

A0001418

Figure 2-3.   743 KSR Terminal Hookup
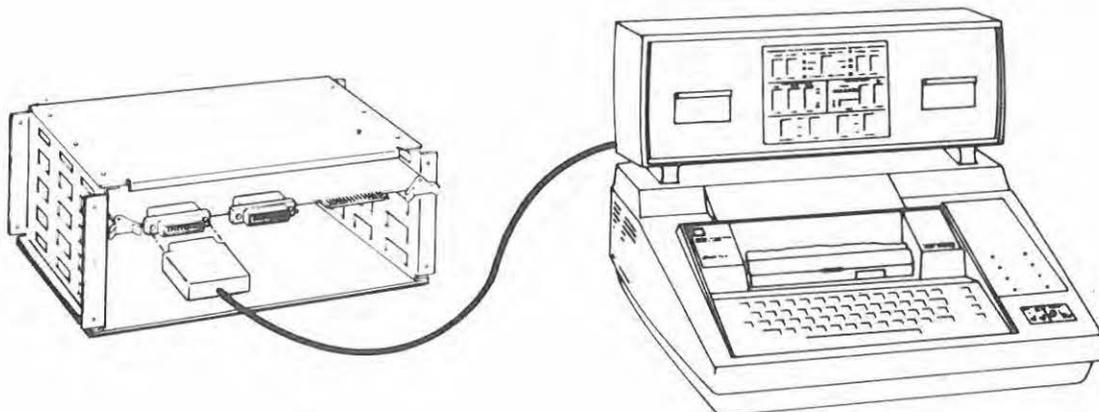
Figure 2-4.   Connector P2 Connected to RS-232-C Device (Model 733 ASR)
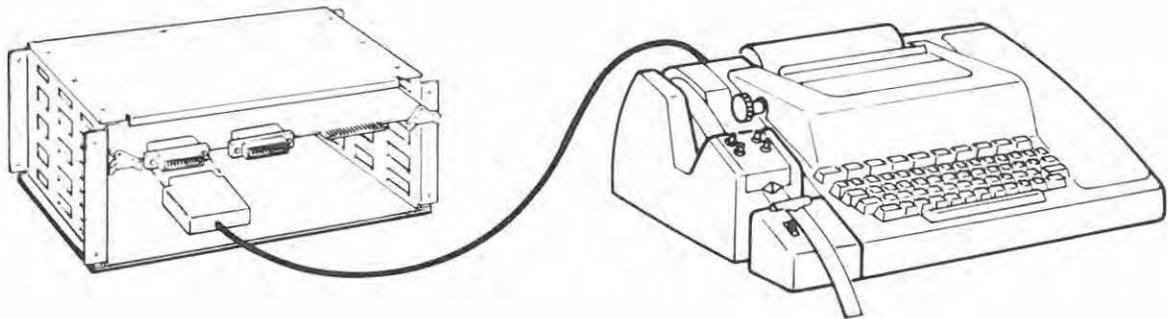
Figure 2-5.   Connector P2 Connected to TTY Device

The jumper marked EIA/MD, pins E38-E40, should be in the EIA position, pins E39 to E40, at all times unless the multidrop interface is used. If connecting a RS-232 terminal, remove the TTY jumper at E36-E37, if connecting a Teletype terminal, then insert the TTY jumper at E36-E37.

The TIBUG monitor operates the local I/O port at one of the following baud rates:

    110, 300, 600, 1200, 2400, 4800, 9600 or 19200 baud.

There is a 200 ms delay following a carriage return for all baud rates at or below 1200 baud. The delay allows for printhead travel.

The TMS 9902 asynchronous communication controller is initialized by TIBUG for a seven-bit ASCII character, even parity, and two stop bits (for compatibility with all terminals). At the terminal, set the baud rate of the terminal to one of the above speeds.

TIBUG also uses conversational mode full-duplex communication. Set the communications mode of your terminal to FULL DUPLEX, and set the OFF/ON LINE switch to ON LINE, or the functional equivalents.

2.4.3 FIVE-SWITCH DIP AND STATUS LED

A five-switch DIP and a programmable LED are accessed through the Communications Register Unit (CRU). Programming these is further explained in subsections 5.7 and 5.8 respectively.

2.5 OPERATION

2.5.1 VERIFICATION

Verify the following conditions before applying power:

- Power connected to correct pins on P1 connector.
- Terminal cable between P2 connector (NOT P3) and terminal.
- Jumpers in correct positions (see Table 2-1).
- Baud rate and communications mode are correctly set at the terminal; terminal is ON LINE.

2.5.2 POWER-UP/RESET

a. Apply power to the board and the data terminal.

b. Activate the RESET switch near the corner of the microcomputer board (see Figure 1-1). This activates the TIBUG monitor.

c. Press the "A" key on the terminal (it may be more convenient to press the carriage return key instead; this is also acceptable). TIBUG measures the time of the start bit and determines the baud rate. A carriage return time delay of 200 ms will be provided for all baud rates at or slower than 1200 baud.

d. TIBUG prints the TIBUG banner message and, on a new line, a question mark. This is a request to input a command to the TIBUG command scanner. Commands are explained in detail in Section 3, and the assembly language is described in Section 4.

NOTE
If control is lost during operation, return control to the TIBUG monitor by repeating steps b, c, and d.

2.6 SAMPLE PROGRAMS

The following sample programs can be used immediately to test the microcomputer board. Other sample programs that can be loaded and executed are provided in Figures 5-15 (interrupt timer message) and 5-22 (L.E.D. blink). Appendix K contains example programs that demonstrate microcomputer performance.

2.6.1 SAMPLE PROGRAM 1

The following sample program can be input using the TIBUG "M" command (paragraph 3.2.8), "R" command (paragraph 3.2.9), and "E" command (paragraph 3.2.4).

a. Enter the M command with a hexadecimal memory address of $FE00_{16}$.

b. Enter the following values into memory, typing the new values then using the space bar as described in paragraph 3.2.8.

Location	Enter Value	Assembly Language	
FE00	2FA0	XOP @ FE08,14	PRINT MSG
FE02	FE08		
FE04	0460	B @ 0080	GO TO TIBUG
FE06	0080		
FE08	4849	TEXT 'HI'	MESSAGE
FE0A	0D0A	DATA 0D0A	CR/LF
FE0C	0700	DATA 0700	BELL/END

Exit the M command with a carriage return after entering the last value above. The monitor will print a question mark.

c. Use the R command to set the address value 'FE00" into the P register (program counter).

d. Use the E command to execute the program.

e. The message 'HI' will print on the printer, followed by a carriage return, line feed, and a bell. Your terminal printout should resemble the following:

```
TIBUG REV.A

?M FE00
FE00=D300   2FA0
FE02=6300   FE08
FE04=0381   0460
FE06=23D2   0080
FE08=0300   4849
FE0A=03C2   0D0A
FE0C=7300   0700
?R
W=FFC6
P=01AC   FE00
?E HI

?
```

You can re-execute your program by repeating steps c. and d.

2.6.2 SAMPLE PROGRAM 2

Using steps 1 to 5 in paragraph 2.6.1 above, enter and execute the following program which has been assembled by the optional TM 990/402 line-by-line assembler.

```
FE00 2FA0 XOP @ FE08,14
FE02 FE08
FE04 0460 B @ 0080
FE06 0080
FE08 434F 'CONGRATULATIONS. YOUR PROGRAM WORK !
FE0A 4E47
FE0C 5241
FE0E 5455
FE10 4C41
FE12 5449
FE14 4F4E
FE16 532E
FE18 2059
FE1A 4F55
FE1C 5220
FE1E 5052
FE20 4F47
FE22 5241
FE24 4D20
FE26 574F
FE28 524B
FE2A 5321
FE2C 0707 + 0707
FE2E 0700 + 0700
```

You can re-execute your program by repeating c. and d. in paragraph 2.6.1 above.

2.7 DEBUG CHECKLIST

If the microcomputer does not respond correctly, turn the power OFF. Do not turn the power ON again until you are reasonably sure the problem has been found. The following is a checklist of points to verify.

* Check POWER circuits:
  - Proper power supply voltages and current capacity.
  - Proper connections from the power supply to the P1 edge connector. Check pin numbers on P1. Check plug positions at your power supply. Look for short circuits. Look for broken connections. Make sure board is seated in chassis or edge connector socket correctly. Be certain that the edge connector socket (if used) is not upside down.

* Check TERMINAL circuits:
  - Proper cable hookup to P2 connector, and to terminal. Verify with data in Appendixes A and B. One of the most common errors is that the terminal cable is not plugged in.
  - Check for power at the terminal. This is another common error – the terminal is not turned ON.
  - Terminal is ON LINE mode, or equivalent.
  - Terminal is in FULL DUPLEX mode, or equivalent. If the terminal is in HALF DUPLEX mode, it will print everything you type twice, or it may print garbage when you type. Put the terminal in FULL DUPLEX mode.

- EIA/MD jumper in EIA position (E30).
- Check BAUD RATE of terminal - it must be 110, 300, 600, 1200, 2400, 4800, 9600, or 19200 BAUD.

● Check jumper plug positions against Table 2-1.

● Be sure TIBUG EPROM's are in place correctly (U42 and U44).

● Check all socketed parts for correctly inserted pins. Be sure there aren't any bent under or twisted pins. Check pin 1 location.

If nothing happens, feel the components for excessive heat. Be careful as burns may occur if a defective component is found. If the cause of inoperation cannot be found, turn power OFF and call your TI distributor. Before calling, though, please be sure that your power supply, terminal, and all connectors (use a volt-ohmmeter) are working properly.

SECTION 3

TIBUG INTERACTIVE DEBUG MONITOR

3.1 GENERAL

TIBUG is a debug monitor which provides an interactive interface between the user and
the TM 990/101M. It is supplied by the factory on assembly TM 990/101M-1 only and is
available as an option, supplied on two 2708 EPROM's.

TIBUG occupies EPROM memory space from memory address (M.A.) 0080_{16} as shown in Figure
3-1. TIBUG uses four workspaces in 40 words of RAM memory. Also in this reserved RAM
area are the restart vectors which initialize the monitor following single step
execution of instructions.

The TIBUG monitor provides seven software routines that accomplish special tasks.
These routines, called in user programs by the XOP machine instruction, perform tasks
such as writing characters to a terminal. XOP utility instructions are discussed in
detail in paragraph 3.3.

All communication with TIBUG is through a 20 mA current loop or RS-232-C device. TIBUG
is initialized as follows:

- Press the RESET pushbutton (Figure 1-2). The monitor is called up through
  interrupt trap 0.

- Enter the character 'A' at the terminal. TIBUG uses this input to measure
  the width of the start bit and set the TMS 9902 Asynchronous Communication
  Controller (ACC) to the correct baud rate.

- TIBUG prints an initialization message on the terminal. On the next line it
  prints a question mark indicating that the command scanner is available to
  interpret terminal inputs.

- Enter one of the commands as explained in paragraph 3.2.

3.2 TIBUG COMMANDS

TIBUG commands are listed in Table 3-1.

Table 3-1. TIBUG Commands

INPUT	RESULTS	PARAGRAPH
B	Execute under Breakpoint	3.2.1
C	CRU Inspect/Change	3.2.2
D	Dump Memory to Cassette/Paper Tape	3.2.3
E	Execute	3.2.4
F	Find Word/Byte in Memory	3.2.5
H	Hex Arithmetic	3.2.6
L	Load Memory from Cassette/Paper Tape	3.2.7
M	Memory Inspect/Change	3.2.8
R	Inspect/Change User WP, PC, and ST Registers	3.2.9
S	Execute in Step Mode	3.2.10
T	1200 Baud Terminal	3.2.11
W	Inspect/Change Current User Workspace	3.2.12

MEMORY
ADDRESS

Address		Area
0000	INTERRUPT VECTOR (RESET)	TIBUG EPROM AREA
	INTERRUPT VECTORS 1 TO 15	USER EPROM AREA
0040	XOP VECTORS 0 AND 1	TIBUG EPROM AREA
0048	XOP VECTOR 2 TO 7	USER EPROM AREA
0060	XOP VECTORS 8 TO 15 MONITOR UTILITIES	
007E		
0080	TIBUG MONITOR	TIBUG EPROM AREA
07FE		

Address		Area
FFB0	MONITOR WORKSPACES	
FFFC	WP	TIBUG RAM AREA
FFFE	PC	

RESTART VECTORS { FFFC, FFFE }

Figure 3-1. Minimum Memory Requirements for TIBUG

3-2

Conventions used to define command syntax in this paragraph are listed in Table 3-2.

Table 3-2.   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

NOTE

Except where indicated otherwise, no space is necessary between the parts of these commands. All numeric input is assumed to be hexadecimal; the last four digits input will be the value used. Thus a mistaken numerical input can be corrected by merely making the last four digits the correct value. If fewer than four digits are input, they are right justified.

3.2.1 EXECUTE UNDER BREAKPOINT (B)

3.2.1.1 Syntax
B <address> <(CR)>

3.2.1.2 Description
This command is used to execute instructions from one memory address to another (the stopping address is the breakpoint). When execution is complete, WP, PC, and ST register contents are displayed and control is returned back to the monitor command scanner. Program execution begins at the address in the PC (set by using the R command). Execution terminates at the address specified in the B command, and a banner is output showing the contents of the hardware WP, PC, and ST registers in that order.

The address specified must be in RAM and must be the address of the first word of an instruction. The breakpoint is controlled by a software interrupt, XOP 15, which is executed when program execution is at the breakpoint address.

If no address is specified, the B command defaults to an E command, where execution continues with no halting point specified.

EXAMPLE:

```
?B FC06
BP     FFB0    FC06    E400
?
```

3.2.2 CRU INSPECT/CHANGE (C)

3.2.2.1 Syntax

C < CRU address > , < count > < (CR) >

3.2.2.2 Description

The Communication Register Unit (CRU) input bits are displayed right justified in a 16-bit hexadecimal representation. CRU addresses of the displayed bits will be:

    from "CRU Software Base Address"
    to "CRU Software Base Address" + 2(Count) - 2

"CRU Software Base Address" is the contents of register 12, bits 0 to 15, as used by the CRU instructions (paragraph 5.5). Up to 16 CRU bits may be displayed. Following display of the sensed CRU input bits, corresponding CRU output bits at that address may be specified by keying in a desired hexadecimal pattern of 1 to 16 bits, right justified. A carriage return following data display forces a return to the command scanner. A minus sign (-) or a space causes the same CRU input bits to be displayed again. Defaults are 0000_{16} for "Software Base Address" and 0 (count of 16) for "Count" (the latter is a hexadecimal value of 0 to F with 0 indicating a decimal 16 bits).

The CRU inspect/change command displays from 1 to 16 CRU bits, right justified. The command syntax includes the CRU software base address and the number of CRU bits to be displayed. The CRU software base address is the 16-bit contents of R12 as explained in paragraph 5.5 (vs. the "CRU hardware base address" on bits 3 to 14 of R12); thus, the user must use 2 X CRU hardware base address. This is shown in Figure 3-2 where 100_{16} is specified in the command to display values beginning with CRU bit 80_{16}.
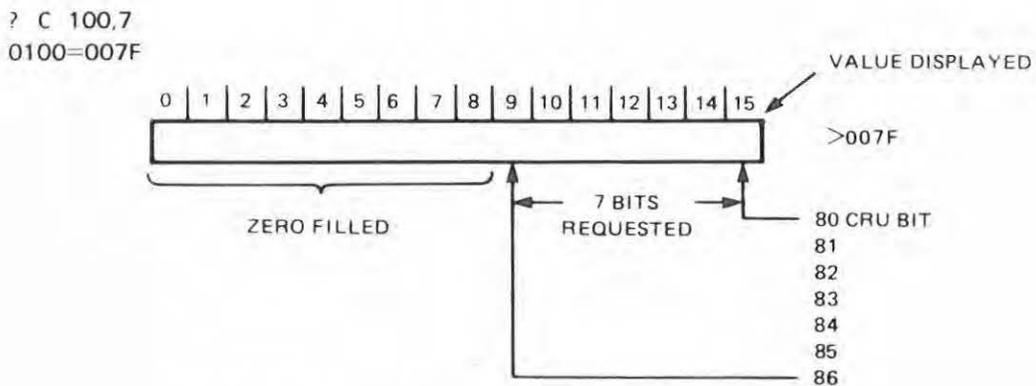
Figure 3-2. CRU Bits Inspected By C Command

3-4

EXAMPLES:

(1) Examine eight CRU input bits. CRU software base address is 20_{16}.

```
?C 20,8
0020=00FF ←── CARRIAGE RETURN ENTERED
?
```

(2) Set value of eight CRU output bits at CRU software base address 20_{16}; new value is 02_{16}.

```
?C 20,8        ┌── CHANGE 00FF TO 0002
0020=00FF  2 ←── 2 FOLLOWED BY CARRIAGE RETURN
?
```

(3) Check changes in CRU input bit 0.

```
?C 0,8
0000=0001  ─┐
0000=0001  ─│
0000=0001  ─├ MINUS SIGN ENTERED
0000=0001  ─│
0000=0001  ─┘
0000=0001 ←── CARRIAGE RETURN ENTERED
?
```

(4) Check to see if the TMS 9901 is in the interrupt mode (zero) or clock mode (one);

```
?C 100
0100=FFFE ←── ZERO IN LSB INDICATES INTERRUPT MODE
```

(5) Check the contents of the I/O ports on the TMS 9901 (bits 1 to 14).

```
?C 120,E
0120=000E
?
```

3.2.3 DUMP MEMORY TO CASSETTE/PAPER TAPE (D)

3.2.3.1 Syntax

```
                                                     ┌─MONITOR PROMPT
D <start address> { . } <stop address> { ^ } <entry address> { . } IDT = <name> < . >
```

NOTE

The termination given after IDT is a space bar. A carriage return or some other termination will cause the instruction to function incorrectly.

Memory is dumped from "start address" to "stop address." "Entry address" is the address in memory where it is desired to begin program execution. After entering a space or comma following the entry address, the monitor responds with an "IDT=" prompt asking for an input of up to eight characters that will identify the program. This program ID will be output. When the program is loaded into memory using the TIBUG loader, code will be dumped as non-relocatable data in 990 object record format with absolute load ("start address") and entry addresses specified. When loading this code once more, the LOAD will occur at the start address specified in the D instruction. If a user specifies a starting address while loading the object code previously dumped, the loader will ignore the user's input and load at the starting address specified during the 'D' command. Object record format is explained in Appendix G.

After entering the D command, the monitor will respond with "READY Y/N" and wait for a Y keyboard entry indicating that the receiving device is ready. This allows the user to verify switch settings, etc., before proceeding with the dump.

3.2.3.3 Dump to Cassette Example
The terminal is assumed to be a Texas Instruments 733 ASR or equivalent. The terminal must have automatic device control (ADC). This means that the terminal recognizes the four tape control characters DC1, DC2, DC3, and DC4.

The following procedure is carried out prior to answering the "READY Y/N" query (Figure 3-3):

(1) Load a cassette in the left (No. 1) transport on the 733 ASR.

(2) Place the transport in the "RECORD" mode.

(3) Rewind the cassette.

(4) Load the cassette. If the cassette does not load, it may be write protected. The write protect hole is on the bottom right side of the cassette (Figure 3-4). Cover it with the tab provided with the cassette. Now repeat steps 1 through 4.

(5) The KEYBOARD, PLAYBACK, RECORD, and PRINTER LOCAL/OFF/LINE switches must be in the LINE position.

(6) Place the TAPE FORMAT switch in the LINE position.

(7) Answer the "READY Y/N" query with a "Y"; the "Y" will not be echoed.
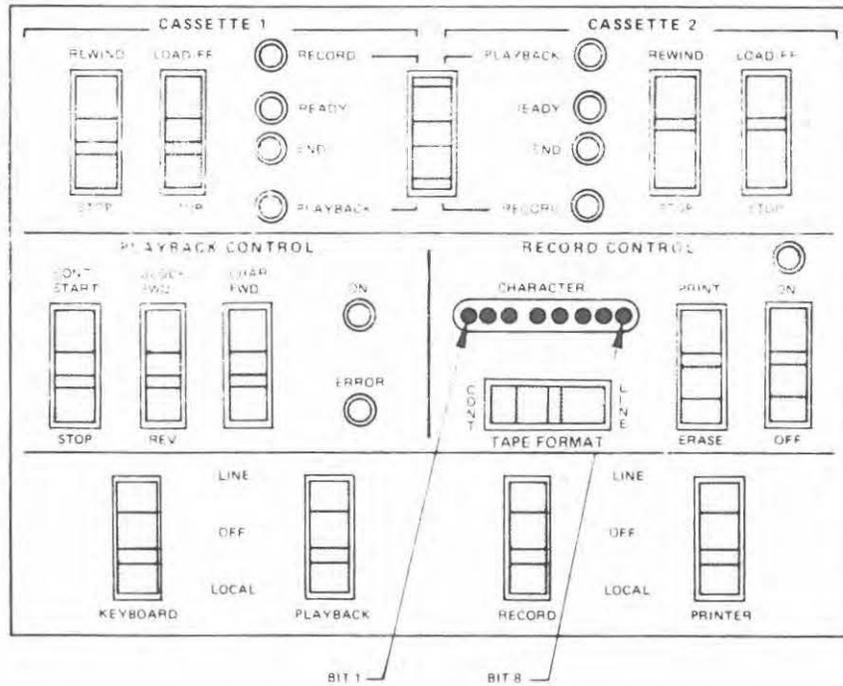
Figure 3-3. 733 ASR Module Assembly (Upper Unit) Switch Panel
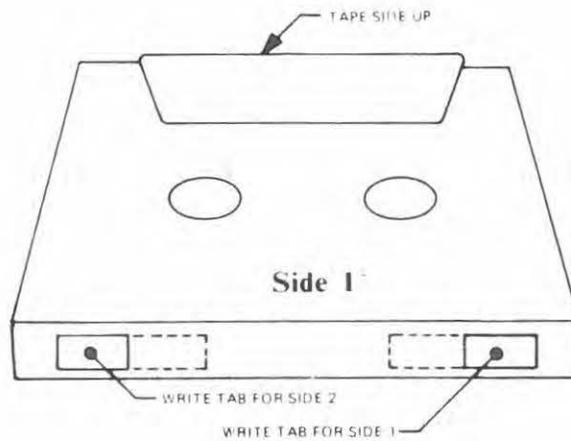
Figure 3-4. Tape Tabs

3.2.3.4 Dump to Paper Tape

The terminal is assumed to be an ASR 33 teletypewriter. The following steps should be completed carefully to avoid punching stray characters:

(1) Enter the command as described in paragraph 3.2.3.1. Do not answer the "READY Y/N" query yet.

(2) Change the teletype mode from ON LINE to LOCAL.

(3) Turn on the paper tape punch and press the RUBOUT key several times, placing RUBOUTS at the beginning of the tape for correct-reading/program-loading.

(4) Turn off the paper tape punch, and reset the teletype mode to LINE. (This is necessary to prevent punching stray characters).

(5) Turn on the punch and answer the "READY Y/N" query with "Y". The Y will not be echoed.

(6) Punching will begin. Each file is followed by 60 rubout characters. When these characters appear (identified by the constant punching of all holes) the punch must be turned off.

3.2.4 EXECUTE COMMAND (E)

3.2.4.1 Syntax

E

3.2.4.2 Description

The E command causes task execution to begin at current values in the Workspace Pointer and Program Counter.

Example: E

3.2.5 FIND COMMAND (F)

3.2.5.1 Syntax

F < start address > { $\overset{\wedge}{,}$ }< stop address > { $\overset{\wedge}{,}$ }< value > { $_{(C\overline{R})}$ }

3.2.5.2 Description

The contents of memory locations from "start address" to "stop address" are compared to "value". The memory addresses whose contents equal "value" are printed out. Default value for start address is 0. The default for "stop address" is 0. The default for "value" is 0.

If the termination character of "value" is a minus sign, the search will be from "start address" to "stop address" for the right byte in "value". If the termination character is a carriage return, the search will be a word mode search.

EXAMPLE:

```
?F 0,20 FFFF ←————————— CARRIAGE RETURN ENTERED
0006
000C
0012
0016
?F 0 20 FF- ←————————— MINUS SIGN ENTERED
0006
0007
000C
000D
0012
0013
0016
0017
?
```

3.2.6 HEXADECIMAL ARITHMETIC (H)

3.2.6.1 Syntax

H $<$ number 1 $>$ $\{ \wedge \atop , \}$ $<$ number 2 $>$ $<$ (CR) $>$

3.2.6.2 Description
The sum and difference of two hexadecimal numbers are output.

EXAMPLE:

```
?H 200,100 ←————————— CARRIAGE RETURN ENTERED
H1+H2=0300   H1-H2=0100
?
```

3.2.7 LOAD MEMORY FROM CASSETTE OR PAPER TAPE (L)

3.2.7.1 Syntax

L $<$ bias $>$ $<$ (CR) $>$

3.2.7.2 Description
Data in 990 object record format (defined in Appendix G) is loaded from paper tape or cassette into memory. Bias is the relocation bias (starting address in RAM). Its default is 0_{16}. Both relocatable and absolute data may be loaded into memory with the L command. After the data is loaded, the module identifier (see tag 0 in Appendix G) is printed on the next line.

3.2.7.3 Loading From Texas Instruments 733 ASR Terminal Cassette
The 733 ASR must be equipped with automatic device control (ADC). The following procedure is carried out prior to executing the L command:

(1) Insert the cassette in one of the two transports on the 733 ASR (cassette 1 in Figure 3-2).

(2) Place the transport in the playback mode.

(3) Rewind the cassette.

(4) Load the cassette.

(5) Set the KEYBOARD, PLAYBACK, RECORD, and PRINTER LOCAL/LINE switches to LINE.

(6) Set the TAPE FORMAT switch to LINE.

(7) Loading will be at 1200 baud.

Execute the L command.

3.2.7.4 Loading From Paper Tape (ASR33 Teletype)

Prior to executing the L command, place the paper tape in the reader and position the tape so the reader mechanism is in the null field prior to the file to be loaded. Enter the load command. If the ASR33 has ADC (automatic device control), the reader will begin to read from the tape. If the ASR33 does not have ADC, turn on the reader, and loading will begin.

Each file is terminated with 60 rubouts. When the reader reaches this area of the tape, turn it off. The loader will then pass control to the command scanner.

The user program counter (P) is loaded with the entry address if a 1 tag or a 2 tag is found on the tape.

EXAMPLE:

```
?L  0000 ◄───────────── CARRIAGE RETURN ENTERED
PROGRAM ◄────────────── PROGRAM ID FROM TAPE
?
```

3.2.8 MEMORY INSPECT/CHANGE, MEMORY DUMP (M)

3.2.8.1 Syntax

*   Memory Inspect/Change Syntax

        M < start address > { ^ } < stop address > < (CR) >

*   Memory Dump Syntax

        M < address > < (CR) >

3.2.8.2 Description

Memory inspect/change "opens" a memory location, displays it, and gives the option of changing the data in the location. The termination character causes the following:

*   If a carriage return, control is returned to the command scanner.

*   If a space, the next memory location is opened and displayed.

*   If a minus sign, the previous memory location is opened and displayed.

If a hexadecimal value is entered before the termination character, the displayed memory location is updated to the value entered.

Memory dump address directs a display of memory contents from "start address" to "stop address". Each line of output consists of the address of the first data word output followed by eight data words. Memory dump can be terminated at any time by typing any character on the keyboard.

EXAMPLES:

(1)

```
?M FE00 ◄──────────── CARRIAGE RETURN ENTERED
FE00=FF0F
FE02=0012  FFFF ◄──── NEW CONTENTS ENTERED
FE04=0311  - ◄─────── MINUS SIGN ENTERED
FE02=FFFF ◄────────── NEW CONTENTS
FE04=0311
FE06=0032  EEAA ◄──── CARRAGE RETURN ENTERED
?
```

(2)

```
?M 20 30
0020=0020  0030  0000  0005    0030  0D00  0000  0024
0030=0001
?
```

3.2.9 INSPECT/CHANGE USER WP, PC, AND ST REGISTERS (R)

3.2.9.1 Syntax

    R < (CR) >

3.2.9.2 Description
The user workspace pointer (WP), program counter (PC), and status register (ST) are inspected and changed with the R command. The output letters WP, PC, and ST identify the values of the three principal hardware registers passed to the TMS 9900 microprocessor when a B, E, or S command is entered. WP points to the workspace register area, PC points to the next instruction to be executed (Program Counter), and ST is the Status Register contents.

The termination character causes the following:

●   A carriage return causes control to return to the command scanner.

●   A space causes the next register to be opened.

Order of display is W, P, S.

EXAMPLES:

    (1)

```
?R
W=0020   100  ◄──── SPACE ENTERED
P=0000   200  ◄──── CARRIAGE RETURN ENTERED
?
```

    (2)

```
?R
W=0100 ◄──────────── SPACE ENTERED
P=0200 ◄────────┘
S=0000 ◄──────────── SPACE OR CARRIAGE RETURN ENTERED
?
```

3.2.10   EXECUTE IN SINGLE STEP MODE (S)

3.2.10.1   Syntax

    S

3.2.10.2   Description

Each time the S command is entered, a single instruction is executed at the address in the Program Counter, then the contents of the Program Counter, Workspace Pointer, and Status Register (after execution) are printed out. Successive instructions can be executed by repeated S commands. Essentially, this command executes one instruction then returns control to the monitor.

EXAMPLE:

```
?R
W=FFC6
P=FE10   FE00 ◄─── SPACES ENTERED
                         WORKSPACE POINTER
S=260A                   PROGRAM COUNTER
?S       FFC6 ◄── FE02 ◄── 860A ◄── STATUS REGISTER
?S       FFC6     FE04     860A
?S       FFC6     FE08     860A
?S       FFC6     FE0C     860A
?
```

NOTE

    Incorrect results are obtained when the S command causes execution of an XOP instruction (see paragraph 4.6.9) in a user program. To avoid this problem, use the B command (breakpoint) to the XOP vectors to execute any XOP's in a program (rather than the S command) with the appropriate XOP parameter previously loaded into R11 of the XOP workspace.

3.2.11 TI 733 ASR BAUD RATE (T)

3.2.11.1 Syntax

    T

3.2.11.2 Description
The T command is used to alert TIBUG that the terminal being used is a 1200 baud
terminal which is not a Texas Instruments' 733 ASR (e.g., a 1200 baud CRT). To revoke
the T command, enter it again.

3.2.11.3 Use
T is used only when operating with a true 1200 baud peripheral device. Note that T is
never used when operating at other baud rates.

In TIBUG the baud rate is set by measuring the width of the character 'A' input from a
terminal. When an 'A' of 1200 baud width is measured, TIBUG is set up to automatically
insert three nulls for every character output to the terminal. These nulls are
inserted to allow correct operation of the TM 990/101M with Texas Instruments 733ASR
data terminals.

3.2.12 INSPECT/CHANGE USER WORKSPACE (W)

3.2.12.1 Syntax

    W [REGISTER NUMBER] < (CR) >

3.2.12.2 Description
The W command is used to display the contents of all workspace registers or display
one register at a time while allowing the user to change the register contents. The
workspace begins at the address given by the Workspace Pointer.

The W command, followed by a carriage return, causes the contents of the entire
workspace to be printed. Control is then passed to the command scanner.

The W command followed by a register number in hexadecimal and a carriage return
causes the display of the specified register's contents. The user may then enter a new
value into the register by entering a hexadecimal value. The following are termination
characters whether or not a new value is entered:

    ●   A space causes display of the next register.

    ●   A minus sign causes display of the previous register.

    ●   A carriage return gives control to the command scanner.

EXAMPLES:

    (1)

        ?W ◄─────────────────── CARRIAGE RETURN ENTERED
        R0=F942   R1=0084   R2=FA2A   R3=0020   R4=FB5E   R5=0098   R6=1300   R7=0084
        R8=FAA0   R9=3600   RA=0EA6   RB=0000   RC=01C0   RD=0084   RE=FA30   RF=C600
        ?

(2)

```
?W 2 ◄─────────── CARRIAGE RETURN ENTERED
R2=0284   3456 ⎫
R3=001B   100  ⎬ SPACE ENTERED
R4=1608        ⎪
R5=0460   800F ⎭
R6=F800   0 ◄─────────── CARRIAGE RETURN ENTERED
```

3.3 USER ACCESSIBLE UTILITIES

TIBUG contains seven utility subroutines that perform I/O functions as listed in Table 3-3. These subroutines are called through the XOP (extended operation) assembly language instruction. This instruction is covered in detail in paragraph 4.6.9. In addition, locations for XOP's 0 and 1 contain vectors for utilities that drive the TM 990/301 microterminal, and XOP 15 is used by the monitor for the breakpoint facility.

Table 3-3. User Accessible Utilities

XOP	FUNCTION	PARAGRAPH
8	Write 1 Hexadecimal Character to Terminal	3.3.1
9	Read Hexadecimal Word from Terminal	3.3.2
10	Write 4 Hexadecimal Characters to Terminal	3.3.3
11	Echo Character	3.3.4
12	Write 1 Character to Terminal	3.3.5
13	Read 1 Character from Terminal	3.3.6
14	Write Message to Terminal	3.3.7

NOTE

All characters are in ASCII code.

NOTES

1. Initially, TIBUG will conduct I/O through the TMS 9902 connected to connector P2: in this mode, 0080_{16} is in TIBUG's R12 located at memory address (M.A.) $FFDE_{16}$. To change this configuration, change the contents of M.A. $FFDE_{16}$ before executing the I/O XOP. For example, to use the auxiliary TMS 9902 at P3, change M.A. $FFDE_{16}$ contents to 0180_{16}. CRU programming is discussed in paragraph 5.5.

2. The write character XOP (XOP 12) activates the REQUEST TO SEND signal of the TMS 9902. This signal is never deactivated by TIBUG so that modems may be used.

3. Most of the XOP format examples herein use a register for the source address, however, all XOP's can also use a symbolic memory address or any of the addressing forms available for the XOP instruction.

3.3.1 WRITE ONE HEXADECIMAL CHARACTER TO TERMINAL (XOP 8)

Format: XOP Rn,8

The least significant four bits of user register Rn are converted to their ASCII coded hexadecimal equivalent (0 to F) and output on the terminal. Control returns to the instruction following the extended operation.

EXAMPLE:

Assume user register 5 contains $203C_{16}$. The assembly language (A.L.) and machine language (M.L.) values are shown below.

A.L. XOP   R5,8   **SEND 4 LSB'S OF R5 TO TERMINAL**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
M.L.	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	$>$2E05

Terminal Output: C

3.3.2 READ HEXADECIMAL WORD FROM TERMINAL (XOP 9)

Format: XOP   Rn,9
            DATA   NULL   ADDRESS OF CONTINUED EXECUTION IF
                             NULL IS ENTERED
            DATA   ERROR   ADDRESS OF CONTINUED EXECUTION IF
                             NON HEX NO. ENTERED
            (NEXT INSTRUCTION) EXECUTION CONTINUED HERE IF VALID HEX
                             NUMBER AND TERMINATOR ENTERED

Binary representation of the last four hexadecimal digits input from the terminal is accumulated in user register Rn. The termination character is returned in register Rn+1. Valid termination characters are space, minus, comma, and a carriage return. Return to the calling task is as follows:

• If a valid termination character is the only input, return is to the memory address contained in the next word following the XOP instruction (NULL above).

- If a non-hexadecimal character or an invalid termination character is input, control returns to the memory address contained in the second word following the XOP instruction (ERROR above).

- If a hexadecimal string followed by a valid termination character is input, control returns to the word following the DATA ERROR statement above.

EXAMPLE

```
A.L.     XOP      R6,9      READ HEXADECIMAL WORD INTO R6
         DATA     >FFC0     RETURN ADDRESS, IF NO NUMBER
         DATA     >FFC6     RETURN ADDRESS, IF ERROR
```

M.L.		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
M.A.	FFB0	0	0	1	0	1	1	1	0	0	1	0	0	0	1	1	0	2146
	FFB2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	FFC0
	FFB4	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	0	FFC6

If the valid hexadecimal character string 12C is input from the terminal followed by a carriage return, control returns to memory address (M.A.) $FFB6_{16}$ with register 6 containing $012C_{16}$ and register 7 containing $0D00_{16}$.

If the hexadecimal character string 12C is input from the terminal followed by an ASCII plus (+) sign, control returns to location $FFC6_{16}$. Registers 6 and 7 are returned to the calling program without being altered. "+" is an invalid termination character.

If the only input from the terminal is a carriage return, register 6 is returned unaltered while register 7 contains $0D00_{16}$. Control is returned to address $FFC0_{16}$.

3.3.3 WRITE FOUR HEXADECIMAL CHARACTERS TO TERMINAL (XOP 10)

Format: XOP   Rn,10

The four-digit hexadecimal representation of the contents of user register Rn is output to the terminal. Control returns to the instruction following the XOP call.

EXAMPLE:

Assume register 1 contains $2C46_{16}$.

A.L.   XOP   R1,10   WRITE HEX NUMBER

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
M.L.	0	0	1	0	1	1	1	0	1	0	0	0	0	0	0	1	>2E81

Terminal Output: 2C46

3.3.4 ECHO CHARACTER (XOP 11)

Format: XOP   Rn, 11

This is a combination of XOP's 13 (read character) and 12 (write character). A
character in ASCII code is read from the terminal, placed in the left byte of Rn, then
written (echoed back) to the terminal. Control returns to the instruction following
the XOP after a character is read and written. By using a code to determine a
character string termination, a series of characters can be echoed and stored at a
particular address:

```
CLR     R2              CLEAR R2
LI      R1, >FE00       SET STORAGE ADDRESS
XOP     R2, 11          ECHO USING R2
CI      R2, >0D00       WAS CHARACTER A CR?
JEQ     $+6             YES, EXIT ROUTINE
MOVB    R2,*R1+         NO, MOVE CHAR TO STORAGE
JMP     $-10            REPEAT XOP
```

3.3.5 WRITE ONE CHARACTER TO TERMINAL (XOP 12)

    Format: XOP   Rn,12

The ASCII character in the left byte of user register Rn is output to the terminal.
The right byte of Rn is ignored. Control is returned to the instruction following the
call.

3.3.6 READ ONE CHARACTER FROM TERMINAL (XOP 13)

    Format: XOP   Rn,13

The ASCII representation of the character input from the terminal is placed in the
left byte of user register Rn. The right byte of register Rn is zeroed. When this
utility is called, control is returned to the instruction following the call only
after a character is input.

3.3.7 WRITE MESSAGE TO TERMINAL (XOP 14)

    Format: XOP   @MESSAGE,14

MESSAGE is the symbolic address of the first character of the ASCII character string
to be output. The string must be terminated with a byte containing binary zeroes.
After the character string is output, control is returned to the first instruction
following the call.

Assuming the following program:

MEMORY ADDRESS (Hex)	OP CODE (Hex)	A.L. MNEMONIC
FE00	2FA0	XOP @ > FEE0,14
FE02	FEE0	
FE04		
.	.	
.	.	
.	.	
FEE0	5445	TEXT 'TEST'
FEE2	5354	
FEE4	00	BYTE 0

During the execution of this XOP, the character string 'TEST' is output on the terminal and control is then returned to the instruction at location $FE04_{16}$. TEXT is an assembler directive to transcribe characters into ASCII code.

3.4 TIBUG ERROR MESSAGES

Several error messages have been included in the TIBUG monitor to alert the user to incorrect operation. In the event of an error, the word 'ERROR' is output followed by a single digit representing the error number.

Table 3-4 outlines the possible error conditions.

Table 3-4. 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.

In the event of errors 0 or 1, the program load process is terminated. If the program is being input from a 733 ASR, possible causes of the errors are a faulty cassette tape or dirty read heads in the tape transport. If the terminal device is an ASR33, chad may be caught in a punched hole in the paper tape. In either case repeat the load procedure.

In the event of error 2, the command is terminated. Reissue the command and parameters with a valid termination character.

Error 3 is the result of the user inputting a null field for either the start address, stop address, or the entry address to the dump routine. It also occurs if the ending address is less than the beginning address. The dump command is terminated. To correct the error, reissue the dump command and input all necessary parameters.

TM 990/101M INSTRUCTION EXECUTION

4.1 GENERAL

This section covers the instruction set used with the TM 990/101M including assembly language and machine language. This instruction set is compatible with other members of the 990 family.

Other topics include:

- Hardware and software registers (paragraphs 4.3 and 4.4).

- CRU addressing (paragraph 4.7)

- Interrupts (paragraph 4.10)

The TM 990/101M microcomputer is designed for use by a variety of users with varying technical backgrounds and available support equipment. Because a TM 990/101M user has the capability of writing his programs in machine language and entering them into memory using the TIBUG monitor, emphasis is on binary/hexadecimal representations of assembly language statements. The assembly language described herein can be assembled on a 990 family assembler. If an assembler is used, this section assumes that the user will be aware of all prerequisites for using the particular assembler.

It is also presumed that all users learning this instruction set have a working knowledge in:

- ASCII coded character set (described in Appendix C).

- Decimal/hexadecimal, binary number system (described in Appendix D).

Further information on the 990 assembly language is provided in the Model 990 Computer/TMS 9900 Microprocessor Assembly Language Programmer's Guide (P/N 943441-9701).

4.2 USER MEMORY

Figure 4-1 shows the user RAM space in memory available for execution of user programs. Note that the memory address value is the number of bytes beginning at 0000; thus, all word addresses are even values from 0000 to $FFFE_{16}$.

Programs in EPROM's can be read by the processor and executed; however, EPROM memory cannot be modified (written to). Therefore, workspace register areas are in RAM where their values can be modified. Restart vectors and TIBUG workspaces utilize the last 40 words of RAM memory space as shown in Figure 4-1.

4.3 HARDWARE REGISTERS

The TM 990/101M uses three major hardware registers in executing the instruction set: Program Counter (PC), Workspace Pointer (WP), and Status Register (ST).

4.3.1 PROGRAM COUNTER (PC)

This register contains the memory address of the next instruction to be executed. After an instruction image is read in for interpretation by the processor, the PC is incremented by two so that it "points" to the next sequential memory word.

Figure 4-1. Memory Map
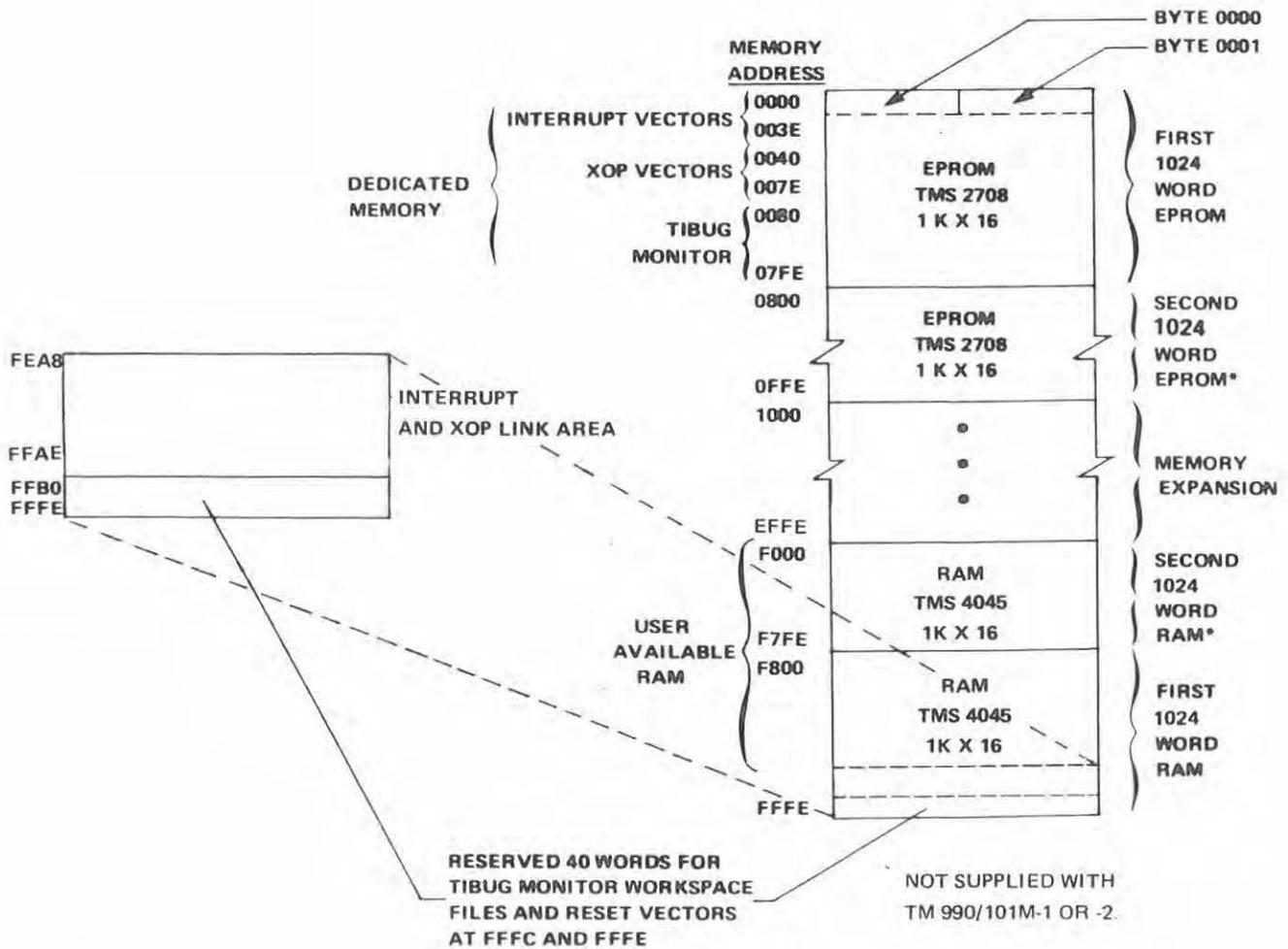
DEDICATED MEMORY

ADDRESS (HEX)	PURPOSE
0000-003F	Vectors for interrupts 0 (RESTART) to 15
0040-0047	Vectors for XOP's 0 and 1 (Microterminal I/O)
0048-005F	Vectors for XOP's 2 to 8 (Programmed by User)
0060-007F	Vectors for XOP's 8 to 15 (TIBUG utilities)
0080-07FF	TIBUG monitor
FEA8-FFAF	Interrupt and XOP linking area
FFB0-FFFB	Four overlapping monitor work spaces
FFFC-FFFF	Restart (load) vectors

BOARD MEMORY MAP

ADDRESS (HEX)	MEMORY TYPE	ENABLE SIGNAL	COMMENT
0000-07FF*	ROM (2708)	ROM1	TIBUG monitor area
0000-0FFF*	ROM (2716)	ROM1	Main EPROM, blank TMS 2716
0800-0FFF*	ROM (2708)	ROM2	Expansion EPROM
1000-1FFF*	ROM (2716)	ROM2	Expansion EPROM, blank TMS 2716
F000-F7FF	RAM (4045)	RAM2	Expansion RAM
F800-FFFF	RAM (4045)	RAM1	Standard RAM

*EPROM pairs (e.g., U42, U44 and U43, U45) must be of the same type — both TMS 2708's or both TMS 2716's. The two EPROMpairs, main and expansion, may be of different type if the appropriate jumper settings are made. This situation means selecting the 2716 memory map jumper option.

4.3.2 WORKSPACE POINTER (WP)

This register contains the memory address of the register file currently being used by the program under execution. This workspace consists of 16 contiguous memory words designated registers 0 to 15. The WP points to register 0. Paragraph 4.4 explains a workspace in detail.

4.3.3 STATUS REGISTER (ST)

The Status Register contains relevant information on preceding instructions and current interrupt level. Included are:

- Results of logical and two's complement comparisons (many instructions automatically compare the results to zero).

- Carry and overflow.

- Odd parity found (byte instructions only).

- XOP being executed.

- Lowest priority interrupt level that will be currently recognized by the processor.

The Status Register is shown in Figure 4-2.

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

L>	LOGICALLY GREATER THAN	OV	OVERFLOW
A>	ARITHMETICALLY GREATER THAN	OP	ODD PARITY
EQ	EQUAL	X	XOP BEING EXECUTED
C	CARRY		
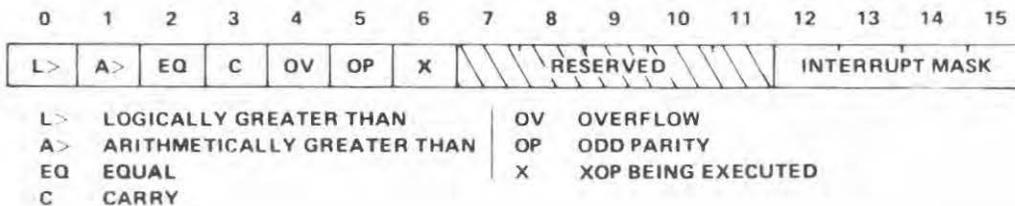
A0001421

Figure 4-2.   Status Register

4.3.3.1 Logical Greater Than

This bit contains the result of a comparison of words or bytes as unsigned binary numbers. Thus the most significant bit (MSB) does not indicate a positive or negative sign. The MSB of words being logically compared represents 2^{15} (32,768), and the MSB of bytes being logically compared represents 2^7 (128).

4.3.3.2 Arithmetic Greater Than

The arithmetic greater than bit contains the result of a comparison of words or bytes as two's complement numbers. In this comparison, the MSB of words or bytes being compared represents the sign of the number, zero for positive, or one for negative.

4.3.3.3 Equal
The equal bit is set when the words or bytes being compared are equal.

4.3.3.4 Carry
The carry bit is set by a carry out of the MSB of a word or byte (sign bit) during arithmetic operations. The carry bit is used by the shift operations to store the value of the last bit shifted out of the workspace register being shifted.

4.3.3.5 Overflow
The overflow bit is set when the result of an arithmetic operation is too large or too small to be correctly represented in two's complement (arithmetic) representation. In addition operations, overflow is set when the MSB's of the operands are equal and the MSB of the result is not equal to the MSB of the destination operand. In subtraction operations, the overflow bit is set when the MSB's of the operands are not equal, and the MSB of the result is not equal to the MSB of the destination operand. For a divide operation, the overflow bit is set when the most significant sixteen bits of the dividend (a 32-bit value) are greater than or equal to the divisor. For an arithmetic left shift, the overflow bit is set if the MSB of the workspace register being shifted changes value. For the absolute value and negate instructions, the overflow bit is set when the source operand is the maximum negative value, 8000_{16}.

4.3.3.6 Odd Parity
The odd parity bit is set in byte operations when the parity of the result is odd, and is reset when the parity is even. The parity of a byte is odd when the number of bits having a value of one is odd; when the number of bits having a value of one is even, the parity of the byte is even.

4.3.3.7 Extended Operation
The extended operation bit of the Status Register is set to one when a software implemented extended operation (XOP) is initiated.

4.3.3.8 Status Bit Summary
Table 4-1 lists the instruction set and the status bits affected by each instruction.

4.4 SOFTWARE REGISTERS
Registers used by programs are contained in memory. This speeds up context-switch time because the content of only one register (WP hardware register) needs to be saved instead of the entire register file. The WP, PC, and ST register contents are saved in a context switch.

A workspace is a contiguous 16 word area; its memory location can be designated by placing a value in the WP register through software or a keyboard monitor command. A program can use one or several workspace areas, depending upon register requirements.

More than three-fourths of the instructions can address the workspace register file; all shift instructions and most immediate operand instructions use workspace registers exclusively.

Figure 4-3 is an example of a workspace file in high-order memory (RAM). A workspace in ROM would be ineffective since it could not be written into. Note that several registers are used by particular instructions.

Table 4-1. Status Bits Affected by Instructions

MNEMONIC	L>	A>	EQ	C	OV	OP	X	MNEMONIC	L>	A>	EQ	C	OV	OP	X
A	X	X	X	X	X	—	—	LDCR	X	X	X	—	—	1	—
AB	X	X	X	X	X	X	—	LI	X	X	X	—	—	—	—
ABS	X	X	X	X	X	—	—	LIMI	—	—	—	—	—	—	—
AI	X	X	X	X	X	—	—	LREX	—	—	—	—	—	—	—
ANDI	X	X	X	—	—	—	—	LWPI	—	—	—	—	—	—	—
B	—	—	—	—	—	—	—	MOV	X	X	X	—	—	—	—
BL	—	—	—	—	—	—	—	MOVB	X	X	X	—	—	X	—
BLWP	—	—	—	—	—	—	—	MPY	—	—	—	—	—	—	—
C	X	X	X	—	—	—	—	NEG	X	X	X	X	X	—	—
CB	X	X	X	—	—	X	—	ORI	X	X	X	—	—	—	—
CI	X	X	X	—	—	—	—	RSET	—	—	—	—	—	—	—
CLR	—	—	—	—	—	—	—	RTWP	X	X	X	X	X	X	X
COC	—	—	X	—	—	—	—	S	X	X	X	X	X	—	—
CZC	—	—	X	—	—	—	—	SB	X	X	X	X	X	X	—
DEC	X	X	X	X	X	—	—	SBO	—	—	—	—	—	—	—
DECT	X	X	X	X	X	—	—	SBZ	—	—	—	—	—	—	—
DIV	—	—	—	—	X	—	—	SETO	—	—	—	—	—	—	—
IDLE	—	—	—	—	—	—	—	SLA	X	X	X	X	X	—	—
INC	X	X	X	X	X	—	—	SOC	X	X	X	—	—	—	—
INCT	X	X	X	X	X	—	—	SOCB	X	X	X	—	—	X	—
INV	X	X	X	—	—	—	—	SRA	X	X	X	X	—	—	—
JEQ	—	—	—	—	—	—	—	SRC	X	X	X	X	—	—	—
JGT	—	—	—	—	—	—	—	SRL	X	X	X	X	—	—	—
JH	—	—	—	—	—	—	—	STCR	X	X	X	—	—	1	—
JHE	—	—	—	—	—	—	—	STST	—	—	—	—	—	—	—
JL	—	—	—	—	—	—	—	STWP	—	—	—	—	—	—	—
JLE	—	—	—	—	—	—	—	SWPB	—	—	—	—	—	—	—
JLT	—	—	—	—	—	—	—	SZC	X	X	X	—	—	—	—
JMP	—	—	—	—	—	—	—	SZCB	X	X	X	—	—	X	—
JNC	—	—	—	—	—	—	—	TB	—	—	X	—	—	—	—
JNE	—	—	—	—	—	—	—	X	2	2	2	2	2	2	2
JNO	—	—	—	—	—	—	—	XOP	2	2	2	2	2	2	2
JOC	—	—	—	—	—	—	—	XOR	X	X	X	—	—	—	—
JOP	—	—	—	—	—	—	—								

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.

WP REGISTER

MEMORY
ADDRESS
(HEXADECIMAL)

		12	15		
FC00	FC00		SHIFT COUNT	R 0	BITS 12-15 USED BY SHIFT INSTRUCTIONS
	FC02			R 1	
	FC04			R 2	
	FC06			R 3	
	FC08			R 4	
	FC0A			R 5	
	FC0C			R 6	
	FC0E			R 7	
	FC10			R 8	
	FC12			R 9	
	FC14			R 10	
	FC16			R 11	USED BY XOP'S AND BRANCH RETURN
	FC18			R 12	USED IN CRU ADDRESSING
	FC1A			R 13	USED IN CONTEXT SWITCHING (XOP, BLWP, RTWP)
	FC1C			R14	
	FC1E			R 15	
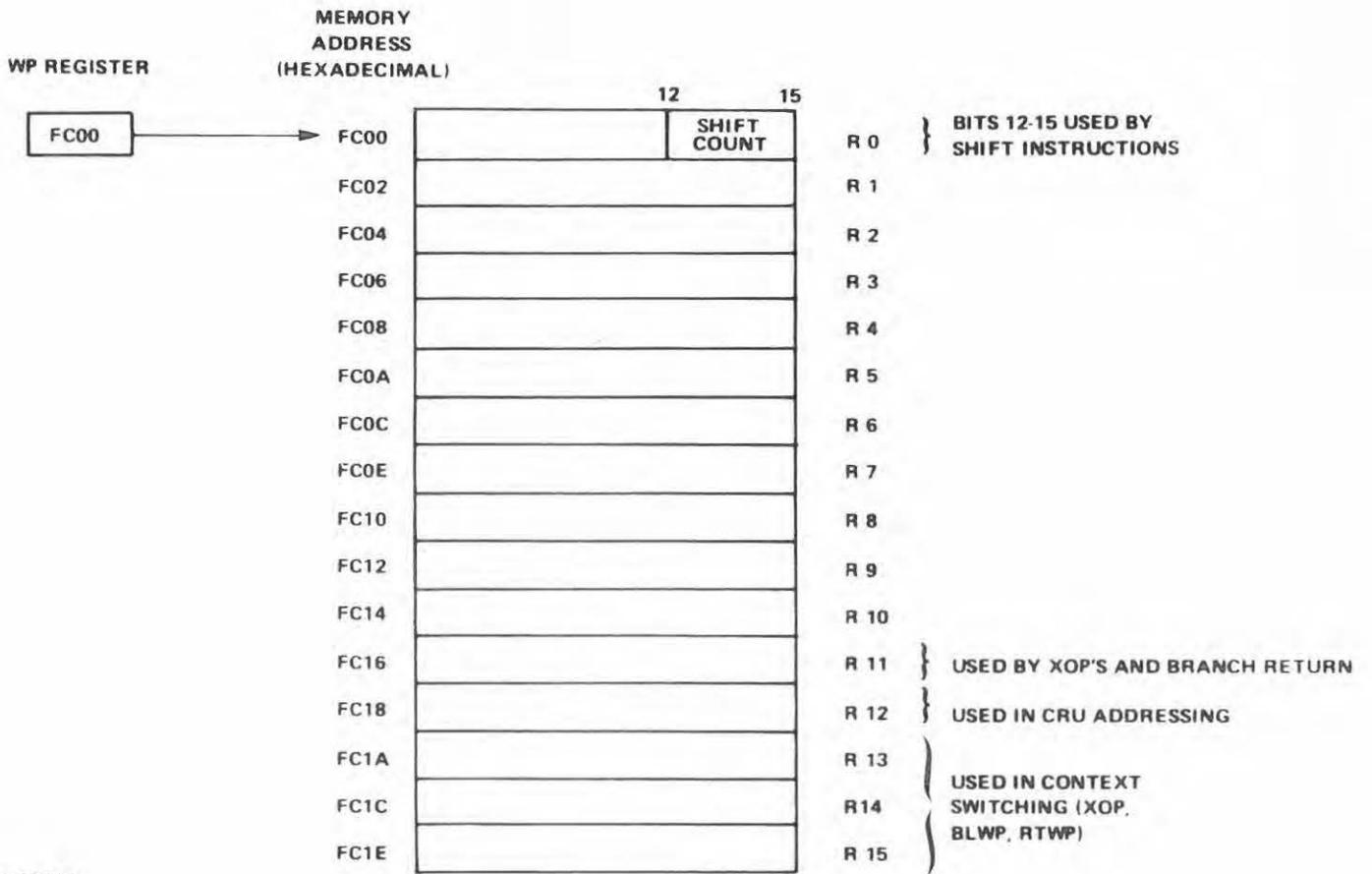
A0001422

Figure 4-3.   Workspace Example

4-6

4.5 INSTRUCTION FORMATS AND ADDRESSING MODES

The instructions used by the TM 990/101M are contained in 16-bit memory words and require one, two, or three words for full definition. The first word (or the single word) of an instruction will describe the purpose of the instruction while the succeeding one or two words will be numbers that are referenced by the initial instruction word. A word describing an instruction is interpreted by the Central Processing Unit (CPU) by decoding the various fields within the 16 bits. These fields are shown in Figure 4-4 for the 9900 instruction set which is also categorized into nine instruction formats as shown in the figure.

In order to construct instructions in machine language, the programmer must have a knowledge of the fields and formats of the instructions. This knowledge is often very important in debugging operations because it allows the programmer to change bits within an instruction in order to solve an execution problem.

The fields within an instruction word contain the following information (see Figure 4-4):

- Op code which identifies the desired operation to be accomplished when this instruction is executed.

FORMAT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	GENERAL USE
1	OP CODE			B	T_D		DR				T_S		SR				ARITHMETIC
2	OP CODE								SIGNED DISPLACEMENT								JUMP
3	OP CODE					WR				T_S		SR				LOGICAL
4	OP CODE					C				T_S		SR				CRU
5	OP CODE						C				R				SHIFT
6	OP CODE								T_S		SR				PROGRAM
7	OP CODE									NOT USED				CONTROL
8	OP CODE									N	R				IMMEDIATE
9	OP CODE					DR			T_S		SR				MPY, DIV, XOP

OP CODE	OPERATION CODE
B	BYTE INDICATOR (1 BYTE)
T_D	DESTINATION ADDRESS TYPE*
DR	DESTINATION REGISTER
T_S	SOURCE ADDRESS TYPE*
SR	SOURCE REGISTER
C	CRU TRANSFER COUNT OR SHIFT COUNT
R	REGISTER
N	NOT USED

*T_D OR T_S	ADDRESS MODE TYPE
00	DIRECT REGISTER
01	INDIRECT REGISTER
10	PROGRAM COUNTER RELATIVE, NOT INDEXED (SR OR DR = 0)
	PROGRAM COUNTER RELATIVE + INDEX REGISTER (SR OR DR>0)
11	INDIRECT REGISTER, AUTOINCREMENT REGISTER

A0001423

Figure 4-4.   TM 990/101M Instruction Formats

4-7

- B code which identifies whether the instruction will affect a full 16-bit word in memory or an 8-bit byte. A one indicates a byte will be addressed, while a zero indicates a word will be addressed.

- T fields identified by TD for the destination T field and TS for the source T field. The T field is a two-bit code which identifies which of five different addressing modes will be used (direct register, indirect register, memory address, memory address indexed, and indirect register autoincremented). These modes are described in detail in paragraphs 4.5.1 through 4.5.5. The source T field is the code for the source address and the destination T field is the code for the destination address. As shown in Figure 4-4, only five instruction formats use a T field.

- Source and destination register fields which contain the number of the register affected (0 through 15).

- Displacement fields that contain a bias to be added to the program counter in program counter relative addressing. This form of addressing is further described in paragraph 4.5.7.

- Fields that contain counts for indicating the number of bits that will be shifted in a shift instruction or the number of Communication Register Unit (CRU) bits that will be addressed in a CRU instruction.

4.5.1 DIRECT REGISTER ADDRESSING (T=00$_2$)

In direct register addressing, execution involves data contained within one of the 16 workspace registers. In the first example in Figure 4-5, both the source and destination operands are registers as noted in the assembly language example at the top of the figure. Both T fields contain 00$_2$ to denote direct register addressing and their associated register fields contain the binary value of the number of the register affected. The 110$_2$ in the op code field identifies this instruction as a move instruction. Since the B field contains a zero, the data moved will be the full 16 bits of the register (a byte instruction addressing a register would address the left byte of the register). The instruction specifies moving the contents of register 1 to register 4, thus changing the contents of register 4 to the same value as in register 1. Note that the assembly language statement is constructed so that the source register is the first item in the operand while the destination register is the second item in the operand. This order is reversed in the machine language construction with the destination register and its T field first and the source register and its T field second.

4.5.2 INDIRECT REGISTER ADDRESSING (T=01$_2$)

In indirect register addressing, the register does not contain the data to be affected by the instruction; instead, the register contains the address within memory of where that data is stored. For example, the instruction in Figure 4-6 specifies to move the contents of register 1 to the address which is contained in register 4 (indirect register 4). Instead of moving the value in register 1 to register 4 as was the case in Figure 4-5, the CPU must first read in the 16-bit value in register 4 and use that value as a memory address at which location the contents of register 1 will be stored. In the example, register 4 contains the value FD00$_{16}$. This instruction stores the value in register 1 into memory address (M.A.) FD00$_{16}$.

Indirect register addressing is specified in assembly language source code by preceding the register number with an asterisk (*). For example, A *R1,*R2 means to add the contents of the memory address in register 1 to the contents of the memory address in register 2, leaving the sum in the memory address contained in register 2.

In direct register addressing, the contents of a register are addressed. In indirect register addressing, the CPU goes to the register to find out what memory location to address. This form of addressing is especially suited for repeating an instruction while accessing successive memory addresses. For example, if you wished to add a series of numbers in 100 consecutive memory locations, you could place the address of the first number in a register, and execute an add indirect through that register, causing the contents of the first memory address (source operand) to be added to another register or memory address (destination operand). Then you could increment the contents of the register containing the address of the number, loop back to the add instruction, and repeat the add, only this time you will be adding the contents of the next memory address to the accumulator (destination operand). This way a whole string of data can be summed using a minimum of instructions. Of course, you would have to include control instructions that would signal when the entire list of 100 addresses have been added, but there are obvious advantages in speed of operation, better utilization of memory space, and ease in programming.

EXAMPLE 1

ASSEMBLY LANGUAGE:

MOV   R1,R4   MOVE THE CONTENTS OF R1 (SOURCE) TO R4 (DESTINATION)

SOURCE OPERAND
DESTINATION OPERAND

T CODE FOR
DIRECT REGISTER
REGISTER 4

T CODE FOR
DIRECT REGISTER
REGISTER 1

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	> C101

| OP CODE | B | T_D | DR | T_S | SR |

M.A.

FC00	R0
FC02	R1
FC04	R2
FC06	R3
FC08	R4
FC0A	R5

PLACE R1 BINARY
IMAGE IN R4

EXAMPLE 2

ASSEMBLY LANGUAGE:

A   R4,R10   ADD THE CONTENTS OF R4 (SOURCE) AND R10 (DESTINATION)

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	> A284

| OP CODE | B | T_D | DR | T_S | SR |

A0001424

Figure 4-5. Direct Register Addressing Example

4-9

ASSEMBLY LANGUAGE:

    MOV   R1,*R4   MOVE THE CONTENTS OF RI (SOURCE) TO ADDRESS IN R4 (DESTINATION)

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	1	>C501

  OP CODE   B T_D DR T_S SR
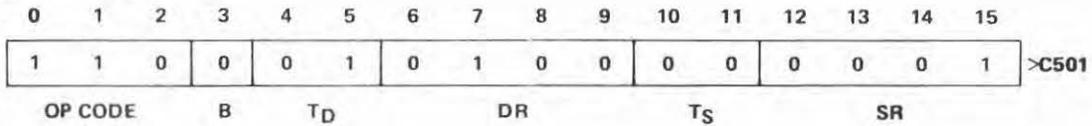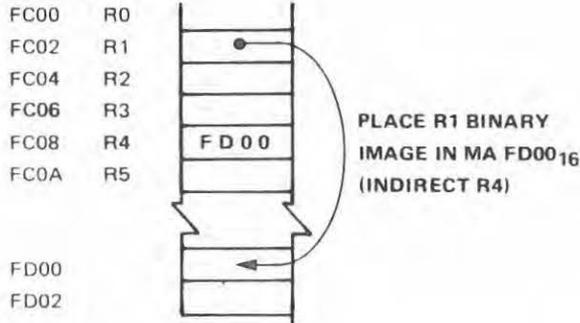
A0001425

Figure 4-6. Indirect Register Addressing Example

ASSEMBLY LANGUAGE:

    MOV   R1,*R4+   MOVE THE CONTENTS OF RI TO ADDRESS CONTAINED IN R4,
                        INCREMENT ADDRESS BY 2

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	1	> CD01

  OP CODE   B T_D DR T_S SR
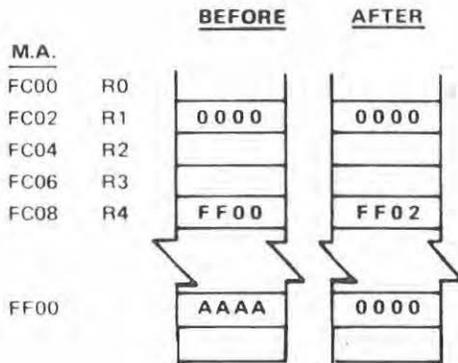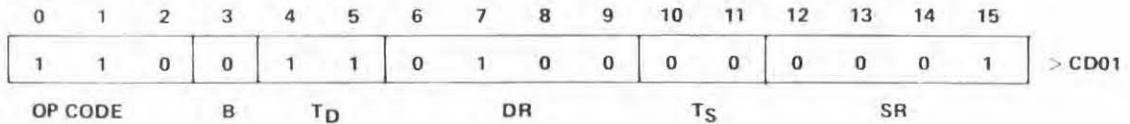
A0001427

Figure 4-7. Indirect Register Autoincrement Addressing Example

4.5.3 INDIRECT REGISTER AUTOINCREMENT ADDRESSING (T=11$_2$)
Indirect register autoincrement addressing is the same as indirect register addressing (paragraph 4.5.2) except for an additional feature - automatic incrementation of the register. This saves the requirement of adding an increment (by one or two) instruction to increment the register being used in the indirect mode. The increment will be a value of one for byte instructions (e.g., add byte or AB) or a value of two for full word instructions (e.g., add word or A)

In assembly language, the register number is preceded by an asterisk (*) and followed by a plus sign (+) as shown in Figure 4-7. Note in the figure that the contents of register 4 was incremented by two since the instruction was a move word (vs. byte) instruction. If the example used a move byte instruction, the contents of the register would be incremented by one so that successive bytes would be addressed (the 16-bit word addresses in memory are always even numbers or multiples of two since each contains two bytes). Bytes are also addressed by various instructions of the 990 instruction set.

Note that only a register can contain the indirect address.

4.5.4 SYMBOLIC MEMORY ADDRESSING, NOT INDEXED (T=10$_2$)
This mode does not use a register as an address or as a container of an address. Instead, the address is a 16-bit value stored in the second or third word of the instruction. The SR or DR fields will be all zeroes as shown for the destination register field in the first example of Figure 4-8. When the T field contains 10$_2$, the CPU retrieves the contents of the next memory location and uses these contents as the effective address. In assembly language, a symbolic address is preceded by an at sign (@) to differentiate a numerical memory address from a register number. All alphanumeric labels must be preceded by an @ sign; numerical values preceded by an @ sign will be assembled as an absolute address (the TM 990/402 Line-By-Line Assembler does not recognize alphanumeric symbols but does recognize absolute memory addresses).

In the second example in Figure 4-8, both the source and destination operands are symbolic memory addresses. In this case, the source address is the first word following the instruction and the destination is the second word following the instruction in machine language.

4.5.5 SYMBOLIC MEMORY ADDRESSING, INDEXED (T=10$_2$)
Note that the T field for indexed as well as non-indexed symbolic addressing is the same (10$_2$). In order to differentiate between the two different modes, the associated SR or DR field is interrogated; if this field is all zeroes (0000$_2$), non-indexed addressing is specified; if the SR or DR field is greater than zero, indexing is specified and the non-zero value is the index register number. As a result, register 0 cannot be used as an index register.

In assembly language, the symbolic address is followed by the number of the index register in parentheses. In the example in Figure 4-9, the source operand is non-indexed symbolic memory addressing while the destination operand is indexed symbolic memory addressing. In this case, the destination effective address is the sum of the FF02$_{16}$ value in the source memory address word plus the value in the index register (0004$_{16}$). The effective address in this case is FF06$_{16}$ as shown by the addition in the left part of the figure.

Note that only symbolic addressing can be indexed.

EXAMPLE 1

ASSEMBLY LANGUAGE:

MOV   R1,@>FF00   MOVE THE CONTENTS OF RI TO ADDRESS >FF00

NOTE

The > sign indicates hexidecimal representation.

MACHINE LANGUAGE:

		OP CODE			B	T_D				DR			T_S			SR			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1st WORD	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	>C801
2nd WORD	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	>FF00

<u>M.A.</u>

RO
R1
R2

FEFE
FF00

PLACE R1 BINARY
IMAGE IN
MA >FF00
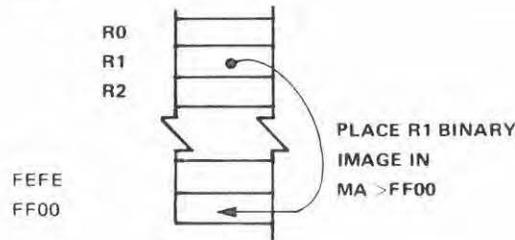
EXAMPLE 2

ASSEMBLY LANGUAGE:

MOV   @>FF0A,@>FF08   MOVE THE CONTENTS OF >FF0A TO >FF08

MACHINE LANGUAGE:

		OP CODE			B	T_D				DR			T_S			SR			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1st WORD	1	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	>C820
2nd WORD	1	1	1	1	1	1	1	1	0	0	0	0	1	0	1	0	>FF0A (SOURCE)
3rd WORD	1	1	1	1	1	1	1	1	0	0	0	0	1	0	0	0	>FF08 (DESTINATION)

<u>M.A.</u>	**BEFORE**	**AFTER**
FF08	FFFF	0000
FF0A	0000	0000
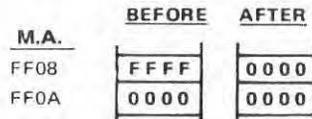
A0001428

Figure 4-8.   Direct Memory Addressing Example

ASSEMBLY LANGUAGE:

        MOV   @>FF00,@>FF02(R1)   MOVE THE CONTENTS OF >FF00 TO >FF02 + RI CONTENTS

MACHINE LANGUAGE:

OP CODE			B	T_D		DR			T_S			SR				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	0	1	0	0	0	0	1	1	0	0	0	0	0	·C860
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	·FF00 (SOURCE)
1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	0	>FF02 (DESTINATION)


```
                                    BEFORE    AFTER
              M.A.
                        R0
                        R1      0004      0004
                        R2

>FF02 (D)
+ 0004 (R1)
  >FF06           FF00    FFEE      FFEE
                  FF02    0000      0000
                  FF04    0000      0000
              →FF06    0000      FFEE

A0001429
```
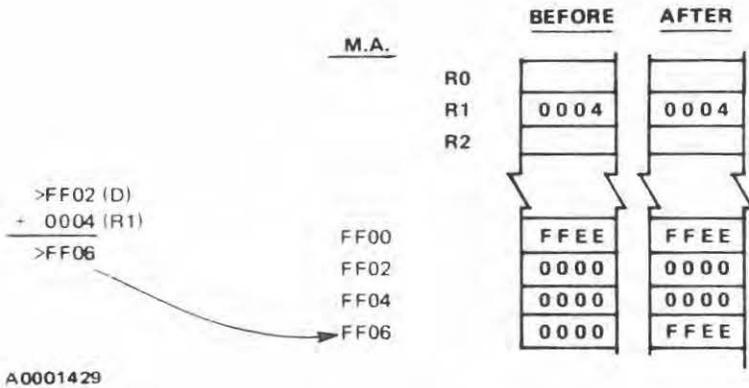
Figure 4-9.   Direct Memory Addressing, Indexed Example

4.5.6   IMMEDIATE ADDRESSING

This mode allows an absolute value to be specified as an operand; this value is used in connection with a register contents or is loaded into the WP or the Status Register interrupt mask. Examples are shown below:

        LI   R2,100   LOAD 100 INTO REGISTER 2
        CI   R8,>100   COMPARE R8 CONTENTS TO >100, RESULTS IN ST
        LWPI   >3C00   SET WP TO MA >3C00

4.5.7   PROGRAM COUNTER RELATIVE ADDRESSING

This mode allows a change in Program Counter contents, either an unconditional change or a change conditional on Status Register contents. Examples are shown below:

        JMP $+6   JUMP TO LOCATION, 6 BYTES FORWARD
        JMP   THERE   JUMP TO LOCATION LABELLED THERE
        JEQ $+4   IF ST EQ BIT 1, JUMP 4 BYTES (MA + 4)
        JMP   >3E26   JUMP TO M.A. >3E26 (LINE-BY-LINE ASSEMBLER ONLY)

The dollar symbol ($) means "from this address"; thus, $+6 means "this address plus 6 bytes."

4.6 INSTRUCTIONS

Table 4-2 lists terms used in describing the instructions of the TM 990/101M. Table 4-3 is an alphabetical list of instructions. Table 4-4 is a numerical list of instructions by op code. Examples are shown in both assembly language (A.L.) and machine language (M.L.). The greater-than sign (>) indicates hexadecimal.

Table 4-2. Instruction Description Terms

TERM	DEFINITION
B	Byte indicator (1 = byte, 0 = word)
C	Bit count
DR	Destination address register
DA	Destination address
IOP	Immediate operand
LSB(n)	Least significant (right most) bit of (n)
M.A.	Memory Address
MSB(n)	Most significant (left most) bit of (n)
N	Don't care
PC	Program counter
Result	Result of operation performed by instruction
SR	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
WR or R	Workspace register
WRn or Rn	Workspace register n
(n)	Contents of n
a → b	a is transferred to b
(a) → b	Contents of a is transferred to be
[n]	Absolute value of n
+	Arithmetic addition
−	Arithmetic subtraction
AND	Logical AND
OR	Logical OR
⊕	Logical exclusive OR
n̄	Logical complement of n
>	Hexadecimal value

ASSEMBLY LANGUAGE MNEMONIC	MACHINE LANGUAGE OP CODE	FORMAT	STATUS REG. BITS AFFECTED	RESULT COMPARED TO ZERO	INSTRUCTION	PARAGRAPH
A	A000	1	0-4	X	Add (word)	4.6.1
AB	B000	1	0-5	X	Add (byte)	4.6.1
ABS	0740	6	0-2	X	Absolute Value	4.6.6
AI	0220	8	0-4	X	Add Immediate	4.6.8
ANDI	0240	8	0-2	X	AND Immediate	4.6.8
B	0440	6	—		Branch	4.6.6
BL	0680	6	—		Branch and Link (R11)	4.6.6
BLWP	0400	6	—		Branch; New Workspace Pointer	4.6.6
C	8000	1	0-2		Compare (word)	4.6.1
CB	9000	1	0-2,5		Compare (byte)	4.6.1
CI	0280	8	0-2		Compare Immediate	4.6.8
CKOF	03C0	7	—		User Defined	4.6.7
CKON	03A0	7	—		User Defined	4.6.7
CLR	04C0	6	—		Clear Operand	4.6.6
COC	2000	3	2		Compare Ones Corresponding	4.6.3
CZC	2400	3	2		Compare Zeroes Corresponding	4.6.3
DEC	0600	6	0-4	X	Decrement (by one)	4.6.6
DECT	0640	6	0-4	X	Decrement (by two)	4.6.6
DIV	3C00	9	4		Divide	4.6.3
IDLE	0340	7	—		Computer Idle	4.6.7
INC	0580	6	0-4	X	Increment (by one)	4.6.6
INCT	05C0	6	0-4	X	Increment (by two)	4.6.6
INV	0540	6	0-2	X	Invert (One's Complement)	4.6.6
JEQ	1300	2	—		Jump Equal (ST2=1)	4.6.2
JGT	1500	2	—		Jump Greater Than (ST1=1), Arithmetic	4.6.2
JH	1B00	2	—		Jump High (ST0=1 and ST2=0), Logical	4.6.2
JHE	1400	2	—		Jump High or Equal (ST0 or ST2=1), Logical	4.6.2
JL	1A00	2	—		Jump Low (ST0 and ST2=0), Logical	4.6.2
JLE	1200	2	—		Jump Low or Equal (ST0=0 or ST2=1), Logical	4.6.2
JLT	1100	2	—		Jump Less Than (ST1 and ST2=0), Arithmetic	4.6.2
JMP	1000	2	—		Jump Unconditional	4.6.2
JNC	1700	2	—		Jump No Carry (ST3=0)	4.6.2
JNE	1600	2	—		Jump Not Equal (ST2=0)	4.6.2
JNO	1900	2	—		Jump No Overflow (ST4=0)	4.6.2
JOC	1800	2	—		Jump On Carry (ST3=1)	4.6.2

Table 4-3. Instruction Set, Alphabetical Index

ASSEMBLY LANGUAGE MNEMONIC	MACHINE LANGUAGE OP CODE	FORMAT	STATUS REG. BITS AFFECTED	RESULT COMPARED TO ZERO	INSTRUCTION	PARAGRAPH
JOP	1C00	2	—		Jump Odd Parity (ST5÷1)	4.6.2
LDCR	3000	4	0-2,5	X	Load CRU	4.6.4
LI	0200	8	—	X	Load Immediate	4.6.8
LIMI	0300	8	12-15		Load Interrupt Mask Immediate	4.6.8
LREX	03E0	7	12-15		Load and Execute	4.6.7
LWPI	02E0	8	—		Load Immediate to Workspace Pointer	4.6.8
MOV	C000	1	0-2	X	Move (word)	4.6.1
MOVB	D000	1	0-2,5	X	Move (byte)	4.6.1
MPY	3800	9	—		Multiply	4.6.3
NEG	0500	6	0-2	X	Negate (Two's Complement)	4.6.6
ORI	0260	8	0-2	X	OR Immediate	4.6.8
RSET	0360	7	12-15		Reset AU	4.6.7
RTWP	0380	7	0-15		Return from Context Switch	4.6.7
S	6000	1	0-4	X	Subtract (word)	4.6.1
SB	7000	1	0-5	X	Subtract (byte)	4.6.1
SBO	1D00	2	—		Set CRU Bit to One	4.6.2
SBZ	1E00	2	—		Set CRU Bit to Zero	4.6.2
SETO	0700	6	—		Set Ones	4.6.6
SLA	0A00	5	0-4	X	Shift Left Arithmetic	4.6.5
SOC	E000	1	0-2	X	Set Ones Corresponding (word)	4.6.1
SOCB	F000	1	0-2,5	X	Set Ones Corresponding (byte)	4.6.1
SRA	0800	5	0-3	X	Shift Right (sign extended)	4.6.5
SRC	0B00	5	0-3	X	Shift Right Circular	4.6.5
SRL	0900	5	0-3	X	Shift Right Logical	4.6.5
STCR	3400	4	0-2,5	X	Store From CRU	4.6.4
STST	02C0	8	—		Store Status Register	4.6.8
STWP	02A0	8	—		Store Workspace Pointer	4.6.8
SWPB	06C0	6	—		Swap Bytes	4.6.6
SZC	4000	1	0-2	X	Set Zeroes Corresponding (word)	4.6.1
SZCB	5000	1	0-2,5	X	Set Zeroes Corresponding (byte)	4.6.1
TB	1F00	2	2		Test CRU Bit	4.6.2
X	0480	6	—		Execute	4.6.6
XOP	2C00	9	6		Extended Operation	4.6.9
XOR	2800	3	0-2	X	Exclusive OR	4.6.3

Table 4-3. Instruction Set, Alphabetical Index (Concluded)

Table 4-4. Instruction Set, Numerical Index

MACHINE LANGUAGE OP CODE (HEXADECIMAL)	ASSEMBLY LANGUAGE MNEMONIC	INSTRUCTION	FORMAT	STATUS BITS AFFECTED
0200	LI	Load Immediate	8	0-2
0220	AI	Add Immediate	8	0-4
0240	ANDI	And Immediate	8	0-2
0260	ORI	Or Immediate	8	0-2
0280	CI	Compare Immediate	8	0-2
02A0	STWP	Store WP	8	—
02C0	STST	Store ST	8	—
02E0	LWPI	Load WP Immediate	8	—
0300	LIMI	Load Int. Mask	8	12-15
0340	IDLE	Idle	7	—
0360	RSET	Reset AU	7	12-15
0380	RTWP	Return from Context Sw.	7	0-15
03A0	CKON	User Defined	7	—
03C0	CKOF	User Defined	7	—
03E0	LREX	Load & Execute	7	—
0400	BLWP	Branch; New WP	6	—
0440	B	Branch	6	—
0480	X	Execute	6	—
04C0	CLR	Clear to Zeroes	6	—
0500	NEG	Negate to Ones	6	0-2
0540	INV	Invert	6	0-2
0580	INC	Increment by 1	6	0-4
05C0	INCT	Increment by 2	6	0-4
0600	DEC	Decrement by 1	6	0-4
0640	DECT	Decrement by 2	6	0-4
0680	BL	Branch and Link	6	—
06C0	SWPB	Swap Bytes	6	—
0700	SETO	Set to Ones	6	—
0740	ABS	Absolute Value	6	0-2
0800	SRA	Shift Right Arithmetic	5	0-3
0900	SRL	Shift Right Logical	5	0-3
0A00	SLA	Shift Left Arithmetic	5	0-4
0B00	SRC	Shift Right Circular	5	0-3
1000	JMP	Unconditional Jump	2	—
1100	JLT	Jump on Less Than	2	—
1200	JLE	Jump on Less Than or Equal	2	—
1300	JEQ	Jump on Equal	2	—
1400	JHE	Jump on High or Equal	2	—
1500	JGT	Jump on Greater Than	2	—
1600	JNE	Jump on Not Equal	2	—
1700	JNC	Jump on No Carry	2	—
1800	JOC	Jump on Carry	2	—
1900	JNO	Jump on No Overflow	2	—
1A00	JL	Jump on Low	2	—
1B00	JH	Jump on High	2	—
1C00	JOP	Jump on Odd Parity	2	—
1D00	SBO	Set CRU Bits to Ones	2	—
1E00	SBZ	Set CRU Bits to Zeroes	2	—
1F00	TB	Test CRU Bit	2	2
2000	COC	Compare Ones Corresponding	3	2

Table 4-4. Instruction Set, Numerical Index (Concluded)

MACHINE LANGUAGE OP CODE (HEXADECIMAL	ASSEMBLY LANGUAGE MNEMONIC	INSTRUCTION	FORMAT	STATUS BITS AFFECTED
2400	CZC	Compare Zeroes Corresponding	3	2
2800	XOR	Exclusive Or	3	0-2
2C00	XOP	Extended Operation	9	6
3000	LDCR	Load CRU	4	0-2,5
3400	STCR	Store CRU	4	0-2,5
3800	MPY	Multiply	9	
3C00	DIV	Divide	9	4
4000	SZC	Set Zeroes Corresponding (Word)	1	0-2
5000	SZCB	Set Zeroes Corresponding (Byte)	1	0-2,5
6000	S	Subtract Word	1	0-4
7000	SB	Subtract Byte	1	0-5
8000	C	Compare Word	1	0-2
9000	CB	Compare Byte	1	0-2,5
A000	A	Add Word	1	0-4
B000	AB	Add Byte	1	0-5
C000	MOV	Move Word	1	0-2
D000	MOVB	Move Byte	1	0-2,5
E000	SOC	Set Ones Corresponding (Word)	1	0-2
F000	SOCB	Set Ones Corresponding (Byte)	1	0-2,5

4.6.1 FORMAT 1 INSTRUCTIONS

These are dual operand instructions with multiple addressing modes for source and destination operands.

GENERAL FORMAT:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE			B	T_D		DR					T_S		SR		
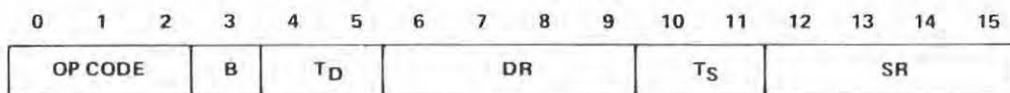
If B = 1, the operands are bytes and the operand addresses are byte addresses. If B = 0, the operands are words and the operand addresses are word addresses.

MNEMONIC	OP CODE 0	OP CODE 1	OP CODE 2	B 3	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
A	1	0	1	0	Add	Yes	0-4	$(SA)+(DA) \rightarrow (DA)$
AB	1	0	1	1	Add bytes	Yes	0-5	$(SA)+(DA) \rightarrow (DA)$
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 appropriate status bits
MOV	1	1	0	0	Move	Yes	0-2	$(SA) \rightarrow (DA)$
MOVB	1	1	0	1	Move bytes	Yes	0-2,5	$(SA) \rightarrow (DA)$
S	0	1	1	0	Subtract	Yes	0-4	$(DA) - (SA) \rightarrow (DA)$
SB	0	1	1	1	Subtract bytes	Yes	0-5	$(DA) - (SA) \rightarrow (DA)$
SOC	1	1	1	0	Set ones corresponding	Yes	0-2	$(DA) \text{ OR } (SA) \rightarrow (DA)$
SOCB	1	1	1	1	Set ones corresponding bytes	Yes	0-2,5	$(DA) \text{ OR } (SA) \rightarrow (DA)$
SZC	0	1	0	0	Set zeroes corresponding	Yes	0-2	$(DA) \text{ AND } (\overline{SA}) \rightarrow (DA)$
SZCB	0	1	0	1	Set zeroes corresponding bytes	Yes	0-2,5	$(DA) \text{ AND } (\overline{SA}) \rightarrow (DA)$

EXAMPLES

(1) ASSEMBLY LANGUAGE:

    A   @>100,R2   **ADD CONTENTS OF MA >100 & R2, SUM IN R2**

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	>A0A0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	>0100

(2) ASSEMBLY LANGUAGE:

    CB   R1,R2   **COMPARE BYTE R1 TO R2, SET ST**

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	>9081

NOTE

In byte instruction designating a register, the left byte is used. In the above example, the left byte (8 MSB's) of R1 is compared to the left byte of R2, and the ST set to the results.

4.6.2 FORMAT 2 INSTRUCTIONS

4.6.2.1 Jump Instructions

Jump instructions cause the PC to be loaded with the value (PC+2 (signed displacement)) if bits of the Status Register are at specified values. Otherwise, no operation occurs and the next instruction is executed since the PC was incremented by two and now points to the next instruction. The signed displacement field is a word (not byte) count to be added to PC. Thus, the jump instruction has a range of -128 to 127 words (-256 to 254 bytes) from the memory address following the jump instruction. No ST bits are affected by a jump instruction.

GENERAL FORMAT:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE								SIGNED DISPLACEMENT (WORDS)							
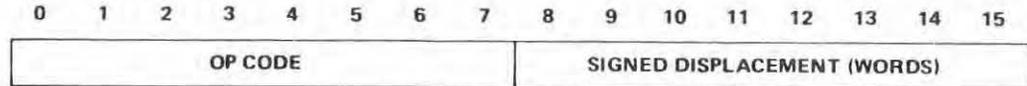
MNEMONIC	OP CODE 0 1 2 3 4 5 6 7	MEANING	ST CONDITION TO CHANGE PC
JEQ	0 0 0 1 0 0 1 1	Jump equal	ST2 = 1
JGT	0 0 0 1 0 1 0 1	Jump greater than	ST1 = 1
JH	0 0 0 1 1 0 1 1	Jump high	ST0 = 1 and ST2 = 0
JHE	0 0 0 1 0 1 0 0	Jump high or equal	ST0 = 1 or ST2 = 1
JL	0 0 0 1 1 0 1 0	Jump low	ST0 = 0 and ST2 = 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

In assembly language, $ in the operand indicates "at this instruction". Essentially JMP $ causes an unconditional loop to the same instruction location, and JMP $+2 is essentially a no-op ($+2 means "here plus two bytes"). Note that the number following the $ is a byte count while displacement in machine language is in words.

EXAMPLES:

(1) ASSEMBLY LANGUAGE:
      JEQ   S+4   IF EQ BIT SET, SKIP 1 INSTRUCTION

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	1	>1301

PC POINTS TO →   JEQ $+4   IF STATUS REGISTER BIT 2 = 1   SKIP NEXT INSTRUCTION
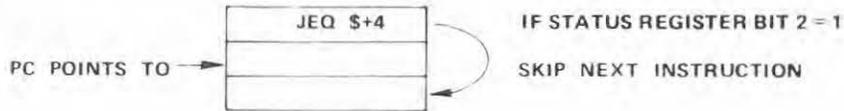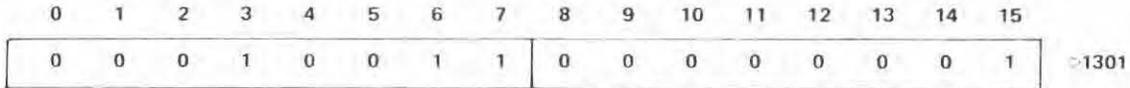
The above instruction continues execution 4 bytes (2 words) from the instruction location or, in other words, two bytes (one word) from the Program Counter value (incremented by 2 and now pointing to next instruction while JEQ executes). Thus, the signed displacement of 1 word (2 bytes) is the value to be added to the PC.

(2) ASSEMBLY LANGUAGE:
      JMP $ REMAIN AT THIS LOCATION

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	1	0	0	0	0	1	1	1	1	1	1	1	1	>10FF

PC −1 WORD →   JMP $ CONTINUOUS LOOP

PC POINTS TO →   TO JMP $ (>FF = −1 WORD)
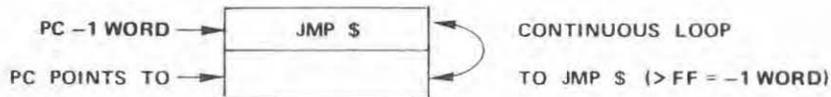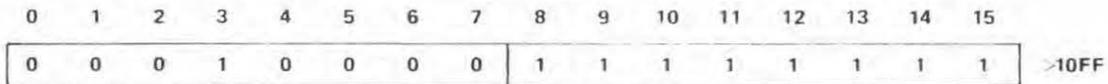
This causes an unconditional loop back to one word less than the Program Counter value (PC + FF = PC−1 word). The Status Register is not checked. A JMP $+2 means "go to the next instruction" and has a displacement of zero (a no-op). No-ops can substitute for deleted code or can be used for timing purposes.

4.6.2.2 CRU Single-Bit Instructions

These instructions test or set values at the Communications Register Unit (CRU). The CRU bit is selected by the CRU address in bits 3 to 14 of register 12 plus the signed displacement value. The selected bit is set to a one or zero, or it is tested and the bit value placed in equal bit (2) of the Status Register. The signed displacement has a value of −128 to 127.

NOTE

CRU addressing is discussed in detail in paragraph 5.5. CRU multibit instructions are defined in paragraph 4.6.4.

General Format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			OP CODE							SIGNED DISPLACEMENT					
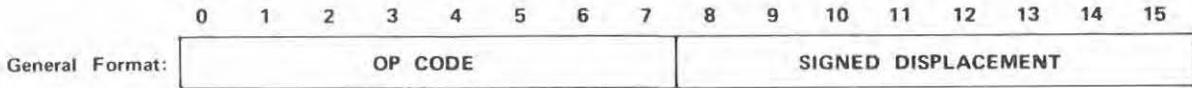
MNEMONIC	OP CODE 0 1 2 3 4 5 6 7	MEANING	STATUS BITS AFFECTED	DESCRIPTION
SBO	0 0 0 1 1 1 0 1	Set bit to one		Set the selected CRU output bit to 1.
SBZ	0 0 0 1 1 1 1 0	Set bit to zero		Set the selected CRU output bit to 0.
TB	0 0 0 1 1 1 1 1	Test bit	2	If the selected CRU input bit 1, set ST2.

EXAMPLE

R12, BITS 3 TO 14 = >100

ASSEMBLY LANGUAGE:

        SBO   4   SET CRU ADDRESS >104 TO ONE

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	1	1	0	1	0	0	0	0	0	1	0	0

>1D04
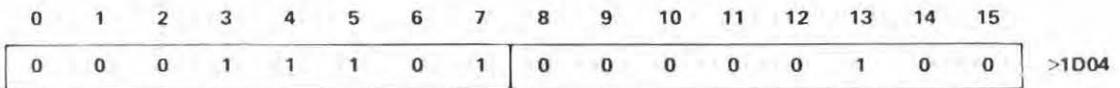
4.6.3 FORMAT 3/9 INSTRUCTIONS

These are dual operand instructions with multiple addressing modes for the source operand, and workspace register addressing for the destination. The MPY and DIV instructions are termed format 9 but both use the same format as format 3. The XOP instruction is covered in paragraph 4.6.9.

General Format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		OP CODE					DR (REGISTER ONLY)			T_S			SR		

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	Compare ones corresponding	No	2	Test (DR) 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 (DR) to determine if 0's are in each bit position where 1's are in (SA). If so, set ST2.
XOR	0 0 1 0 1 0	Exclusive OR	Yes	0-2	(DR) $\oplus$ (SA) → (DR)
MPY	0 0 1 1 1 0	Multiply	No		Multiply unsigned (DR) by unsigned (SA) and place unsigned 32-bit product in DR (most significant) and DR + 1 (least significant). If WR15 is DR, the next word in memory after 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 (DR), perform no operation and set ST4. Otherwise divide unsigned (DR) and (DR) by unsigned (SA). Quotient → (DR), remainder → (DR+1). If DR 15, the next word in memory after WR15 will be used for the remainder.

Exclusive OR Logic $1 \oplus 0 = 1$
                               $0 \oplus 0 = 0$
                               $1 \oplus 1 = 0$

(1) ASSEMBLY LANGUAGE:

      MPY   R2,R3   MULTIPLY CONTENTS OF R2 AND R3, RESULT IN R3 AND R4

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	0	38C2

	BEFORE	AFTER	
R2	0002	0002	
R3	0003	0000	} 32-BIT
R4	N	0006	} RESULT

The destination operand is always a register, and the values multiplied are 16-bits, unsigned. The 32-bit result is placed in the destination register and destination register +1, zero filled on the left.

(2) ASSEMBLY LANGUAGE:

      DIV   @>FE00,R5   DIVIDE CONTENTS OF R5 AND R6 BY VALUE AT M.A. >FE00

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	1	1	1	0	1	0	1	1	0	0	0	0	0	>3D60
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	>FE00

	BEFORE	AFTER	
M.A. >FE00	0005	0005	
R5	0000	0003	
R6	0011	0002	← REMAINDER

The unsigned 32-bit value in the destination register and destination register +1 is divided by the source operand value. The result is placed in the destination register. The remaindewr is placed in the destination register +1.

(3) ASSEMBLY LANGUAGE:
    COC   R10,R11   ONES IN R10 ALSO IN R11?

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	0	0	0	1	0	1	1	0	0	1	0	1	0	>22CA

Locate all binary ones in the source operand. If the destination operand also has ones in these positions, set the equal flag in the Status Register; otherwise, reset this flag. The following sets the equal flag:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R10	1	0	1	0	1	0	1	0	0	0	0	0	1	1	0	0	>AA0C
R11	1	1	1	0	1	1	1	1	1	1	0	0	1	1	0	1	>EFCD

Set EQ bit in Status Register to 1.

4.6.4   FORMAT 4 (CRU MULTIBIT) INSTRUCTIONS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
General Format:			OP CODE					C			T_S			SR		

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 are not affected. Ts and SR 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 (even number) address. If the source is addressed in the workspace register indirect autoincrement mode, the workspace register is incremented by 1 if C = 1 through 8, and is incremented by 2 otherwise.

NOTE

CRU addressing is discussed in detail in paragraph 5.5. CRU single bit instructions are defined in paragraph 4.6.2.2.

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 $1 \leqslant C \leqslant 8$.

EXAMPLE

ASSEMBLY LANGUAGE:
    LDCR   @>FE00,8   LOAD 8 BITS ON CRU FROM M.A. >FE00

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	1	0	0	1	0	0	0	1	0	0	0	0	0	>3220
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	>FE00

NOTE
CRU addressing is discussed in detail in paragraph 5.5.

4.6.5 FORMAT 5 (SHIFT) INSTRUCTIONS

These instructions shift (left, right, or circular) the bit patterns in a workspace register. The last bit value shifted out is placed in the carry bit (3) of the Status Register. If the SLA instruction causes a one to be shifted into the sign bit, the ST overflow bit (4) is set. The C field contains the number of bits to shift.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
General Format:	OP CODE								C				R			

If C = 0, bits 12 through 15 of R0 contain the shift count. If C = 0 and bits 12 through 15 of WR0 = 0, the shift count is 16.

MNEMONIC	OP CODE								MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
	0	1	2	3	4	5	6	7				
SLA	0	0	0	0	1	0	1	0	Shift left arithmetic	Yes	0-4	Shift (R) left. Fill vacated bit positions with 0.
SRA	0	0	0	0	1	0	0	0	Shift right arithmetic	Yes	0-3	Shift (R) right. Fill vacated bit positions with original MSB of (R).
SRC	0	0	0	0	1	0	1	1	Shift right circular	Yes	0-3	Shift (R) right. Shift previous LSB into MSB.
SRL	0	0	0	0	1	0	0	1	Shift right logical	Yes	0-3	Shift (R) right. Fill vacated bit positions with 0's.

EXAMPLES

(1) ASSEMBLY LANGUAGE:
     SRA   R1,2   SHIFT R1 RIGHT 2 POSITIONS, CARRY SIGN

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	>0841

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R1 BEFORE	1	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	>8F0F
R1 AFTER	1	1	1	0	0	0	1	1	1	1	0	0	0	0	1	1	>E3C3

SIGN BIT CARRIED IN
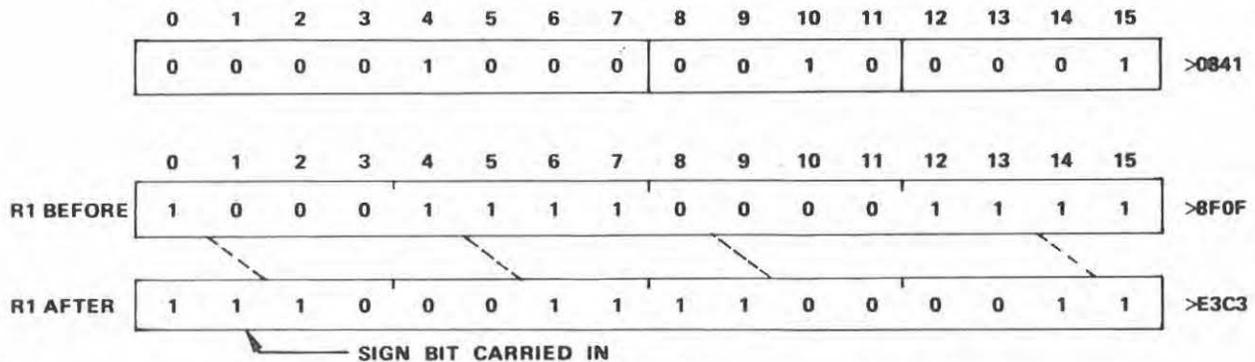
(2) ASSEMBLY LANGUAGE:
     SRC   R5,4   CIRCULAR SHIFT R5 4 POSITIONS

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	1	>0845

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R5 BEFORE	0	0	0	0	1	0	0	1	0	0	0	0	1	1	1	1	>090F
R5 AFTER	1	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	>F090

SLA   R1,0   SHIFT COUNT IN R0

SHIFT COUNT

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	1	1	CCC3
R1 (BEFORE)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
R1 (AFTER)	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	

VACATED BITS ZERO FILLED
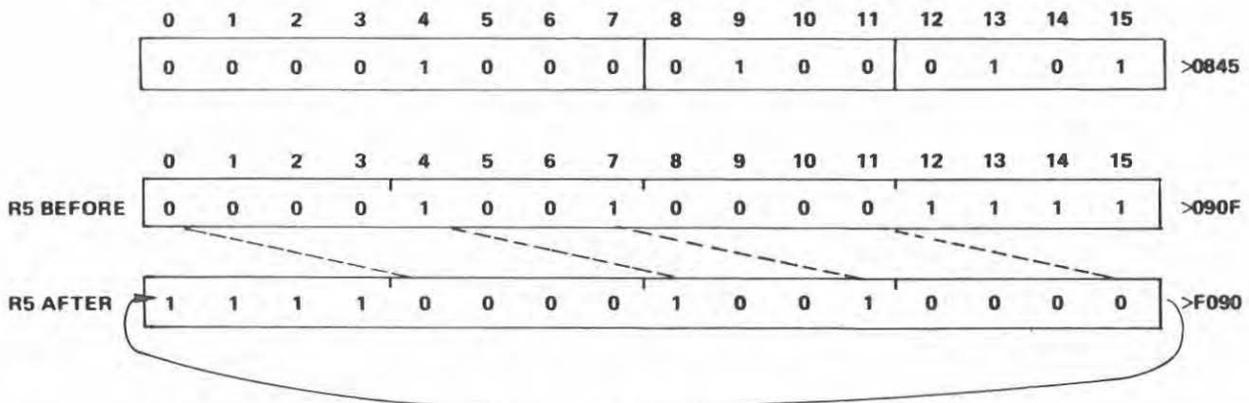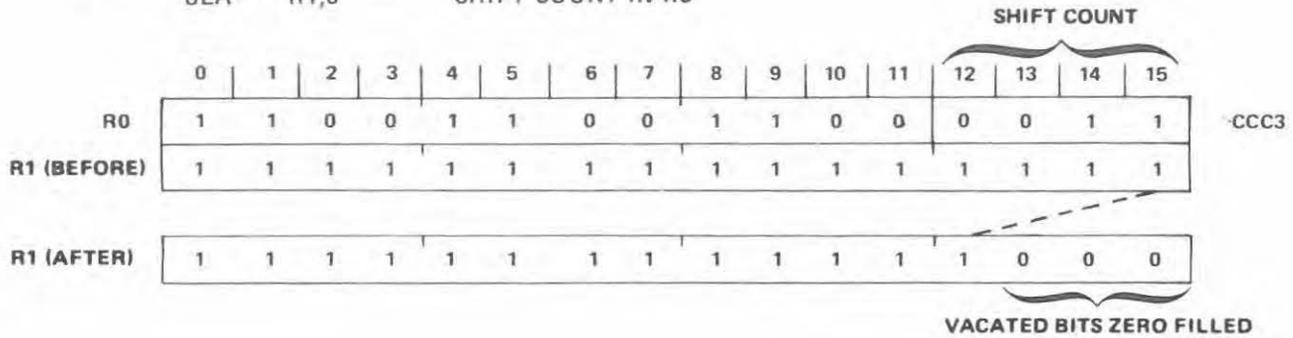
4.6.6 FORMAT 6 INSTRUCTIONS
These are single operand instructions.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
General Format:						OP CODE						T_S			SR	
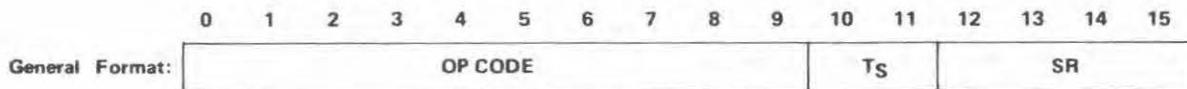
The T_S and S fields provide multiple mode addressing capability for the source operand.

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) → (R11); 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 (INTREQ) is not tested upon completion of the BLWP instruction.
CLR	0 0 0 0 0 1 0 0 1 1	Clear operand	No		0000 → (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	$\overline{(SA)}$ → (SA) (ONE'S complement)
NEG	0 0 0 0 0 1 0 1 0 0	Negate	Yes	0-4	— (SA) → (SA)(TWO'S complement)
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 compared to zero for setting the status bit (i.e., before execution).

†If additional memory words for the execute instruction are required to define the operands of the instruction located at SA, these words will be accessed from PC and the PC will be updated accordingly. The instruction acquisition signal (IAQ) will not be true when the TMS 9900 accesses the instruction at SA. Status bits are affected in the normal manner for the instruction executed.

NOTE

Jumps, branches, and XOP's are compared in Table 4-5.

(1) ASSEMBLY LANGUAGE:
    B   *R2   BRANCH TO M.A. IN R2

MACHINE LANGUAGE:

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	0	1	0	0	1	0	> 0452

R2   F D D O

       B   *R2   PC | F D D O | (AFTER)

M.A. >FDDO   NEXT INSTR.
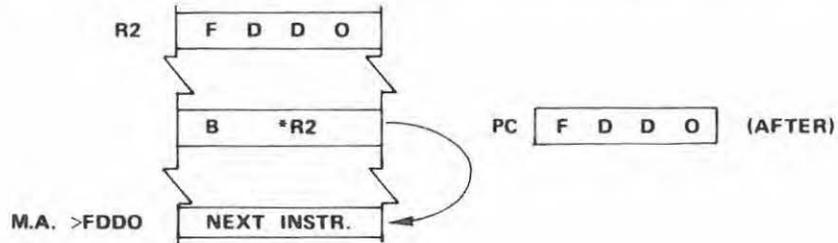
(2) ASSEMBLY LANGUAGE:
    BL   @>FF00   BRANCH TO M.A. >FF00, SAVE OLD PC VALUE (AFTER EXECUTION) IN R11

MACHINE LANGUAGE:

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	1	0	0	0	0	0	>04A0
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	>FF00

R11   F C 0 4 ←──── OLD PC VALUE

M.A. >FC00   BL @ >FF00   PC | F F 0 0 | (AFTER)

>FC02   F F 0 0

>FC04

>FF00   NEXT INSTR.

TO RETURN
EXECUTE
B *R11

     B *R11
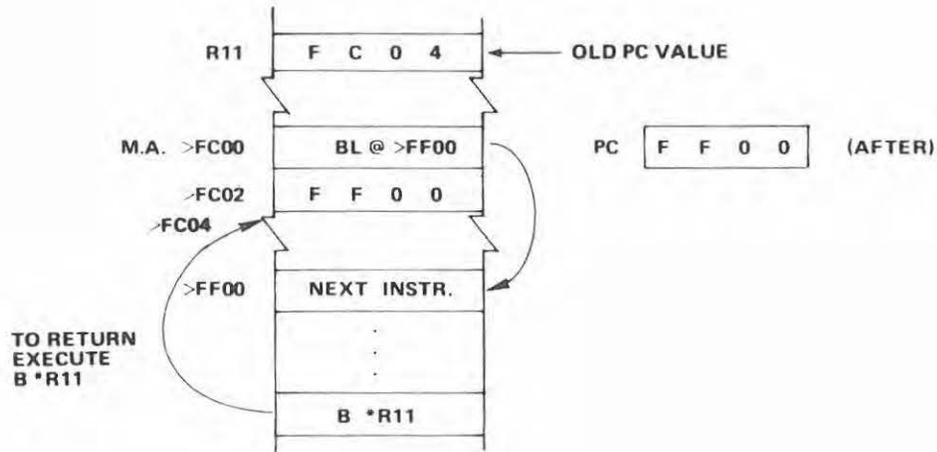
(3) ASSEMBLY LANGUAGE:
    BLWP   @>FD00   BRANCH, GET NEW WORKSPACE AREA

MACHINE LANGUAGE:

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	1	0	0	0	0	0	>0420
1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	>FD00

This context switch provides a new workspace register file and stores return values in the new workspace. See Figure 4-10 . The operand (>FD00 above) is the M.A. of a two-word transfer vector, the first word the new WP value, the second word the new PC value.

BLWP @>FD00   BRANCH WITH NEW WORKSPACE
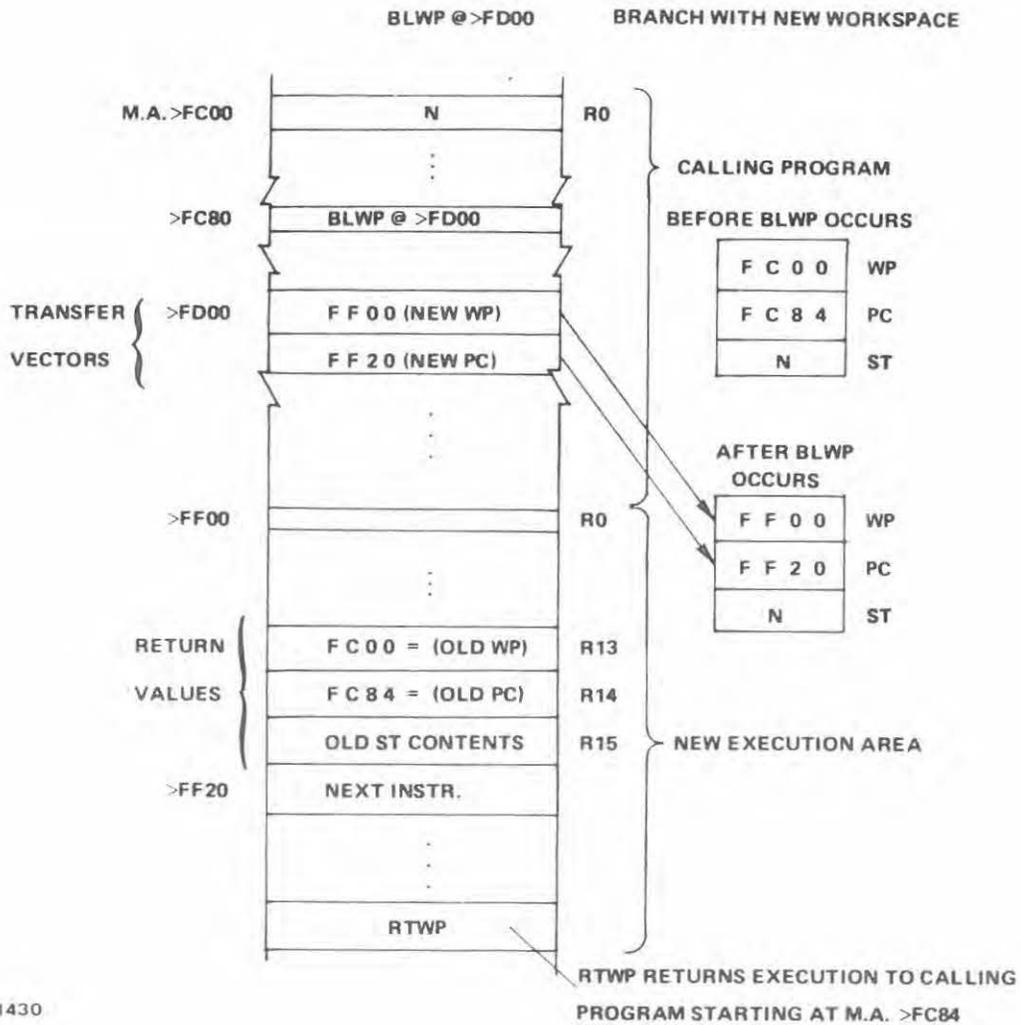
Figure 4-10.   BLWP Example

Essentially, the RTWP instruction is a return to the next instruction that follows the BLWP instruction (i.e., RTWP is a return from a BLWP context switch, similar to the B *R11 return from a BL instruction). BLWP provides the necessary values in registers 13, 14, and 15 (see Figure 4-10.

Table 4-5. Comparison of Jumps, Branches, XOP's

MNEMONIC	PARAGRAPH	DEFINITION SUMMARY
JMP	4.6.2	One-word instruction, destination restricted to +127, −128 words from Program Counter value.
B	4.6.6	Two-word instruction, branch to any memory location.
BL	4.6.6	Same as B with PC return address in R11.
BLWP	4.6.7	Same as B with new workspace; old WP, PC and ST contents (return vectors) are in new R13, R14, R15.
XOP	4.6.9	Same as BLWP with address of parameter (source operand) in new R11. Sixteen XOP vectors outside program in M.A. 40_{16} to $7E_{16}$; can be called by any program.

4.6.7 FORMAT 7 (RTWP, CONTROL) INSTRUCTIONS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
General Format:						OP CODE								N		
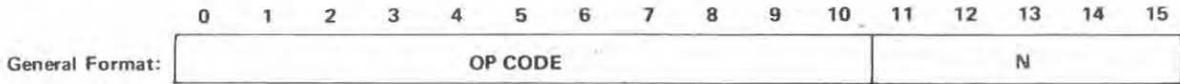
External instructions cause the three most-significant address lines (A0 through A2) to be set to the levels described in the table below and cause the CRUCLK line to be pulsed, allowing external control functions to be interpreted during CRUCLK at A0, A1, and A2. The RSET instruction resets the I/O lines on the TMS 9901 to input lines; the TMS 9902 is not affected. RSET also clears the interrupt mask in the Status Register. The LREX instruction causes a delayed load interrupt, delayed by two IAQ cycles after LREX execution. The load operation gives control to the monitor. Note, that although included here because of its format, the RTWP instruction is not classified as an external instruction because it does not affect the address lines or CRUCLK.

$\overline{CKOF}$ and $\overline{CKON}$ can be used by monitoring pins 9 and 10 respectively of U25. See sheet 2 of the schematics in Appendix F.

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 an interrupt, $\overline{LOAD}$, or $\overline{RESET}$ occurs	L H L
RSET	0 0 0 0 0 0 1 1 0 1 1	Reset I/O & SR	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	Load interrupt		Control to *TIBUG*	H H H
RTWP	0 0 0 0 0 0 1 1 1 0 0	Return from Subroutine	0–15	(R13) → (WP) (R14) → (PC) (R15) → (ST)	

*These outputs from the TMS 9900 go to a SN74LS138 as shown in Figure 5-6

RTWP   RETURN FROM CONTEXT SWITCH

MACHINE LANGUAGE:

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

RTWP   **RETURN TO PREVIOUS WP (R13), PC (R14), ST (R15) VALUES**

```
R13     F C 0 0
R14     F C 8 4          AFTER
R15     STATUS        F C 0 0      WP
                       F C 8 4      PC
                       STATUS       ST
M.A. >FF40   RTWP
```
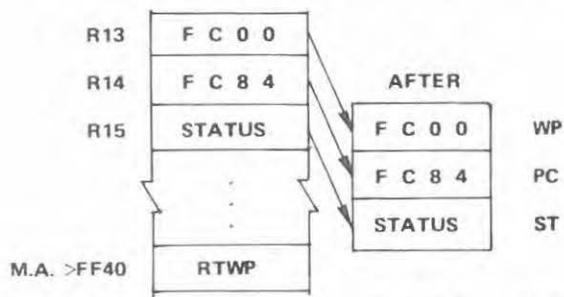
EXECUTION BEGINS AT M.A. >FC84
WITH R0 AT M.A. >FC00.

4.6.8 FORMAT 8 (IMMEDIATE, INTERNAL REGISTER LOAD/STORE) INSTRUCTIONS

4.6.8.1 Immediate Register Instructions

General format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
					OP CODE						N		R		
						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	(R) + IOP → (R)
ANDI	0 0 0 0 0 0 1 0 0 1 0	AND immediate	Yes	0-2	(R) AND IOP → (R)
CI	0 0 0 0 0 0 1 0 1 0 0	Compare immediate	Yes	0-2	Compare (R) 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 → (R)
ORI	0 0 0 0 0 0 1 0 0 1 1	OR immediate	Yes	0-2	(R) OR IOP → (R)

AND Logic: $0 \cdot 1, 1 \cdot 0 = 0$ OR Logic: $0 + 1, 1 + 0 = 1$
                 $0 \cdot 0 = 0$ $1 + 1 = 1$
                 $1 \cdot 1 = 1$ $0 + 0 = 0$

4.6.8.2 Internal Register Load Immediate Instructions

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

General format:
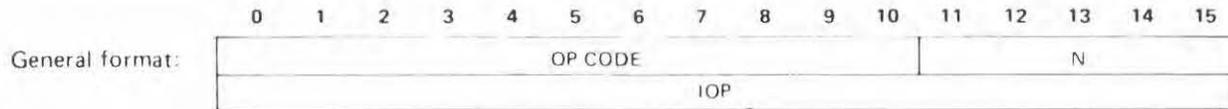
OP CODE	N
IOP	

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

4.6.8.3 Internal Register Store Instructions

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

General format:
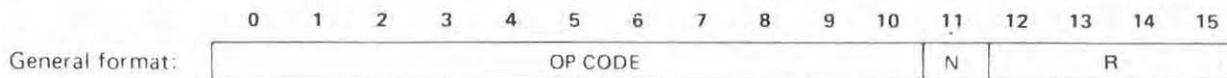
OP CODE	N	R

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) → (R)
STWP	0 0 0 0 0 0 1 0 1 0 1	Store workspace pointer	(WP) → (R)

EXAMPLES

(1) ASSEMBLY LANGUAGE:

AI   R2,>FF   ADD >FF TO CONTENTS OF R2

MACHINE LANGUAGE:

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	0	0	1	0	>0222
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	>00FF

	BEFORE		AFTER
R2	0 0 0 F		0 1 0 E

(2) ASSEMBLY LANGUAGE:

CI   R2,>10E   COMPARE R2 TO >10E

MACHINE LANGUAGE:

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	0	0	1	0	>0282
0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	0	>010E

R2 contains "after" results (>10E) of instruction in Example (1) above; thus the ST equal bit becomes set.

(3) *ASSEMBLY LANGUAGE:*
       LWPI   >FC00   WP SET AT >FC00 (M.A. OF R0)

MACHINE LANGUAGE:

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
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	>FC00

(4) *ASSEMBLY LANGUAGE:*
       STWP   R2   STORE WP CONTENTS IN R2

MACHINE LANGUAGE:

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	0	0	1	0	>02A2

This places the M.A. of R0 in a workspace register.

4.6.9 FORMAT 9 (XOP) INSTRUCTIONS
Other format 9 instructions (MPY, DIV) are explained in paragraph 4.6.3 (format 3).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
General Format:	0	0	1	0	1	1		D (XOP NUMBER)			T$_S$			SR		

The TS and SR fields provide multiple mode addressing capability for the source operand. When the XOP is executed, ST6 is set and the following transfers occur:

$(40_{16} + 4D) \rightarrow (WP)$	First vector at 40_{16}
$(42_{16} + 4D) \rightarrow (PC)$	Each vector uses 4 bytes (2 words)
$SA \rightarrow$ (new R11)	
(old WP) $\rightarrow$ (new WR13)	
(old PC) $\rightarrow$ (new WR14)	
(old ST) $\rightarrow$ (new WR15)	

The TMS 9900 does not test interrupt request ($\overline{\text{INTREQ}}$) upon completion of the XOP instruction.

An XOP is a means of calling one of 16 subtasks available for use by any executing task. The EPROM memory area between M.A. 40_{16} and $7E_{16}$ is reserved for the transfer vectors of XOP's 0 to 15 (see Figure 4-1). Each XOP vector consists of two words, the first a WP value, the second a PC value, defining the workspace pointer and entry point for a new subtask. These values are placed in their respective hardware registers when the XOP is executed.

The old WP, PC, and ST values (of the XOP calling task) are stored (like the BLWP instruction) in the new workspace, registers 13, 14, and 15. Return to the calling routine is through the RTWP instruction. Also stored, in the new R11, is the M.A. of the source operand. This allows passing a parameter to the new subtask, such as the memory address of a string of values to be processed by the XOP-called routine. Figure 4-11 depicts calling an XOP to process a table of data; the data begins at M.A. $FF00_{16}$. This XOP example uses XOP vectors that point directly to the XOP service routine WP and PC. The TM 990/101M comes with interrupt and XOP vectors pointing to linking areas that point to the service routine. The use of these linking areas is explained in subsection 5.9.

XOP's 0, 1 and 8 to 15 are used by the TIBUG monitor, calling software routines (supervisor calls) as requested by tasks. This user-accessible software performs tasks such as write to terminal, convert binary to hex ASCII, etc. These monitor XOP's are discussed in Section 3.3. XOP vectors 2 through 7 are programmed with memory vector values, but reserved for the user. See Section 5.9 for an explanation of the Interrupt/XOP linking area.

ASSEMBLY LANGUAGE:
    XOP   @>FF00,4

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	0	1	1	0	1	0	0	1	0	0	0	0	0	>2D20
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	>FF00
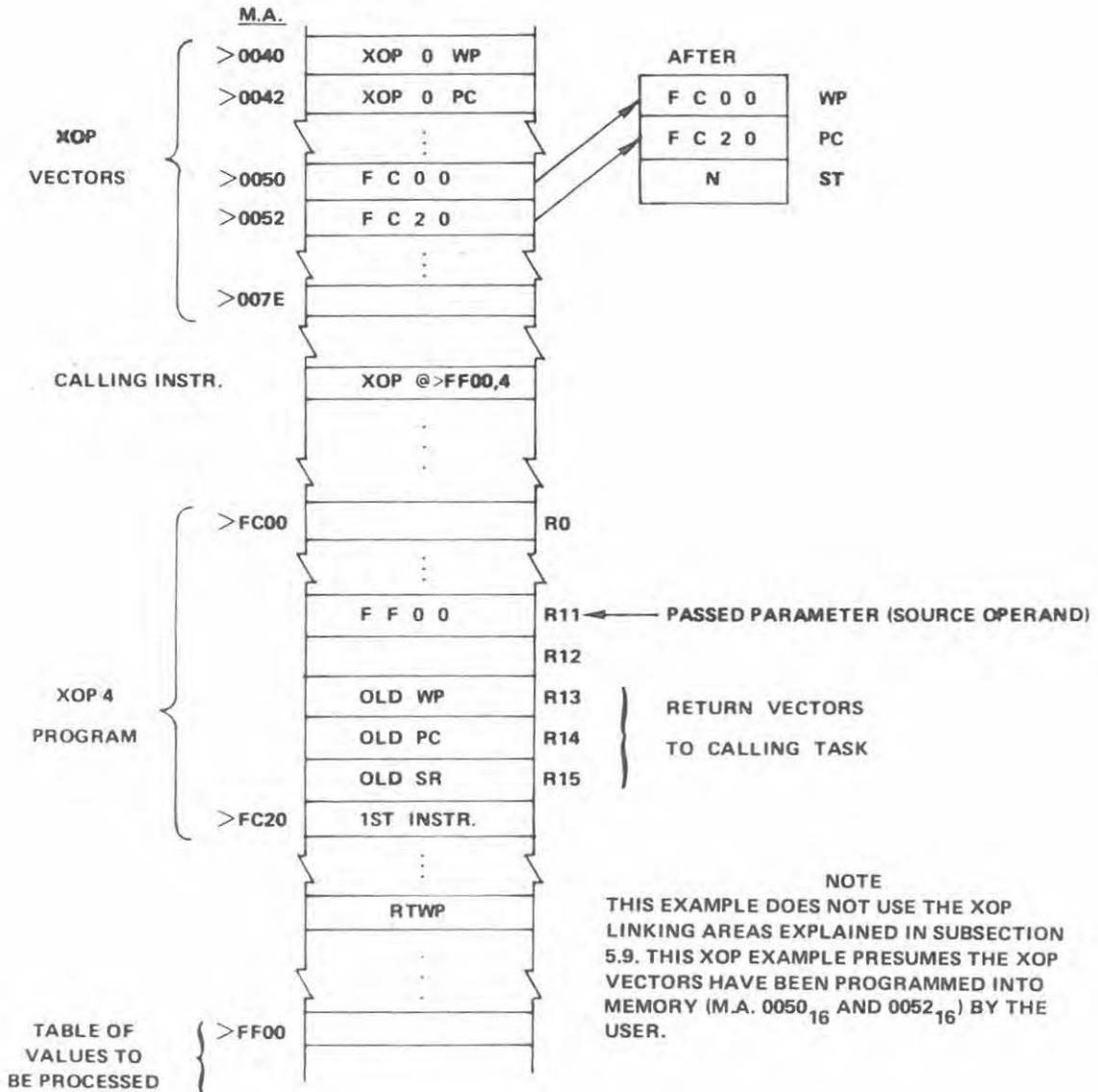
Figure 4-11.   XOP Example

A0001431

4-35

PROGRAMMING

5.1 GENERAL

This section is designed to familiarize the user with programming the TM 990/101M. Explanations about the programming environment, using TIBUG XOP's, supporting special features of the hardware, and certain programming practices are included. Programs are provided as examples for the user to analyze and follow, and possibly combine into the user's system. This section is divided into, roughly, two areas: the first part gives background information on the programming environment and shows suggested coding practices for a variety of situations, and the second part gives specific program examples using special features of the hardware.

For clarity, source listing examples in this section use assembler directives recognized by larger assemblers but not recognized by the TM 990/402 Line-By-Line Assembler (LBLA). These directives are not explained in the section on the 990 instruction set (Section 4), but are explained in detail in the Model 990 Computer, TMS 9900 Microprocessor Assembly Language Programmer's Guide. A synopsis of their definitions is included here. These directives are explained in Table 5-1.

Table 5-1. Assembler Directives Used In Examples

Label	Opcode	Operand	Meaning
	AORG	XXXX	Assemble code that follows so that it is loaded beginning at M.A. XXXX. This is similar to the absolute load (slash) request of the LBLA.
	DATA	YYYY	Place the value YYYY in this location (if preceded by the greater-than sign ( ) the quantity is a hexadecimal representation).
	DATA	LABEL	If LABEL represents a memory address, the memory address value is placed at this location aligned on an even address (word boundary).
	END		Signifies end of program for assembler.
AAAA	EQU	BBBB	Wherever the symbol AAAA is found, substitute the value BBBB.
	IDT	'NAME'	Program will be identified by NAME.
	TEXT	'ABCD123'	The ASCII value of the specified character string is assembled in successive bytes.
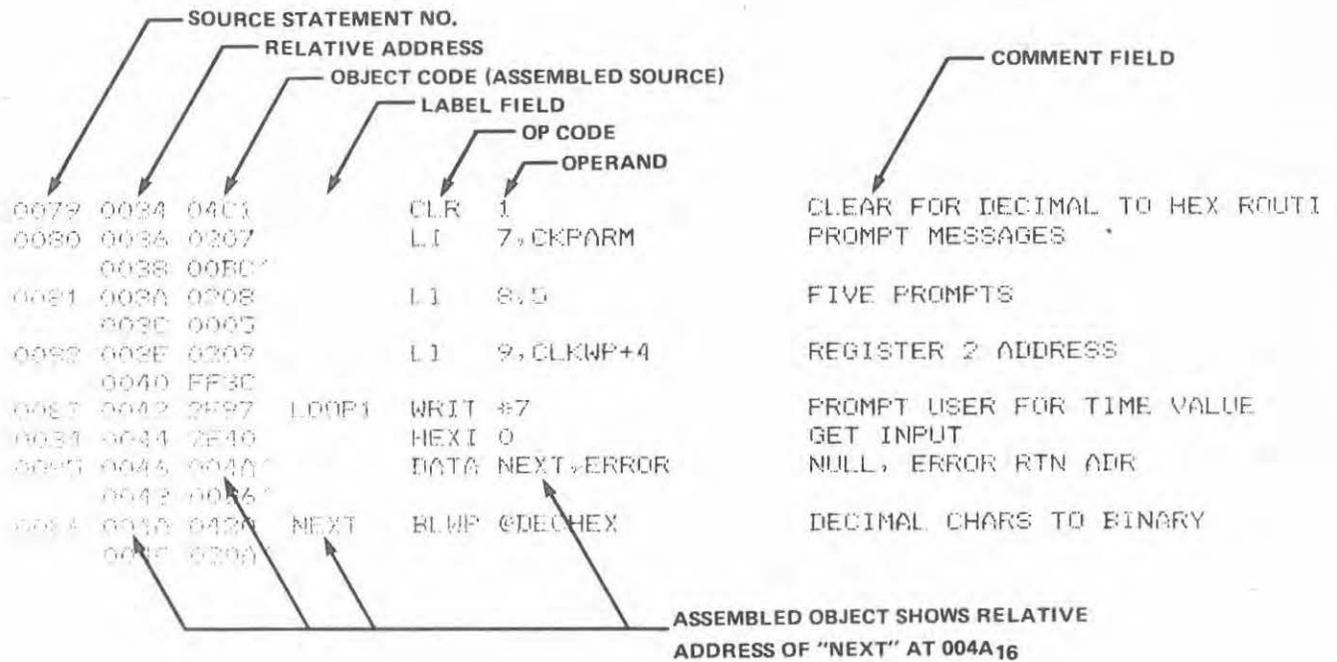
Figure 5-1. Source Listing

Figure 5-1 is part of a source listing used in this section, as assembled by TI's TXMIRA assembler. Unless specified otherwise by directive, the TXMIRA assembler will begin assembling code relative to memory address 0000_{16} (second column). When resolving an address for an instruction, as shown at the bottom of the figure, the instruction address operator is the same as the relative address in column two of the listing. Thus, for the label NEXT, the address $004A_{16}$ is assembled which is the relative address within the listing. This is useful when determining such addresses as the destination of a labelled BLWP instruction. Note that the Line-By-Line Assembler does not use labelled addressing, but assembles the absolute address given.

5.2 PROGRAMMING CONSIDERATIONS

5.2.1 PROGRAM ORGANIZATION

Programs should be organized into two major areas:

- Procedure area of executable code and data constants (never modified)

- Data area of program data and work areas whose contents will be modified.

The executable code and constant data section can be debugged as a separate entity, and then programmed into EPROM. The work area can be placed at any other address in RAM, and that address does not have to be contiguous with the program code area, and can even be dynamically allocated by a Get Memory supervisor call of some kind. Even if the program parts are loaded and executed together, the organization and debug ease are enhanced.

In this programming section, all example programs are coded, with one exception, in this manner. The only work area is the register set, which is arbitrarily fixed to a RAM address. The one exception, the Two-Terminal routine, is coded to reside entirely in RAM because the workspace is a part of the contiguous extent of code. This method of coding is used in RAM-intensive systems because the operating system need not manage workspaces as might be necessary in a system with very little RAM.

5.2.2 EXECUTING TM 990/100M PROGRAMS ON THE TM 990/101M

Programs developed on the TM 990/100M board use a different interrupt and XOP trap configuration than the TM 990/101M. This must be taken into consideration when executing programs on the TM 990/101M that were developed for running on TM 990/100M. On the TM 990/100M, interrupt vectors are programmed into PROM for INT3 and INT4 (vectors $FF68_{16}$ and $FF88_{16}$ for INT3 and $FFAC_{16}$ and $FFAC_{16}$ for INT4). This allows immediate use of these interrupt traps such as with the TMS 9901 and TMS 9902 interval timers. XOP vectors on the TM 990/100M are programmed for XOP's 0, 1 and 8 to 15 for use by TIBUG. User XOP's (XOP 2-7) are not programmed.

On the TM 990/101M board, however, all interrupt and XOP vectors are programmed, and the linking scheme in RAM is different. Consult the interrupt linking section (paragraph 5.9) for the scheme used. The TM 990/100M scheme is described in the User's Guide for that microcomputer.

5.2.3 REQUIRED USE OF RAM IN PROGRAMS

All memory locations that will be written to must be in RAM-type memory (this is important to consider when the program is to be programmed into ROM). Areas to be located in RAM include all registers as well as the destination operands of format 1 instructions and the source operands of most format 6 instructions.

For example, in the following source lines:

```
MOV     @>0700,@>FC00      MOVE DATA
CLR     @ >FC00            CLEAR MEMORY ADDRESS
ABS     @ >FC00            SET TO ABSOLUTE VALUE
INCT    @ >FC00            INCREMENT BY TWO
S       R1,@ >FC00         (>FC00) - R1, ANSWER IN >FC00
```

the address $FC00_{16}$ will be written to; thus, it has to be in RAM.

5.3 PROGRAMMING ENVIRONMENT

The programming environment of a computer is loosely defined as the set of conditions imposed on a programmer by either or both the hardware and systems software, but it is also the facilities available to the programmer because of the design of the hardware and software. The environment in which a program resides usually determines how that program is coded. This section gives explanations of the major areas of the TM 990/101M design from a programmer's point of view. Note all program examples given are for a full assembler (e.g., PXRASM, TXMIRA, or SDSMAC vs. the Line-By-Line Assembler) so that labels can be used for reader comprehension.

5.3.1 HARDWARE REGISTERS

The TMS 9900 family of processors are designed around a memory-to-memory architecture philosophy; consequently, the only hardware registers inside the processor affecting the programmer are the Workspace Pointer (WP) register, the Program Counter (PC) register and the Status (ST) register. There are no accumulators or general purpose registers which reside physically inside the microprocessor. All manipulations of data are accomplished by using these three registers as described below.

5.3.1.1 Workspace Pointer (WP) Register

The Workspace Pointer is a register which holds the address of a sixteen word area in memory; this memory area serves as a general purpose register set. A memory area is designated as a workspace or general purpose register set by loading the address of the first word (register 0) of the 16-word space into the WP register. Thus the programmer's register set is in memory, and can be referred to with register addressing, or if the WP value is known, with memory addressing. The registers are simply a data area in a program with the special privileges usually given to processor registers. This approach has several advantages for the programmer.

1. Register save areas need no longer be kept in programs, since the actual program registers are already in memory, and are maintained by the hardware during program linking by the use of a special class of instructions.

2. Program debugging is greatly heightened since the registers of a questionable program remain intact in memory during debugging. The debug monitor has its own set of registers, in memory, and there is no question of which of many program modules has tampered with the processor registers, since each program in question can have its own registers.

3. Recursive, re-entrant, and ROM resident code is much easier to write since program calls are handled by special instructions, and new workspace areas, linked together by the hardware, are available for use at each program call.

4. Linked-list structuring of workspaces is automatically done by the hardware, reducing system software overhead.

5. Very fast interrupt handling is possible since only three processor registers (WP, PC, ST) are stored by the hardware during the interrupt (instead of a whole register set) usually by a software instruction or routine.

5.3.1.2 Program Counter (PC) Register

The Program Counter (PC) register holds the address of the next instruction to be executed by the processor. As such, it is no different than the PC in any other processor and is incremented while fetching instructions unless modified by a program branch or jump, or during an interrupt sequence.

5.3.1.3 Status (ST) Register

The Status Register holds the processor status and is the only one of the three processor registers which has nothing to do with memory, directly. It is divided into two parts: the status bits, which are set to reflect the attributes of data being handled by the processor, and the interrupt mask, which governs the priority structure of interrupt processing. The ST is organized as shown in Figure 4-2.

5.3.2 ADDRESS SPACE

The TMS 9900 microprocessor addresses 65,536 (64K) bytes (8-bits each). Although the data bus is 16 bits wide, and the instruction set is mainly word (16-bits) oriented, the basic unit of address is a byte. The actual memory architecture is 32,768 (32K) words of two bytes each, and byte processing is accomplished within the processor after fetching a word from memory. Because the instruction set is mainly arithmetically oriented, and usually operates on 16-bit words, it is probably best to view the address space as a collection of words, each containing, usually for I/O purposes, two bytes.

> NOTE
> This subsection covers the interrupt and XOP environments
> in general; programming of interrupts and XOPs is covered
> in detail in subsection 5.9.

5.3.3 VECTORS (INTERRUPT AND XOP)

Interrupt and XOP vectors are located beginning with address 0000_{16} and extend through $007F_{16}$. The first half, addresses 0000_{16} through $003F_{16}$, contain the interrupt vectors. There are 16 prioritized interrupts. Level 0 is the highest priority, with a vector pair at 0000_{16} and 0002_{16}. Level 15 is the lowest priority, with its vector pair at $003C_{16}$ and $003E_{16}$. Level 0 interrupt is synonomous with the RESET function. A vector pair consists of a workspace pointer and a program counter, both values identifying the interrupt program environment.

Before an interrupt can occur, the processor must recognize it as having an equal or higher priority than the interrupt mask in the Status Register. After a valid interrupt has occurred the interrupt vector values are retrieved from memory, and the hardware equivalent of a BLWP instruction takes place.

There is one additional vector pair, at $FFFC_{16}$ and $FFFE_{16}$, for the $\overline{LOAD}$ function. When signaled, this interrupt always occurs and cannot be disabled by the Status Register interrupt mask. Note also that RESET being level zero, cannot be disabled, since its Status Register priority value of zero is always equal to or higher than any value in the interrupt mask field.

The XOP vectors work in a similar manner. Vector location begins at 0040_{16} and extend through $007F_{16}$. These vectors are triggered by execution of the XOP instruction, with a number from 0 to 15. There is no prioritizing; these are software-triggered interrupts, and XOP service routines may freely execute other XOP's. One additional event happens during the vector action: the source operand of the XOP instruction is evaluated as an address and placed in the new Workspace Register 11. This provides a parameter to the XOP routine.

The TIBUG monitor uses several XOP's for I/O service from the terminal; some of these are available for the user as explained in subsection 3.3. In addition, the programmer may wish to program interrupt and XOP vectors for special functions.

5.3.4 WORKSPACE REGISTERS

The actual workspace registers, in memory, provide general working areas for a program. Some registers can also be used for special purposes; these are listed in Table 5-2.

Table 5-2. Register Reserved Application

Register	Application
0:	Bits 12-15 (least significant half-byte) provide the shift count for shift instructions coded to refer to this register. This register cannot be used for indexed addressing.
11:	Holds return address following execution of a BL instruction. During XOP service routine, it holds resolved memory address of argument in XOP instruction.
12:	CRU Base Address.
13:	During BLWP, RTWP, interrupts, and XOP's: holds old WP contents.
14:	During BLWP, RTWP, interrupts, and XOP's: holds old PC contents.
15:	During BLWP, RTWP, interrupts, and XOP's: holds old ST contents.

In general, then, registers 1 to 10 are available for unrestricted use, although the programmer can use the reserved registers for other purposes, if proper consideration is given.

One advantage of the workspace concept is that one program can request an almost unlimited number of register sets, or, alternatively, every little module in a program system can have at least one set of its own registers. Programs are usually written to take advantage of the benefits associated with program operands in registers.

5.4 LINKING INSTRUCTIONS

These are of vital interest to a programmer for they answer the all important question of how to get in and out of a program. These instructions are:
- B (paragraph 5.4.1)   Branch
- BL (paragraph 5.4.2)   Branch with return link in R11
- BLWP (paragraph 5.4.3)   Branch, new workspace, return link in R13 to R15
- RTWP (paragraph 5.4.4)   Return, use vectors in R13 and R14
- XOP (paragraph 5.4.5)   Branch, new workspace, vectors in low memory

5.4.1 BRANCH INSTRUCTION (B)

Though not normally considered a program linking instruction, the branch instruction can be used to link to programs in a known location, such as TIBUG. Since the Workspace Pointer is not affected by the instruction, program systems using this convention usually delegate the responsibility for establishing workspaces to each program. Thus we may have branches to various programs as shown in Figure 5-2. Note that each program sets up its own WP (LWPI instruction). The AORG and EQU directives are explained in paragraph 5.1.
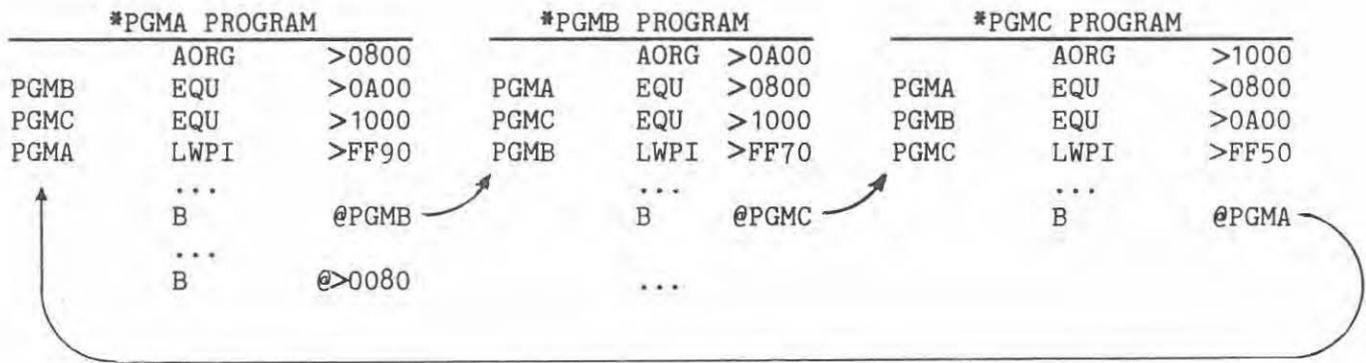
```
      *PGMA PROGRAM              *PGMB PROGRAM              *PGMC PROGRAM
          AORG   >0800              AORG  >0A00                 AORG   >1000
PGMB      EQU    >0A00     PGMA     EQU   >0800        PGMA     EQU    >0800
PGMC      EQU    >1000     PGMC     EQU   >1000        PGMB     EQU    >0A00
PGMA      LWPI   >FF90     PGMB     LWPI  >FF70        PGMC     LWPI   >FF50
          ...                       ...                        ...
          B      @PGMB              B     @PGMC                 B      @PGMA
          ...
          B      @>0080             ...
```

Figure 5-2. Example of Separate Programs Joined By Branches to Absolute Addresses

5.4.2 BRANCH AND LINK (BL)

The BL instruction is designed mainly for the calling of subprograms with a convenient means of returning back to the calling program. Since the processor puts the address of the next instruction in register 11 (it effectively transfers the PC to R11) before branching, the return path is established. To return (using the same workspace) simply execute a B *R11 (or RT instruction).

Note, though, that only one level of subroutine call is possible if only one workspace area is used, unless register 11 is saved by the first subroutine wishing to branch and link to a second subroutine.

```
CALLING PROGRAM          FIRST LINK               SECOND LINK
BL    @FE00        FE00     LI    R6,47     FD00      CI      R5,22
          .                  .                         .
          .                  .                         .
          .                  .                         .
                           MOV   R11,R10              B       *R11
                           BL    @>FD00
                             .
                             .
                             .
                        B    *R10
```
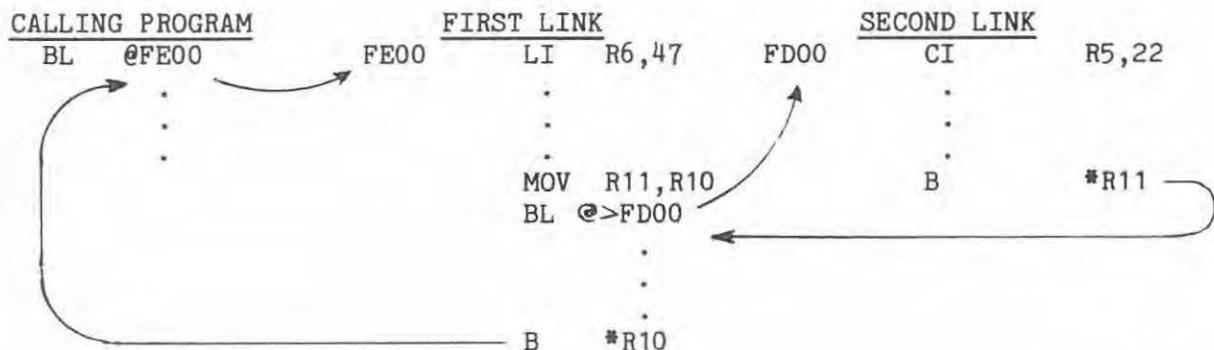
The BL subroutine can include XOP instructions to provide special services needed to accomplish the subroutine function, as in the following example:

```
CALLING PROGRAM                          SUBROUTINE
                     RDNUM     XOP     R1,13        READ A CHARACTER
     BL @RDNUM                 CI      R1,>3000     IS IT BELOW A ZERO?
        .                      JL      RDNUM        YES, GO BACK
        .                      CI      R1,>3900     IS IT ABOVE A NINE?
        .                      JH      RDNUM        YES, GO BACK
                               XOP     R1,12        ECHO THE CHARACTER
                               B       *11          RETURN
```

The very simple routine shown above reads a character from the terminal and checks for a decimal digit 0-9. If the character is acceptable, it is 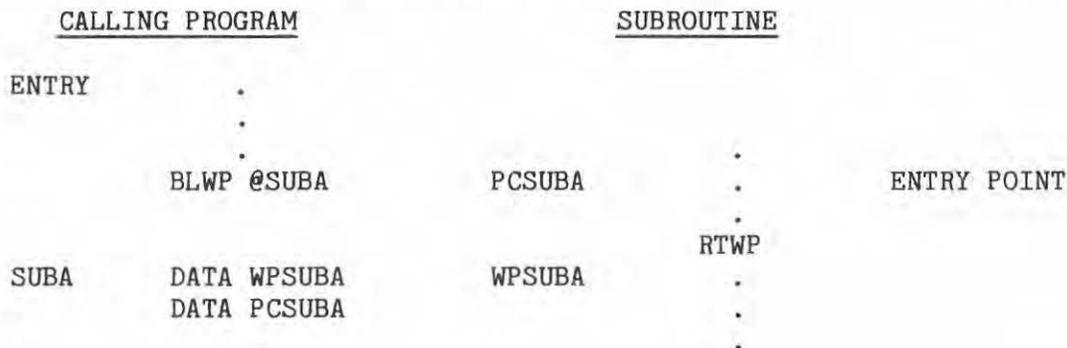echoed back to the terminal, and then control is returned to the calling program. If the character is unacceptable, the routine drops it and requests another; the bad character is not echoed to show the user that another character must be typed.

5.4.3 BRANCH AND LOAD WORKSPACE POINTER (BLWP)

This is the most sophisticated linking instruction in that it causes a complete program environment change (context switch), and automatically links the old workspace to the new, also preserving the old processor status. As such, it behaves in the same way as the interrupt sequence or XOP sequence, and it is therefore possible to vector to an interrupt or XOP service routine without actually causing an interrupt or executing an XOP. For example, executing a BLWP @0 will vector to the RESET interrupt handler, which if TIBUG is resident, causes the user to set the baud rate and start TIBUG again.

Since the TMS 9900 is a linked-list rather than a stack machine, those used to a stack for systems programming may need some readjustment, but the superior flexibility of linked-lists is simplified by the fact that the programmer can move nodes around, whereas in a stack, the nodes are fixed in Last-In First-Out (LIFO) order. The transition is made painlessly since the hardware completes program linking with the execution of one instruction, and very little effort is required on the part of the programmer.

There are two immediate possibilities to discuss in using the BLWP instruction. For simple subroutine linking, the following is an example:

```
        CALLING PROGRAM                         SUBROUTINE

ENTRY           .
                .
                .
        BLWP @SUBA          PCSUBA          .          ENTRY POINT
                                            .
                                            RTWP
SUBA    DATA WPSUBA         WPSUBA          .
        DATA PCSUBA                         .
                                            .
```

Note the double word vector pointed to by the BLWP operand, the values WPSUBA and PCSUBA. These two DATA statements provide the memory addresses of these vectors. The latter (PCSUBA) is the entry point, and is well defined. However, the WP value is shown here without a definition. This raises a fundamental question: if there are many programs operating together, such as TIBUG, possibly a user-written monitor, and a collection of application programs and subroutines, who is responsible for managing the workspaces? If each individual program is responsible, then the following definition would be added to the above subroutine:

```
WPSUBA    EQU                     >FF70
```

Note this defines WPSUBA as M.A. $FF70_{16}$ and ties down one area of memory to the subroutine; thus, no other program in the system can call this subroutine without chancing some conflict by using the same workspace. Thus, it is reserved for one subroutine.

A second approach is to code a value which is designated as a common workspace for whoever is in control at the time. In the EQU statement above, the value could be, by agreement, the common workspace. This implies that there are now two entities - the reserved workspace, which must be carefully mapped out ahead of time so there is no overlap, and the common workspace, of which there may be one or more, and whose status is such that any program can use it, but if control leaves that program, then that workspace is no longer considered needed, and thus can be used by another program.

Note the previous discussion assumes that the program code is in EPROM. If the code is to be executed from RAM, then writing the program is simple; put the workspace at the end of the program as a data area.

In either case, the user is responsible for partitioning his memory such that workspaces do not overlap or interfere with TIBUG or the XOP's defined by TIBUG, along with any user defined workspaces.

5.4.4 RETURN WITH WORKSPACE POINTER (RTWP)

The RTWP instruction can be used to both return from a program, and to link to a program. Since the instruction reloads the processor WP, PC, ST registers from workspace registers 13, 14, and 15, then the contents of these registers governs where control will go. If those registers were initialized by a BLWP instruction, then the action can be seen as a return, but if special values are placed in these registers, the action can be viewed as a subroutine call. Actually, program calls are not limited to a nesting structure, as in stack architectures, but are generalized so that chains and even rings may be formed. The TIBUG monitor uses the RTWP instruction in this manner. Using the "R" command, the user fills TIBUG's registers 13, 14, and 15. Using the "E" command causes TIBUG to execute a RTWP instruction using the values in these registers.

Since the RTWP does not affect the new workspace at all, there is no way for the called program to return to the caller unless the caller had initialized the new workspace registers before executing the RTWP. This type of program transfer is thus in a "forward" direction only, and is usually suitable only for a monitor program in a fixed location such as TIBUG.

5.4.5 EXTENDED OPERATION (XOP)

The XOP instruction works almost like a BLWP instruction, except that the address containing the double-word vector area is between 0040_{16} and $007F_{16}$, and is selected by an argument of from 0 to 15, and that the new workspace register 11 is initialized with the fully resolved address of the first operand of the XOP instruction. This means that if the operand is a register, the actual memory address is computed and placed in the new register 11.

The XOP instruction is meant as a "supervisor call" or special function operation. As such, a programmer might wish to implement routines which perform some standard process such as a character string search or setting the system timer, on the next page.

```
CALLING PROGRAM                         XOP TRAPS AND SUBROUTINE
                      *AT M.A.  0048:   FF90₃  TIMER ROUTINE WP   XOP 2
                      *AT M.A.  004A:   10AE₃  TIMER ROUTINE PC   VECTORS

LI    R0,11719        *AT M.A.  10AE:   IDT    'TIMER'
XOP   R0,2                      ENTRY   MOV    #11,11             GET VALUE
                                        LI     12,>0100           ADDRESS 9901
                                        SLA    11,1               SHIFT CLOCK COUNT
                                        ORI    11,1               SET CLOCK MODE
                                        LDCR   11,15              START CLOCK
                                        SBZ    0                  SET INTERRUPT MODE
                                        SBO    3                  ENABLE INT3 MASK
                                        RTWP
```

The main program requests 11719 clock counts, which is a desired time of 0.25 second.
This number is found by taking the system clock frequency, dividing it by 64 to find
the timer frequency, then reciprocating that to give the timer interval, then dividing
the desired time delay by the timer interval to find the clock counter value. It is
assumed here that XOP 2 is available for this function. The timer routine translates
the request and starts the system timer. One quarter second later, an interrupt
through INT3 will be generated.

TIBUG supplies definitions for XOPs 0, 1, and 8 through 15, leaving 2 through 7
available for the user. XOP's 2 through 7 are programmed according to a scheme
described in subsection 5.9.

5.4.6 LINKED-LISTS
A linked list is a data organization where a collection of related data, called a
node, contains information which links it to other nodes. The prime example here is a
workspace register set, it contains sixteen words of data. If there are many
workspaces present at one time connected by BLWP instructions, then every register 13
contains the address of the previous workspace, forming a linked list. At the same
time, the BLWP also places the previous program counter value in register 14,
providing a means of returning back to the previous program environment.

For example, the E or execute TIBUG command uses the RTWP instruction to begin program
execution at the WP, PC, and ST values in current registers 13, 14, and 15. The R or
register inspect/change TIBUG command can be used to set up these registers prior to
the execute command. In the example in Figure 5-3, program PGMA is executed using the
TIBUG E command; it later gives control to program PGMB using the BLWP command. In
doing so, the processor forges links back to PGMA by placing return WP, PC, and ST
values in registers 13, 14, and 15 of PGMB. Likewise, PGMB branches to PGMC with
return links to PGMB forged into R13 to R15 of PGMC. Each can return to the previous
program by executing an RTWP instruction, and the processor can travel up the linked
list until PGMA is reached again.

5.5 COMMUNICATIONS REGISTER UNIT (CRU)
Input and output is mainly done on the TM 990/101M using the Communications Register
Unit or CRU. This is a separate hardware structure with its own data and control
lines. Thus the TMS 9900 microprocessor has one address bus, but two sets of control
and data busses. One set, the memory set, has a 16-bit parallel bidirectional data bus
and three control lines, MEMEN, DBIN, and WE.

The other set the CRU I/O set, uses two lines, one line for input (CRUIN), and one for
output (CRUOUT). There is one control line, CRUCLK, used to strobe a bit being output
on CRUOUT. A bit being input on CRUIN has no strobe and is simply sampled by the
microprocessor at its discretion.

CRU devices are run on one phase of the system clocks, and therefore, the rate of data transfer on the CRUIN line is a function of the system clock. Since the CPU also uses this system clock, it will sample the CRUIN line at a rate that is a function of the system clock when doing a CRU read operation (executing a CRU read instruction - STCR or TB).
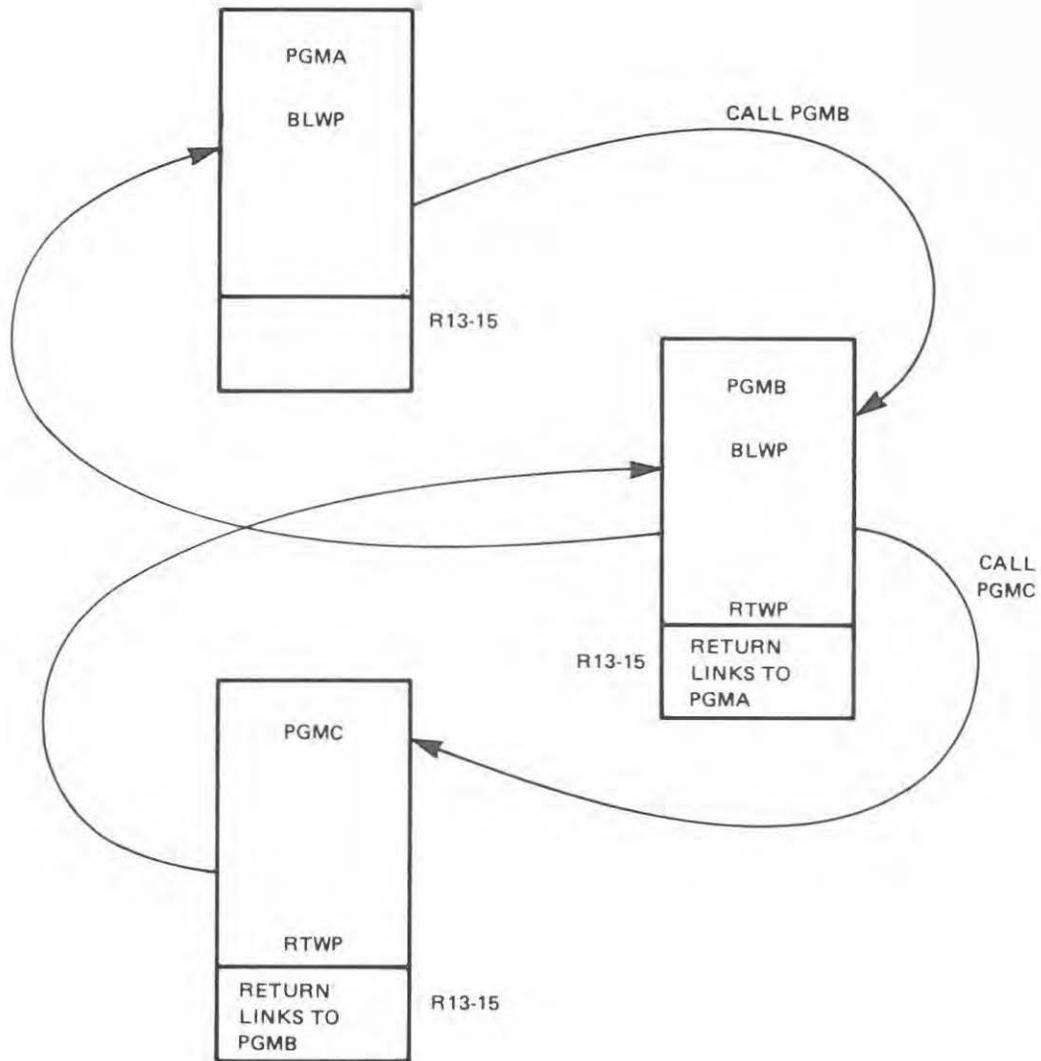

Figure 5-3. Linked List Example

Thus, the CRU data group consists of three lines - CRUIN, CRUOUT, and CRUCLK. The address bus supplies CRU addresses as well as memory addresses; which operation being performed is determined by the presence of the proper control signals. Memory operations use address bits 0 through 14 externally, bit 15 is used inside the microprocessor for byte operations. CRU operations, however, use only bits 3 through 14; bits 0, 1, and 2 are set to zero, and bit 15 of an address is totally ignored.

When CRU instructions are executed, data is written or read through the CRUOUT or CRUIN pins, respectively, of the TMS 9900 to or from designated devices addressed via the address bus of the microprocessor.

The CRU software base address is maintained in register 12 (bits 0 to 15) of the workspace register area. Only bits 3 to 14 of the register are interpreted by the CPU for the CRU address, and this 12-bit value is called the CRU hardware base address. When the displacement is added to the hardware base address, the result is the CRU bit address further explained in paragraph 5.5.1.

The CRU address is maintained in register 12 of the workspace register area. Only bits 3 through 14 of the register are interpreted by the CPU for the desired CRU address, and this 12-bit value is called the CRU bit address.

TM 990/101M devices driven off of the CRU interface include the TMS 9901 parallel interface and the TMS 9902 serial interface, which are accessed through the CRU addresses noted in Table 5-3. This table also lists the functions of the other CRU addresses which can be used for on-board or off-board I/O use. Addressing the TMS 9901 and TMS 9902 for use as interval timers is explained, along with programming examples, in subsections 5.9.3 and 5.9.4. Further detailed information on these two devices can be obtained from their respective data manuals.

Table 5-3.   TM 990/101M Predefined CRU Addresses

Function	CRU Hardware Base Address (R12, bits 3-14)	CRU Software Base Address (R12, bits 3-14)
Status L.E.D	0000	0000
Unit I.D. Switch	0020	0040
TMS 9902, Main I/O (Lower Half)	0040	0080
TMS 9902, Main I/O (Upper Half)	0050	00A0
TMS 9901 Interrupt Mask, System Timer	0080	0100
TMS 9901 Parallel I/O	0090	0120
RESET Interrupt 6	00A6	014C
TMS 9902, Auxiliary I/O (Lower Half)	00C0	0180
TMS 9902, Auxiliary I/O (Upper Half)	00D0	01A0
RS-232 Handshaking Signals	00E0	01C0
Offboard CRU	0100	0200

NOTES

1. Besides theexamples used herein, Appendix J contains examples of the various CRU instructions programmed to drive the on-board TMS 9901 or monitor signals to the TMS 9901.

2. The CRU software base address is equal to 2X the hardware base address, or the hardware base address is 1/2 the software base address.

5.5.1 CRU ADDRESSING

The CRU software base address is contained in the 16 bits of register 12.
From the CRU software base address, the processor is able to determine the CRU
hardware base address and the resulting CRU bit address. These concepts are
illustrated in Figure 5-4.

5.5.1.1 CRU Address
The CRU bit address is the address that will be placed on the address bus at
the beginning of a CRU instruction. This is the address bus value that,
when decoded by hardware attached to the address bus, will enable the device
so that it can be driven by the CRU I/O and clock lines. The CRU bit
address is the sum of the displacement value of the CRU instruction
(displacement applies to single-bit instructions TB, SBO, and SBZ only) and
the CRU hardware base address in bits 3 to 14 of R12. Note that the sign
bit of the eight-bit displacement is extended to the left and added as part of
the address. The resulting CRU hardware bit address is then placed on address
lines A3 to A14; address lines A0 to A3 will always be zeroes in CRU
instruction execution.

5.5.1.2 CRU Hardware Base Address
The CRU hardware base address is the value in bits 3 to 14 of R12. For
instructions that do not specify a displacement (LDCR and STCR do not), the
CRU hardware base address is the same as the first CRU bit address (see
above). An important aspect of the CRU hardware base address is that it does
not use the least significant bit of register 12 (bit 15); this bit is ignored
in deriving the CRU bit address.

5.5.1.3 CRU Software Base Address
The CRU software base address is the entire 16-bit contents of R12. In
essence, this is the CRU hardware base address divided by two. Bits 0, 1,
2, and 15 of the CRU software base address are ignored in deriving the CRU
hardware base address and the CRU bit address.
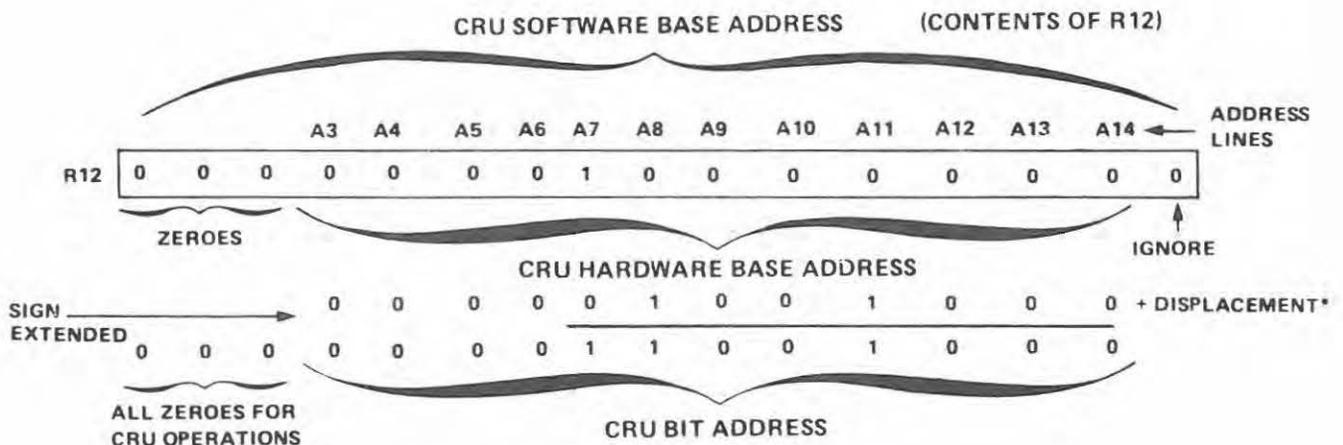
*The displacement added to the CRU hardware base address is a signed eight-bit value,
with sign extended, used only when executing one of the single-bit CRU instructions
(TB, SBO, and SBZ).

FIGURE 5-4. CRU BASE AND BIT ADDRESSES

Because bit 15 of R12 is not used, some confusion can result in programming. Instead of loading the CRU address in bits 0 to 15 of register 12 (e.g., LI R12,>80 to address the TMS 9901 at CRU address 80_{16}), the programmer must shift the base address value one bit to the left so that it is in bits 3 to 14 instead of in bits 4 to 15. Several programming methods can be used to ensure this correct placement, and all of the following examples place the TMS 9901 bit address of 80_{16} correctly in R12.

```
LI   R12,>100     PLACES >80 IN BITS 3 TO 14
  or
LI   R12,>80*2    MULTIPLY BASE ADDRESS BY 2 (NOT RECOGNIZED BY LINE-BY-
                  LINE ASSEMBLER)
  or
LI   R12,>80      BASE ADDRESS IN BITS 4 TO 15
SLA  R12,1        SHIFT BASE ADDRESS ONE BIT TO THE LEFT
```

From a programming standpoint, it may be best to view addressing of the CRU through the entire 16 bits of R12. In this context, blocks of a maximum of 16 CRU bits can be addressed, and in order to address an adjacent 16-bit block, a value of 0020_{16}, must be added or subtracted from R12. For example, with R12 containing 0000_{16}, CRU bits 0 to F_{16} can be addressed. By adding 0020_{16} to R12, CRU bits 10_{16} to $1F_{16}$ can be addresses, etc.

5.5.2 CRU Timing
CRU timing is shown in Figure 5-5. Timing phases (ϕ1 to ϕ4) are shown at the top of the figure. The CRU address is valid on the address bus beginning at the start of of ϕ2, and stays valid for eight timing phases (two clock cycles). At the start of the next ϕ2 phase, CRUCLK at the TMS 9900 goes high for two phases to provide timing for CRUOUT sampling. Note that for LDCR and STCR instructions, the address bus is incremented for each data bit to be output or input. For input operations, the address is placed on the address bus at the beginning of phase ϕ2, and the input is sampled between phases ϕ4 and ϕ1.

5.5.3 CRU Instructions
The five instructions that program the CRU interface are:

- LDCR   Place the CRU hardware base address on address lines A3 to A14. Load from memory a pattern of 1 to 16 bits and serially transmit this pattern through the CRUOUT pin of of the TMS 9900. Increment the address on A3 to A14 after each CRUOUT transmission.

- STCR   Place the CRU hardware base address on lines A3 to A14. Store into memory a pattern of 1 to 16 bits obtained serially at the CRUIN pin of the TMS 9900. Increment the address on A3 to A14 after each CRUIN sampling.

- SBO   Place the CRU hardware base address plus the instruction's signed displacement on address lines A3 to A14. Send a logical one through the CRUOUT pin of the TMS 9900.

- SBZ   Place the CRU hardware base address plus the instruction's signed displacement on address lines A3 to A14. Send a logical zero through the CRUOUT pin of the TMS 9900.

- TB   Place the CRU hardware base address plus the instruction's signed displacement on address lines A3 to A14. Sample the

CRUIN pin of the TMS 9900 and place the bit read into ST2, the
Equal Bit of the Status register.

NOTE

Examples of single- and multi-bit CRU instruction execution using
the TMS 9901 are presented graphically in Appendix J.

5.5.3.1 CRU Multibit Instruction

The two multibit instructions, LDCR and STCR, address the CRU devices by
placing bits 3 through 14 (hardware base address) of R12 on address lines A3
through A14. A0, A1, and A2 are set to zero for all CRU operations. The first
operand is the source field address and the second operand is the number of
bits in the operation.

If the length is coded as from 1 through 8 bits, only the left byte of the
source or receiving field takes part in the operation, and bits are shifted in
or out from the least significant bit of that left byte. Thus a LDCR R2, 1
outputs bit 7 of R2 to the CRU at the address derived from register 12. An
STCR R5,2 would receive two bits of data serially and insert them into bit 7
and then bit 6 of register 5. The CRU address lines are automatically
incremented to address each new CRU bit, until the required number of bits are
transferred. In an STCR instruction, unused bits of the byte or word are
zeroed. In this last example, bits 0-5 are zeroed, the right byte is
unaffected.
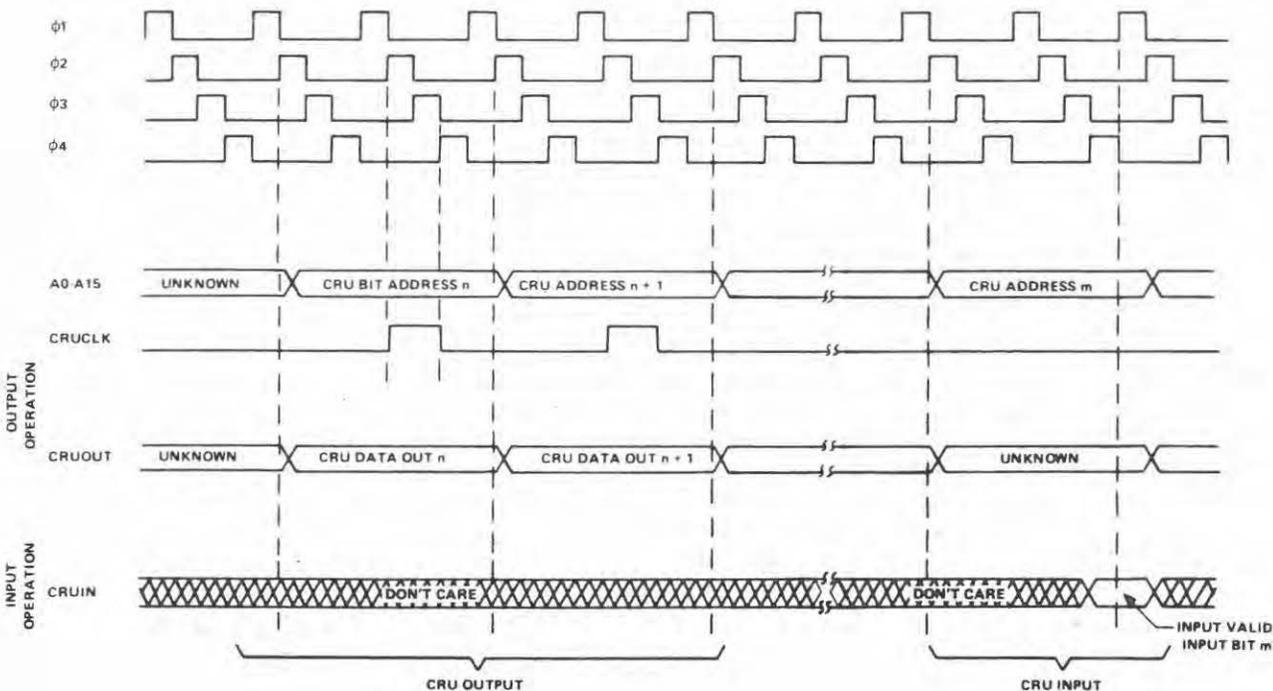
FIGURE 5-5. TMS 9900 CRU INTERFACE TIMING

An LDCR loads the CRU device serially from memory over CRUOUT timed by CRUCLK. An STCR stores data into memory obtained serially through CRUIN from the addressed CRU device. Figures 5-6 and 5-7 show this operation graphically. The TMS 9901 is used in the example as the CRU device because it most simply shows the bit transfers involved.
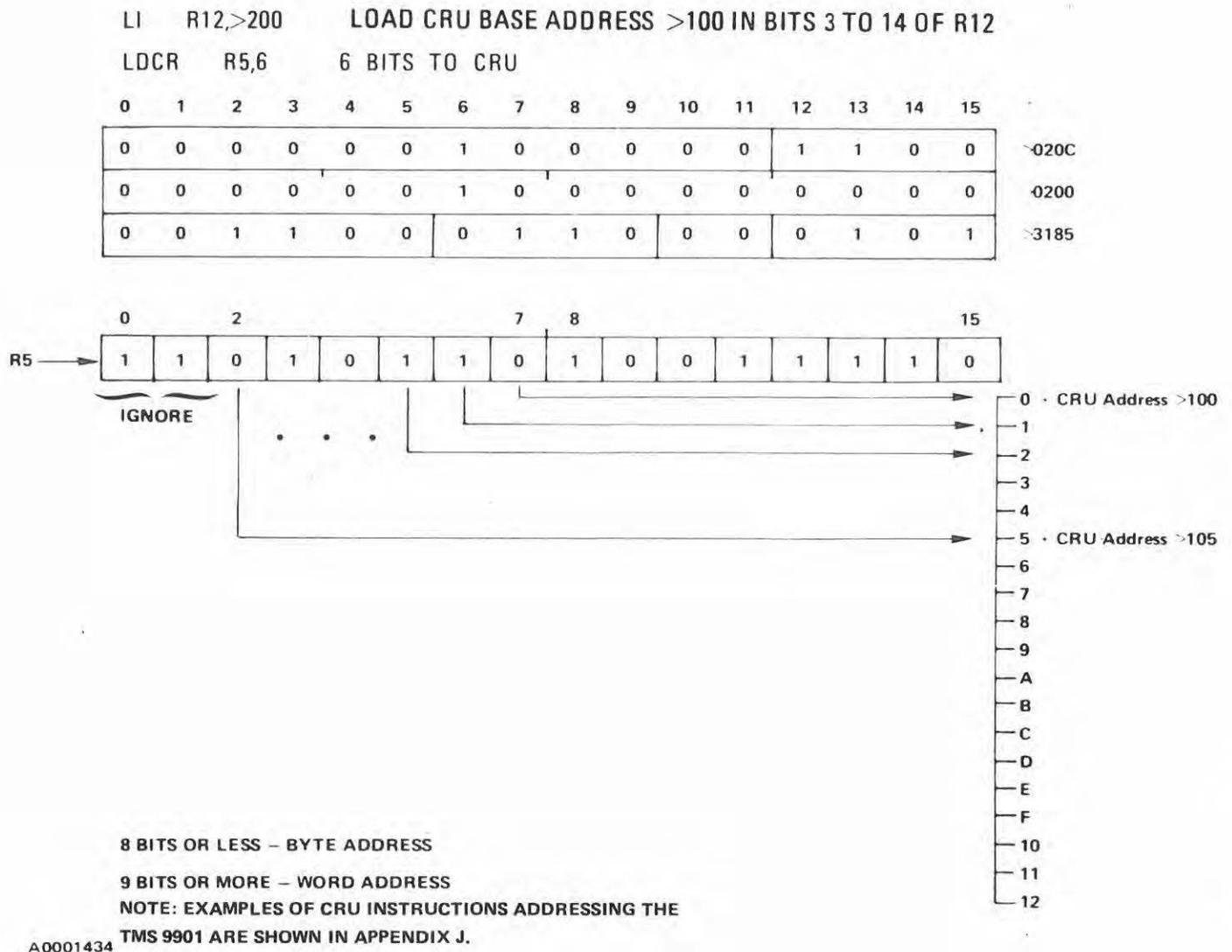
Figure 5-6. LDCR Instruction

LI R12,>120*2 LOAD CRU BASE ADDRESS >120 IN BITS 3 TO 14 OF R12

STCR R4,10   10 BITS FROM CRU TO R4

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	1	1	0	0	>020C
0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	>0240
0	0	1	1	0	1	1	0	1	0	0	0	0	1	0	0	>3684

R4 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | | |

ZERO FILL
UNUSED LEFT-SIDE BITS

0 ←CRU Address >120
1
2
3
4
5
6
7
8
9 ←CRU Address >129
A
B
C
D
E
F
10
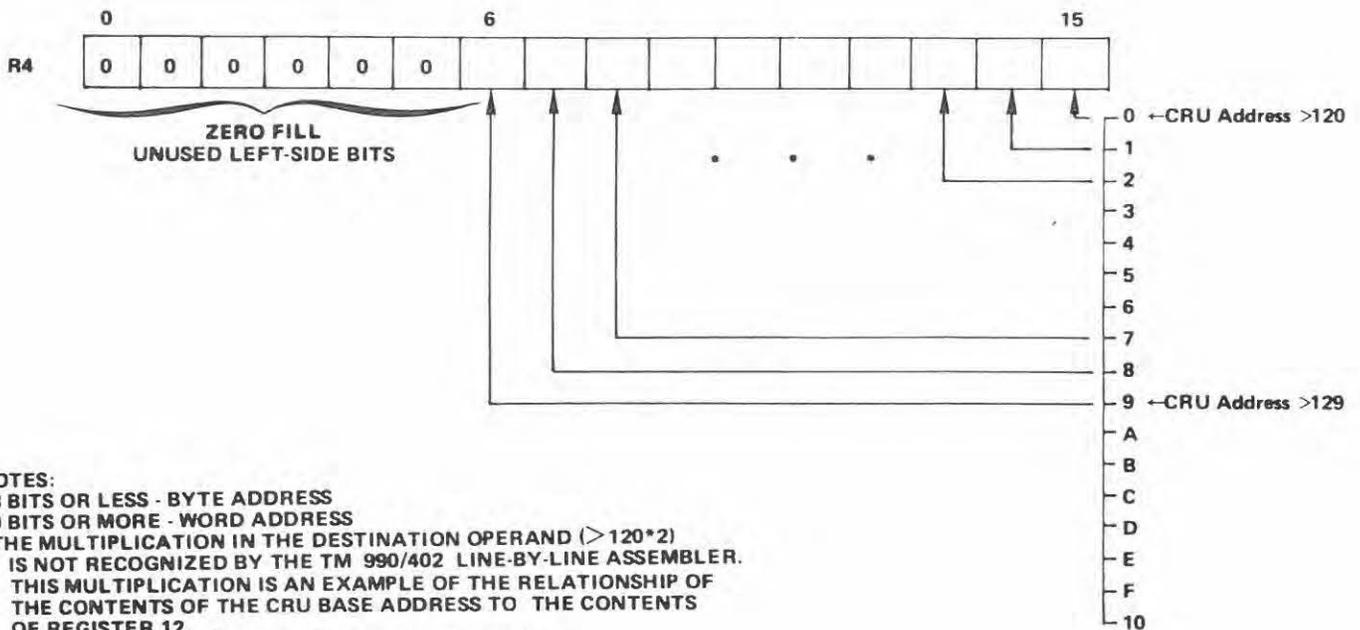
NOTES:
8 BITS OR LESS - BYTE ADDRESS
9 BITS OR MORE - WORD ADDRESS
THE MULTIPLICATION IN THE DESTINATION OPERAND (>120*2)
 IS NOT RECOGNIZED BY THE TM 990/402 LINE-BY-LINE ASSEMBLER.
 THIS MULTIPLICATION IS AN EXAMPLE OF THE RELATIONSHIP OF
 THE CONTENTS OF THE CRU BASE ADDRESS TO THE CONTENTS
 OF REGISTER 12.
EXAMPLES OF CRU INSTRUCTIONS ADDRESSING THE
TMS 9901 ARE SHOWN IN APPENDIX J.

Figure 5-7. STCR Instruction

5.5.3.2 CRU Single-Bit Instructions

The three single-bit instructions are SBZ (set bit to zero), SBO (set bit to one), and TB (test bit). The first two are output instructions, and the last one is an input instruction. All three instructions have only one operand, which is assembled into an eight-bit signed displacement to be added to the CRU hardware base address to provide the CRU bit address. The SBZ instruction sets the addressed bit to zero (zero on CRUOUT), and the SBO instruction sets the addressed bit to one (one on CRUOUT). The TB instruction reads the logical value on the CRUIN line and places this value in bit 2 (EQ) of the status register; the test can be proven by using the JEQ or JNE instructions.

The operand value is treated as a signed, eight-bit number, and thus has a range of values of -128 to +127. This number is added to the CRU hardware base address derived from bits 3 to 14 of R12, and the result is placed on the address lines. This process is illustrated in Figure 5-8.

Notice that after execution of a TB instruction, a JEQ instruction will cause a jump if the logic value on CRUIN was a one, and JNE will cause a jump if the logic value was a zero.
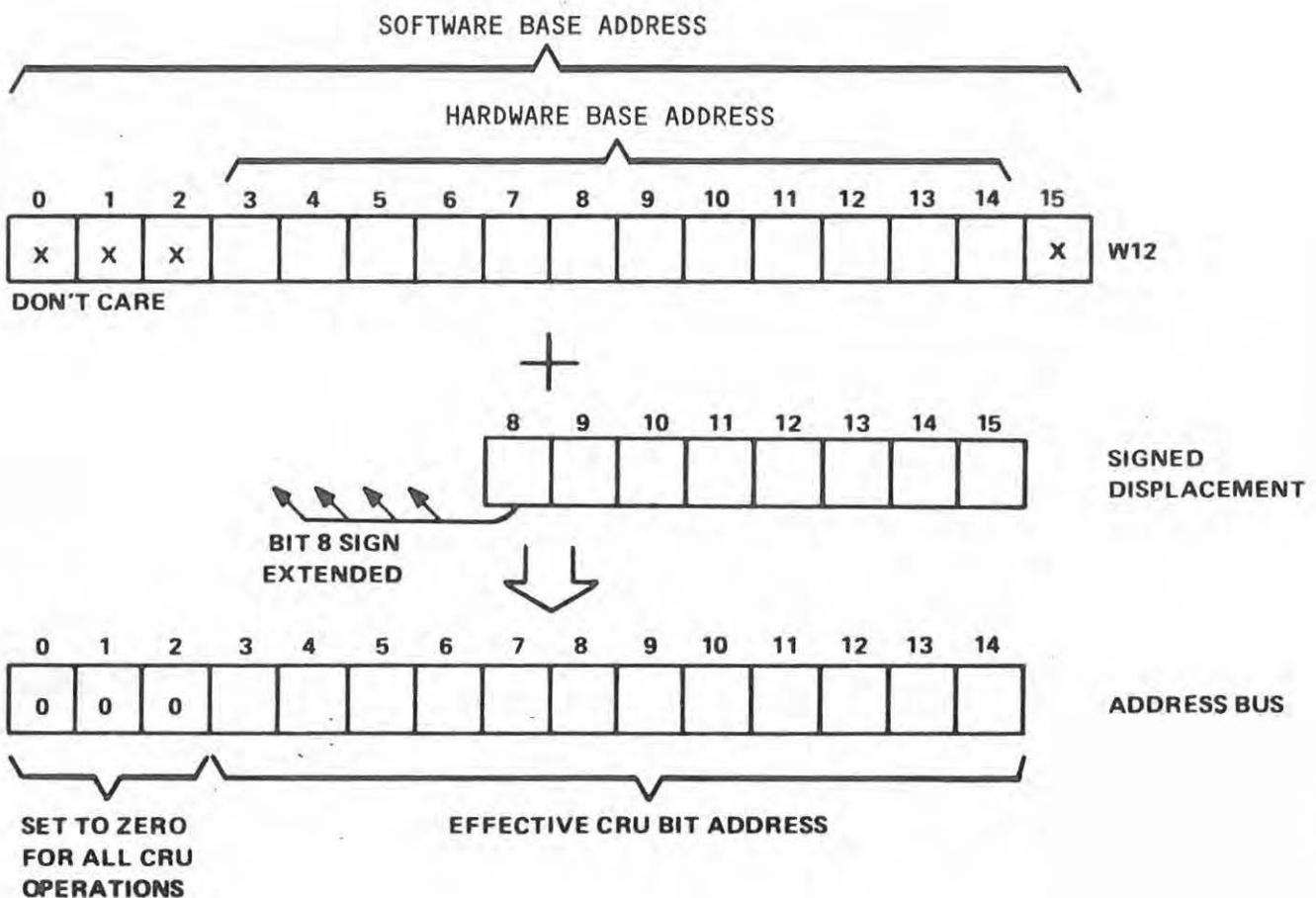

Figure 5-8.   Addition Of Displacement And R12 Contents To Drive CRU Bit Address

5.6 DYNAMICALLY RELOCATABLE CODE

Most programs written for the TM 990/101M will contain references in memory. These references are given by means of a symbolic name preceded by an at (@) sign. Examples are @>FE00 (M.A. FE00$_{16}$, recognized by the LBLA) or @SUM (recognized by a symbol-reading assembler, not the LBLA).

For example, a short program, located at M.A. 0900$_{16}$ to 090F$_{16}$, adds two memory addresses then branches to the monitor:

M.A.			
0900	MOV	@>090C,R1	MOVE VALUE AT M.A. 090C TO R1
0904	A	@>090E,R1	ADD VALUE AT M.A. 090E TO R1 (R1=ANSW)
0908	B	@>0080	RETURN TO MONITOR
090C	DATA	100	FIRST NUMBER
090E	DATA	200	SECOND NUMBER

In this program, a number in EPROM is moved to a register in RAM, and another number in EPROM is added to that register (the destination of an add must be in RAM in order for the sum to be written into it). If it is desired to move this entire program to another address (such as to RAM for debugging purposes to allow data changes as desired), then the locations in the code must be changed to reflect the new addresses. For example, to relocate the above example to start at address FC00$_{16}$, each of the addresses of the numbers must be changed before the program can execute; otherwise, the program will try to access numbers in M.A. 090C$_{16}$ and 090E$_{16}$ when they have been relocated to M.A. FC0C$_{16}$ and FC0E$_{16}$ respectively.

For a variety of reasons, it may be advantageous to have code that is "self-relocating," that is, it can be relocated anywhere in memory and execute correctly. Such "position-independent" or "dynamic-relocating" code is of great advantage when the code is programmed into EPROM. In this manner, the EPROMs can be installed in _any_ socket, responding to _any_ address, and the program will still execute correctly. Such programs are possible with the TM 990/101M by merely beginning the program with the code segment shown below (register 10 is used in the following examples). Thereafter, memory addresses can be indexed, relative to the beginning of the program (using R10 at the index register, in this case). This code is shown in Figure 5-9.

	M.A.		OPCODE/OPERANDS		COMMENTS
Base Reg. Setup	0000	START	LWPI	FE00	R0 AT M.A. FE00
	0004		LI	R10,START	LOOK AT START ADDR.
	0008		JEQ	RELOC	IF NOT BIASED, NEED RELOCATING
	000A		CLR	R10	LOADER HAS BIAS, CLEAR BASE REG.
	000C		JMP	STARTX	GO TO PROGRAM
	000E	RELOC	LI	R10,>045B	B *R11 OPCODE IN R10
	0012		BL	R10	PC VALUE TO R11
	0014	RELOCX	AI	R11,START-RELOCX	PC-10 = PROGRAM START
	0018		MOV	R11,R10	PROGRAM START TO R10
Relo-catable Program	001E	STARTX	MOV	@>001A(R10),R1	MOVE FIRST NUMBER TO R1
	0012		A	@>001C(R10),R2	ADD 2D NO. TO R1, ANSW IN R1
	0016		B	@>0080	RETURN TO MONITOR
	001A		DATA	100	FIRST NUMBER
	001C		DATA	200	SECOND NUMBER

Figure 5-9. Example Of Program With Coding Added to Make it Relocatable

This coding first sets up a program base register which computes the address of the beginning of the program. This is accomplished by:

- establishing the beginning workspace register address with LWPI

- placing the opcode for the instruction B *R11 in the designated index register address (R10 above)

- execute a branch and link to R10; this places the address of the next instruction following BL R10 into register 11; a branch to R10 means a return indirect through R11

- compute the beginning address of the program by subtracting 10_{16} from the address in register 11.

- move this beginning address to R10, allowing R11 to be further used as a linking register.

- Index all future relocatable addresses using R10.

There are several considerations. Absolute addresses (e.g., beginning of monitor at 0080_{16}) need not be indexed, and other types of memory indexing should consider the contents of the base register; it may be necessary to add the contents of the base register to another indexing register. Also, an immediate load of an address into a register will require that the base address in the index register be added to the register also. For example:

```
LI      R2,>0980      ADDRESS OF VALUES IN R2
A       R10,R2        ADD BASE ADDRESS
```

Figure 5-10 is an example of a program that searches a table of numbers for a value. The example is shown in both relocatable and in non-relocatable code, for comparison. Symbolic addressing is used.

```
*NON SELF-RELOCATING                          *SELF-RELOCATING
*NO BASE REGISTER USED                         *R10 IS BASE REGISTER
        LI   R3,TABLE   POINT TO TABLE                 LI   R3,TABLE      POINT TO TABLE
*                                                      A    R10,R3        ADD BASE REG.
*REMAINDER OF CODE NOT INDEXED                 *REMAINDER OF CODE INDEXED
        MOV  @COUNT,R2  GET COUNT                      MOV @COUNT(R10),R2 GET COUNT
SEARCH  C    R1,*R3+    (R1) IN TABLE?         SEARCH  C    R1,*R3+       (R1) IN TABLE?
        JEQ  FOUND      YES                            JEQ  FOUND         YES
        DEC  R2         NO, DEC COUNTER                DEC R2             NO, DEC COUNTER
        JNE  SEARCH     LOOK AGAIN                     JNE SEARCH         LOOK AGAIN
        .    .                                         .    .
        .    .                                         .    .
COUNT   DATA 6                                 COUNT   DATA 6
TABLE   DATA 12,15,59,62,73,92                 TABLE   DATA 12,15,59,62,73,92
```

Figure 5-10. Examples of Non Self-Relocating Code and
Self-Relocating Code

Great care must be taken with B, BL, and BLWP. If linking to other modules is needed, these modules must be part of a system which is linked together by the linker program (e.g., TXLINK on the FS990 system), and all modules must be coded as self-relocating.

When programming the EPROM's, the code must be loaded such that the address START has the value ZERO, i.e. The code must appear biased at location 0000_{16}.

5.7 PROGRAMMING HINTS
In any programming environment there are several ways to accomplish a task. Table 5-4 contains alternate coding practices; some have an advantage over conventional coding.

Table 5-4. Alternate Programming Conventions

PURPOSE	CONVENTIONAL CODE	ALTERNATE CODE	ALTERNATE CODE ADVANTAGE
Compare Register Contents to 0	CI RX,0	MOV RX,RX	Saves one word
Increment A Register by 4	INCT RX INCT RX	C *RX+,*RX+	Saves one word
Access old workspace registers		MOV @N(R13),R1	N is twice the number of the old register wanted

Swap two registers	MOV RX,RHOLD
	MOV RY,RX
	MOV RHOLD,RY
		XOR RX,RY
		XOR RY,RX
		XOR RX,RY
			Saves a register: "RHOLD" Not needed

Clear a register	CLR RX CLR RX	XOR RX,RX SUB RX,RX	(None) (None)

5.8 INTERFACING WITH TIBUG
The TIBUG monitor provides a starting point for the programmer to consider when looking for program examples. The monitor contains some basic user facilities, and the user will probably enter and exit programs through TIBUG.

5.8.1 PROGRAM ENTRY AND EXIT
To execute a program under TIBUG, use the "R" and "E" commands as explained in Section 3 of this manual.

Exit from a program to TIBUG can be through:

                               B @>0080

TIBUG will print the prompting question mark. Note that the power-up initialization routine is not entered; instead, control goes directly to TIBUG's command scanner.

5.8.2 I/O USING MONITOR XOP's

5.8.2.1 Character I/O
Four XOP's deal specifically with character I/O:

- Echo Character   XOP 11
- Write Character   XOP 12
- Read Character   XOP 13
- Write Message   XOP 14

The echo XOP (11) is a read character XOP (13) followed by a write character XOP (12). The following code reads in a character from a terminal. If an A or E is found, the character is writen back to the terminal and program execution continues; otherwise, the program loops back waiting for another keyboard entry.

```
GETCHR        XOP       R1,13         READ CHARACTER
              CI        R1,>4100      COMPARE R1 to ASCII "A"
              JEQ       OK            IF "A" FOUND JUMP
              CI        R1,>4500      COMPARE R1 TO ASCII "E"
              JEQ       OK            IF "E" FOUND, JUMP
              JMP       GETCHR        RETURN TO READ ANOTHER CHARACTER
OK            XOP       R1,12         WRITE CHARACTER AS ECHO
              . .                 . .
```

XOP 14 causes a string of characters to be written to the terminal. Characters are written until a byte of all zeroes is found.

XOP 13 reads one character and stores it into the left byte of a work; the right byte is zero filled. The previous coding example could also have been completed with the following:

```
              OK      XOP      R1,14
```

Instructions are written in uninterrupted form; thus, messages should be grouped in a block separated from the continuous executable code. Each message must be delimited by a byte of all zeroes:

```
        **MESSAGES
        CRLF      BYTE       >0D
        LF        BYTE       >0A,>00
        MSG1      TEXT       'BEGIN PGMA'
                  BYTE       0
        MSG2      TEXT       'END PGMA'
                  BYTE       0
        MSG3      TEXT       '# ERRORS (IN HEX):'
                  BYTE       0
        MSG4      TEXT       'ERROR EXP VALUE='
                  BYTE       0
        MSG5      TEXT       ',RCV VALUE='
                  BYTE       0
```

Note in the preceding example, that if it is desired to send a carriage return and a line feed, use the following: XOP @CRLF,14. But if only a line feed is wanted, use: XOP @LF,14.

5.8.2.2 Hexadecimal I/O
Three XOP's handle hexadecimal numbers.

- Write one hexadecimal character   XOP 8
- Read a four-digit hexadecimal word   XOP 9
- Write four hexadecimal characters   XOP 10

Using the message block in paragraph 5.8.2.1, an example code segment might be:

```
*ERROR ROUTINE
ERROR      XOP      @MSG4,14        START ERROR LINE
           XOP      R1,10           PRINT CORRECT EXPECTED VALUE
           XOP      @MSG5,14        MORE ERROR LINE
           XOP      R2,10           PRINT ERRORED RCV VALUE
           XOP      @CRLF,14        DO CARRIAGE RETURN/LINE FEED
           XOP      @LF,14          ONE MORE LF FOR DOUBLE SPACE
```

XOP 8 is actually called four times by XOP 10, after positioning the next digit to be written into the least significant four bits of the work register.

The following shows how to input values to a program by asking for inputs from the terminal.

```
GET        XOP      R4,9            CALL TO GET HEX # ROUTINE
           DATA     NULL,ERROR      NO INPUT/BAD INPUT ADDRESSES
OK         A        R3,R4           ADD OLD NUMBER IN
           JMP      XXX             CONTINUE PROGRAM
NULL       LI       R4,>3AF1        LOAD DEFAULT VALUE
           XOP      @DEFMSG,14      PRINT DEFAULT MESSAGE
           JMP      OK
ERROR      XOP      @ERRMSG,14      PRINT ERROR MSG
           JMP      GET             TRY AGAIN
           . .      . .
DEFMSG     TEXT     'DEFAULT USED'
           BYTE     0
ERRMSG     TEXT     'ERROR: USE 0-9, A-F ONLY'
           BYTE     o
```

Note that the XOP 9 routine stores only the last four digits typed before the termination character (delimiter) is typed. This means if a wrong number is entered, continue typing until four correct digits are entered; then type a delimiter (space, carriage return, or minus sign). Typing fewer than four digits total (but at least one digit) causes leading zeroes to be inserted. Typing only a delimiter gives control to the first address following the XOP, and typing an illegal character at any time causes control to go to the address specified in the second word following the XOP call.

5.9 INTERRUPTS AND XOPS

5.9.1 INTERRUPT AND XOP LINKING AREAS

When an interrupt or XOP instruction is executed, program control is passed to WP and PC vectors located in lower memory. Interrupt vectors are contained in M.A. 0000_{16} to $003F_{16}$; and XOP vectors are contained in M.A. 0040_{16} to $007F_{16}$. User-available interrupt and XOP vectors are preprogrammed in the EPROM chip with WP and PC values that allow the user to implement interrupt service routines (ISR's) and XOP service routines (XSR's). This includes programming an intermediate linking area as well as the ISR or XSR code.

When an interrupt or XOP is executed, it first passes control to the vectors which point to the linking area. The linking area directs execution to the actual ISR or XSR. The linking areas are shown in Table 5-5. The linking area is designed to leave as much space free as possible when not using all the interrupts. That is, the most frequently used areas are butted up against TIBUG area, the least frequently used areas extend downward into RAM.

Return from the ISR or XSR is through return vectors in R13, R14, and R15 at the ISR or XSR workspace and at the linking area workspace.

How to program these linking areas is explained in the following paragraphs.

NOTE
Interrupts 3 and 4 are used by the timers at the TMS 9901 and TMS 9902 respectively.

Table 5-4. Preprogrammed Interrupt And User XOP Trap Vectors

M.A.	Int.	VECTORS WP	VECTORS PC	M.A.	XOP	VECTORS WP	VECTORS PC
0000	INT0	TIBUG	TIBUG	0048	XOP2	FF48	FF5A
0004	INT1	FF5A	FF7A	004C	XOP3	FF3A	FF4C
0008	INT2	FF4E	FF6E	0050	XOP4	FF2C	FF3E
000C	INT3	FF8A	FFAA	0054	XOP5	FF1E	FF30
0010	INT4	FF7E	FF9E	0058	XOP6	FF10	FF22
0014	INT5	FF72	FF92	005C	XOP7	FF02	FF14
0018	INT6	FF66	FF86				
001A	INT7	FEEE	FF0E				
0020	INT8	FEE2	FF02				
0024	INT9	FED6	FEF6				
0028	INT10	FECA	FEEA				
002C	INT11	FEBE	FEDE				
0030	INT12	FEB2	FED2				
0034	INT13	FEA6	FEC6				
0038	INT14	FE9A	FEBA				
003C	INT15	FE8E	FEAE				

Table 5-5.   Interrupt and User XOP Linking Areas

M.A.	BYTE							
	0-1	2-3	4-5	6-7	8-9	A-B	C-D	E-F
↑ FE90	USER RAM AREA							
FEA0					INT15	INT15	INT15	INT15
FEB0	INT15	INT15	INT14	INT14	INT14	INT14	INT14	INT14
FEC0	INT13	INT13	INT13	INT13	INT13	INT13	INT12	INT12
FED0	INT12	INT12	INT12	INT12	INT11	INT11	INT11	INT11
FEE0	INT11	INT11	INT10	INT10	INT10	INT10	INT10	INT10
FEF0	INT9	INT9	INT9	INT9	INT9	INT9	INT8	INT8
FF00	INT8	INT8	INT8	INT8	INT7	INT7	INT7	INT7
FF10	INT7	INT7	XOP7	XOP7	XOP7	XOP7	XOP7	XOP7
FF20	XOP7	XOP6	XOP6	XOP6	XOP6	XOP6	XOP6	XOP6
FF30	XOP5	XOP5	XOP5	XOP5	XOP5	XOP5	XOP5	XOP4
FF40	XOP4	XOP4	XOP4	XOP4	XOP4	XOP4	XOP3	XOP3
FF50	XOP3	XOP3	XOP3	XOP3	XOP3	XOP2	XOP2	XOP2
FF60	XOP2	XOP2	XOP2	XOP2	INT2	INT2	INT2	INT2
FF70	INT2	INT2	INT1	INT1	INT1	INT1	INT1	INT1
FF80	INT6	INT6	INT6	INT6	INT6	INT6	INT5	INT5
FF90	INT5	INT5	INT5	INT5	INT4	INT4	INT4	INT4
FFA0	INT4	INT4	INT3	INT3	INT3	INT3	INT3	INT3
FFB0 FFFB	TIBUG WORKSPACE							

5.9.1.1 Interrupt Linking Areas

When one of the programmable interrupts (INT1 to INT15) is executed, it traps to an interrupt linking area in RAM. Each linking area consists of six words (12 bytes) as shown in Figures 5-11 and 5-12. The first three words contain the last three registers of the called interrupt vector workspace (R13, R14, and R15), and the second three words, located at the interrupt vector PC address, are intended to be programmed by the user to contain code for a BLWP instruction, a second word for the BLWP destination address, and an RTWP instruction code (all three words to be entered by the user). When the ISR is completed, control returns to this linking area where the return values (to the interrupted program) are loaded into the linking area's three registers (R13 to R15), then the BLWP instruction (at the PC vector address) is executed using the M.A. provided by the user (the BLWP instruction consists of two words, the BLWP operator and the destination address; the destination address points to a two-word area also programmed by the user).

Return from the interrupt service routine is through the RTWP instruction (routine's last instruction). This places the (previous) WP and PC values at the time of the BLWP instruction (in the six-word linking area) into the WP and PC registers. Thus, the RTWP code that follows the BLWP instruction will now be executed, causing a second return routine to occur, this time to the interrupted program using the return values in R13, R14, and R15 of the interrupt link area. This is shown graphically in Figure 5-11.

① INTERRUPT NO. 1 RECOGNIZED

M.A. 0000 | WP
0002 | PC
0004 | FF5A
0006 | FF7A

INTERRUPT VECTORS IN EPROM

FF5A | R0 ← FIRST REGISTER IN WORKSPACE

6-WORD INTERRUPT LINK AREA

②

R13 (OLD WP)
R14 (OLD PC)
R15 (OLD ST)

③

FF7A | BLWP
FF7C | XXXX
FF7E | RTWP

⑥

⑤

INTERRUPTED PROGRAM

XXXX | WP YYYY
PC ZZZZ

YYYY | R0

④

R13 FF5A
R14 FF7E
R15 (OLD ST)

ZZZZ

RTWP

INTERRUPT SERVICE ROUTINE
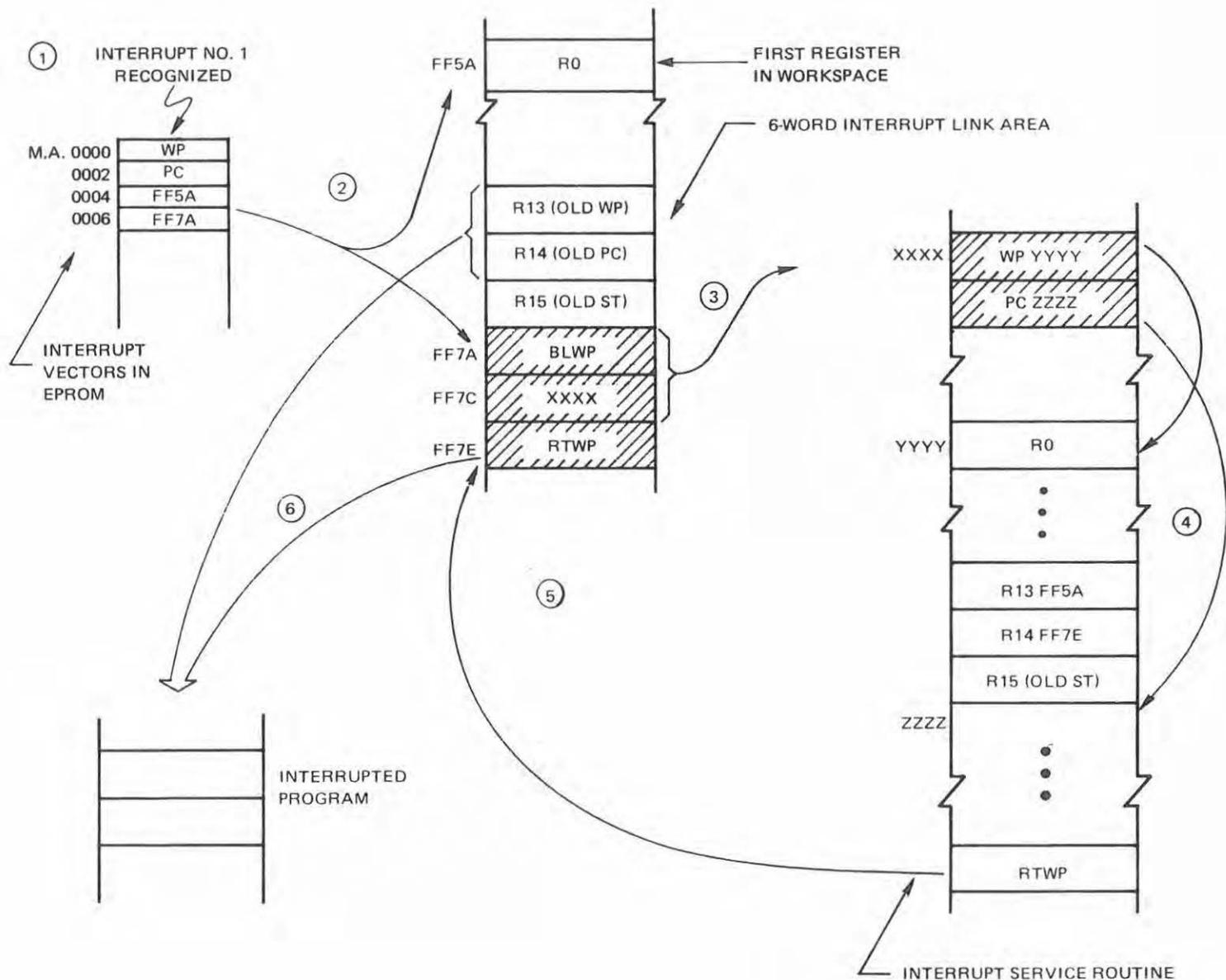
1,2 INTERRUPT EXECUTION TRAPS TO 6-WORD INTERRUPT LINK AREA.
3,4 BLWP EXECUTED TO 2-WORD VECTORS TO INTERRUPT SERVICE ROUTINE (ISR)
5   RTWP FROM ISR TRAPS BACK TO 6-WORD LINK AREA.
6   RTWP FROM LINK AREA RETURNS BACK TO INTERRUPTED PROGRAM.

= LINKAGE PROGRAMMED BY USER

Figure 5-11. Interrupt Sequence

Each interrupt linking area is set up so that it can be programmed in this manner. In summary, each six-word linking area can be programmed as follows:

- Determine the location of the linking area as shown by the WP and PC vectors in Table 5-4.

- The PC vector will point to the last three words of the six-word area. The user must program these three words respectively with 0420_{16} for a BLWP instruction, the address (BLWP operand) of the 2-word vector pointing to the interrupt service routine, and 0380_{16} for an RTWP instruction as shown in Figure 5-12.

- At the vector address for the BLWP operand, place the WP and PC values respectively of the interrupt handler.

EXAMPLE USING INT1 LINKING AREA (WP = FF5A, PC = FF7A)

M.A.	
FF5A	

(ACTUAL ADDRESS OF R0 OF INTERRUPT VECTOR WP)

FF74	R13 (OLD WP)
FF76	R14 (OLD PC)
FF78	R15 (OLD ST)

USED TO SAVE RETURN VALUES (TO INTERRUPTED PROGRAM)

TO BE PROGRAMMED BY USER

FF7A	0420 (BLWP)
FF7C	XXXX
FF7E	0380 (RTWP)

INT1 VECTOR PC ADDRESS (CONTAINS BLWP)

ADDRESS OF 2 - WORD VECTOR POINTING TO WP AND PC VALUES OF ISR

RETURN PC VALUE IN ISR POINTS TO THIS RTWP INSTR.

NOTE

DO NOT USE R0—R12 OF THE LINKING AREA WORKSPACE, BECAUSE THE OVERLAPPING STRUCTURE WILL DESTROY THE CONTENTS OF A LINKING AREA FOR ANOTHER INTERRUPT OR XOP.

Figure 5-12. Six-Word Interrupt Linking Area

Example coding to program the linkage to the interrupt service routine for INT1 is as follows:

```
*PROGRAM POINTER TO INT1 SERVICE ROUTINE FOLLOWING BLWP INSTRUCTION
     AORG      >FF7A      INT1 PC VECTOR ADDRESS
     DATA      >0420      HEX VALUE OF BLWP OP CODE
     DATA      >FA00      LOCATION OF 2-WORD VECTORS TO ISR (EXAMPLE)
     DATA      >0380      HEX VALUE OF RTWP OP CODE

*PROGRAM POINTER TO 2-WORD VECTORS TO INTERRUPT SERVICE ROUTINE (EXAMPLE)
     AORG      >FA00
     DATA      >FB00      WP OF INTERRUPT SERVICE ROUTINE (EXAMPLE)
     DATA      >FA04      PC OF INTERRUPT SERVICE ROUTINE (EXAMPLE)

*INT1 ISR FOLLOWS (BEGINS AT M.A.  FA04)
```

The interrupt service routine which begins at M.A. $FA04_{16}$ will terminate with an RTWP instruction.

5.9.1.2   XOP Linking Area

The XOP linking area contains seven words (14 bytes), of which the first two and the fourth words must be programmed by the user. Each XOP vector pair contains the pointer to the new WP (in the first word) and a pointer to the new PC (in the second word) which points to the first instruction to be executed.

In the seven-word XOP linking area, the first word is the destination of the XOP PC vector. The last three words are the final three registers (R13, R14, and R15) of the linking area workspace which will contain the return vectors back to the program that called the XOP. The third word of the seven-word area is R11, which contains the parameter being passed to the XOP service routine. This is shown in Figure 5-13.

For example, when XOP2 is executed, the PC vector points to the BLWP instruction shown at M.A. $FF5A_{16}$ in Figure 5-13. This executes, transferring control to the preprogrammed WP and PC values at the address in the next word (YYYY as shown in Figure 5-13). To obtain the parameter passed to R11 of the vector WP (M.A. $FF5E_{16}$ in Figure 5-13), use the following code in the XOP service routine:

MOV *R14+,R1   MOVE PARAMETER TO R1

This moves the parameter to R1 from the old R11 (the old PC value in R14 was pointing to this address following the BLWP instruction immediately above it, effectively to R11), and increments the XOP service routine PC value in its R14 to the RTWP instruction at M.A. $FF60_{16}$. Thus an RTWP return from the XOP service routine will branch back to the RTWP instruction at $FF60_{16}$ which returns control back to the instruction following the XOP.

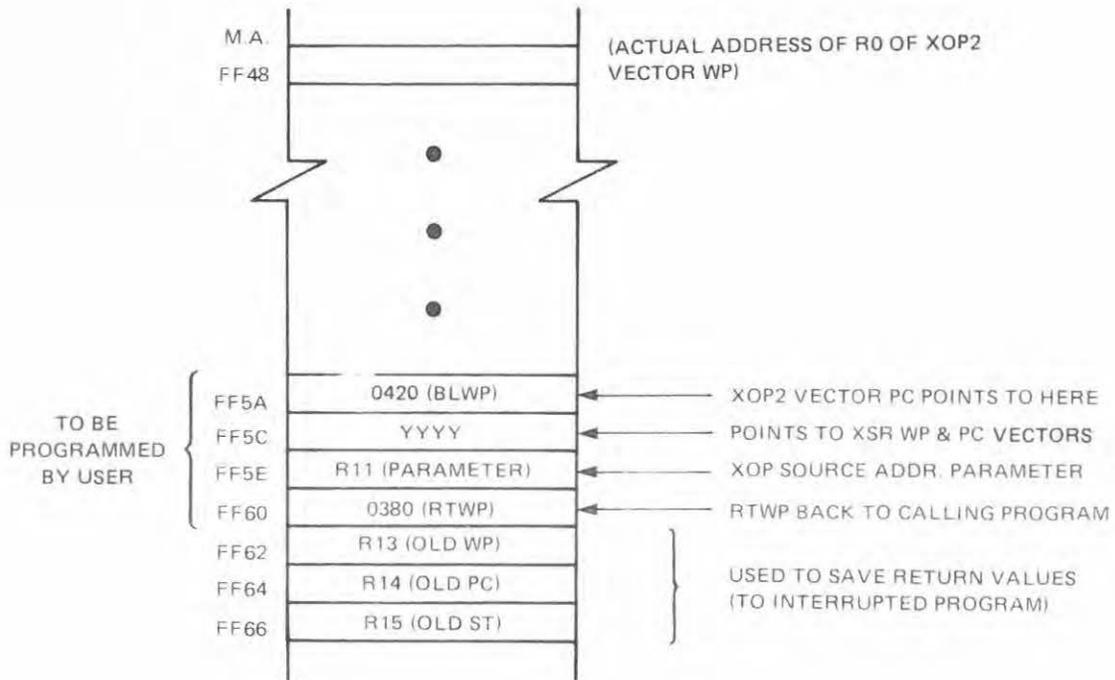EXAMPLE USING XOP 2 LINKING AREA (WP = FF48, PC = FF5A)

M.A.
FF48   (ACTUAL ADDRESS OF R0 OF XOP2 VECTOR WP)

TO BE PROGRAMMED BY USER

FF5A	0420 (BLWP)	← XOP2 VECTOR PC POINTS TO HERE
FF5C	YYYY	← POINTS TO XSR WP & PC VECTORS
FF5E	R11 (PARAMETER)	← XOP SOURCE ADDR. PARAMETER
FF60	0380 (RTWP)	← RTWP BACK TO CALLING PROGRAM
FF62	R13 (OLD WP)	
FF64	R14 (OLD PC)	USED TO SAVE RETURN VALUES (TO INTERRUPTED PROGRAM)
FF66	R15 (OLD ST)	

Figure 5-13. Seven-Word XOP Interrupt Linking Area

In summary, the seven-word XOP linking area can be programmed as follows:

● Determine the value of the PC vector for the XOP as shown in Table 5-4.

● The PC value will point to the first word of the seven-word linkage area. The user must program three of the first four words of this area respectively with 0420_{16} for a BLWP instruction, the address of the two-word vector that points to the XOP service routine, ignore the third word, and 0380_{16} for an RTWP instruction in the fourth word.

● At the address of the BLWP destination in the second word, place the WP and PC values respectively to the XOP service routine.

An example of coding to program the XOP linkage for XOP 2 as shown in Figure 5-13 is as follows:

```
*PROGRAM POINTER TO XOP SERVICE ROUTINE AT XOP2 LINK AREA
    AORG    >FF5A       XOP2 PC VECTOR ADDRESS
    DATA    >0420       HEX VALUE OF BLWP CODE
    DATA    >FA00       LOCATION OF 2-WORD VECTORS TO XSR (EXAMPLE)
    DATA    0           IGNORE
    DATA    >0380       HEX VALUE OF RTWP CODE

*PROGRAM POINTER TO 2-WORD VECTORS TO XOP2 SERVICE ROUTINE (EXAMPLE)
    AORG    >FA00       LOCATION OF VECTORS
    DATA    >FB00       WP OF XOP SERVICE ROUTINE (EXAMPLE)
    DATA    >FA04       PC OF XOP SERVICE ROUTINE (EXAMPLE

*XSR CODE FOLLOWS (BEGINS AT M.A.  FA04)
```

At the XOP service routine, the following code uses the PC return value (in R14 of the XOP service routine workspace) to obtain the parameter in R11 (in the link area) as well as set the return PC value in R14 (in the XOP service routine workspace) to the RTWP in the link area:

```
    MOV     *R14+,R1  MOVE OLD R11 CONTENTS TO R1 OF XOP SERVICE ROUTINE
```

Now R14 points to the RTWP instruction in the link area. The last instruction in the XOP service routine is RTWP. RTWP execution causes a return to the link area where a second RTWP executes, returning control to the next instruction following the XOP.

5.9.2 TMS 9901 INTERVAL TIMER INTERRUPT PROGRAM

A detailed discussion of the TMS 9901 interval timer can be found in the TMS 9901 data manual. There are several possible sequences of coding that can program and enable the interrupt 3 interval timer, and since the timer has a maximum period of 349 milliseconds before issuing an interrupt, the programmer must decide whether to set the interval period in the calling program or in the code handling the interrupt. If the interrupt period desired is longer than 349 milliseconds, then it may be advantageous to reset the timer in the interrupt subroutine which also triggers the interrupt and returns control back to the interrupted program. In any case, the timer must be initially set and triggered following the general sequence below:

1. Set the CRU address of the TMS 9901 in bits 3 to 14 of R12.

2. Set up the interrupt 3 linking area.

3. Enable the clock interrupt at the TMS 9901 (interrupt 3).

4. Set the Status Register interrupt mask to a value of 3 or greater.

5. Set a register to the value of the interval desired (bits 1 to 14) with bit 15 set to one to enable the clock as shown in Figure 5-14. This figure shows the code and a representation of the CRU for setting a time of 250 milliseconds and for setting the TMS 9901 to the clock mode. The first bit serially brought in on the CRU will be a value of one in bit 15 of the register which sets the TMS 9901 to the clock mode; successive bits (1 to 14) then set the clock interval value. The final bit brought in triggers the timer.

6. When the interrupt occurs, the interrupt handler must reset the interrupt at the TMS 9901 before returning to the interrupted program.

LI   R12,>100   CRU ADDRESS OF TMS 9901 (2 X >80 = >100)

LI   R1,>5B8F   CLOCK, >2DC7 COUNTS, AND SET CLOCK MODE BIT

LDCR R1, 15   SET CLOCK VALUE AT CLOCK REGISTER
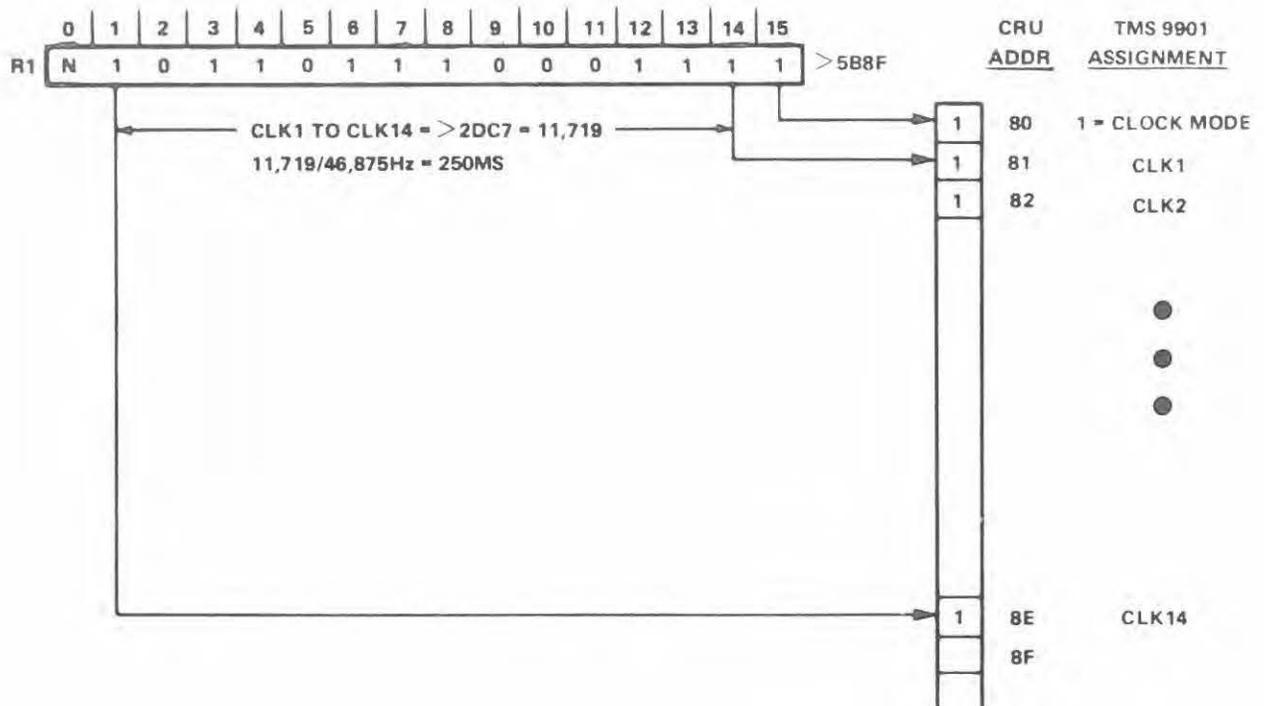
NOTE:

THE FIRST SERIAL INPUT FROM CRU (A ONE IN BIT 15 OF R1) SETS CLOCK MODE.

LAST INPUT TO CLOCK REGISTER (CLK1 TO CLK14) STARTS THE CLOCK.

A0001436

Figure 5-14. Enabling and Triggering TMS 9901 Interval Timer

The clock decrements the value set in step 5 at the rate of o/64 (approximately 46,875 Hz with a 3 MHz clock). The maximum interval register value of all ones in 14 bits (16,383) takes approximately 349 milliseconds to decrement to zero.

The code in Figure 5-15 is an example of a code to set up and call the TMS 9901 interval timer and also the code of the interrupt handling subroutine. Note that the calling program first clears the counting register (R0) of the interrupt workspace. Then it sets up the interrupt masks at the TMS 9901 and TMS 9900 after setting the TMS 9901 address in R12. Then the calling program sets an initial value in the timer register (CLK1 to CLK14 as shown in the TMS 9901 data manual). Because the desired output on the terminal is a message every 15 seconds, a minimum interval is set in the calling program while the interrupt handler is responsible for setting the time and clearing the interrupt after it occurs. The handler keeps a count of the intervals to determine the 15 seconds.

At the bottom of the figure is the interrupt linking area. Since all the code in this figure is loaded as if at absolute memory address values (using the AORG assembler directive) data statements are used here at the appropriate memory address. This program can be loaded and executed by placing the machine-language assembler output in the third column at the address shown in the second column. Then execute with the program start at M.A. FD0016.

The TMS 9901 can also be used as an event timer by starting the counter at the beginning of an interval and reading the counter after the event has occurred. To read the current value in the counter, the TMS 9901 must be taken out of the clock mode and put into the interrupt mode for at least 21.4 usec (1 TMS 9901 clock period). After that, putting the 9901 back into clock mode and reading the clock/int mask bits gives the current clock value (elapsed bit count divided by 46,875 equals elapsed time in seconds).

5.9.3 EXAMPLE OF PROGRAMMING TIMER INTERRUPTS FOR TMS 9901 AND TMS 9902
This subsection explains how to use the interrupt vector scheme to program the TMS 9901 and TMS 9902 timers. These timers use, respectively, interrupts 3 and 4 to trap to interrupt service routines following timer countdown.

The program described in the following paragraphs is an example that does the following:

- Initializes the interrupt linking areas for the TMS 9901 and TMS 9902 timers (interrupts 3 and 4 respectively).

- Loads the timers with interval values.

- Triggers the timers which cause interrupts when the countdown is complete.

- Contains interrupt service routines (ISR's) which execute when interrupts 3 or 4 are executed.

- Provides modules that perform hexadecimal-to-decimal conversions and decimal-to-hexadecimal conversions.

The individual modules of this program are summarized in Table 5-6. Please read these descriptions before continuing. The listing of this example program is provided in Figure 5-16, sheets 1 to 12.

```
0001          *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
0002          *  THIS PROGRAM CAUSES AN INTERRUPT THROUGH INT3        *
0003          *  EVERY 15 SECONDS USING THE INTERVAL TIMER IN THE     *
0004          *  TMS 9901. THE AORG DIRECTIVE CAUSES THE CODE TO BE   *
0005          *  ASSEMBLED BY THE TXMIRA ASSEMBLER BEGINNING AT THE   *
0006          *  ADDRESS SPECIFIED (SAME AS SLASH COMMAND ON THE      *
0007          *  LINE-BY-LINE ASSEMBLER). THIS PROGRAM CAN BE EXE-    *
0008          *  CUTED BY LOADING THE PROGRAM WITH THE TIBUG "M"      *
0009          *  COMMAND AND EXECUTING WITH THE "E" COMMAND AT PC     *
0010          *  ADDRESS >FD00. LOAD OBJECT IN THIRD COLUMN OF        *
0011          *  THIS LISTING AT ADDRESS IN 2D COLUMN. J.WALSH        *
0012          *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
0013                        IDT  'TIMER'
0014          *
0015          * REGISTER EQUATES
0016          *
0017      0000  R0     EQU  0
0018      0001  R1     EQU  1
0019      000C  R12    EQU  12
0020          *
0021          * PROGRAM CALLING THE INTERRUPT
0022          *
0023 FD00                  AORG >FD00              BEGIN ASSEMBLY AT M.A. >FD00
0024 FD00 02E0             LWPI >FD20              DEFINE WORKSPACE ADDRESS
     FD02 FD20
0025 FD04 04E0             CLR  @>FE60             CLEAR INTERRUPT REG 0
     FD06 FE60
0026 FD08 020C             LI   R12,>0100          9901 CRU ADDRESS IN R12
     FD0A 0100
0027 FD0C 1E00             SBZ  0                  9901 TO INTERRUPT MODE
0028 FD0E 1D03             SBO  3                  ENABLE INTERRUPT 3
0029 FD10 0300             LIMI 3                  ENABLE INT3 AT TMS 9900
     FD12 0003
0030 FD14 0201             LI   R1,3               2 ONES TO TMS 9901
     FD16 0003
0031 FD18 33C1             LDCR R1,15              ENABLE CLOCK AT 9901
0032 FD1A 10FF             JMP  $                  LOOP HERE, WAIT FOR INTERRUPT
0033          *
0034          * INTERRUPT SUBROUTINE
0035          *
0036 FE00                  AORG >FE00              BEGIN ASSEMBLY AT M.A.>FE00
0037 FE00 FE60             DATA >FE60              BLWP WP VECTOR FOR INT
0038 FE02 FE04             DATA >FE04              BLWP PC VECTOR FOR INT
0039 FE04 0300             LIMI 0                  DISABLE INTERRUPTS
     FE06 0000
0040 FE08 0280             CI   R0,60              COUNT = 60 = 15 SECONDS?
     FE0A 003C
0041 FE0C 130B             JEQ  >FE24              YES, PRINT MESSAGE
0042 FE0E 0580             INC  R0                 NO, INCREMENT COUNTER
0043 FE10 020C             LI   R12,>100           9901 CRU ADDRESS
     FE12 0100
0044 FE14 0201             LI   R1,>5B9F           CLOCK COUNT OF 11,719
     FE16 5B9F
0045 FE18 33C1             LDCR R1,15              APPLY COUNT, START COUNTER
```

Figure 5-15. Example of Code to Run TMS 9901 Interval Timer (Sheet 1 of 2)

```
0046 FE1A 1E00        SBZ   0              9901 TO INTERRUPT MODE
0047 FE1C 1D03        SBO   3              CLEAR INTERRUPT AFTER EXECUTED
0048 FE1E 0300        LIMI  3              RESET INT MASK AT TMS 9900
     FE20 0003
0049 FE22 0380        RTWP                 RETURN TO CALLING PROGRAM
0050 FE24 2FA0        XOP   @>FE2E,14      WRITE MESSAGE
     FE26 FE2E
0051 FE28 04C0        CLR   R0             RESET TIMER COUNT
0052 FE2A 0460        B     @>FE04         BEGIN AT INTERRUPT START
     FE2C FE04
0053 FE2E   31        TEXT  '15 SECONDS HAVE ELAPSED.'
     FE2F   35
     FE30   20
     FE31   53
     FE32   45
     FE33   43
     FE34   4F
     FE35   4E
     FE36   44
     FE37   53
     FE38   20
     FE39   48
     FE3A   41
     FE3B   56
     FE3C   45
     FE3D   20
     FE3E   45
     FE3F   4C
     FE40   41
     FE41   50
     FE42   53
     FE43   45
     FE44   44
     FE45   2E
0054 FE46 0707        DATA  >0707,>0707    BELLS
     FE48 0707
0055 FE4A   00        BYTE  0              END OF MESSAGE DELIMITER
0056             *
0057             * INTERRUPT LINK AREA PROGRAMMING
0058             *
0059 FFAA            AORG  >FFAA          BEGIN ASSEMBLY AT M.A. >FFAA
0060 FFAA 0420        DATA  >0420          BLWP INSTRUCTION CODE
0061 FFAC FE00        DATA  >FE00          BLWP VECTORS LOCATION
0062 FFAE 0380        DATA  >0380          RTWP INSTRUCTION CODE
0063                  END

0000 ERRORS
```

NOTE: As an exercise, the user can load and execute this code: (1) load the machine code values shown
        in column 3 into the memory locations shown in column 2, or (2) reassemble : if the Line-By-
        Line Assembler (LBLA) is used, substitute the slash command for the AORG directive and follow
        the DATA and TEXT statement conventions for the LBLA. Execute using the E TIBUG command.

Figure 5-15.   Example of Code to Run TMS 9901 Interval Timer (Sheet 2 of 2)

Table 5-6. Interrupt Example Program Description

Module	Sheet Number of Figure 5-16	Program Description
Interrupt Link	1	This module sets up the interrupt linkage areas for interrupts 3 and 4, loads vectors pointing to Module REALCK for interrupt 3 and to Module KYBDSC for interrupt 4. This is the first program called, and it calls Module User Start.
User Start	2 to 4	"User Start" routine; this is the start of the general user control program. This contains mainline code to the timers, and calls KYINIT before starting the timers.
Timer, TMS 9901	5	This module sets TMS 9901 timer to specified value, starts countdown (countdown completion causes interrupt through interrupt level 3).
Timer, TMS 9902	6	This module sets TMS 9902 timer of local I/O port to specified value, starts countdown (countdown completion causes interrupt through interrupt 4).
Real Time Clock ISR	7 and 8	This Real-Time Clock routine is the Interrupt Service Routine (ISR) for interrupt 3. It accumulates counts at one-fifth second intervals to keep a real time clock count; time values are initialized by User Start.
Keyboard Initialization	8	This module initializes I/O buffer for keyboard input.
Keyboard Scan ISR	9 and 10	This is the Keyboard Scan Routine ISR for interrupt 4. It polls the keyboard unit for a new character, and then puts the character in buffer. Backspace and delete monitoring is provided.
Hex/Decimal Conversions	11 and 12	These modules convert decimal numbers to hexadecimal equivalents (sheet 11) and hexadecimal numbers to decimal equivalents (sheet 12).

5.9.3.1 Interrupt Linking Area Set-Up (Figure 5-16, Sheet 1)

This module sets up the interrupt linking areas that point to the two interrupt service routines for the timers in the TMS 9901 and TMS 9902. The workspace for this module is the space just below the INT3 and INT4 linking areas. Since this example uses only interrupts 3 and 4, the linking areas for interrupts 1, 2, and 5 through 15 are free space.

5.9.3.2 User Start Program (Figure 5-16, Sheets 2, 3, 4)

This module organizes the other modules into a user program. It sets up control functions and calls other modules in a prescribed sequence. This program receives control after the interrupt linking areas are initialized as described in paragraph 5.9.2.1. It then sets the timing values for the TMS 9901 timer and begins the countdown by a BLWP @TIME01. It also calls the keyboard initialization module (BLWP @KYINIT) which calls the TMS 9902 set and execute module (BLWP @TIME02).

NOTE

This User Start Program is for example purposes, and is intended only as a vehicle to demonstrate usage of the following subroutine modules.

5.9.3.3 TMS 9901 Timer Set Routine (Figure 5-16, Sheet 5)

This module sets and executes the interval timer of the TMS 9901. The calling routine specifies the number of 21.333-microsecond periods (at 3 MHz) to be counted by loading its own register 0. The TIME01 routine then picks this number (limited to 14 bits) by indirect addressing through R13 (return WP value = R0). It shifts it while in R9, supplies the correct control bit (bit 0 = 1 by ORing), starts the timer (LDCR instruction) and enables the interrupt. Control returns to the calling program, which will be interrupted by the timer interrupt when the count reaches zero. The calling sequence to the timer set routine is:

```
LI      R0,9375       1/5TH SECOND INTERVALS
BLWP    @TIME01       SET TIMER
```

The interrupt service routine for interrupt 3 is in paragraph 5.9.3.5.

5.9.3.4 TMS 9902 Timer Set Routine (Figure 5-16, Sheet 6)

This module sets and executes interval timer of the TMS 9902. The calling routine specifies (in its own register 0) the number of 64 microsecond periods (at 3 MHz, with the TMS 9902's CLK4M control bit zeroed) to be counted before generating the interrupt. This routine then picks this number up (through WP return value in R13, old R0), puts it in the left byte of R9, sets the LDIR (Load Interval Register) flag to enable loading of the timer value, resets LDCTRL (Load Control Register) to bypass loading the control register, loads the timer which begins the count, and then enables interrupt 4 on the TMS 9901. Notice that the user must have a jumper plug between pins E2 and E3 for an interrupt to occur. Control returns to the calling program which will be interrupted by the timer sometime later (called ISR described in paragraph 5.9.3.6).

5.9.3.5 TMS 9901 24-Hour Real-Time Clock Service Routine (Figure 5-16, Sheet 7)
In this module, the TMS 9901 timer is used as a real time clock; an interrupt occurs
every fifth of a second and a fractions counter is updated. The calling program
initially sets the second-interval counter (R1) to 5. Every five counts, the seconds
counter is updated; every sixty seconds the minutes counter is updated, etc. Note that
since the initial period (one-fifth second) is long, the execution time of this
service routine is trivial from a system throughput standpoint. Note also that because
this timer is associated with interrupt 3, it has higher priority than the TMS 9902
timer, which will be used for miscellaneous timing purposes in this example. This
ensures the integrity of the real time clock recording the elapsed time from system
initialization.

5.9.3.6 TMS 9902 Used To Poll Keyboard Service Routine (Figure 5-16, Sheets 9 and 10)
In this module, the TMS 9902 timer is being used as a general purpose delay timer. The
service routine samples an ASCII encoded keyboard's output, and if a set time has
elapsed and a strobe change occurred, it reads the character. The time delay and
strobe change ensure a new character has been sent from the keyboard. The strobe for
any one character is assumed to last longer than the interval set in the timer for
scanning, and a flag is used in the software to simulate an edge-triggered data
capture condition. The ASCII encoded keyboard is assumed to be connected to the TMS
9901 through connector P4.

When the strobe goes from high to low, data is read, and the flag turned on. Only when
the strobe goes high again is the flag reset and a new character can be received.

5.9.3.7 Decimal To Hexadecimal Conversion (Figure 5-16, Sheet 11)
This module is a sample decimal-to hexadecimal-conversion routine. The calling program
places the least significant four digits in its register 0, and the most significant
(fifth) digit is right-justified in its register 1. A BLWP @DECHEX instruction gives
control to the conversion routine.

The called routine isolates each decimal digit and uses it to index a loop which adds
the proper place value (10,100,1000, etc.) to the result register. As each digit is
isolated, a table pointer is bumped through the decimal powers. The resultant
hexadecimal number is returned to the caller routine's register 0. The caller's
register 1 is not disturbed.

5.9.3.8 Hexadecimal To Decimal Conversion (Figure 5-16, Sheet 12)
This module is a sample hexadecimal to decimal conversion routine. The calling routine
places the hexadecimal number in its own register 0, then performs a BLWP @HEXDEC. The
converted result is placed back in the caller's register 0 (through address in R13),
with a fifth digit (most significant) in register 1 of the calling program. Both
registers in the calling program are always altered.

The routine repeatedly divides the number by 10, and collects the remainders. These
remainders, properly collected by the shift and SOC instructions, form the decimal
number.

```
0001                                 IDT   'TEST'
0002                         *-----------------------------------------------------------
0003                         *        INTERRUPT LINKING AREA INITIALIZATION ROUTINE.
0004                         *        THIS ROUTINE INITIALIZES THE INTERRUPT LINKING
0005                         *        AREA IN HIGH RAM FOR INTERRUPTS 3 AND 4.
0006                         *        A "BLWP" INSTRUCTION IS BUILT, WITH THE
0007                         *        ADDRESS OF THE PARTICULAR INTERRUPT SERVICE
0008                         *        ROUTINE WHICH WILL THEN RECEIVE CONTROL
0009                         *        WHEN THE INTERRUPT IS ACTIVATED. TO COMPLETE
0010                         *        THE RETURN PATH, A "RTWP" INSTRUCTION IS
0011                         *        BUILT IN RAM ALSO.
0012                         *-----------------------------------------------------------
0013 0000 02E0       ENTRY   LWPI  >FF78           GET WORKSPACE
     0002 FF78
0014 0004 0300               LIMI  0               CUT OFF INTERRUPTS
     0006 0000
0015                         *        THE FOLLOWING CODE LOADS THE REGISTERS WITH THE
0016                         *        PROPER VALUES FOR INITIALIZING THE RAM AREA.
0017 0008 C060               MOV   @>000E,1        GET INT 3 PC PTR
     000A 000E
0018 000C C0A0               MOV   @>0012,2        GET INT 4 PC PTR
     000E 0012
0019 0010 0203               LI    3,>0420         LOAD BLWP OPCODE
     0012 0420
0020 0014 0204               LI    4,>0380         LOAD RTWP OPCODE
     0016 0380
0021 0018 0205               LI    5,INT3VC        ADDR OF 9901 TIMER ROUTINE
     001A 0148'
0022 001C 0206               LI    6,INT4VC        ADDR OF 9902 TIMER ROUTINE
     001E 01A8'
0023                         *        THE FOLLOWING CODE TAKES THE INFORMATION IN THE
0024                         *        REGISTERS AND MOVES IT OUT TO INITIALIZE THE
0025                         *        RAM LINKING AREA. FIRST INTERRUPT 3 AREA IS
0026                         *        INITIALIZED, THEN THE INTERRUPT 4 AREA.
0027                         *
0028                         *        INTERRUPT 3 - TMS 9901 TIMER
0029 0020 CC43               MOV   3,*1+           MOVE "BLWP" OPCODE
0030 0022 CC45               MOV   5,*1+           MOVE SERVICE ROUTINE ADDRESS
0031 0024 CC44               MOV   4,*1+           MOVE "RTWP" OPCODE
0032                         *        INTERRUPT 4 - TMS 9902 TIMER
0033 0026 CC83               MOV   3,*2+           MOVE "BLWP" OPCODE
0034 0028 CC86               MOV   6,*2+           MOVE SERVICE ROUTINE ADDRESS
0035 002A CC84               MOV   4,*2+           MOVE "RTWP" OPCODE
0036                         *        RETSORE INTERRUPTS
0037 002C 0300               LIMI  4               TURN INTERRUPTS BACK ON
     002E 0004
```

Figure 5-16. Example Program Using Timer Interrupts 3 and 4 (Sheet 1 of 12)

```
0039                    *----------------------------------------------------------
0040                    *       MAIN ROUTINE
0041                    *       THIS ROUTINE IS A SMALL SAMPLE OF WHAT TYPE
0042                    *       OF CODE SHOULD BE USED TO CONTROL THE FUNCTIONS
0043                    *       OF THE VARIOUS PARTS OF THE SYSTEM BEING
0044                    *       USED IN THIS EXAMPLE. PLEASE KEEP IN MIND
0045                    *       THAT THIS ENTIRE PROGRAMMING EXAMPLE IS
0046                    *       STILL ONLY AN EXAMPLE OF HOW THE FACILITIES
0047                    *       OF THE MICROCOMPUTER CAN BE USED; IT IS NOT
0048                    *       INTENDED TO SERVE AS A SOFTWARE BASE FOR
0049                    *       A USER APPLICATION PROGRAM.
0050                    *----------------------------------------------------------
0051                    *       THIS MAIN ROUTINE RECEIVES CONTROL AFTER
0052                    *       THE INTERRUPT LINKING AREA IS INITIALIZED.
0053                    *       IT CALLS THE KEYBOARD INITIALIZATION
0054                    *       ROUTINE, AND STARTS BOTH TIMERS GOING.
0055                    *       IT THEN INTERROGATES THE NEW-LINE FLAG
0056                    *       AND "DISPOSES" OF THE USER DATA BY
0057                    *       PRINTING IT. (OF COURSE, AN APPLICATION
0058                    *       PROGRAM WOULD DO MORE WITH THE DATA).
0059                    *----------------------------------------------------------
0060                    *
0061                    *       WORK AREA DEFINITIONS
0062                    *
0063      FF18   KYBDWP EQU   >FF18              KEYBOARD ROUTINE WORKSPACE
0064      FEF3   KYBUF  EQU   >FEF3              KEYBOARD BUFFER
0065      FF38   CLKWP  EQU   >FF38              REAL-TIME CLOCK WORKSPACE
0066      FF78   COMRG  EQU   >FF78              TRANSIENT ROUTINE COMMON WORKS
0067      FF58   MAINRG EQU   >FF58              MAIN REGS FOR THIS ROUTINE
0068                    *
0069                    *       XOP DEFINITIONS
0070                    *
0071                           DXOP READ,11      READ ONE CHARACTER
0072                           DXOP WRIT,14      WRITE A STRING
0073                           DXOP HEXI,9       HEX # INPUT
0074                           DXOP HEXO,10      HEX # OUTPUT
0075                    *
0076                    *       ENTRY POINT
0077                    *
0078 0030 02E0  USERST LWPI CLKWP               CLOCK REGS FOR INITIALIZATION
     0032 FF38
0079 0034 04C1         CLR  1                   CLEAR FOR DECIMAL TO HEX ROUTI
0080 0036 0207         LI   7,CKPARM            PROMPT MESSAGES
     0038 00BC'
0081 003A 0208         LI   8,5                 FIVE PROMPTS
     003C 0005
0082 003E 0209         LI   9,CLKWP+4           REGISTER 2 ADDRESS
     0040 FF3C
0083 0042 2F97  LOOP1  WRIT *7                  PROMPT USER FOR TIME VALUE
0084 0044 2E40         HEXI O                   GET INPUT
0085 0046 004A'        DATA NEXT,ERROR          NULL, ERROR RTN ADR
     0048 00B6'
0086 004A 0420  NEXT   BLWP @DECHEX             DECIMAL CHARS TO BINARY
     004C 020A'
```

Figure 5-16. Example Program Using Timer Interrupts 3 And 4 (Sheet 2 of 12)

```
0087 004E CE40              MOV   0,*9+         PUT VALUE IN CLOCK REGISTERS
0088 0050 2FA0              WRIT  @CRLF         DO CARRIAGE RETURN / LINE FEED
     0052 0100'
0089 0054 0227              AI    7,12          NEXT PROMPT IN TABLE
     0056 000C
0090 0058 0608              DEC   8             ONE LESS TO GO
0091 005A 16F3              JNE   LOOP1         GO BACK IF NOT DONE
0092 005C 2F97              WRIT  *7            READY, GET SET, GO !
0093 005E 2EC9              READ  9             USER RESPONSE STARTS CLOCK
0094 0060 2FA0              WRIT  @CRLF         NEW LINE
     0062 0100'
0095 0064 0200              LI    0,9375        ONE-FIFTH SECOND INTERVALS
     0066 249F
0096 0068 0420              BLWP  @TIME01       SET TIMER
     006A 0104'
0097 006C 0201              LI    1,5           INTERRUPTS / SECOND
     006E 0005
0098 0070 02E0              LWPI  MAINRG        NOW USE THIS ROUTINE'S REGS
     0072 FF58
0099 0074 0420              BLWP  @KYINIT       START SCANNING KEYBOARD
     0076 0184'
0100 0078 C820   WAIT       MOV   @KYBDWP,@KYBDWP   LOOK AT LINE FLAG
     007A FF18
     007C FF18
0101 007E 13FC              JEQ   WAIT          NOT COMPLETE LINE YET
0102 0080 8820              C     @KYBUF,@TI    TIME REQUEST?
     0082 FEF3
     0084 00FE'
0103 0086 1305              JEQ   TIME          GO PRINT REAL TIME
0104 0088 2FA0              WRIT  @CRLF         FINISH LINE
     008A 0100'
0105 008C 2FA0              WRIT  @KYBUF        SPILL THE BUFFER
     008E FEF3
0106 0090 10F3              JMP   WAIT          WAIT FOR MORE TYPED STUFF
0107 0092 0207   TIME       LI    7,CKPARM      PROMPT STRINGS NOW HEADINGS
     0094 00BC'
0108 0096 0208              LI    8,5           # OF ITEMS
     0098 0005
0109 009A 0209              LI    9,CLKWP+4     CLOCK REGISTERS 2,3,4,5,6
     009C FF3C
0110 009E 2F97   LOOP2      WRIT  *7            PRINT HEADING
0111 00A0 C039              MOV   *9+,0         GET TIME PARM FROM CLOCK
0112 00A2 0420              BLWP  @HEXDEC       CONVERT BINARY TO DECIMAL
     00A4 0252'
0113 00A6 2E80              HEXO  0             PRINT TIME
0114 00A8 2FA0              WRIT  @CRLF         FINISH LINE
     00AA 0100'
0115 00AC 0227              AI    7,12          NEXT HEADING
     00AE 000C
0116 00B0 0608              DEC   8             ONE LESS TO GO
0117 00B2 16F5              JNE   LOOP2         GO BACK IF NOT DONE
0118 00B4 10E1              JMP   WAIT          DONE, GO WAIT
0119 00B6 2FA0   ERROR      WRIT  @CRLF         DO CR / LF
     00B8 0100'
```

Figure 5-16. Example Program Using Timer Interrupts 3 and 4 (Sheet 3 of 12)

```
0120 00BA 10C3              JMP  LOOP1
0121            *
0122            *     DATA CONSTANTS
0123            *
0124 00BC  53   CKFARM TEXT 'SECONDS
0125 00C7  00          BYTE 0
0126 00C8  4D          TEXT 'MINUTE
0127 00D3  00          BYTE 0
0128 00D4  48          TEXT 'HOUR
0129 00DF  00          BYTE 0
0130 00E0  44          TEXT 'DAY NUMBER
0131 00EB  00          BYTE 0
0132 00EC  59          TEXT 'YEAR
0133 00F7  00          BYTE 0
0134 00F8  47          TEXT 'GO ? '
0135 00FD  00          BYTE 0
0136 00FE  54   TI     TEXT 'TI'
0137 0100  0D   CRLF   BYTE >D,>A,0
     0101  0A
     0102  00
```

Figure 5-16. Example Program Using Timer Interrupts 3 and 4 (Sheet 4 of 12)

```
0139                   *-----------------------------------------------------------------
0140                   *      TMS 9901 TIMER SET ROUTINE
0141                   *      THIS ROUTINE SETS THE INTERVAL TIMER ON THE TMS9901
0142                   *      WITH A VALUE PASSED BY THE CALLING PROGRAM. THE
0143                   *      VALUE PASSED IS SIMPLY AN INTEGER COUNT OF THE
0144                   *      NUMBER OF 21.333 MICROSECOND PERIODS DESIRED. THIS
0145                   *      ROUTINE TAKES CARE OF LOADING THE TIMER REGISTER
0146                   *      PROPERLY, AND ENABLING THE TIMER INTERRUPT.
0147                   *-----------------------------------------------------------------
0148 0104 FF78   TIME01 DATA >FF78,ENT01
     0106 0108'
0149 0108 0300   ENT01  LIMI 0                  TURN OFF INTERRUPTS
     010A 0000
0150 010C C25D          MOV  *13,9              GET TIMER VALUE
0151 010E 020C          LI   12,>0100           ADDRESS 9901
     0110 0100
0152 0112 0A19          SLA  9,1                SHIFT CLOCK COUNT
0153 0114 0269          ORI  9,1                SET CLOCK MODE
     0116 0001
0154 0118 33C9          LDCR 9,15               START CLOCK
0155 011A 1E00          SBZ  0                  INTERRUPT MODE
0156 011C 1D03          SBO  3                  ENABLE INT 3 REQ MASK
0157 011E 0300          LIMI 4                  TURN INTERRUPTS BACK ON
     0120 0004
0158 0122 0380          RTWP                    RETURN TO CALLER
```

Figure 5-16. Example Program Using Timer Interrupts 3 and 4 (Sheet 5 of 12)

```
0160                    *---------------------------------------------------------------
0161                    *       TMS 9902 INTERVAL TIMER SET ROUTINE
0162                    *       THIS PROGRAM SETS THE INTERVAL TIMER OF THE TMS 9902
0163                    *       USING THE VALUE PASSED BY THE CALLING PROGRAM.
0164                    *       THE PROGRAM LOADS THE VALUE PROPERLY AND ENABLES
0165                    *       THE APPROPRIATE INTERRUPT.
0166                    *---------------------------------------------------------------
0167 0124 FF78    TIME02 DATA >FF78,ENTO2
     0126 0128
0168 0128 0300    ENTO2  LIMI 0                    CUT OFF INTERRUPTS
     012A 0000
0169 012C C25D           MOV  *13,9                GET TIMER VALUE
0170 012E 06C9           SWPB 9                    PUT IN LEFT BYTE FOR LDCR
0171 0130 020C           LI   12,>0080             POINT TO 9902
     0132 0080
0172 0134 1D0D           SBO  13                   SET LDIR TO LOAD VALUE
0173 0136 1E0E           SBZ  14                   RESET LDCTRL, BYPASS CONTROL R
0174 0138 3209           LDCR 9,8                  LOAD TIMER, BEGIN COUNT
0175 013A 1D14           SBO  20                   SET TIMENB FOR INTERRUPT
0176 013C 0A1C           SLA  12,1                 POINT TO 9901
0177 013E 1E00           SBZ  0                    SET INTERRUPT MODE
0178 0140 1D04           SBO  4                    ENABLE INT 4 MASK
0179 0142 0300           LIMI 4                    GIVE BACK INTERRUPTS
     0144 0004
0180 0146 0380           RTWP                      RETURN
```

Figure 5-16. Example Program Using Timer Interrupts 3 and 4 (Sheet 6 of 12)

```
0182                  *-------------------------------------------------------------
0183                  *        TMS 9901 REAL TIME CLOCK ROUTINE
0184                  *        THIS ROUTINE IS ACTIVATED WHEN THE TMS 9901
0185                  *        INTERVAL TIMER COUNTS DOWN TO ZERO, CAUSING
0186                  *        INTERRUPT 3. THIS ROUTINE COUNTS THE NUMBER
0187                  *        OF ONE-FIFTH SECOND INTERVALS OCCURRING AND
0188                  *        UPDATES THE APPROPRIATE COUNTER. AT THE END
0189                  *        OF A SECOND, THE MINUTE COUNTER IS CHECKED,
0190                  *        AND UPDATED IF NECESSARY. THIS PROCEDURE IS
0191                  *        REPEATED FOR EACH SUCCESSIVELY LARGER TIME
0192                  *        UNIT, UP TO A YEAR. LEAP YEARS DON'T COUNT.
0193                  *-------------------------------------------------------------
0194 0148 FF38  INT3VC DATA CLKWP,IN3PC
     014A 014C'
0195 014C 020C  IN3PC  LI   12,>0100          POINT TO 9901
     014E 0100
0196 0150 1E00         SBZ  0                  INTERRUPT MODE
0197 0152 1D03         SBO  3                  ACKNOWLEDGE INTERRUPT
0198 0154 0601         DEC  1                  DOCK FRACTION COUNTER
0199 0156 1615         JNE  RETURN             NOT DONE WITH A SECOND YET
0200                 *      NEW SECOND
0201 0158 0201         LI   1,5                NEW SECOND COUNTDOWN
     015A 0005
0202 015C 0582         INC  2                  ADD ONE SECOND TO CLOCK
0203 015E 0282         CI   2,60               60 SECONDS YET?
     0160 003C
0204 0162 160F         JNE  RETURN             NO, GO RETURN
0205                 *      NEW MINUTE
0206 0164 04C2         CLR  2                  NEW MINUTE: CLEAR SECONDS
0207 0166 0583         INC  3                  ADD ONE MINUTE
0208 0168 0283         CI   3,60               60 MINUTES YET?
     016A 003C
0209 016C 160A         JNE  RETURN             NO,RETURN
0210                 *      NEW HOUR
0211 016E 04C3         CLR  3                  NEW HOUR: CLEAR MINUTES
0212 0170 0584         INC  4                  ADD ONE HOUR
0213 0172 0284         CI   4,24               MIDNIGHT YET?
     0174 0018
0214 0176 1605         JNE  RETURN             NO
0215                 *      NEW DAY
0216 0178 0585         INC  5                  ADD ONE DAY
0217 017A 0285         CI   5,366              END OF YEAR?
     017C 016E
0218 017E 1601         JNE  RETURN             NO, RETURN
0219                 *      NEW YEAR
0220 0180 0586         INC  6                  NEXT YEAR
0221 0182 0380  RETURN RTWP
```

Figure 5-16. Example Program Using Timer Interrupts 3 and 4 (Sheet 7 of 12)

```
0223                    *------------------------------------------------------------
0224                    *       KEYBOARD INITIALIZATION ROUTINE
0225                    *       THIS ROUTINE INITIALIZES THE WORK AREA USED BY THE
0226                    *       KEYBOARD SCANNING ROUTINE WHEN THE TMS 9902 TIMER
0227                    *       TIMES OUT. THE TMS 9902 TIMER IS DEDICATED TO TIMING
0228                    *       THE INTERVAL BETWEEN KEYBOARD SCANS. IT IS SET
0229                    *       IN THIS ROUTINE, AND THE KEYBOARD CHARACTER BUFFER
0230                    *       .IS CLEARED OUT, AS WELL AS THE APPROPRIATE FLAGS RESE
0231                    *------------------------------------------------------------
0232 0184 FF18   KYINIT DATA KYBDWP,KYENT
     0186 0188
0233 0188 0209   KYENT  LI    9,37          # WORDS IN BUFFER
     018A 0025
0234 018C 0208          LI    8,KYBUF       KEYBOARD INPUT BUFFER
     018E FEF3
0235 0190 04F8   LOOP   CLR   *8+           WIPE TWO BYTES OUT
0236 0192 0609          DEC   9             # OF WORDS LEFT
0237 0194 16FD          JNE   LOOP          GO BACK
0238 0196 04C2          CLR   2             CLEAR INDEX PTR: NEW LINE
0239 0198 04C3          CLR   3             CLEAR STROBE FLAG
0240 019A 04C0          CLR   0             CLEAR NEW-LINE FLAG
0241 019C 04C1          CLR   1             CLEAR DATA AREA
0242 019E 0200          LI    0,208         75 SCANS / SECOND
     01A0 00D0
0243 01A2 0420          BLWP  @TIMEO2       GO START TIMER
     01A4 0124
0244 01A6 0380          RTWP                DONE
```

Figure 5-16. Example Program Using Timer Interrupts 3 and 4 (Sheet 8 of 12)

```
0246                     *----------------------------------------------------------------
0247                     *          KEYBOARD SCANNING ROUTINE
0248                     *          THIS ROUTINE SCANS AN ASCII-ENCODED KEYBOARD
0249                     *          CONNECTED DIRECTLY TO THE PARALLEL I/O PORT, P4.
0250                     *          I/O BITS 0-7 ARE ASCII DATA, AND BIT 8 IS AN
0251                     *          EDGE-TRIGGERED (HIGH-TO-LOW) STROBE.
0252                     *          THIS ROUTINE IS ENTERED WHEN THE INTERVAL TIMER
0253                     *          IN THE TMS 9902 TIMES OUT. THE INTERRUPT IS
0254                     *          ACKNOWLEDGED, AND THE STATE OF THE STROBE FLAG
0255                     *          IS SENSED. IF PREVIOUSLY INACTIVE AND NOW ACTIVE,
0256                     *          A NEW CHARACTER HAS APPEARED ON THE I/O PORT,
0257                     *          WHICH IS READ IMMEDIATELY. IF THE STROBE IS
0258                     *          INACTIVE, OR IF PREVIOUSLY ACTIVE AND STILL ACTIVE,
0259                     *          THEN THE I/O PORT IS IGNORED. WHEN A NEW CHARACTER
0260                     *          IS READ, THE STROBE FLAG IS SET, AND IS RESET
0261                     *          ONLY AFTER THE STROBE GOES INACTIVE.
0262                     *          CHARACTERS ARE COLLECTED IN THE KEYBOARD BUFFER
0263                     *          AND WHEN A CARRIAGE RETURN IS INPUT, OR WHEN
0264                     *          THE BUFFER IS FULL, THE NEW-LINE FLAG IS SET.
0265                     *          IT IS ASSUMED THERE IS A ROUTINE SOMEWHERE
0266                     *          WHICH INSPECTS THE NEW-LINE FLAG, AND USES
0267                     *          THE COLLECTED DATA FOR SOME PURPOSE.
0268 01A8 FF18    INT4VC DATA KYBDWP,IN4PC
     01AA 01AC
0269                     *          ADDRESS THE TMS 9902, TURN OFF INTERRUPT
0270 01AC 020C    IN4PC  LI   12,>0080           POINT TO 9902
     01AE 0080
0271 01B0 1D14           SBO  20                 RESET INTERRUPT
0272                     *          ADDRESS THE TMS 9901, AND POLL THE KEYBOARD STATUS
0273 01B2 020C           LI   12,>0120           PARALLEL I/O 9901
     01B4 0120
0274 01B6 C0C3           MOV  3,3                CHECK STROBE FLAG
0275 01B8 1304           JEQ  SCAN               RESET: SCAN KEYBOARD
0276 01BA 1F08           TB   8                  LOOK AT STROBE
0277 01BC 1617           JNE  GOBACK             STILL LOW FROM LAST CHAR
0278 01BE 04C3           CLR  3                  HIGH: DONE WITH OLD CHAR
0279 01C0 1015           JMP  GOBACK             SINCE NO CHAR, RETURN
0280                     *          STROBE FLAG WAS RESET, SO SCAN KEYBOARD
0281 01C2 1F08    SCAN   TB   8                  LOOK AT STROBE
0282 01C4 1313           JEQ  GOBACK             HIGH: NO CHAR YET
0283 01C6 0703           SETO 3                  SET STROBE FLAG, NEW CHAR
0284 01C8 3601           STCR 1,8                GRAB BYTE FROM KEYBOARD
0285 01CA 0241           ANDI 1,>7F00            STRIP PARITY BIT
     01CC 7F00
0286 01CE 0281           CI   1,>0800            BACKSPACE?
     01D0 0800
0287 01D2 130D           JEQ  BS                 GO DO BACKSPACE
0288 01D4 0281           CI   1,>7F00            DELETE LINE?
     01D6 7F00
0289 01D8 130C           JEQ  DEL                GO DELETE LINE
0290 01DA D881           MOVB 1,@KYBUF(2)        PUT CHAR IN BUFFER
     01DC FEF3
0291 01DE 0582           INC  2                  CHAR PTR TO NXT LOC
0292 01E0 0282           CI   2,72               END OF BUFFER?
```

Figure 5-16. Example Program Using Timer Interrupts 3 and 4 (Sheet 9 of 12)

```
          01E2 0048
0293 01E4 1308              JEQ    LINE              YES, FORCE LINE DONE
0294 01E6 0281              CI     1,>0D00           CARRIAGE RETURN ?
          01E8 0D00
0295 01EA 1308              JEQ    LINEX             YES, SET END-OF-LINE
0296 01EC 0380      GOBACK  RTWP                     DONE
0297               *        SPECIAL CHARACTER HANDLING ROUTINES
0298 01EE 0602      BS      DEC    2                 MOVE INDEX BACK
0299 01F0 10FD              JMP    GOBACK
0300 01F2 04C2      DEL     CLR    2                 CLEAR INDEX
0301 01F4 10C6              JMP    RETURN
0302               *        BUFFER OVERFLOW HANDLING ROUTINE
0303 01F6 D8A0      LINE    MOVB   @CRX,@KYBUF(2)    FORCE <CR>
          01F8 0208
          01FA FEF3
0304 01FC 0582      LINEX   INC    2                 BUMP POINTER FOR NULL BYTE
0305 01FE D8A0              MOVB   @CRX+1,@KYBUF(2)  NULL OUT END OF LINE
          0200 0209
          0202 FEF3
0306 0204 0700      CR      SETO   0                 TURN LINE FLAG ON
0307 0206 10BD              JMP    RETURN
0308 0208 0D00      CRX     DATA   >0D00
```

Figure 5-16.   Example Program Using Timer Interrupts 3 and 4 (Sheet 10 of 12)

```
0310                   *----------------------------------------------------------------------
0311                   *        DECIMAL TO HEXADECIMAL CONVERSION ROUTINE
0312                   *----------------------------------------------------------------------
0313 020A FF78   DECHEX DATA >FF78,DECH1
     020C 020E'
0314 020E C03D   DECH1  MOV  *13+,0           GET 4 LSD'S
0315 0210 C05D          MOV  *13,1            GET 1 MSD
0316 0212 064D          DECT 13               RESTORE OLD WP
0317 0214 0202          LI   2,4              SET UP COUNTER
     0216 0004
0318 0218 0203          LI   3,MULT           ADDRESS OF MULTIPLY TABLE
     021A 0248'
0319 021C 04C4          CLR  4                CLEAR SUM
0320 021E C173   DECH2  MOV  *3+,5            GET MULTIPLIER
0321 0220 C180          MOV  0,6              COPY OVER INPUT
0322 0222 0246          ANDI 6,>F             STRIP WANTED DIGIT
     0224 000F
0323 0226 C186          MOV  6,6              IS NEW DIGIT ZERO ?
0324 0228 1303          JEQ  DECH4            YES, SKIP ADDITIONS
0325 022A A105   DECH3  A    5,4              ADD INTO SUM
0326 022C 0606          DEC  6                DECREMENT COUNTER
0327 022E 16FD          JNE  DECH3            IF NOT DONE, JUMP BACK
0328 0230 0940   DECH4  SRL  0,4              MOVE NEXT DIGIT OVER
0329 0232 0602          DEC  2                DECREMENT DIGIT COUNTER
0330 0234 16F4          JNE  DECH2            IF NOT ALL DIGITS, JUMP
0331 0236 0241          ANDI 1,>F             LOOK AT MSD ONLY
     0238 000F
0332 023A 1304          JEQ  DECH6            IF ZERO, EXIT
0333 023C C153          MOV  *3,5             GET 10 K VALUE
0334 023E A105   DECH5  A    5,4              ADD IT ON
0335 0240 0601          DEC  1                DECREMENT THE COUNTER
0336 0242 16FD          JNE  DECH5            IF NOT ZERO, JUMP
0337 0244 C744   DECH6  MOV  4,*13            PUT DATA IN OLD REGS.
0338 0246 0380          RTWP                  RETURN
0339 0248 0001   MULT   DATA 1,10,100,1000,10000
     024A 000A
     024C 0064
     024E 03E8
     0250 2710
```

Figure 5-16.   Example Program Using Timer Interrupts 3 and 4 (Sheet 11 of 12)

```
0341                   *----------------------------------------------------------------
0342                   *     HEXADECIMAL TO DECIMAL CONVERSION ROUTINE
0343                   *----------------------------------------------------------------
0344 0252 FF78  HEXDEC DATA >FF78,HEXD1
     0254 0256'
0345 0256 C0BD  HEXD1  MOV  *13+,2          GET HEX VALUE
0346 0258 04C0         CLR  0               CLEAR RETURN VALUE
0347 025A 0204         LI   4,4             SET UP COUNTER
     025C 0004
0348 025E 0205         LI   5,10            DIVISOR IS 10
     0260 000A
0349 0262 0B40  HEXD2  SRC  0,4             MAKE ROOM FOR NEW DATA
0350 0264 C082         MOV  2,2             IS QUOTIENT > 0 ?
0351 0266 130C         JEQ  HEXD3           IF NO, JUMP
0352 0268 C0C2         MOV  2,3             SET UP FOR NEXT DIVIDE
0353 026A 04C2         CLR  2               CLEAR UPPER HALF OF DOUBLEWORD
0354 026C 3C85         DIV  5,2             DIVIDE BY 10
0355 026E E003         SOC  3,0             PUT NEW DATA IN 0
0356 0270 0604         DEC  4               DECREMENT COUNTER
0357 0272 16F7         JNE  HEXD2           IF NOT DONE, JUMP BACK
0358 0274 0B40         SRC  0,4             MOVE DATA OVER 1 NIBBLE
0359 0276 C042  HEXD4  MOV  2,1             SET UP MSD
0360 0278 C741         MOV  1,*13           PUT DATA IN CALLER REG.1
0361 027A 064D         DECT 13              OLD WP ADDRESS
0362 027C C740         MOV  0,*13           PUT DATA IN CALLER REG.0
0363 027E 0380         RTWP                 EXIT
0364 0280 0B40  HEXD3  SRC  0,4             MOVE DATA OVER
0365 0282 0604         DEC  4               DECREMENT COUNTER
0366 0284 16FD         JNE  HEXD3           IF NOT DONE, CONTINUE SHIFTING
0367 0286 10F7         JMP  HEXD4           GO XFER DATA AND EXIT
0368            *
0369            *       PROGRAM END
0370            *
0371                   END
```

0000 ERRORS

Figure 5-16. Example Program Using Timer Interrupts 3 and 4 (Sheet 12 of 12)

5-10   MOVE BLOCK FOLLOWING PASSING OF PARAMETERS

The coding in Figure 5-17 is an example of a called subroutine that will move a block of data from one location to another. The three parameters of (1) move-from address, (2) move-to address, and (3) length of block are provided to the subroutine either through registers 0 to 2, or by the three words following the calling program's BLWP insruction, or by a combination of both. The block move subroutine first interrogates the words following the calling program's BLWP instruction; if a zero is found, it looks in a register for the parameter. In Figure 5-17, the calling program provides the move-from and block length parameters in registers, and the move-to parameter in the second word following the BLWP instruction.

```
        .
        .
        .
        LI      R0,>F100        MOVE-FROM ADDRESS
        LI      R2,125          MOVE 125 BYTES
        BLWP    @MVBLK          BRANCH TO SUBROUTINE
        DATA    0               MOVE-FROM ADDR IN R0
        DATA    >F200           MOVE-TO ADDRESS
        DATA    0               BYTE COUNT IN R2
        .
```

(a)   Calling Program

```
        .
        .
        .

MVBLK   DATA    >FF90,MVBLK1    WP, PC OF SUBROUTINE
MVBLK1  MOV     13,12           SAVE WP
        MOV     *14+,1          GET "FROM" ADR
        JNE     MVBLK2          NON-ZERO: PARM IN-LINE
        MOV     *13+,1          PICK UP FROM REG INSTEAD
MVBLK2  MOV     *14+,2          GET "TO" ADR
        JNE     MVBLK3          PARM IN IN-LINE CODE
        MOV     *13+,2          GET FROM REGS
MVBLK3  MOV     *14+,3          GET LENGTH
        JNE     MVBLK4          IN-LINE PARM
        MOV     *13,3           GET FROM REGS
MVBLK4  MOVB    *1+,*2+         MOVE BYTE
        DEC     3               ONE LESS TO GO
        JNE     MVBLK4          NOT DONE YET
        MOV     12,13           RESTORE WP
        RTWP                    RETURN TO CALLING PROGRAM
        .
        .
        .
```

(b)   Move Block Subroutine

Figure 5-17.   Move Block of Bytes Example Subroutine

5.11 BLOCK COMPARE SUBROUTINE

Figure 5-18 shows a sample block-compare subroutine which accepts three parameters from the calling program, in the same manner as the block-move subroutine (paragraph 5.10.1). This compare subroutine inspects two strings, comparing successive bytes until an unequal byte is found or until the specified string length is exhausted. The Status Register bits in register 15 are updated accordingly, and the subroutine returns to the calling routine with the altered status bits, which may be used immediately for conditional jumps.

The sample calling program is at the top of Figure 5-18. Note that the conditional jumps follow directly after the calling code, so the calling program simply compares (through the subroutine) and jumps, in the normal programming manner.

```
              .
              .
              .
        LI      R0,>F100    FIRST BLOCK START ADDRESS
        LI      R1,>F200    SECOND BLOCK START ADDRESS
        BLWP    @CMBLK      BRANCH TO SUBROUTINE
        DATA    0           START ADDR. IN R0 (1ST BLOCK)
        DATA    0           START ADDR. IN R1 (2ND BLOCK)
        DATA    100         COMPARE 100 BYTES
        JLE     $+10        IF LESS THAN OR EQUAL, JUMP
        JGT                 IF GREATER THAN, JUMP
              .
```

(a) Calling Program

```
              .
              .
              .

CMBLK   DATA    >FF90,CMBLK1  WP, PC OF SUBROUTINE
CMBLK1  MOV     13,12         SAVE WP
        MOV     *14+,1        GET "A" ADR
        JNE     CMBLK2
        MOV     *13+,1        GET IN CALLER REG
CMBLK2  MOV     *14+,2        GET  "B" ADDR
        JNE     CMBLK3
        MOV     *13+,2        GET FROM IN CALLER REG
CMBLK3  MOV     *14+,3        GET LENGTH
        JNE     CMBLK4
        MOV     *13,3         GET FROM REG
CMBLK4  CB      *1+,*2+       LOOK AT STRINGS
        JNE     CMBLK5        FOUND UNEQUAL
        DEC     3             ONE LESS BYTE
        JNE     CMBLK4        STILL MORE TO LOOK AT
CMBLK5  STST    15            STORE FINAL STATUS
        RTWP                  RETURN TO CALLING PROGRAM
              .
              .
              .
```

(b) Compare Block Subroutine

Figure 5-18. Compare Blocks of Bytes Example Subroutine

5.12 UNIT ID DIP-SWITCH

The Unit ID switch is a very versatile piece of hardware. The practical uses of this small device are limited only by the imagination. The proper way to read the switch settings is shown in Figure 5-19.

One example use of the switch is in a multidrop environment where each board on the communications line is assigned an ID number through the settings on the switch. The same software can be used in all the boards in the system, instead of having to maintain up to 32 separate copies, each unique only in an I.D. field. Figure 5-20 shows an example program segment in a communications routine.

Another example for use is in systems configuration. Whereas the main communications port (P2) is designed for use specifically for a terminal, the auxiliary communications port (P3) is a general purpose RS-232 port and can be connected to modems, serial line printers, device interfaces such as cassette or floppy disk controllers, etc., as well as terminals. The switch can be set to indicate the nature and baud rate of the device attached to the remote port. Figure 5-21 shows a program segment example.

5.13 CRU ADDRESSABLE LED

The light-emitting diode (LED) DS1 on the TM 990/101M is addressable through the CRU at software base address 0000_{16}. Writing a zero to the LED turns it on and writing a one turns it off. Figure 5-22 shows a sample routine to blink the LED on and off once a second, using the TMS 9901 timer. The LED is on for one-quarter second and off for three-quarters of a second.

5.14 CRU ADDRESSABLE LED

The TIBUG XOP routines (XOP 8 to 14) are written to accomplish input and output through a TMS 9902. When the TIBUG monitor is entered, the address for all I/O is directed to the main TMS 9902 (through connector P2). Any time a user program branches back into TIBUG at address 0080_{16} or when a RESET function is activated, the CRU address is set to the main TMS 9902. However, a user program may use all of the above-mentioned XOP calls to program any TMS 9902 in the system by first moving the software base address of the desired TMS 9902 into R12 of the I/O routines; this register is located at M.A. $FFDE_{16}$. In other words, move the software base address for the TMS 9902 (board addresses shown in Table 5-3) into memory address $FFDE_{16}$. Figure 5-23 is an example where both serial I/O ports of the TM 990/101M are activated for conversation to each other. Two terminals are assumed to be connected, one to each port, and the operators may type messages to each other. This principle can be expanded to support any of a number of TMS 9902-controlled serial I/O ports. (A variety of custom line interfaces may be used with a TMS 9902.)

The write character XOP service routine first ensures that the Request-to-Sent signal is active. This signal is not deactivated by TIBUG so that modem users will retain their data carrier. If a modem user wishes to drop the data carrier, the affected TMS 9902 must be addressed by the user program, and then the Request-to-Send signal is deactivated through the CRU.

Only the main TMS 9902, at CRU software base address 0080_{16} is initialized by TIBUG; other TMS 9902's in a system must be initialized by the user. Note the first portion of the example program in Figure 5-23. Part of TIBUG's initialization is to sense the baud rate of the attached terminal. If the baud rate is 110, 300, or 1200 baud, then the XOP routine waits 200 milliseconds after transmitting a carriage return. In addition, 1200 baud causes every character transmitted to be followed by 25 milliseconds of delay time. Only at 2400 and 9600 baud are characters transmitted without delays.

For 110, 300, and 1200 baud, the monitor ASRFLAG is set to one to cause a wait state following writing of a carriage return. If the TIBUG I/O XOP routines are used for other I/O ports, the state of the monitor's ASRFLAG will also govern delay loops used by the Write Character XOP. The user should then swap out the contents of the ASRFLAG (memory address $FFF4_{16}$) with one of the three values of ASRFLAG as listed in Table 5-7.

SWITCH 5 LSB WHEN READ
SET TO ON, ZERO READ (GROUNDED)

SET TO OFF, ONE READ
SWITCH 1 MSB WHEN READ
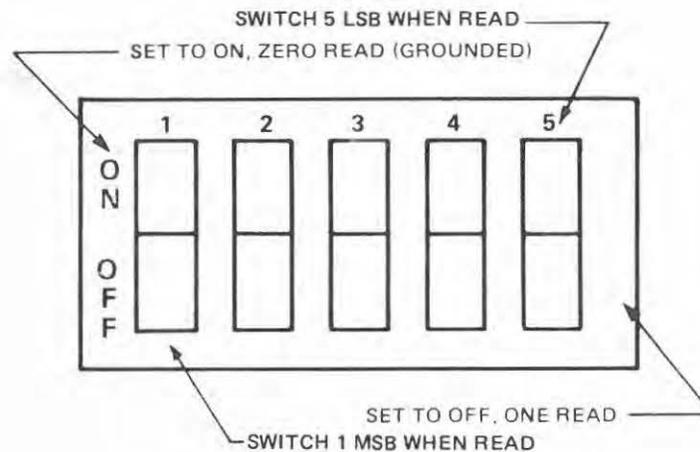
NOTE

If all five switch settings are stored (using CRU), switch 1 would be the MSB and switch 5 would be the LSB. For example, if switch 1 was set to OFF, and the others set to ON, storage of the five settings would be represented by 10_{16} or 10000_2. Code to store the switch contents in register 0 is shown below:

```
*READING THE DIP SWITCH
      CLR     R0          CLEAR HOLDING AREA
      LI      R12,>40     DIP SWITCH ADDRESS
      STCR    R0,5        SWITCH VALUES IN REGISTER 0
```

Figure 5-19. Reading the DIP Switch

5-53

```
* MULTIDROP SYSTEM WITH DIP SWITCH
* REGISTER 1 CONTAINS DESIRED ID VALUE
          CLR   RO              CLEAR HOLDING AREA
          LI    R12,>40         DIP SWITCH ADDRESS

          STCR  RO,5            SWITCH VALUES IN REG. O
          C     RO,R1           IS MESSAGE FOR ME?
          JEQ   PROCES          YES, GO PROCESS IT
          BLWP  @CLRBUF         NO, CLEAR BUFFER

          RTWP                  RETURN BACK TO RESCHEDULE
```

Figure 5-20. Example Code The Check Board ID at DIP Switch
for Multidrop Environment

```
* SYSTEMS CONFIGURATION EXAMPLE
          CLR   RO              CLEAR HOLDING AREA
          LI    R12,>40         DIP SWITCH CRU ADDRESS

          STCR  RO,5            SWITCH VALUES IN REG. O
          LI    R1,>10          LOAD "1" BIT FOR WALKING COMPA

          CZC   R1,RO           IS REMOTE PORT USED?
          JNE   NOTUZD          NO, JUMP OUT OF ROUTINE
          SRL   R1,1            SET TO >08 FOR CHECK
          CZC   R1,RO           ID2: IS TERMINAL CONNECTED?
          JEQ   TERMNL          YES, ID3, ID4, ID5 = BAUD RATE
          SRL   R1,1            NO, SET TO >04 FOR CHECK
          CZC   R1,RO           ID3: MODEM CONNECTED?
          JEQ   MODEM           YES, ID4, ID5 = MODEM TYPE
          SRL   R1,1            NO, SET TO >02 TO CHECK ID4
          CZC   R1,RO           ID4: I/O DEVICE CONTROLLER?
          JEQ   IODEV           YES, ID5, 1 = TAPE, O = FLOPPY
          SRL   R1,1            NO, SET TO >01 TO CHECK ID4
          CZC   R1,RO           ID5: SERIAL LINE PRINTER?
          JEQ   PRNTR           YES
          XOP   @SYSERR,14      NO, PRINT ERROR MESSAGE

*                               BECAUSE WRONG SWITCH SETTINGS
```

Figure 5-21. Coding Example to Ascertain System Configuration
Through DIP Switch Settings

```
0001                              IDT  'BLINK'
0002                 *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
0003                 *     THIS PROGRAM SETS UP THE INTERRUPT LINKING AREA AND THE
0004                 *     TIMER AT THE TMS 9901. IT EXECUTES THE TIMER. WHEN THE
0005                 *     THE TIMER COUNTS DOWN, AN INTERRUPT IS EXECUTED THROUGH
0006                 *     INTERRUPT TRAP 3 WHICH TRANSFERS CONTROL TO THE ISR AT
0007                 *     THE BOTTOM OF THIS LISTING. THE CALLING PROGRAM AND ISR
0008                 *     USE THE SAME WORKSPACE (>FF00). THIS PROGRAM IS CODED
0009                 *     AT ABSOLUTE ADDRESSES USING THE AORG ASSEMBLER DIRECTIVE
0010                 *     THUS, IT CAN BE CODED USING THE LINE-BY-LINE ASSEMBLER
0011                 *     WITH THE SLASH COMMAND USED INSTEAD OF THE AORG COMMAND.
0012                 *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
0013                 *
0014                 *  CALLING PROGRAM
0015                 *
0016 FC00                     AORG >FC00              BEGIN CODE AT M.A. >FC00
0017                 ** SET UP INT3 LINKING AREA
0018 FC00 02E0               LWPI >FF00              WORKSPACE ADDR (FOR BOTH PGMS)
     FC02 FF00
0019 FC04 C060                MOV  @>000E,1           INT3 PC VECTOR TO R1
     FC06 000E
0020 FC08 0202                LI   2,>0420            PLACE BLWP MACH. CODE IN R2
     FC0A 0420
0021 FC0C CC42                MOV  2,*1+              MOVE BLWP CODE TO LINK AREA
0022 FC0E 0202                LI   2,>FD00            ADDRESS OF VECTORS TO ISR
     FC10 FD00
0023 FC12 CC42                MOV  2,*1+              MOVE VECTOR ADDR TO LINK AREA
0024 FC14 0202                LI   2,>0380            PLACE RTWP MACHINE CODE IN R3
     FC16 0380
0025 FC18 C442                MOV  2,*1               MOVE RTWP TO LINK AREA
0026                 ** LOAD AND EXECUTE TIMER AT TMS 9901
0027 FC1A 0300                LIMI 0                  DISABLE INTERRUPTS
     FC1C 0000
0028 FC1E 020C                LI   12,>0100           TMS 9901 CRU ADDRESS
     FC20 0100
0029 FC22 0203                LI   3,>0300            CLOCK MODE, COUNT = 1
     FC24 0300
0030 FC26 0204                LI   4,>0800            INTERRUPT MODE, ENABLE INT3
     FC28 0800
0031 FC2A 0205                LI   5,3                INITIALIZE TIMER COUNTER
     FC2C 0003
0032 FC2E 3104                LDCR 4,4                ENABLE INT3 AT 9901
0033 FC30 3083                LDCR 3,2                START CLOCK AT 9901
0034 FC32 04CC                CLR  12                 POINT TO L.E.D.
0035 FC34 1D00                SBO  0                  TURN L.E.D. OFF
0036 FC36 0300                LIMI 3                  ENABLE INT3 AT TMS 9900
     FC38 0003
0037 FC3A 10FF                JMP  $                  WAIT HERE FOR INTERRUPT
```

Figure 5-22.   Coding Example to Blink L.E.D. On and OFF (Sheet 1 of 2)

```
0039                    *
0040                    *  INTERRUPT SERVICE ROUTINE
0041                    *
0042 FD00                    AORG  >FD00              BEGIN CODE AT M.A. >FD00
0043 FD00 FF00               DATA  >FF00,>FD04        WP, PC OF ISR
     FD02 FD04
0044 FD04 0300               LIMI  0                  DISABLE INTERRUPTS
     FD06 0000
0045 FD08 020C               LI    12,>0100           TMS 9901 CRU ADDRESS
     FD0A 0100
0046 FD0C 1D03               SBO   3                  CLEAR INTERRUPT AT 9901
0047 FD0E 0209               LI    9,15625            1/4 SECOND COUNT FOR TMS 9901
     FD10 3D09
0048 FD12 0A19               SLA   9,1                SHIFT CLOCK COUNT
0049 FD14 0269               ORI   9,1                SET CLOCK MODE
     FD16 0001
0050 FD18 33C9               LDCR  9,15               START CLOCK
0051 FD1A 1E00               SBZ   0                  SET INTERRUPT MODE AT 9901
0052 FD1C 04CC               CLR   12                 L.E.D. CRU ADDRESS
0053 FD1E 0605               DEC   5                  DECREMENT COUNTER
0054                    ** SET L.E.D. TO ON OR OFF STATUS
0055 FD20 C145               MOV   5,5                REG. 5 = ZERO?
0056 FD22 1606               JNE   >FD30              NO, TURN OFF L.E.D.
0057 FD24 1E00               SBZ   0                  YES, TURN ON L.E.D.
0058 FD26 0205               LI    5,3                RELOAD INTERRUPT COUNT
     FD28 0003
0059 FD2A 0300               LIMI  3                  ENABLE INT3
     FD2C 0003
0060 FD2E 0380               RTWP                     RETURN TO PROGRAM
0061 FD30 1D00               SBO   0                  TURN OFF L.E.D
0062 FD32 0300               LIMI  3                  ENABLE INT3
     FD34 0003
0063 FD36 0380               RTWP                     RETURN TO PROGRAM
0064                         END

0000 ERRORS
```

NOTE: As an exercise, the user can load and execute this code:
(1) load the machine code values shown in column 3 into
the memory locations shown in column 2, or (2) reassemble;
if the Line-By-Line Assembler (LBLA) is used, substitute the
slash command for the AORG directive and follow the
DATA and TEXT statement conventions for the LBLA. Ex-
ecute using the E TIBUG command.

Figure 5-22. Coding Example to Blink L.E.D. On and Off (Sheet 2 of 2)

```
0001                          IDT    'TWOTRM'
0002                 *------------------------------------------------------------
0003                 *         TWO TERMINAL PROGRAM EXAMPLE
0004                 *         THIS ROUTINE INITIALIZES THE AUXILIARY I/O PORT
0005                 *         OF THE TM990/101M MICROCOMPUTER. BOTH SERIAL
0006                 *         PORTS ARE THEN USED IN A CONVERSATIONAL MODE
0007                 *         WITH EACH OTHER. THE PROCEDURE IS TO INSPECT
0008                 *         THE RECEIVE BUFFER BIT IN THE ADDRESSED TMS9902
0009   .             *         TO SEE IF A CHARACTER HAS BEEN ASSEMBLED
0010                 *         IN THE UART. IF SO, IT IS ECHOED TO THE
0011                 *         ORIGINATING TERMINAL, AND THEN TRANSMITTED
0012                 *         TO THE OTHER TERMINAL. THEN THE OTHER
0013                 *         TERMINAL IS INSPECTED FOR A CHARACTER, ETC.
0014                 *         THE POINTS TO NOTE ARE:
0015                 *              1)   THE AUXILIARY TMS9902 MUST BE INITIALIZED.
0016                 *              2)   THE OLD "ASR"-FLAG MUST BE SAVED,
0017                 *                   AND A NEW ONE DETERMINED FOR THE
0018                 *                   NEW TERMINAL (AUXILIARY PORT).
0019                 *              3)   EVERY WRITE OPERATION CONSISTS OF
0020                 *                   MOVING THE DESIRED ADDRESS TO TIBUG,
0021                 *                   AND MOVING THE DESIRED "ASR"-FLAG TO TIBUG.
0022                 *------------------------------------------------------------
0023 0000 02E0             LWPI REGS                    USE SPARE SPACE AT END OF PROG
     0002 00CC'
0024 0004 020C             LI   12,>0180                AUXILIARY PORT ADDRESS
     0006 0180
0025                 *         INITIALIZE AUXILIARY SERIAL PORT
0026 0008 1D1F             SBO  31                      RESET UART
0027 000A 1000             NOP                          RESET TIMING DELAY
0028 000C 3220             LDCR @CTL,8                   LOAD CONTROL CHARACTER
     000E 00B6'                                          .
0029 0010 1E0D             SBZ  13                      BYPASS INTERVAL REGISTER
0030 0012 04C0             CLR  0                       BAUD RATE LOOP COUNTER
0031 0014 04C2             CLR  2                       ASR FLAG FOR THIS PORT
0032 0016 1F0F    TSTSP    TB   15                      LOOK AT RIN
0033 0018 13FE             JEQ  TSTSP                   WAIT FOR USER TO TYPE SOMETHIN
0034 001A 0580    SPLOOP   INC  0                       UP BAUD LOOP COUNTER
0035 001C 1F0F             TB   15                      RIN NOW HAS A SPACE:
0036 001E 16FD             JNE  SPLOOP                  DROP OUT ON A MARK
0037 0020 0201             LI   1,TABLE                 BAUD RATE TABLE
     0022 00A2'
0038                 *         NOW INSPECT BAUD RATE TABLE FOR A LOOP
0039                 *         COUNT WHICH MATCHES, THEN LOAD BAUD RATE.
0040 0024 8C40    BDLOOP   C    0,*1+                   LOOK AT ATBLE LOOP COUNT
0041 0026 1202             JLE  MATCH                   IF < OR = WE HAVE A MATCH
0042 0028 05C1             INCT 1                       SKIP BAD BAUD RATE, NEXT LOOP
0043 002A 10FC             JMP  BDLOOP                  LOOK AT NEXT LOOP COUNT
0044 002C 3311    MATCH    LDCR *1,12                   LOAD BAUD RATE CONTROL VALUE
0045 002E C051             MOV  *1,1                    GET VALUE ITSELF
0046 0030 0281             CI   1,>01A0                 1200 BAUD ?
     0032 01A0
0047 0034 1103             JLT  HIRATE                  NO, HIGHER BAUD RATE
0048 0036 1603             JNE  BEGIN                   NO, LOWER BAUD RATE
0049 0038 0702             SETO 2                       SET LOCAL ASR FLAG
```

Figure 5-23. Example Program to Converse Through
Main and Auxiliary TMS 9902's (Sheet 1 of 3)

```
0050 003A 1001            JMP   BEGIN               AND PRINT BEGIN MESSAGE
0051 003C 0582   HIRATE  INC   2                   MARK NO <CR> DELAY
0052                      *     THE AUXILIARY PORT IS NOW UP. PRINT GREETING.
0053 003E C820   BEGIN   MOV   @X180,@XOPCRU        AUX. PORT ADR. TO TIBUG
     0040 00A0'
     0042 FFDE
0054 0044 C0E0           MOV   @ASRFLG,3           SAVE MAIN PORT ASR-FLAG
     0046 FFF4
0055 0048 C802           MOV   2,@ASRFLG           AUX. PORT ASR-FLAG
     004A FFF4
0056 004C 2F40           XOP   0,13                READ BY OLD INIT. CHAR.
0057 004E 2FA0           XOP   @BGNMSG,14          PRINT BEGIN MESSAGE
     0050 00B7'
0058 0052 C820           MOV   @X80,@XOPCRU        MAIN PORT ADR TO TIBUG
     0054 009E'
     0056 FFDE
0059 0058 C803           MOV   3,@ASRFLG           MAIN PORT ASR-FLAG
     005A FFF4
0060 005C 2FA0           XOP   @BGNMSG,14          PRINT BEGIN MESSAGE HERE, TOO
     005E 00B7'
0061                      *     THIS IS THE MAIN LOOP.
0062                      *     FIRST ADDRESS MAIN PORT, THEN THE AUXILIARY PORT
0063 0060 C320   LOOP    MOV   @X80,12             ADDRESS FOR MAIN PORT
     0062 009E'
0064 0064 1F15           TB    21                  CHARACTER TYPED HERE ?
0065 0066 160B           JNE   NEXT                NO, TRY OTHER PORT
0066 0068 C80C           MOV   12,@XOPCRU          YES, GIVE ADDRESS TO TIBUG
     006A FFDE
0067 006C C803           MOV   3,@ASRFLG           MOVE ASR-FLAG
     006E FFF4
0068 0070 2EC0           XOP   0,11               READ/ECHO CHAR TO ORIGINATING
0069 0072 C820           MOV   @X180,@XOPCRU       AUXILIARY PORT ADDRESS
     0074 00A0'
     0076 FFDE
0070 0078 C802           MOV   2,@ASRFLG           AUXILIARY PORT ASR-FLAG
     007A FFF4
0071 007C 2F00           XOP   0,12               WRITE CHARACTER TO OTHER TERMI
0072 007E C320   NEXT    MOV   @X180,12            ADDRESS FOR AUXILIARY PORT
     0080 00A0'
0073 0082 1F15           TB    21                  CHARACTER TYPED HERE ?
0074 0084 16ED           JNE   LOOP                NO, TRY MAIN PORT
0075 0086 C80C           MOV   12,@XOPCRU          YES, GIVE ADDRESS TO TIBUG
     0088 FFDE
0076 008A C802           MOV   2,@ASRFLG           MOVE ASR-FLAG
     008C FFF4
0077 008E 2EC0           XOP   0,11               READ/ECHO CHAR TO ORIGINATING
0078 0090 C820           MOV   @X80,@XOPCRU        MAIN PORT ADDRESS
     0092 009E'
     0094 FFDE
0079 0096 C803           MOV   3,@ASRFLG           MAIN PORT ASR-FLAG
     0098 FFF4
0080 009A 2F00           XOP   0,12               WRITE CHARACTER TO MAIN TERMIN
0081 009C 10E1           JMP   LOOP
0082                      *------------------------------------------------------
```

Figure 5-23. Example Program to Converse Through
Main and Auxiliary TMS 9902's (Sheet 2 of 3)

```
0083                    *       DATA AREA
0084                    *
0085 009E 0080   X80     DATA  >0080              MAIN PORT R12 BASE ADDRESS
0086 00A0 0180   X180    DATA  >0180              AUXILIARY PORT R12 BASE ADDRES
0087      FFF4   ASRFLG  EQU   >FFF4              TIBUG ASR FLAG ADDRESS
0088      FFDE   XOPCRU  EQU   >FFDE              TIBUG XOP R12 ADDRESS
0089 00A2 0010   TABLE   DATA  >10,>34            9600 BAUD
     00A4 0034
0090 00A6 0040           DATA  >40,>D0            2400 BAUD
     00A8 00D0
0091 00AA 0070           DATA  >70,>1A0           1200 BAUD
     00AC 01A0
0092 00AE 0200           DATA  >200,>4D0          300 BAUD
     00B0 04D0
0093 00B2 0400           DATA  >400,>638          110 BAUD
     00B4 0638
0094 00B6   62   CTL     BYTE  >62                9902 CONTROL
0095 00B7   OD   BGNMSG  BYTE  >0D,>0A
     00B8   0A
0096 00B9   42           TEXT  'BEGIN OPERATION'
0097 00C8   OD           BYTE  >0D,>0A,>00
     00C9   0A
     00CA   00
0098 00CC 0000   REGS    DATA  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
     00CE 0000
     00D0 0000
     00D2 0000
     00D4 0000
     00D6 0000
     00D8 0000
     00DA 0000
     00DC 0000
     00DE 0000
     00E0 0000
     00E2 0000
     00E4 0000
     00E6 0000
     00E8 0000
     00EA 0000
0099                     END

0000 ERRORS
```

Figure 5-23. Example Program to Converse Through
Main and Auxiliary TMS 9902's (Sheet 3 of 3)

Table 5-7. ASRFLAG Values

ASRFLAG Value*	Recommended Baud Rate	Description/Recommendations
Positive No.	2400, 9600	No delays. Use for CRT's, modems.
Zero	110, 300	Carriage Return Delay only. Use for hardcopy terminals.
Negative No.	1200	Carriage Return and Character padding delays. Use with "T" command if terminal is not a TI 733.

*ASRFLAG located in RAM at M.A. $FFF4_{16}$

SECTION 6

THEORY OF OPERATION

6.1 GENERAL

This section presents the theory of operation of the TM 990/101M microcomputer. Information in the following manuals can be used to supplement material in this section:

● TMS 9900 Microprocessor Data Manual

● TMS 9901 Programmable Systems Interface Data Manual

● TMS 9902 Asynchronous Communications Controller Data Manual

● TTL Data Book, Second Edition

● Bipolar Microcomputer Components Data Book

● The MOS Memory Data Book.

Figure 6-1 shows a block diagram of the TM 990/101M, highlighting the four major buses:

● Address bus

● Control bus

● Data bus

● Communications register unit (CRU) bus

In normal operation the TMS 9900 microprocessor commands the address bus and most of the control bus; the data bus is bidirectional, driven by both the microprocessor and the memory devices. The two-line CRU bus is not bidirectional; the serial output line is microprocessor driven and the serial input line is driven by the CRU device.

The major features of the TM 990/101M microcomputer board are the clock driver, the microprocessor, the TMS 9901, the two TMS 9902's and peripheral circuitry, the bidirectional and normal backplane buffers, the EPROM, the RAM, the additional CRU devices, and the miscellaneous signals. These features are discussed in the following paragraphs of this section.

6.2 POWER SPECIFICATIONS

Approximate power values required by the TM 990/101M-1 are listed in the following table:

	Current			
	-12V	+5V	+12V	Watts
Typical	0.2A	1.8A	0.25A	15.0W
Maximum	0.35A	2.5A	0.3A	19.7W

Figure 6-1. TM 990/101M Block Diagram

6-2
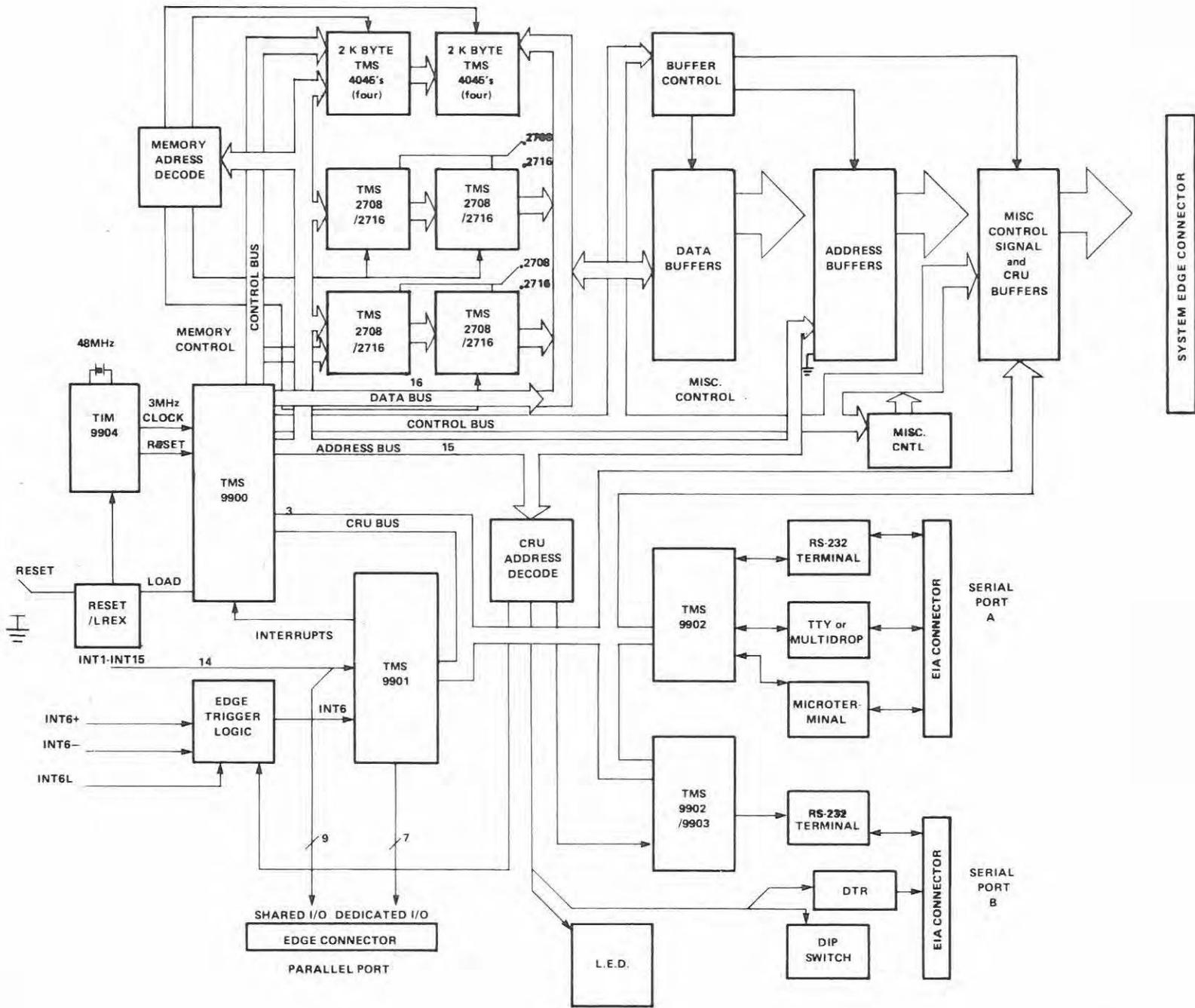
The supply -5V is derived on the board by the UA7905 regulator from the -12V line supplied from off board. The -5V supply is used primarily by the TMS 9900 microprocessor and the TMS 2708/2716 EPROM's for back-biasing the substrate, and by the multidrop interface as a supply voltage. The -12V supply is used for the EIA line drivers as well as for supplying the voltage to the -5V regulator.

The +12V supply is used by the TMS 9900 microprocessor and the TMS 2708/2716 EPROM's as the main voltage supply since these are MOS parts. The +12V also is used for the EIA line drivers.

All integrated circuits on the board, except the EIA line drivers, use the +5V supply, and because of the heavy load this voltage is not derived by an on-board regulator but must be supplied from off the board. The MOS parts use this supply for TTL compatibility, and, in fact, the TMS 9901, 9902, 9903, and 4045 use only this voltage for supply since they contain internal charge pumps, eliminating the need for -5 or +12V in their operation.

Table 6-1 lists the pin assignments of each integrated circuit for the supply voltages each uses.

Table 6-1. Device Supply Voltage Pin Assignments

Device	SUPPLY VOLTAGES TO PIN NUMBER				
	-12V	-5V	GND	+5V	+12V
TMS 9900		1	26,40	2,59	27
TMS 9901			16	40	
TMS 9902			9	18	
TMS 9902/03 socket			9	20	
TMS 9904			3,10	20	13
TMS 4045			9	18	
TMS 2708/2716		21	12	24	19
74LS241, 74LS245			10	20	
75188	1		7		14
75189			7	14	
75154			8	15	
75107		13	7	14	
75112		11	7	14	
74LS138, 153, 251, 259; 74S287			8	16	
74LSXX			7	14	

6.3 SYSTEM STRUCTURE

The block diagram in Figure 6-1 shows the system structure of the TM 990/101M microcomputer board. The microcomputer design centers around five buses: power, control, address, data, and CRU. The major blocks of the system are the processor, the miscellaneous control signals, address decoding, on-board memory, the TMS 9901, and two TMS 9902 serial ports, and the miscellaneous CRU devices.

Functionally, these major blocks represent the processing, memory and I/O portions of the microcomputer.

Throughout the remainder of this section, each block's function is discussed, grouping the explanations into three categories: processing, memory, and I/O. The first subject is the buses since the buses tie all the blocks together.

The power bus is explained in paragraph 6.2 above, so the following paragraph deals with the remaining buses.

6.4 SYSTEM BUSES

The four major buses are subdivided by function in Table 6-2. By referring to the schematics in Appendix G, each random logic line as well as the bus lines can all be traced. All bus signals appear on connector P1.

6.4.1 ADDRESS BUS

The 16-line address bus consists of lines A0 through A15. Only 15 of these, A0 through A14, are normally used for addressing memory. Memory access deals with a 16-bit word, and A15, the byte address bit, is not brought out of the TMS 9900 since byte operations are handled by fetching a 16-bit word into the processor, and modifying the addressed byte, rewriting the 16-bit word back to memory if necessary. Therefore, A15 appears only on connector P1 and is grounded to show a zero off-board, thereby fetching words on even address boundaries.

On the board the address lines are routed to the address decoding PROM which, with MEMEN, selects on-board memory if the address presented lies within the limits of the memory map programmed into the PROM.

Lines A0, A1, and A2 also are routed to the 74LS138 external instruction decoder where, upon a CRUCLK pulse, the state of the address lines determines whether a CRU operation (A0, A1, A2 = 0) or an external instruction is occurring. This leaves A3 through A14 for CRU addressing; A3 through A14 are routed to the I/O decode logic and the CRU devices.

6.4.2 DATA BUS

The data bus consists of 16 bidirectional lines which are routed from the processor to the on-board memory and to the bidirectional buffers for off-board use. D0 is the most significant bit, and D15 is the least significant bit.

6.4.3 CRU BUS

The three lines in the CRU bus are CRUIN, CRUOUT, and CRUCLK. Whenever an address is present on the address bus and MEMEN is not also active, a CRU operation is assumed. Note that even if some CRU device responds to the address bus while it changes value or is in any way invalid, no harm is done because the data presented to CRUIN by the addressed device will be ignored by the processor. Since the processor will poll CRUIN only when required, CRU address decoding is simplified.

Table 6-2. Bus Signals

Signal	Functional Device Connections
Address Bus A0, A1, A2 A3, A4 A5, A6, A7, A8, A9 A10, A11, A12 A13, A14 (A15.B)	Address decode PROM, external instruction decode Address decode PROM, CRU decode logic, TMS 2716 EPROM CRU decode logic, all memory devices All memory devices, TMS 9901, TMS 9902/3, one 74LS251 All memory devices, TMS 9901, TMS 9902/3, both 74LS251's Byte indicator: always zero, off-board signal only
Data Bus D0-D7 D8-D15	Most significant byte, 1 EPROM/byte, 2 TMS 4045/byte Least significant byte, 1 EPROM/byte, 2 TMS 4045/byte
CRU Bus CRUIN CRUOUT CRUCLKB	CRU input line, TMS 9901, TMS 9902/3, 74LS251 (TIM9905) CRU output line, TMS 9901, TMS 9902/3 74LS259 (TIM9906) CRU clock, TMS 9901, TMS 9902/3, 74LS251 (TIM9905), 74LS259 (TIM9906), Edge-triggered logic
Control Bus MEMEN DBIN WE MEMCYC READY	Memory control: address decode PROM Memory control: RAM decode logic, data bus buffer control Memory control: RAM decode logic, all TMS 4045 RAM's Memory control: off-board only Memory control: slow EPROM logic, off-board WAIT state
Auxiliary Control ø1, ø3 EXTCLK.B, CLK.B PRES.B, RESTART.B, RST, LOAD, IORST.B INT1-INT6 INT7-INT15 HOLD, HOLDA IAQ	Clock: TMS 9901, TMS 9902/3, RESET/LOAD logic Clock: off-board only RESET/LOAD logic, TMS 9900, TMS 9901 Interrupt Control: dedicated TMS 9901 Interrupt Control: shared I/O, TMS 9901 Address, Data, Memory Control for DMA: TMS 9900 Miscellaneous: TMS 9900

When an address is present on the address bus and $\overline{\text{MEMEN}}$ is not active and if A0, A1, and A2 are all zero, the CRUCLK pulse is gated through the external instruction decoder, and any data on CRUOUT is strobed into the addressed CRU device. This is a CRU output operation, and it is distinct from an input operation in that CRUCLK is active during output; whereas, it is inactive upon input.

As mentioned above, CRU input is achieved by the processor asserting an address while keeping the $\overline{\text{MEMEN}}$ signal inactive, and then polling CRUIN at the appropriate time.

6.4.4 CONTROL BUS
This bus is not as homogenous as the other buses; therefore it is divided into groups as shown in Table 6-2. Table 6-3 gives a brief explanation of each function.

Table 6-3. Control Bus Functions

Signal	Active State	Group	Purpose
$\overline{\text{MEMEN}}$ (memory enable)	Low	Memory	Enables memory devices, address on address bus is for memory
DBIN (data bus input)	High	Memory	Shows state of processor's data bus: high is input to processor, low is output.
$\overline{\text{WE}}$ (write enable)	Low	Memory	Strobe to memory devices for writing data to memory.
$\overline{\text{MEMCYC}}$ (memory cycle)	Low	Memory	Indicates beginning and end of one memory cycle. For successive memory cycles, $\overline{\text{MEMEN}}$ can be active continuously, $\overline{\text{MEMCYC}}$ goes inactive between each separate memory cycle.
READY	High	Memory	Indicates memory is ready with read data on next clock, or has disposed of data on write cycle. Wait states are generated by pulling this line low.
WAIT	High	Memory	Acknowledges that memory is not ready, indicating a wait state.
$\overline{\text{HOLD}}$	Low	Processor Activity	Requests processor to give up control of address, data buses and $\overline{\text{MEMEN}}$, $\overline{\text{WE}}$, and DBIN.
HOLDA	High	Processor Activity	Acknowledges that processor has given up control of buses given above, and has suspended activity.

Table 6-3. Control Bus Functions (Concluded)

Signal	Active State	Group	Purpose
$\overline{\varnothing 1}$, $\overline{\varnothing 3}$	Low	Clock	TTL level clocks
$\overline{\text{EXTCLK}}$.B	Low	Clock	External TTL clock input to TIM 9904.
$\overline{\text{CLK}}$.B	Low	Clock	Output of internal oscillator of TIM 9904.
$\overline{\text{PRES}}$.B	Low	Reset/Load	Causes reset interrupt
$\overline{\text{RST}}$	Low	Reset/Load	Reset interrupt input, TMS 9900
$\overline{\text{RSET}}$	Low	Reset/Load	External instruction, causes IORST
$\overline{\text{IORST}}$	Low		I/O reset to TMS 9901's. Does not cause reset interrupt
$\overline{\text{RESTART}}$.B	Low	Reset/Load	Causes load function delayed by two IAQ or idle pulses. (LOAD is name of external instruction and load function pulse)
$\overline{\text{LOAD}}$	Low	Reset/Load	
$\overline{\text{INT}}$ 1-15	Low	Interrupt	Request for interrupt to TMS 9901
IAQ	High	Miscellaneous	Signifies this memory cycle to be an instruction fetch.

6.5. SYSTEM CLOCK

The system clock is generated by a crystal and tank circuit tuned to 16 times the desired system frequency. This network is attached to the TIM 9904 clock driver, which counts down the input signal from the tank and crystal into four non-overlapping clock phases at MOS signal levels for the TMS 9900. The inverse of these phases is output to TTL levels for the remainder of the system.

Also on the TIM 9904, the reset function is latched and synchronously presented to the TMS 9900; this ensures synchronization with the correct phase.

The crystal is a third overtone series/parallel-resonant crystal, set in an HC-18U holder (see Figure 6-2).

The TTL clocks are routed to the $\overline{\text{RESET}}$/$\overline{\text{LOAD}}$ and MEMCYC logic, as well as to the P1-connector and the TMS 9901 and TMS 9902/9903's.

```
CAUTION
```

If pins 11 and 12 of the TMS 9904 (∅1 and ∅2) are shorted, the device will overheat and go into thermal runaway almost instantly.
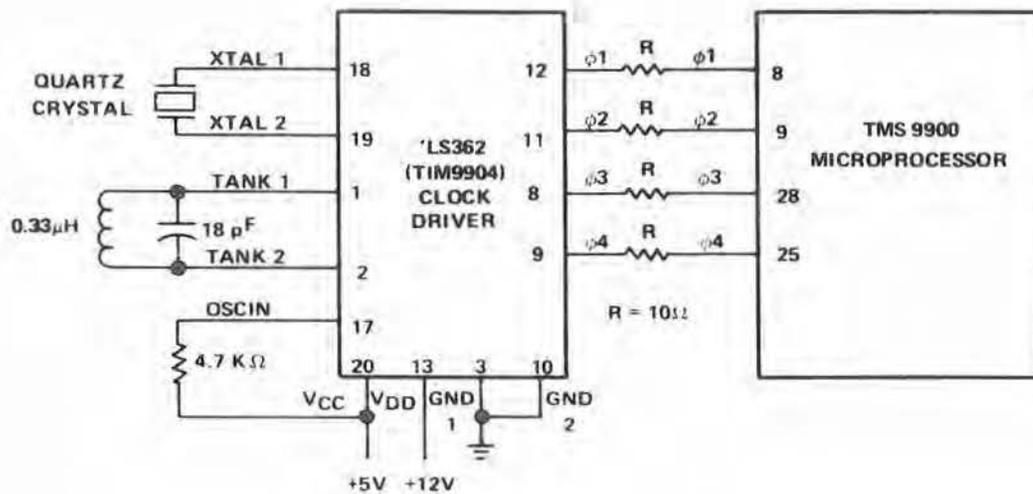
Figure 6-2. Crystal-Controlled Operation

6.6 CENTRAL PROCESSING UNIT

The TMS 9900 microprocessor is the central processing unit (CPU) for the TM 990/101M. The responsibilities of the CPU include:

- Memory, CRU and general bus control

- Instruction acquisition and interpretation

- Timing of most control signals and data

- General system initialization.

Figure 6-3 groups the TMS 9900 pins by function. The address bus addresses memory and the CRU devices, and provides the codes for the external instructions. The data bus carries all memory data, including instruction code as well as program data and addresses. Interrupt requests are encoded as a binary number by the TMS 9901 for presentation to the TMS 9900 microprocessor.

Memory operations are initiated by placing an address on the address lines along with MEMEN, DBIN, and eventually, WE. If the memory cycle is an instruction fetch, IAQ goes active also. READY is sampled and the memory cycle is ended one clock cycle after READY is active.
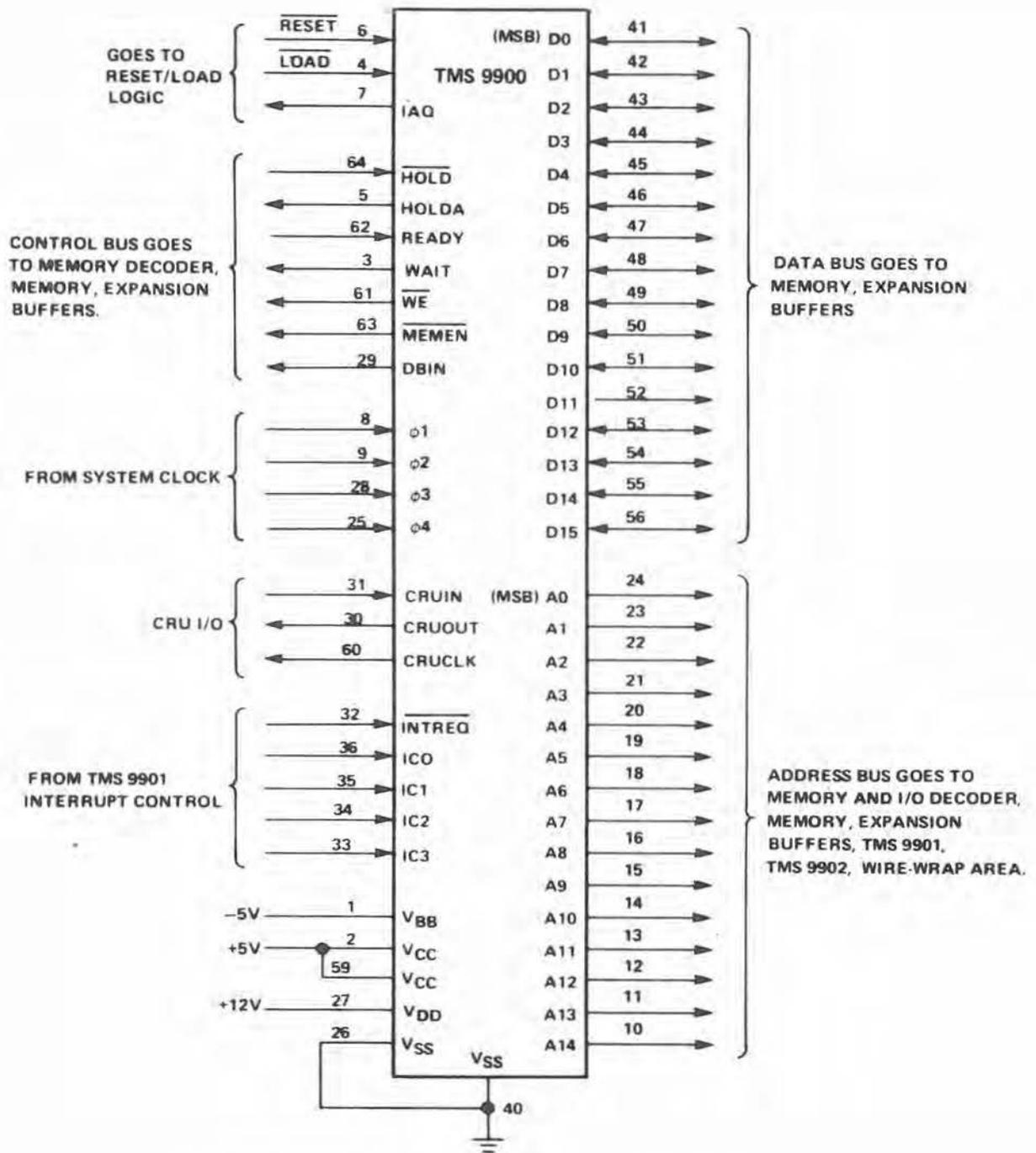
Figure 6-3. TMS 9900 Pin Functions

CRU operations are initiated by placing an address on the address bus. CRUIN is sampled for an input operation; otherwise it is ignored, and for an output operation the datum is placed on CRUOUT and strobed with CRUCLK. Aside from I/O purposes, CRU operations also program the operation of such devices as the TMS 9901, 9902, and 9903.

Figures 6-4 and 6-5 show the data flow and operational flowchart of the microprocessor. Figure 6-6 shows the decoding of the external instructions. For more information, refer to the TMS 9900 Microprocessor Data Manual.

6.7 RESET/LOAD LOGIC

After the clock and the CRU, the next block most closely associated with microcomputer operation is the random logic dealing with RESET and LOAD. This block initializes the system and is also used to return control to TIBUG when using single-step operation (refer to Figure 6-6).

6.7.1 RESET FUNCTION

The RESET pushbutton feeds a latch formed from back-coupled inverters for debouncing. The PRES.B signal from connector P1 joins the RESET pushbutton signal in a Schmitt trigger gate to assure that multiple reset pulses due to noise or bounce do not affect the microcomputer. After being inverted again, the reset signal is routed to the TIM 9904 which then synchronizes it with $\phi3$ and then presents the signal to the microprocessor.

The RESET signal also goes to two flip-flops which generate the IORST signal, which clears TMS 9901's and any other devices attached to it off-board. This IORST signal is also generated by the external instruction RSET, but it is important to realize that the RSET instruction in a program generates only IORST and not a full RESET interrupt. IORST can be active for up to two o3 clock periods.

Reset causes the following to occur:

- Clears I/O devices on IORST line (on board TMS 9901)

- Inhibits memory write and CRU operations

- Sets TMS 9900 status register interrupt mask to 0000

- Processor traps to vectors at 0000 and 0002

Reset is caused by:

- Actuating the RESET switch on the PC board

- Setting the PRES.B signal to a logic ZERO state on connector P1.
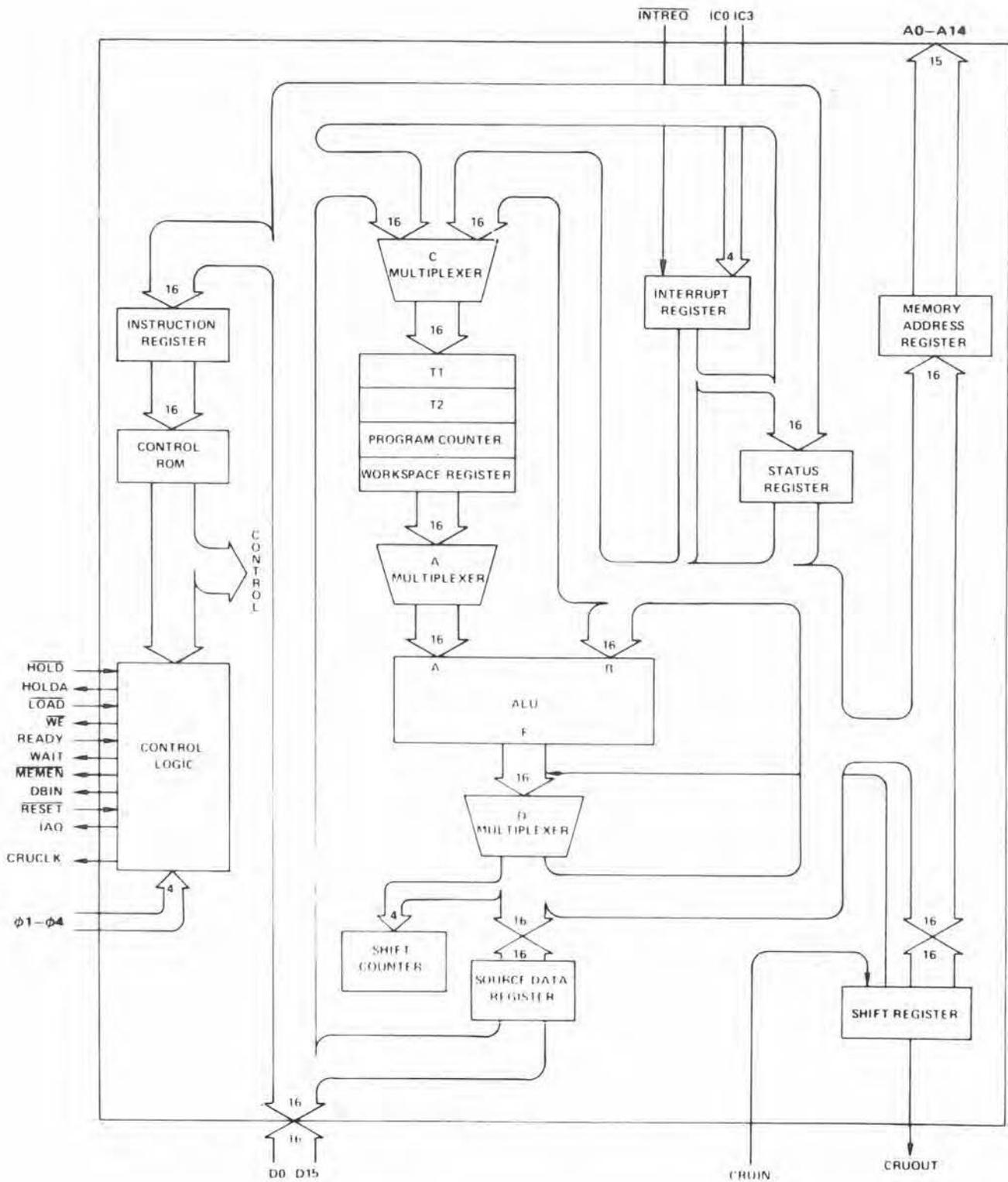
Figure 6-4. TMS 9900 Data and Address Flow
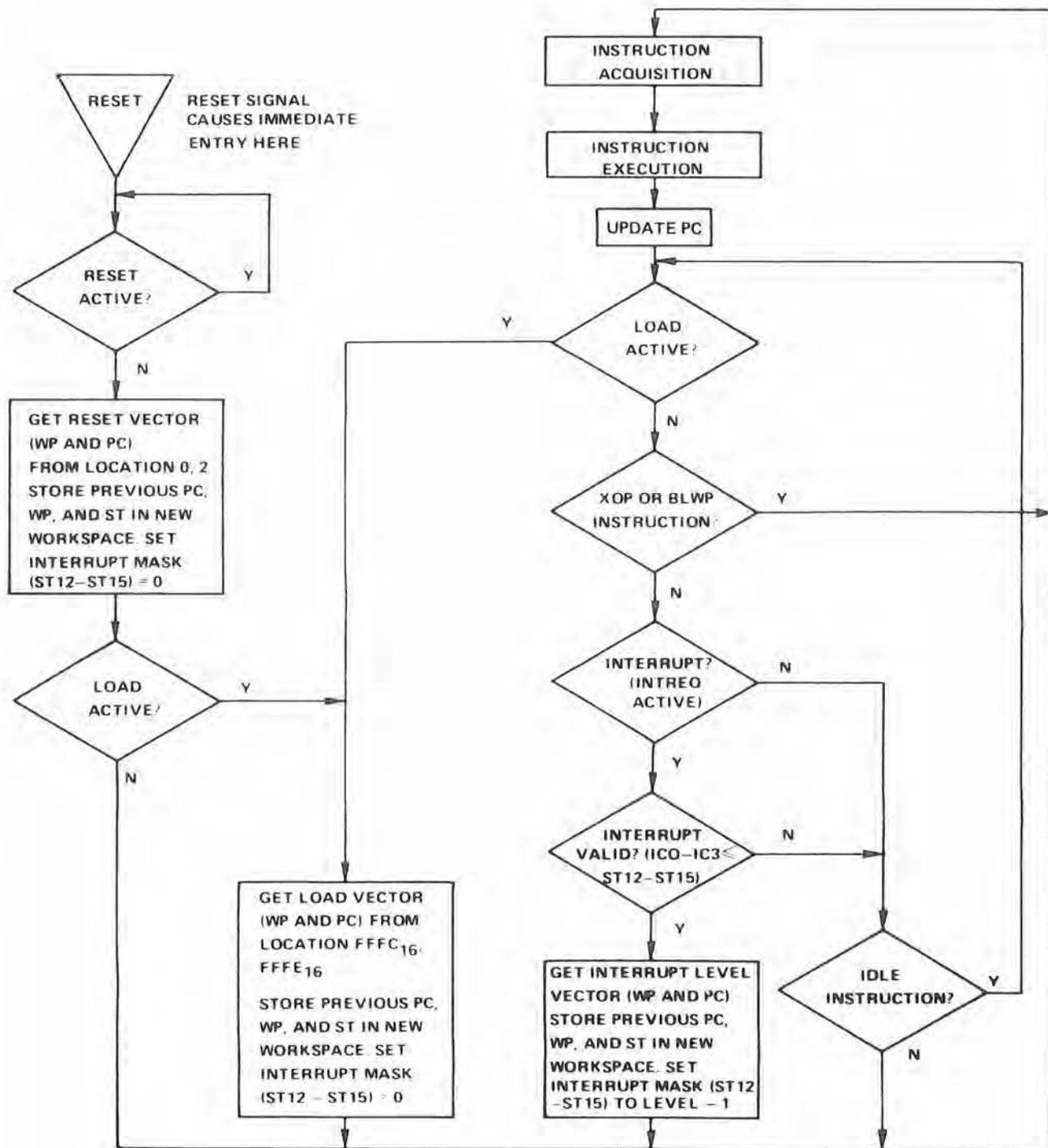
Figure 6-5. TMS 9900 CPU Flowchart
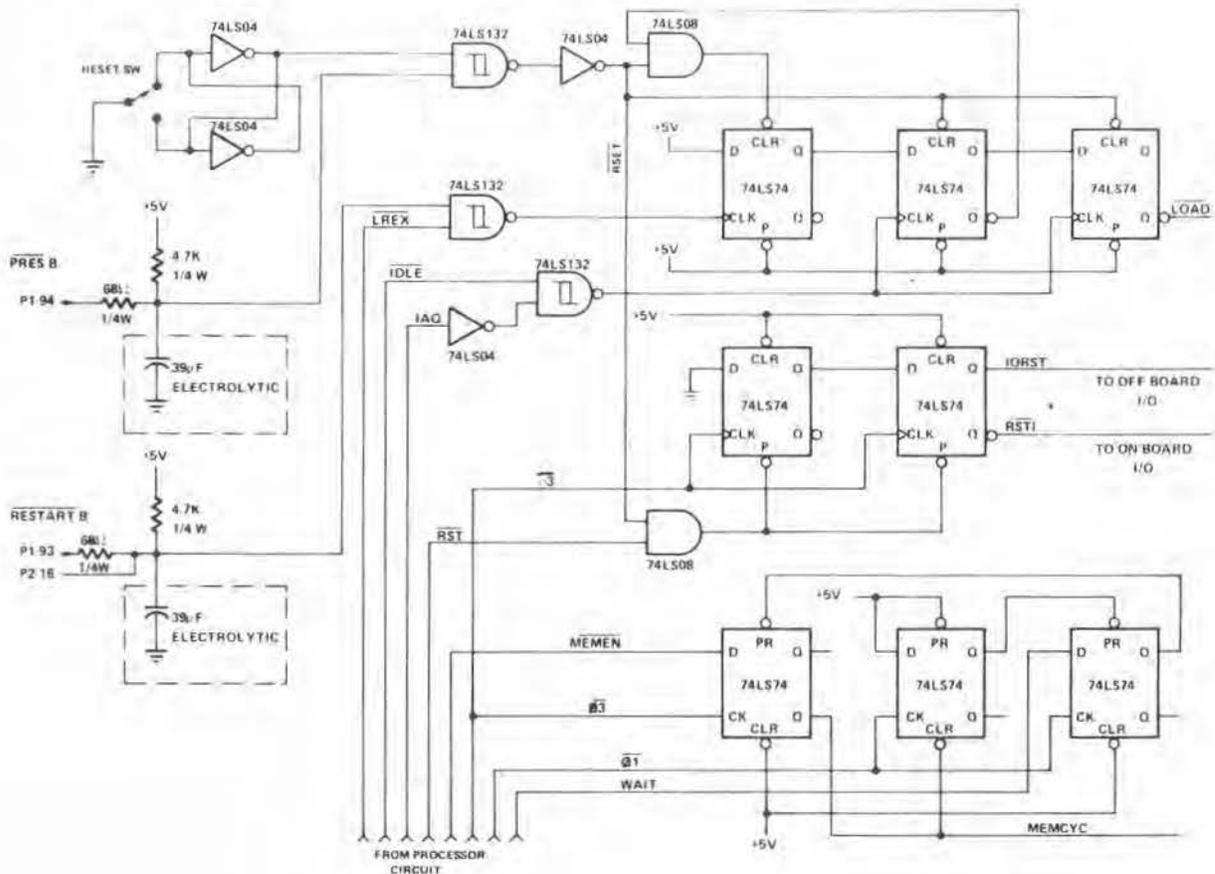
Figure 6-6. RESET and LOAD Logic

6.7.2 LOAD FUNCTION

The LOAD function is triggered by either activating RESTART.B or executing the external instruction LREX. Both of these are combined in the same way the RESET function is. The first flip-flop presents the LOAD request to the second, and the second and third flip-flops count two IAQ or IDLE pulses and then present the LOAD function request to the microprocessor. The second flip-flop clears the first one so that only one LOAD is generated even though, for instance, the RESTART.B signal may be continuously active.

RESET overrides LOAD because a RESET signal clears the LOAD flip-flops. This is important when both requests occur simultaneously.

Load causes the following to occur:

* LOAD function is delayed two instructions (IAQ) or idle pulses (IDLE), then triggered

6-13

● Processor traps to vectors at M.A. >FFFC and >FFFE

Load is caused by the following if RESET is inactive:

● Executing the software instruction LREX

● Setting $\overline{\text{RESTART}}$.B to logic ZERO state on connector P1.

6.7.3 RESET AND LOAD FILTERING

Installing a 39 microfarad capacitor at C18 will debounce the $\overline{\text{PRES}}$.B signal. This would be adequate for manual actuation by an SPST pushbutton to ground.

A 39 microfarad capacitor at C23 debounces the $\overline{\text{RESTART}}$.B signal, suitable for connection to a manually actuated switch in the same way as above.

These capacitors are user options, and these values are suggested values.

6.7.4 CLRCRU SIGNAL

The CLRCRU (clear CRU) signal is a power-up IORST which resets the edge-triggered interrupt 6, the status LED, and remote serial port Data Terminal Ready signal. The status LED is lighted and Data Terminal Ready is inactive.

6.8 EXTERNAL INSTRUCTIONS

The so-called external instructions are those which, when executed by the processor, cause address lines A0, A1, and A2 to be set to a state, and CRUCLK to become active. The instructions and descriptions are listed in Table 6-4.

Table 6-4. External Instructions

Instruction	Opcode	A0	A1	A2	Description
IDLE	0340	0	1	0	Suspends processor until RESET, LOAD, or interrupt occur
RSET	0360	0	1	1	Zeroes TMS 9900 interrupt mask, generates IORST
CKON	03A0	1	0	1	Not used on TM 990/101M
CKOF	03C0	1	1	0	Not used on TM 990/101M
LREX	03E0	1	1	1	Causes LOAD, delayed by two IAQ or IDLE pulses

The CKON and CKOF instructions are used by other 990-family systems to control the system timer. On the TM 990/101M the system timer is incorporated into the TMS 9901; hence, these instructions are not used.

The RSET instruction generates the IORST signal to clear all I/O devices (on board TMS 9901) attached to it. It also clears out the status register interrupt mask, which allows only a RESET interrupt or a LOAD function to be granted.

The LREX instruction causes a LOAD function request to be presented to the processor after two IAQ or IDLE pulses. This means that the LOAD function occurs after two instructions are executed following the LREX. TIBUG uses this function to do single step by executing the LREX, a RTWP to the user, then one user instruction. The LOAD function becomes active and vectors back to TIBUG, which then prints the processor registers.

IDLE causes the processor to suspend operation; it is, in essence, a HALT instruction. An interrupt or LOAD terminates the idle state.

In all cases, note that A0, A1, and A2 are nonzero values so that these instructions are differentiated from a CRU output operation.

6.9   ADDRESS DECODING
This subsection explains address decoding for both memory and CRU I/O along with their memory maps. The memory address map configurations are shown in Figure 6-7.

6.9.1   MEMORY ADDRESS DECODING

6.9.1.1   Memory Address Decoding PROM
The memory map is programmed in a 74S287 PROM as shown in Figure 6-8. The PROM is a 256 x 4 bit memory, and each four-bit word (DO4 to DO1) is used to determine memory to be enabled. The most significant bit of the PROM word, DO4, is the RAM enable control line. Programming a ZERO on DO4 will cause RAM to become active. Since there are two banks of RAM on the board and since there is no room on the PROM to decode the two banks separately, each bank is enabled by the state of address line A4. Therefore, all RAM is decoded by the PROM as a complete block and cannot be separated.

The next two bits of the PROM word (DO3 and DO2) enable each EPROM bank separately and directly. EPROM's are enabled by programming a zero.

The least significant bit of the PROM word (DO1) is a negative-logic "OR" of the other three bits of the PROM word. If any of the other three bits are zero, this bit must be zero also. This signal indicates to data bus buffer control whether memory addressed is on-board or off-board; a zero state indicates on-board memory.

Figure 6-7. TM 990/101M Memory Addressing

a) 2K EPROM (2708's) 2K RAM

Address	Contents	Address
0000	EPROM 1 (TMS 2708)	07FE
0800	EPROM 2 (TMS 2708)	0FFE
1000	OFF-BOARD MEMORY	
F000	RAM 2 TMS 4045	EFFE / F7FE
F800	RAM 1 TMS 4045	FFFE

b) 4K EPROM (2708's) 2K RAM

Address	Contents	Address
0000	EPROM 1 TMS 2716	
1000	EPROM 2 TMS 2716	0FFE
2000	OFF-BOARD MEMORY	1FFE
F000	RAM 2 TMS 4045	EFFE / F7FE
F800	RAM 1 TMS 4045	FFFE

c) 3K EPROM (2708 & 2716) 2K RAM

Address	Contents	Address
0000	EPROM 1 TMS 2708	07FE
0800	EPROM 1 TMS 2708	0FFE
1000	EPROM 2 TMS 2716	
2000	OFF-BOARD MEMORY	1FFE
F000	RAM 2 TMS 4045	EFFE / F7FE
F800	RAM 1 TMS 4045	FFFE

d) 3K EPROM (2708 & 2716) 2K RAM

Address	Contents	Address
0000	EPROM 1 TMS 2716	
1000	EPROM 2 TMS 2708	0FFE / 17FE
1800	EPROM 2 TMS 2708	1FFE
2000	OFF-BOARD MEMORY	
F000	RAM 2 TMS 4045	EFFE / F7FE
F800	RAM 1 TMS 4045	FFFE

NOTES

1. All addresses in hexadecimal.
2. EPROM selection in each bank is a jumper option.

The memory address decoding PROM is enabled by $\overline{\text{MEMEN}}$ when active low, and the lower five input bits are the most significant bits of the address bus (A0 to A4). The PROM thus selects memory in blocks of 1K words. The upper three address bits of the PROM have jumper options to choose between TMS 2708's and TMS 2716's and to select or deselect on-board EPROM, and to configure the memory map either with EPROM in low addresses and RAM in high addresses, or RAM low and EPROM high. There are thus eight different address maps in the PROM controlled by the three jumpers ($2^3 = 8$). Each address map consists of 32 four-bit words, showing the state of each 1K word block in memory.

When MEMEN is inactive, the PROM is disabled.

6.9.1.2 EPROM Selection
There are two basic memory maps for the EPROM - one for the TMS 2708's and the other for TMS 2716's. These correspond to cases (a) and (b) of Figure 6-7. Each bank of EPROM actually consists of two EPROM devices, one for bits 0 to 7 of the addressed word, and the other for bits 8-15. Beginning addresses are shown to the left of the figure; ending addresses are shown to the right. Each EPROM bank is separate and can be programmed into any location by reprogramming the address decode PROM.

Case (c) and (d) of the memory map in Figure 6-7 show what happens if the jumper is configured to "2716" position, and TMS 2708's are used. Case (c) shows that if a word at address 0000 is accessed, that same word can be read at 0800. Likewise, both 0002 and 0802 will address the same word, etc.

On the board, the jumper next to the EPROM's selects the proper pin configuration for the particular EPROM in use. Note that address line A4 is routed to the EPROM when the jumper is in the "2716" position.

To deselect, or ignore, on-board EPROM, move the EPROM select jumper to connect pin E12 to E13. This causes on-board EPROM sockets to disappear completely from the memory map.

6.9.1.3 RAM Selection
The RAM is treated as one block, since it is decoded with only one output line from the address decode PROM. There are four RAM's per bank and two banks in the block. The selection of a specific bank of RAM is decided by the state of address line A4. Selection is accomplished by the gate array shown in Figure 6-8. Each RAM select is set up by the PROM and A4, and becomes valid when $\overline{\text{WE}}$ goes low for a write, or DBIN goes high for a read. Note that DBIN will assert at the same time $\overline{\text{MEMEN}}$ goes low during a read cycle, reference Figure 6-10, but $\overline{\text{WE}}$ will not assert until some time after $\overline{\text{MEMEN}}$ goes to 0. The user should be aware that a chip select will not occur during a write cycle until after $\overline{\text{WE}}$ drops. This is to prevent fast RAMs, which sample WE as soon as they are selected, from sampling $\overline{\text{WE}}$ before it goes low during a write cycle.

At this point, the second jumper option becomes meaningful. This option selects where EPROM and RAM appear in the memory map. In the normal "RAM HI" position, the RAM bank address begins at $F000_{16}$ and EPROM begins at 0000_{16}. Moving the jumper plug to the alternate position causes "2708" EPROMs to be at $F000_{16}$ ("2716" EPROM's begin at $E000_{16}$), and RAM to be at 0000_{16}.
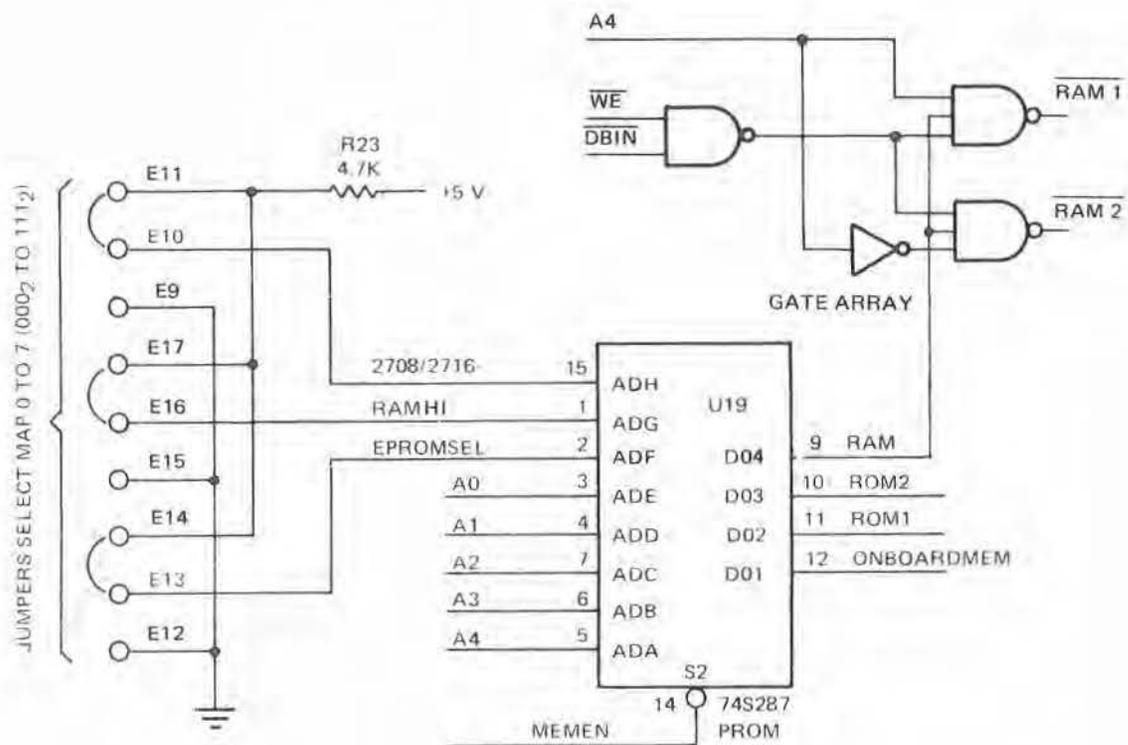
TABLE A. ADDRESS IN/DATA OUT

ADDRESS INPUT ADH TO ADA (LSB)	MAP	PROM OUTPUT (4 BITS EACH)							
00	0	66FF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
20	1	66FF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	CCAA
40	2	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FF66
60	3	CCAA	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FF66
80	4	66FF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
A0	5	66FF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFCA
C0	6	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FF66
E0	7	CAFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FF66

TABLE B. MAP CONFIGURATION (SET BY JUMPERS)

	2708 OR 2716 USED?	LOW OR HIGH RAM?	READ EPROM?
MAP 0 =	TMS 2716	Low RAM	No EPROM
Map 1 =	TMS 2716	Low RAM	High EPROM
Map 2 =	TMS 2716	High RAM	No EPROM
Map 3 =	TMS 2716	High RAM	Low EPROM
Map 4 =	TMS 2708	Low RAM	No EPROM
Map 5 =	TMS 2708	Low RAM	High EPROM
Map 6 =	TMS 2708	High RAM	No EPROM
Map 7 =	TMS 2708	High RAM	Low EPROM

Figure 6-8. Memory Address Decode PROM

6-18

6.9.1.4 Memory Mapping

The memory map can be changed by the user substituting another user programmed PROM in the address decoder socket. (The 74S287 PROM's are available from your Texas Instruments distributor.) Using the guidelines in paragraph 6.9.1, the user can produce many different memory maps. In general, if active output is desired for any particular input combination, the bit code is set to zero. Starting at the initial input address to the PROM, the output states desired are determined. ADA is the least-significant address input, and ADH is the most-significant. DO1 is the least significant output bit, and DO4 is the most-significant.

CAUTION

When planning a memory map, or when using any memory off-board (such as a TM 990/201 or TM 990/206 memory board), the memory devices on the TM 990/101M board must not overlap in address space either with each other or with devices off-board. On-board memory devices MUST be mapped into unique locations, and no other off-board devices may respond to addresses intended for any on-board memory device.

The 74S287 PROM's are field-programmable, fusible-link devices. The PROM's are delivered in a state of all binary ONE's. By blowing a fuse link during programming, a ZERO is programmed. Once a bit is programmed as ZERO, there is no way to restore the bit to a ONE. Be careful to program the device completely; partially programmed devices have been known to have random bits revert back to the ONE state because the fuse link was not blown completely.

MSB and LSB conventions are those used by the 990-family systems hardware and software for PROM and EPROM programming.

6.9.2 CRU SELECT

The CRU I/O decoding is done by a gate array and a 74LS138 decoder as shown in Figure 6-9. Address lines A3 through A9 are decoded, providing eight on-board select lines, each line addressing a block of 32 CRU bits. These select lines, ISEL0 through ISEL7, go to the various on-board CRU devices, with the exception of the ISEL3 line which is reserved for future use. The INTCRU/EXTCRU line is defined by the upper four address bits (A3-A6) and MEMEN; the line activates the 74LS138 decoder and deactivates the 74LS241 buffer with CRUIN.B and CRUOUT.B when an on-board CRU address is asserted. At all other times the buffer is enabled, and the on-board decoder is disabled, allowing some off-board CRU device to respond. Because of this manner of decoding, overlapping CRU addresses off-board will be ignored if they are mapped into on-board CRU space. On-board CRU address space thus is reserved; and because there is no PROM, the CRU address map cannot be changed.
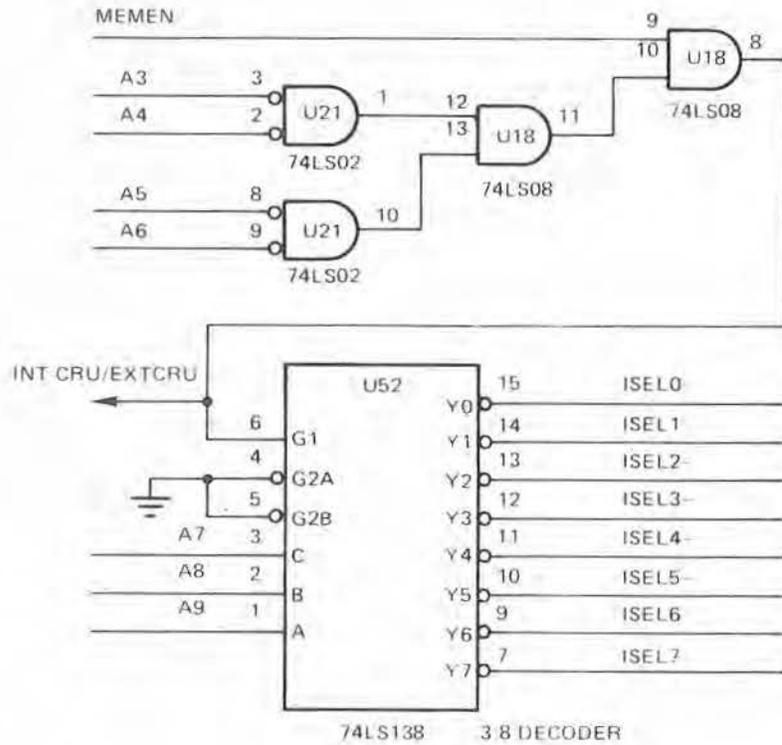
MEMEN

A3 ——— 3
A4 ——— 2 U21 1   12
              74LS02   13 U18 11   9 U18 8
                                         10 74LS08

A5 ——— 8
A6 ——— 9 U21 10
              74LS02   74LS08

INT CRU/EXTCRU

U52

6 G1   Y0 15 ISEL0
4 G2A   Y1 14 ISEL1
5 G2B   Y2 13 ISEL2-
A7 3 C   Y3 12 ISEL3-
A8 2 B   Y4 11 ISEL4-
A9 1 A   Y5 10 ISEL5-
               Y6 9   ISEL6
               Y7 7   ISEL7

74LS138   3:8 DECODER

SIGNAL   ENABLES

ISEL0   L.E.D. Circuit
ISEL1   DIP Switch
ISEL2-   Main TMS 9902 (P2)
ISEL3-   Not Used
ISEL4-   TMS 9901
ISEL5   RESET Edge-Triggered Interrupt
ISEL6-   Auxiliary TMS 9902 (P3)
ISEL7   Auxiliary EIA Signals

Figure 6-9. Decoding Circuitry for CRU I/O Addresses

Table 6-5.   TM 990/101 CRU Map

CRU Software Base Address (Hex)	Bit Address (Hex)	Function	Input	Output
0000_{16}	0000	STATUS LED		
	0001	RESERVED		
		↓		
	001F	RESERVED		
0040_{16}	0020	UNIT ID 4 (LSB)	INPUT ONLY	
	0021	UNIT ID 3		
	0022	UNIT ID 2		
	0023	UNIT ID 1		
	0024	UNIT ID 0 (MSB)	INPUT ONLY	
	0025	GROUNDED		
	0026	GROUNDED		
	0027	GROUNDED		
	0028	RESERVED		
	TO	↓		
	003F	RESERVED		
0080_{16}	0040	SERIAL I/O	RBR0	CTRL0
	0041	PORT A P2	RBR1	CTRL1
	0042	TMS 9902	RBR2	CTRL2
	0043		RBR3	CTRL3
	0044		RBR4	CTRL4
	0045		RBR5	CTRL5
	0046		RBR6	CTRL6
	0047		RBR7	CTRL7
	0048		0	CTRL8
	0049		RCVERR	CTRL9
	004A		RPER	CTRL10
	004B		ROVER	LXDR
	004C		RFER	LRDR
	004D		RFBD	LDIR
	004E		RSBD	LDCTRL
	004F		RIN	TSTMD
	0050		RBINT	RTSON
	0051		XBINT	BRKON
	0052		0	RIENB
	0053		TIMINT	XBIENB
	0054		DSCINT	TIMENB
	0055		RBRL	DSCENB
	0056		XBRE	NOT USED
	0057		XSRE	NOT USED
	0058		TIMERR	NOT USED
	0059		TIMELP	NOT USED
	005A		RTS	NOT USED
	005B		DTR	NOT USED
	005C		CTS	NOT USED
	005D		DSCH	NOT USED
	005E		FLAG	NOT USED
	005F	↓	INT	RESET

Table 6-5. TM 990/101 CRU MAP (Continued)

CRU Software Base Address (Hex)	Bit Address (Hex)	Function	Input	Output
	0060	RESERVED		
	↓			
0100_{16}	007F	RESERVED		
	0080	TMS 9901	CONTROL BIT	CONTROL BIT
	0081	PSI	INT1 /CLK1	MASK1 /CLK1
	0082		INT2 /CLK2	MASK2 /CLK2
	0083		INT3 /CLK3	MASK3 /CLK3
	0084		INT4 /CLK4	MASK4 /CLK4
	0085		INT5 /CLK5	MASK5 /CLK5
	0086		INT6 /CLK6	MASK6 /CLK6
	0087		INT7 /CLK7	MASK7 /CLK7
	0088		INT8 /CLK8	MASK8 /CLK8
	0089		INT9 /CLK9	MASK9 /CLK9
	008A		INT10/CLK10	MASK10/CLK10
	008B		INT11/CLK11	MASK11/CLK11
	008C		INT12/CLK12	MASK12/CLK12
	008D		INT13/CLK13	MASK13/CLK13
	008E		INT14/CLK14	MASK14/CLK14
	008F		INT15/INTREQ	MASK15/RST2
	0090		P0 INPUT	P0 OUTPUT
	0091		P1 INPUT	P1 OUTPUT
	0092		P2 INPUT	P2 OUTPUT
	0093		P3 INPUT	P3 OUTPUT
	0094		P4 INPUT	P4 OUTPUT
	0095		P5 INPUT	P5 OUTPUT
	0096		P6 INPUT	P6 OUTPUT
	0097		P7 INPUT	P7 OUTPUT
	0098		P8 INPUT	P8 OUTPUT
	0099		P9 INPUT	P9 OUTPUT
	009A		P10 INPUT	P10 OUTPUT
	009B		P11 INPUT	P11 OUTPUT
	009C		P12 INPUT	P12 OUTPUT
	009D		P13 INPUT	P13 OUTPUT
	009E	TMS 9901	P14 INPUT	P14 OUTPUT
	009F	PSI	P15 INPUT	P15 OUTPUT
0140_{16}	00A0	RESERVED		
	↓			
	00A5	RESET INT6		
	00A7	RESERVED		
	TO	↓		
	00BF	RESERVED		

6-22

Table 6-5. TM 990/101 CRU Map (Continued)

CRU Software Base Address (Hex)	Bit Address (Hex)	Function	Input	Output
0180₁₆	00C0	SERIAL I/O	RBR0	CTRL0
	00C1	PORT B P3	RBR1	CTRL1
	00C2	(TMS 9902)	RBR2	CTRL2
	00C3		RBR3	CTRL3
	00C4		RBR4	CTRL4
	00C5		RBR5	CTRL5
	00C6		RBR6	CTRL6
	00C7		RBR7	CTRL7
	00C8		0	CTRL8
	00C9		RCVERR	CTRL9
	00CA		RPER	CTRL10
	00CB		ROVER	LXDR
	00CC		RFER	LRDR
	00CD		RFBD	LDIR
	00CE		RSBD	LDCTRL
	00CF		RIN	TSTMD
	00D0		RBINT	RTSON
	00D1		XBINT	BRKON
	00D2		0	RIENB
	00D3		TIMINT	XBIENB
	00D4		DSCINT	TIMENB
	00D5		RBRL	DSCENB
	00D6		XBRE	NOT USED
	00D7		XSRE	
	00D8		TIMERR	
	00D9		TIMELP	
	00DA		RTS	
	00DB		DCD (NOT DSR)	
	00DC		CTS	
	00DD		DSCH	
	00DE		FLAG	NOT USED
	00DF		INT	RESET
01C0₁₆	00E0		DTR	DTR
	00E1		DSR	
	00E2		RI	
	00E3			
	00E4			
	00E5			
	00E6		RI	
	00E7		0	DTR
	00E8		RESERVED	RESERVED
	00E9			
	00EA			
	00EB			
	00EC			
	00ED			
	00EE			
	00EF	PORT B	0	RESERVED

Table 6-5. TM 990/101 CRU Map (Concluded)

CRU Software Base Address (Hex)	Bit Address (Hex)	Function	Input	Output
$01C0_{16}$	00F0 TO 00FF	RESERVED ↓ RESERVED		
0200_{16}	0100 TO 0FFF	OFF BOARD CRU		

CAUTION

Although CRU addresses are decoded into 32-bit blocks, not all CRU devices use or completely decode the entire 32 bits. This can result in a CRU device being enabled by addresses other than those specified. Note the alternate addresses in Table 6-6. This condition may be referred to as implicit decoding, and should be considered where it is necessary to debug a CRU scheme.

Note that address lines A0, A1, and A2 do not enter into the decoding. If an external instruction is being executed, it is true that some CRU device may be addressed by the lines A3 through A14, but since CRUCLK is trapped in the external instruction decoder, no CRU output can be done. Therefore, since CRUCLK is absent from the addressed device, it will assume a CRU input operation, and present a datum to CRUIN, which the processor will ignore. No harm is done in either case, so lines A0, A1, and A2 are don't care conditions.

Table 6-5.   Implicit Decoded CRU Bit Addresses

Device	Normal Address Range (R12, Bits 3 to 14)	Alternate Address Ranges
Status LED	0000	0001 - 001F
Unit ID Switch	0020-0027	0028 - 002F, 0030 - 0037, 0038 - 003F
Edge Trig INT6 Clear	00A6	00A0 - 00BF
DTR (Input)	00E0	00E4, 00E8, 00EC, 00F0, 00F4, 00F8, 00FC
DTR (Output)	00E0	00E1 - 00FF
DSR (Input)	00E1	00E5, 00E9, 00ED, 00F1, 00F9 00FD
RI (Input)	00E2	00E6, 00EA, 00EE, 00F2, 00F6, 00FA, 00FE

NOTES

1.   The above are CRU bit addresses, not R12 contents.

2.   Response to an alternate address (right column) will be the same as to using the normal address (middle column); however, the user should program using only the normal address.

Table 6-6 gives nominal address values for all on-board CRU devices. These are the nominal values which should be used in programs.

Table 6-6.   On-Board Device CRU Address

Device	CRU Address (R12, Bits 0-15) (Hexadecimal)	Maximum Displacement (Decimal)	CRU Bit Address Range (R12, Bits 3-14) (Hexadecimal)
Status LED	0000	0	0000
Unit ID Switch	0040	4	0020 - 0024
Local TMS 9902	0080	31	0040 - 005F
TMS 9901, Interrupt/Timer	0100	15/31*	0080 - 008F
TMS 9901, Parallel I/O	0120	15/31*	0090 - 009F
Reset Interrupt 6	014C	0	00A6
Remote TMS 9902	0180	31	00C0 - 00DF
DTR, DSR, RI	01C0	2	00E0 - 00E2

*The TMS 9901 is shown split into its two separately functional parts; each has a maximum displacement of 15. Together, the device has a maximum displacement of 31.

6.10   MEMORY TIMING SIGNALS
The three memory timing signals are READY, WAIT, and MEMCYC. These are arbitrarily grouped together for a discussion of their theory of operation.

6.10.1   READY
The READY signal is an input to the TMS 9900 microprocessor which indicates that during a memory cycle, the memory devices addressed will be ready at the next ∅1 clock phase for successful disposition of data.

The READY signal is sampled by the processor during ∅1, after MEMEN has gone low. If READY is high when sampled, the 9900 CPU will continue the memory operation in progress as shown by the READ cycle part of Figure 6-10. During a read cycle if READY is sampled and found to be high, the processor will read data from the selected memory device(s) on the leading edge of the next ∅1. During a write cycle, if READY is sampled on the leading edge of ∅1 and found high, the CPU will assume that data has successfully been stored in the selected memory device(s) by the time the next leading edge of ∅1 occurs. If the selected memory device(s) cannot meet this timing constraint, the READY signal can be pulled low, which puts the TMS 9900 CPU into a wait state. The WAIT signal will go high to signify that the processor is in a wait state, and CPU operations will be suspended until READY is sampled high. When READY goes high again, WAIT will drop and the CPU will continue execution from the point where it stopped. (Refer to the write cycle portion of Figure 6-10.)

RD READ DATA
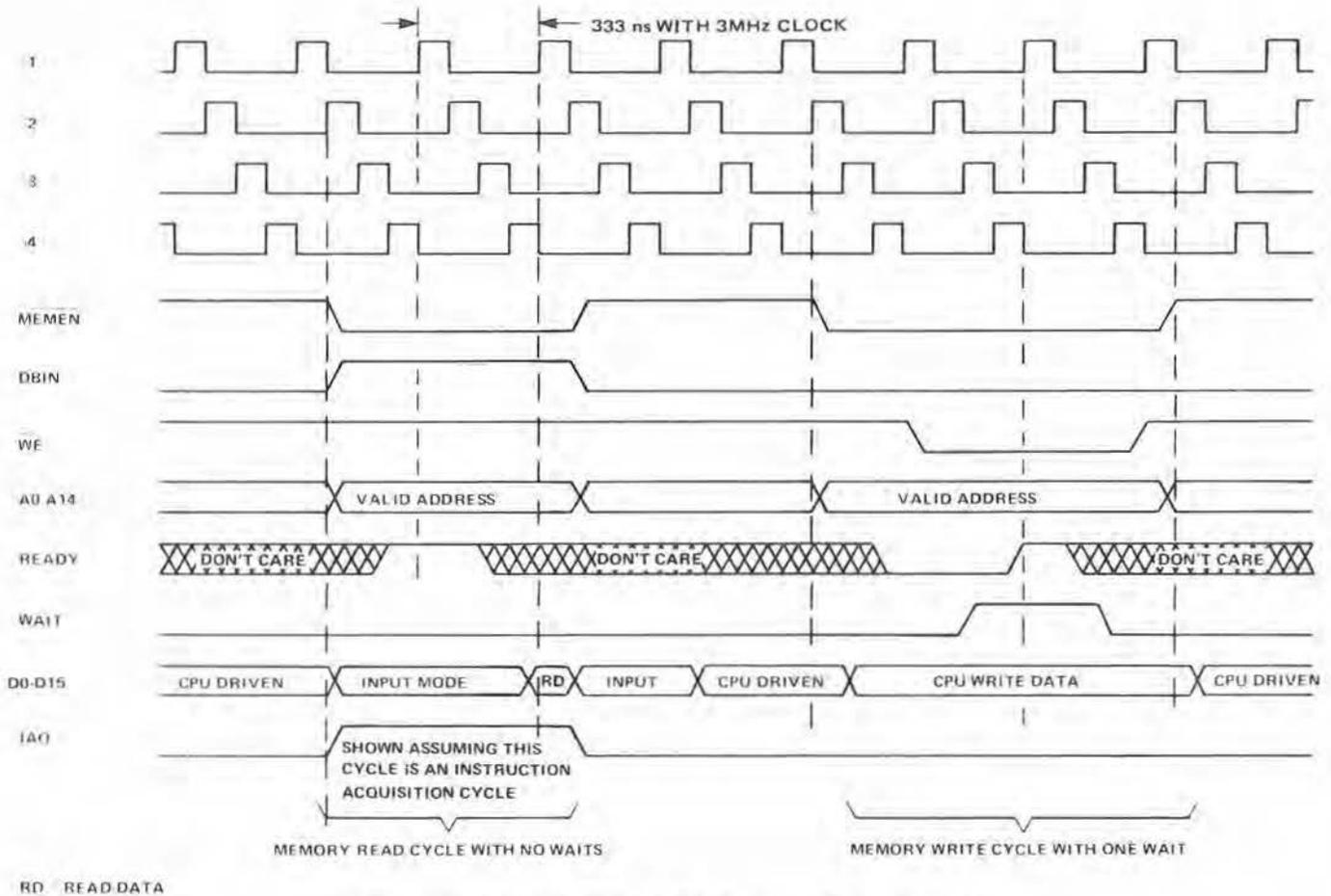
Figure 6-10.   TMS 9900 Memory Bus Timing

The READY line can be held low for any amount of time, so the user can utilize memory devices with very slow access times. As an example, consider the memory cycle times for the TMS 4045 memories resident on the CPU board. With a system clock frequency of 3 MHz, the total time is about 600 nsec between (1) assertion of DBIN, $\overline{\text{MEMEN}}$ and valid address and (2) the actual processor read. When rise and fall times for these signals plus setup times for the data are computed, the memory device should have an access time of 490 nsec or less from valid address. For processor write operations, counting rise and fall times plus data hold times, the cycle time should be less than 600 nsec from valid address. TMS 4045 devices will have data available for the processor to read a maximum of 450 nsec after receiving a valid address. For write operations, the data must be held valid for at least 200 nsec before the $\overline{\text{WE}}$ signal goes high. If Figure 6-10 is examined, the user will notice these constraints are easily met. If the memory devices do not meet these times, wait states can be inserted to hold control, address and data lines valid until the timing criteria for the device is met. Each wait state extends valid control, address and data information by 333 nsec.

For 3 MHz operation, data must be available during a read cycle 490 nanoseconds after the start of the cycle. For a write operation data must be captured by the memory devices 600 nanoseconds after the start of the cycle. If these times cannot be met the processor can be put in a wait state by forcing READY.B low for as long as necessary (indefinitely, if need be). After READY.B becomes high, the memory cycle will occupy one more clock cycle and then be completed. Refer to Figure 6-10.

6.10.2 WAIT
The WAIT signal is output by the processor to acknowledge that addressed memory devices are not ready and that the processor is in a wait state.

Note that if one wait state is required, as is specified by the SLOW jumper, WAIT can be connected to READY. At the start of the cycle, WAIT is inactive and thus low. When the processor samples READY, it sees that memory is not ready because the READY line is low. The processor acknowledges by raising WAIT to high, and being connected to READY, when the processor samples READY again, it finds it high and therefore completes the memory cycle. The SLOW jumper must be inserted for memories which cannot meet the speed requirements listed in paragraph 6.10.1.

6.10.3 $\overline{\text{MEMCYC}}$
It is possible for the TMS 9900 microprocessor to activate $\overline{\text{MEMEN}}$ and accomplish many fetches from memory by shifting the address bus, all while $\overline{\text{MEMEN}}$ is still active. The $\overline{\text{MEMCYC}}$ signal is synchronized to the $\phi3$ clock edge after the beginning of the memory cycle, and goes inactive just before the instant the address bus could change. This signal thus delimits one complete memory cycle and differentiates between separate memory cycles.

The $\overline{\text{MEMCYC}}$ signal is used by dynamic memories which must be able to intervene between memory cycles for burst refresh, if necessary.

6.11 READ-ONLY MEMORY
The two EPROM blocks, shown in Figure 6-11 each contain two devices. Each device provides an eight-bit output; the two in parallel in each block thus provide a 16-bit word. TMS 2708 EPROM's contain 1K x 8 bits; therefore, each block is 1K words. Using TMS 2716 EPROM's, capacity is expanded to 2K words per block. A fully expanded EPROM section thus contains 4K words or 8K bytes of addressability. Each block is separately mapped into the address space as explained in paragraph 6.9.1.2.

6.12 RANDOM ACCESS MEMORY

The two RAM blocks, RAM 1 and RAM 2, each contain four TMS 4045 devices. Each device provides four-bit storage; four devices in parallel in each block provide a 16-bit word. Each TMS 4045 device contains 1K x 4 bits; therefore, each block is 1K words. A fully expanded RAM section thus consists of 2K words. Both blocks are mapped into contiguous address space, and are selected as explained in paragraph 6.9.1.3. Block RAM 2 is shown in Figure 6-12.

6.13 BUFFER CONTROL

Connector P1 is the system bus edge connector. It contains, in approximate order by pins: the system power, interrupt, data, address, and control signals. Table H-1 lists pins and their functions. Power lines are detailed in paragraph 6.2, and interrupts are detailed in paragraph 6.14. This discussion covers the address bus buffers, the data bus buffers, control bus buffers, and a short discussion of HOLD, HOLDA, and direct memory access (DMA).
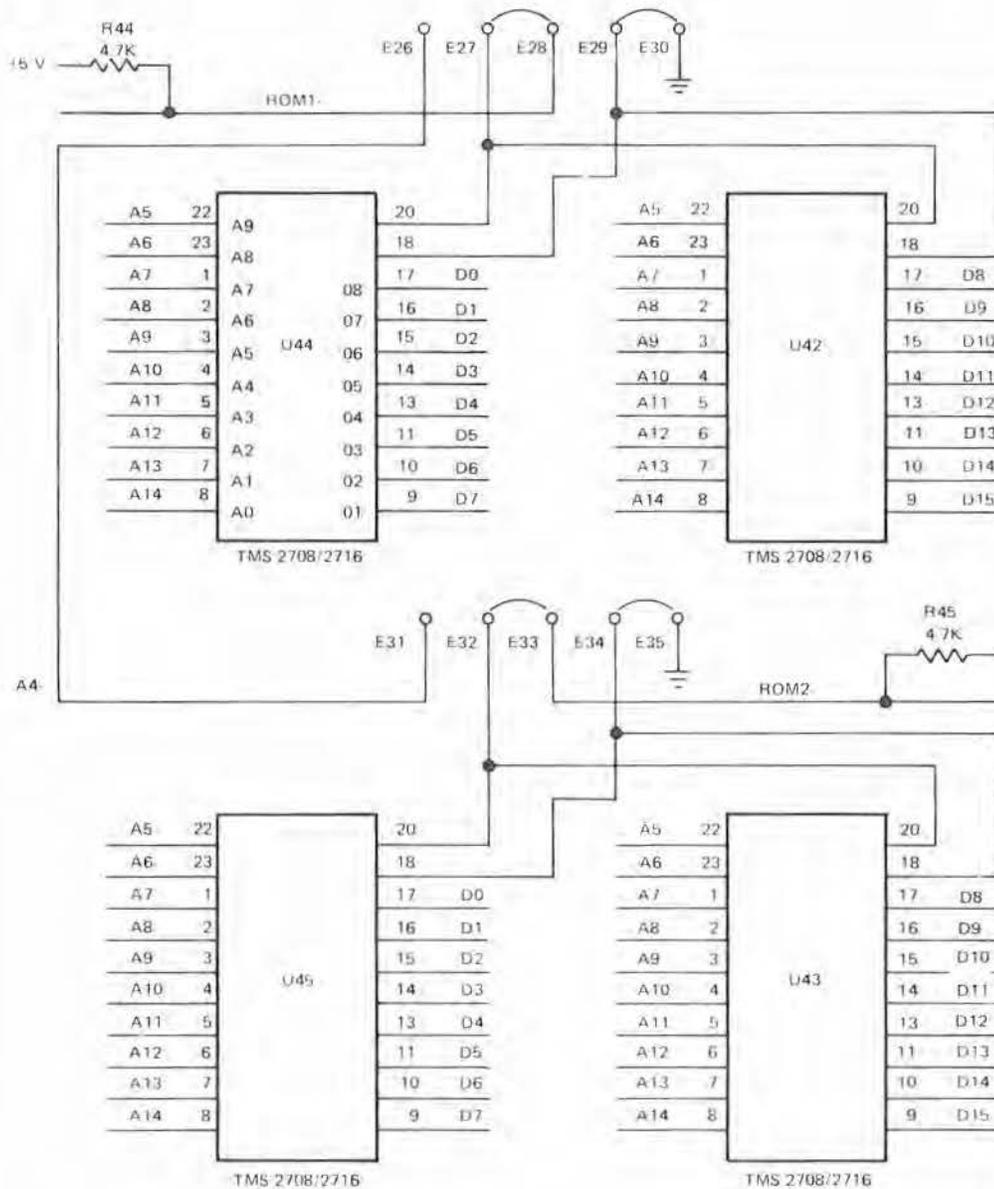
Figure 6-11. Read-Only Memory

A5 15 A9 I/04 11 D0
A6 16 A8 I/03 12 D1
A7 17 A7 I/02 13 D2
A8 1 A6 I/01 14 D3
A9 2 A5 U36
A10 3 A4
A11 4 A3
A12 7 A2
A13 6 A1 CS 8
A14 5 A0 WE 10

TMS 4045

A5 15 11 D8
A6 16 12 D9
A7 17 13 D10
A8 1 14 D11
A9 2 U34
A10 3
A11 4
A12 7
A13 6 8
A14 5 10

TMS 4045

WE-
RAM2-

A5 15 11 D4
A6 16 12 D5
A7 17 13 D6
A8 1 14 D7
A9 2 U30
A10 3
A11 4
A12 7
A13 6 8
A14 5 10

TMS 4045

A5 15 11 D12
A6 16 12 D13
A7 17 13 D14
A8 1 14 D15
A9 2 U28
A10 3
A11 4
A12 7
A13 6 8
A14 5 10

TMS 4045

Figure 6-12. Random Access Memory

6.13.1 ADDRESS AND DATA BUFFERS
The address buffers consist of two 74LS245 octal   bus transceivers. The address lines
normally flow off-board. Upon a HOLDA signal, the direction reverses, allowing a DMA
controller to input an address onto the board for disposition by the address decoder
section. Address and data buffers are shown on sheet 3 of the schematics (Appendix F,
page F-3).

The same devices are used as the data bus buffers. Direction data flow, however, is
governed by the 74LS153 decoder using the states of ONBDMEM and HOLDA (listed in Table
6-8).

<p align="center">Table 6-8. Data Buffers</p>

HOLDA	ONBDMEN	Bus Command	Data Flow		Operation
			(READ)	(WRITE)	
Low	Low	DBIN	On-board	Off-board	Normal off-board
Low	High	Low	Off-board	Off-board	Normal on-board
High	Low	High	On-board	On-board	DMA off-board
High	High	$\overline{\text{DBIN}}$	Off-board	On-board	DMA on-board

Note that during normal off-board operation, the direction is as expected. During
normal on-board operation, the direction of data flow is always off-board so that
off-board data will not interfere with the on-board operation. This also permits an
external logic system to monitor on-board activities for debugging purposes. For
example, illegal op codes can be caught by monitoring the data bus during IAQ time.
Following the same logic, data flow is always on-board during an off-board DMA
operation so that no interference occurs. Finally, on-board DMA requires that the
buffers be in a state opposite that normally expected since the controller is
off-board.

6.13.2 CONTROL BUFFERS
Three types of enabling are used on control line buffers: HOLDA, CRU, and always
enabled. The lines that are always enabled are those whose source is always on-board,
such as the clocks, IAQ, IORST, CRUCLK, and HOLDA.

The second type, the CRU signals, are governed by the INTCRU/$\overline{\text{EXTCRU}}$ signal derived by
the CRU address decoder (see paragraph 6.9.2). Normally enabled, CRUIN.B and CRUOUT.B
are disabled for on-board operation to prevent possible interference during address
and CRU data stabilization.

The third type of control buffer is the type directly affected by CPU or DMA
operations: the memory control signals $\overline{\text{MEMEN}}$, $\overline{\text{WE}}$, and DBIN. Normally enabled flowing
off-board, these lines reverse direction when flowing on-board for DMA operations so
that the DMA controller can command on-board memory. These lines are keyed on the
state of HOLDA.

6.13.4 $\overline{\text{HOLD}}$, HOLDA, and DMA

When an off-board direct memory access controller (DMAC) wishes to initiate operation, it asserts a low state onto the $\overline{\text{HOLD}}$ line. After finishing the current memory cycle, the microprocessor responds by floating its address, data, $\overline{\text{MEMEN}}$, DBIN, and $\overline{\text{WE}}$ lines, and then forces HOLDA (HOLD acknowledge) high.

The DMAC is now free to use the system buses to transfer data directly in and out of memory as it wishes. For a more detailed discussion of DMA operations, refer to Section 8 of the manual, Applications.

6.14 INTERRUPT STRUCTURE

The TM 990/101M provides total of 17 interrupts. The characteristics of each are listed in Table 6-9.

Table 6-9. Interrupt Characteristics

Interrupt	Types	Maskable	Prioritized	Characteristics
RESET	Dedicated	No	Yes	INT 0, resets I/O, TMS 9900 mask
1-5	Dedicated	Yes	Yes	Level triggered, all defined*
6	Dedicated	Yes	Yes	Level or edge-triggered*
7-15	Shared I/O	Yes	Yes	Level-triggered, undefined
LOAD	Dedicated	No	No	Level triggered, will always occur unless locked out by a RESET

*Definitions in Table 6-10

Table 6-10. Dedicated Interrupt Description

Interrupt Level	Purpose
1	Power fail interrupt, brought out on OEM chassis
2	Reserved for future use
3	System timer: TMS 9901
4	Main I/O port: TMS 9902
5	Auxiliary I/O port: TMS 9902/03
6	External device - edge (positive or negative) triggered or level sensitive.

All interrupts except RESET and LOAD are processed by the TMS 9901 Programmable Systems Interface device. This device handles both parallel I/O and interrupt requests. Because of the pinout limitation on the package, the TMS 9901 must share INT7 through INT15 (interrupt requests 7 through 15) with the parallel I/O lines P15 through P7, respectively. This reverse arrangement provides contiguous I/O and interrupt lines if some of the shared lines are used for interrupts and others for I/O (see Figure 6-13).

The basic operation of the interrupt facility must be initialized by the microprocessor through the CRU. The 15-bit interrupt mask is set under program control to allow interrupt requests by writing a ONE state into those mask register positions. The mask bits that contain ZERO will not honor interrupt requests. Note that the condition of the processor's Status Register priority mask is irrelevant if the TMS 9901's Interrupt Mask Register is a ZERO for a particular interrupt: the request will not even be presented to the processor.

When one or more interrupt requests are presented on the INT1 to INT15 lines, only those whose corresponding mask bits are ONE are considered. The highest priority request present is encoded onto lines IC0 through IC3, and INTREQ becomes active (low).

The TMS 9900 receives the coded request and compares its value to the interrupt mask in its status register. If equal or higher priority, (a lower interrupt number) the interrupt is honored, the mask is set to one less than the current interrupt number, and the vector process begins. Note that level 0 is the highest priority, and cannot be masked out since it is a number that is always equal to or lower than any number which can be in the mask register of the processor. The lowest priority is 15.

There is extra logic for INT6 to be triggered either in the normal manner by presenting a low level to P1 pin 20, or in an edge-triggered manner. A low-to-high transition should be presented to P4-8, and a high-to-low transition on P4-6. These edge-triggered signals are converted to level-sensitive signals, and are latched by a pair of flip-flops. The interrupt request line can be set inactive by the interrupt-service routine by writing a bit, either a ONE or a ZERO, to CRU bit address 00A6 (R12 base address $014C_{16}$). These flip-flops are automatically cleared by the CLRCRU signal.

6.15 PARALLEL I/O AND SYSTEM TIMER

The TMS 9901 provides sixteen lines of parallel I/O. The TM 990/101M user can read or write to any single bit of this parallel port because it is under CRU control. For example, eight bits can be used for output at the same time the eight other bits are used for input. This allows applications such as scanning a custom keyboard for input, or outputting multiplexed signals to a seven-segment display device; all under program control. A timer is also integrated into this device.
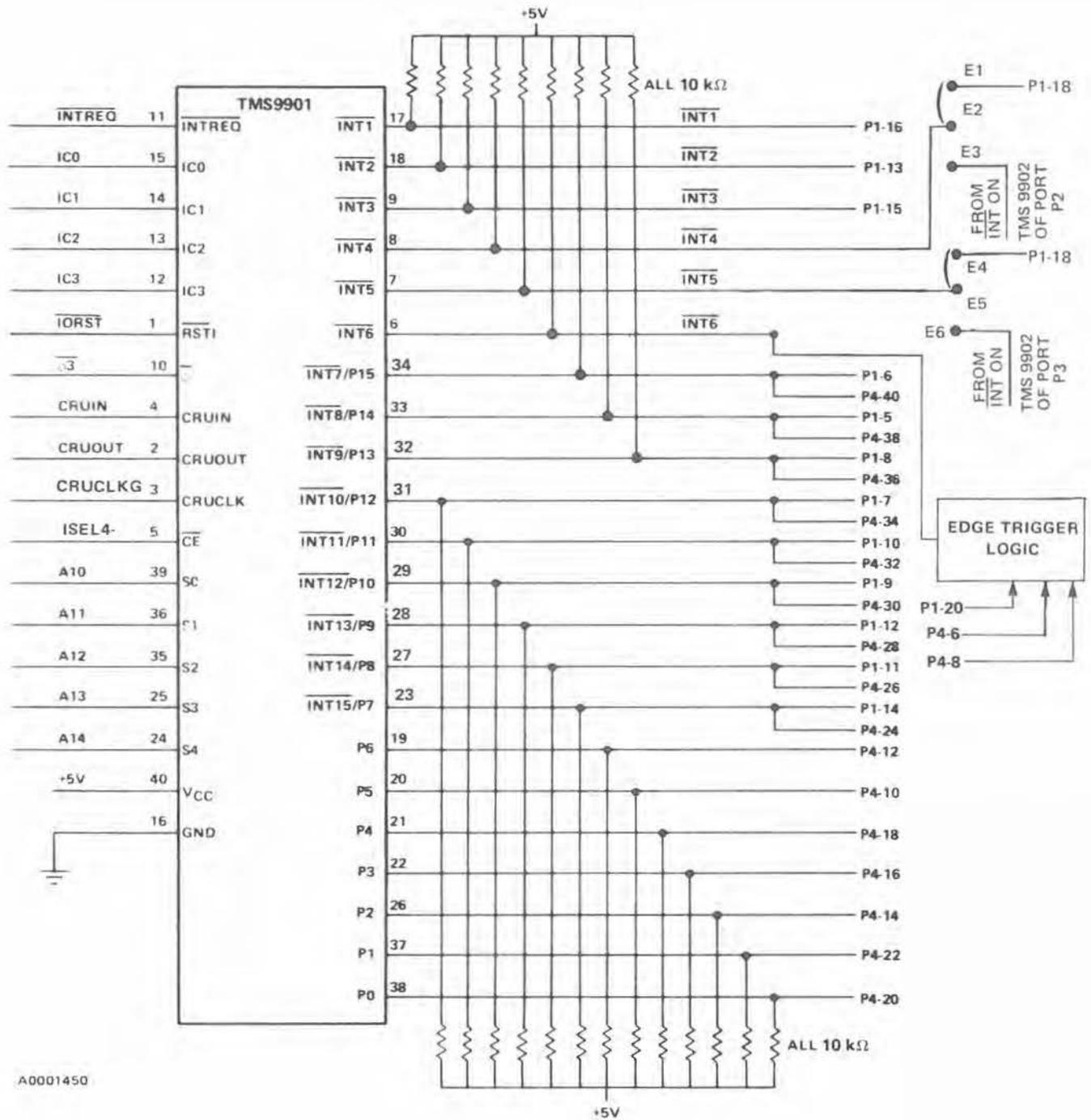
Figure 6-13. TMS 9901

6.15.1 PARALLEL I/O

Lines P0 through P6 are dedicated I/O lines, while P7 through P15 are shared with INT15 through INT7, respectively. When a user system is configured, it must be decided how to allocate these shared lines between interrupts and I/O. When written to, each parallel line remains in the same state until written to again. The parallel I/O lines are initialized by resetting the 9901. This may be done in 3 ways; by

    (1) Activating the RESET switch or pulling $\overline{\text{PRESET.B}}$ to 0

    (2) executing a RSET instruction

    (3) Putting the TMS 9901 in the clock mode and then writing a 0 to CRU bit 15 (refer to Table 1, TMS 9901 manual). Instructions to accomplish this for the TMS 9901 on the /101M CPU board are:

```
LI      R12,>100
SBO      0
SBZ     15
```

After initialization of the 9901, all I/O lines are in the input mode, and all I/O lines are pulled high. Writing to a specific CRU bit programs that bit as an output, and that bit will remain an output until the TMS 9901 is initialized again.

6.15.2   SYSTEM TIMER

The TMS 9901 has an internal real time clock which may be used as an interval timer by the user. It is a decrementer which generates an interrupt when it decrements to 0. To load a value into the 9901 clock register on the 101 board, the user must:

    (1) put the 9901 in the clock mode by writing a 1 to the control bit (CRU bit 0)

    (2) load a 14-bit count value into the counter register (CRU bits 1 through 14)

The counter will start decrementing the counter register value immediately after it is loaded at a rate of 0/64. For a 101 running at 3 MHz, this computes to a decrement every 21.33 microseconds (rounded off). Writing all ones to the counter register gives the maximum time interval of 349.525 milliseconds (rounded off value). An example of loading and starting the timer is:

```
LI      R12,>100
LDCR    R1,15
```

R1 contains the 14-bit timer value, plus a one in the least significant bit position. This least significant one gets loaded first and puts the TMS 9901 in the clock mode. If the least significant bit is a 0, the user will be loading the TMS 9901 interrupt mask register instead of the counter register. Refer to the TMS 9901 manual for more details.

When the TMS 9901 timer decrements to 0, a level 3 ($\overline{\text{INT3}}$) interrupt is generated. For this interrupt to cause a context switch, the 9901 must be in the interrupt mode (CRU bit 0 = 0), the $\overline{\text{INT3}}$ mask bit must be 1 (CRU bit 3 = 1), and the TMS 9900 interrupt mask must be set to accept a level 3 or higher priority interrupt (LIMI 3). Code to do this would look like the following:

```
LI        R12,>100       SET CRU BASE ADDRESS OF 9901 ON 101
SBZ       0              PUT 9901 INTO INTERRUPT MODE
SBO       3              ENABLE INT3
LIMI      3              SET 9900 INTERRUPT MASK FOR LEVEL 3
                         OR HIGHER PRIORITY INTERRUPT.
```

After the interrupt has occurred and a context switch has taken place, the user should disable the timer interrupt at the 9901 by writing a 0 to CRU bit 3. This will prevent INT3 from occurring during the Interrupt Service Routine and possibly cause an infinite loop to the Interrupt Service Routine. Several items of interest regarding the 9901 timer are

(1) after decrementing to 0, the timer reloads itself with the start value and starts decrementing again

(2) when the 9901 timer is being utilized, it generates INT3. Any signals on the INT3 pin (pin 9) of the 9901 are ignored.

(3) if the timer is used for measuring elapsed time or as an event counter, the contents of the counter register must be read. To do this, the 9901 must be put in the interrupt mode (CRU bit 0 = 0) for at least 21.33 microseconds, then placed back in the clock mode (CRU bit 0 = 1) and CRU bits 1-14 are read.

(4) to stop the timer, the 9901 must be put in the clock mode and the counter register (CRU bits 1-14) must be loaded with zeroes.

6.16 MAIN COMMUNICATIONS PORT

The main communications serial I/O port (P2) has two options, depending on the "dash number" ordered by the customer. (Refer to paragraph 1.3, "Product Index," to determine whether the Teletype (TTY) or multidrop (MD) interface circuitry is included on this serial port.) The main I/O port uses the TMS 9902 Asynchronous Communications Controller and is intended for operation with either the "console device" or master terminal for the TM 990/101M user, or with an automated control device using the multidrop interface. For detailed operation instructions for the TMS 9902, refer to the data manual for this device. When pin E2 is connected via jumper to pin E3, the INT pin of U46 is connected to the INT4 pin of the TMS 9901. The TMS 9902 will generate an interrupt on 4 separate conditions, and so if the 9902 at P2 does generate an interrupt, it will appear as INT4.

6.16.1 EIA INTERFACE

The EIA interface consists of 75188 line drivers and 75189A line receivers. The receive-data line goes to P3-2 and the transmit-data line to P3-3. This configuration forms a port suitable for connection to an RS-232-C compatible terminal. A data-terminal-ready (DTR) signal is supplied as an input for handshaking use with a device requiring it. Request-to-send RTS) and clear-to-send (CTS) signals are tied together and brought out to P2-8, which functions as the data-carrier-detect (DCD) signal to the terminal.
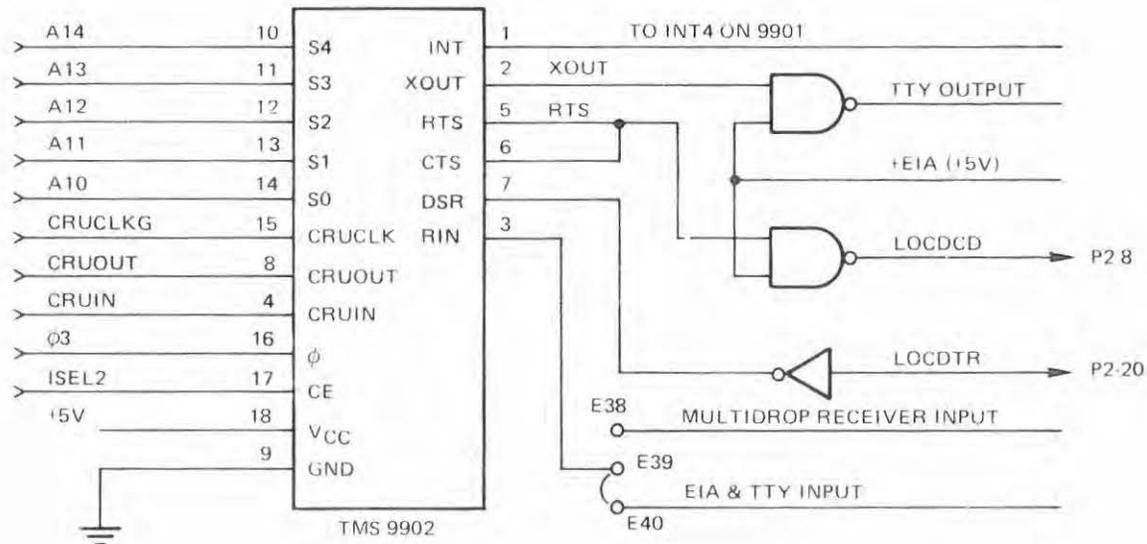
Figure 6-14. Serial I/O Port EIA Interface

6.16.2 TTY INTERFACE

A transistor and 560-ohm resistor form the transmit loop for the 20-mA current loop, TTY interface. The transistor conducts current while the line driver connected to its base is at a mark state. As the line driver goes to the space state, the positive voltage output is clamped to ground through the signal diode on the transistor base, thereby turning off the transistor and the current loop (refer to Figure 6-15).

The receive circuit consists of a line receiver which monitors the receive loop formed by the TTY transmit circuitry and the two supply resistors. The values of these resistors is such that during a mark state, the input to the line receiver is held very close to -12 volts. When the TTY transmit circuitry cuts the loop, the receiver input is pulled up to +12 by the 2.7 kohm resistor.

Note that the TTY jumper must be in place so that the line receiver can monitor the loop voltage. An EIA terminal should not be connected when the TTY jumper is in place.
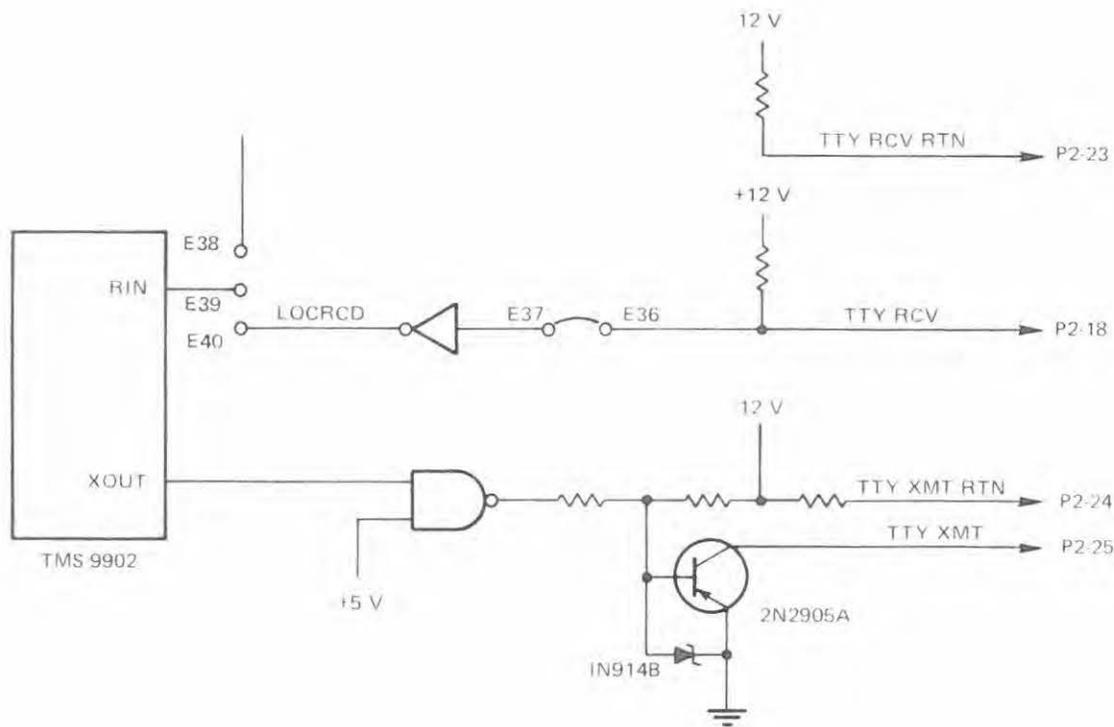
Figure 6-15. Serial I/O Port TTY Interface

6-16.3 MULTIDROP INTERFACE

The multidrop interface (Figure 6-16) may be used for board-to-board communications over long distances. Generally, only a twisted pair line is required between the boards. One pair is necessary for transmitting, and another pair for receiving when in full duplex mode. Connecting the two half-duplex jumpers will loop the transmitter back to the receiver for test or half-duplex applications and only one pair is then required.

More than two boards may be linked together, each one is just "dropped" in place, hence the term multidrop. If more than two boards are used, the boards not at the extreme ends of the twisted pair line (i.e., those "dropped in the middle") are considered nonterminating boards, and the termination resistor jumper plugs should be removed to prevent standing wave patterns which might occur, mostly at the higher baud rates. The two boards at the extremes of the line, regardless of whether additional boards exist in between, should have these resistor jumper plugs installed. Refer to Section 7, Options, for jumper configuration information.
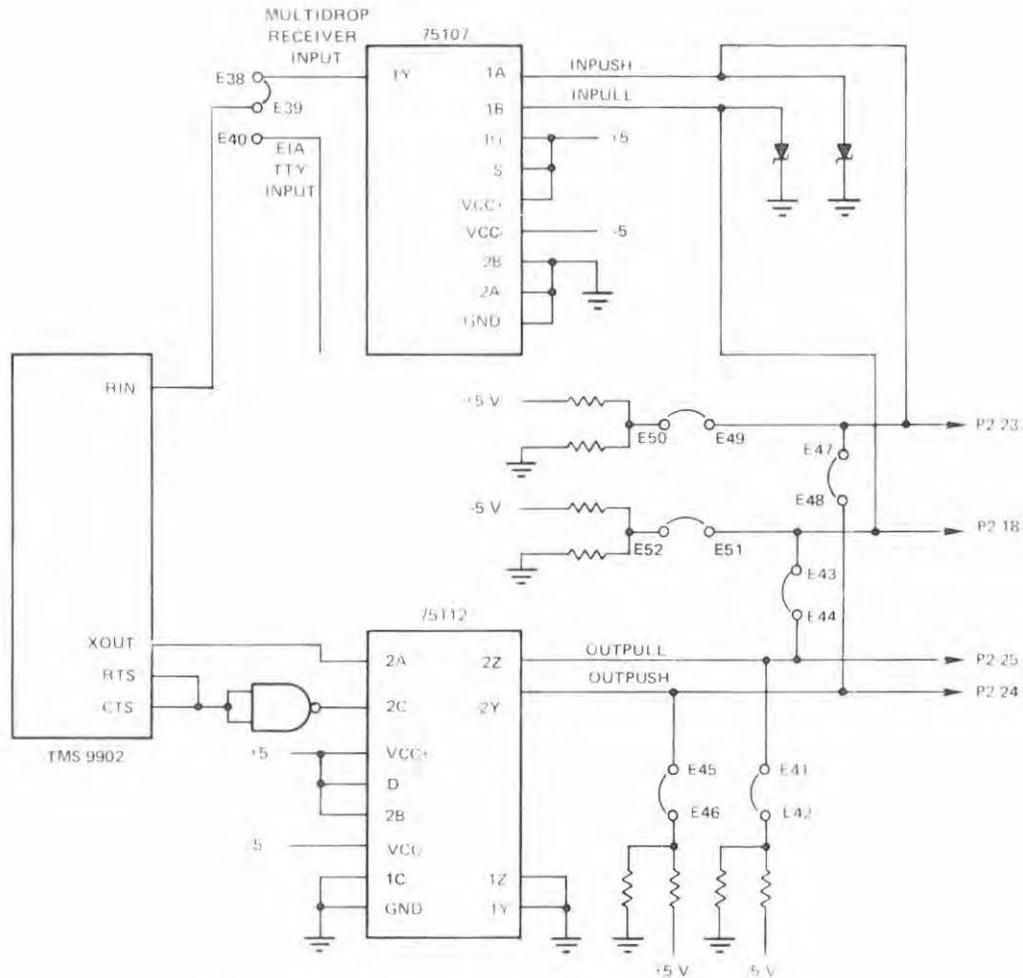
Figure 6-16. Multidrop Interface

The multidrop system, also called the private wire interface, uses a dual set, twisted pair wiring, with operation of these lines in an unbalanced, differential mode. As such, it is a differential line driver/receiver pair which offers higher current drive capability and the noise-free advantages of a balanced line.

6.17 AUXILIARY COMMUNICATIONS PORT

The auxiliary RS-232-C compatible port logic is shown in sheet 6 of the Schematics (Appendix F). All signals for RS-232-C operation are provided. Both terminal and modem communication can be used by proper programming and cable assemblies. Devices such as terminals, modems, and serial line printers, such as the TI 810, all can be attached to this port. Using a TMS 9902, communications are asynchronous. By substituting a TMS 9903 Synchronous Communications Controller, for example, 1200-baud synchronous modems can be used.

This port uses a modified EIA-standard configuration for direct use with RS-232-C-compatible terminals. Signals required by modems are brought out to spare pin positions, which are then rearranged in the special modem cable, the TM 990/506 cable assembly, to the positions required by the modem.

All TMS 9902/9903 signals are brought out to line drivers or receivers. Port. P3 may be configured as either a modem or EIA type interface in the following manner:

(1) If E54 and E55 are jumpered together (terminal position), the $\overline{RTS}$ and $\overline{CTS}$ signals from the TMS 9902/9903 are tied together to form DCD (Data Carrier Detect). The DCD signal is brought out to P3-8. In this configuration, the P3 port appears as a modem to the terminal device. If the user wishes to send characters to a terminal device through the P3 port, he must first make the $\overline{RTS}$ signal to the terminal go low. This is done by writing a 1 to CRU bit 16 of the 9902. By making $\overline{RTS}$ go to 0, the user is also pulling $\overline{CTS}$ to 0, which is the same as asserting DCD. DCD will then be available for terminals requiring that signal for communications.

(2) If E55 and E56 are jumpered together (modem position), $\overline{RTS}$ and $\overline{CTS}$ are distinct signals, both of which are brought out to P3. In this configuration, the P3 port looks like a terminal to the modem connected to P3.

Provisions are made also for Data-Terminal-Ready (P3-21) and Data-Set-Ready (P3-19) and Ring Indicator (P3-22). These three signals are CRU-addressable, outside the range of the TMS 9902/03. DTR is a latched output and the other two are inputs. Use of all signals provided can result in a completely automated communications system. Section 8, Applications, describes several examples for the use of this port, and gives the modem cable configuration as well.

The TMS 9902/9903 at Port P3 can be configured to generate an interrupt at the TMS 9901 by connecting E5 to E6 with the INT5 jumper. If the TMS 9902 is configured in this manner and does generate an interrupt, the interrupt will appear at the TMS 9901 as INT5. Refer to the TMS 9902 or 9903 data manuals for proper interrupt-causing conditions.

6.18 UNIT ID SWITCH
The ID switch is a set of five SPST switches mounted in a DIP packing and connected to a 74LS251 CRU device. Each switch position corresponds to one CRU bit and, in the open or OFF position, represents a logic ONE state. Closing a switch to ground produces a logic ZERO state. Five switches can be set to provide 32 unique codes.

The DIP switch has many applications. Used to pass information to a program, it can function as a "programmer's front panel". Automatic communications systems may have the same software in EPROM for every board in the system: the polling ID for each board is set uniquely in the DIP switch. Alternately, it can be used to pass baud rate and device type information about the auxiliary port to the service programs. The uses for fixing system configuration in the switch, and having one set of standard software, is limited only by the imagination.

6.19 STATUS INDICATOR
The status indicator is a CRU-addressable light emitting diode (LED). Writing a ZERO to CRU address 0000_{16} causes the LED to light; writing a ONE, turns off the LED.

Uses for this feature are again limited only by the imagination. Initialization software can turn it off once initialization is complete. A system error can cause the LED to come on. Test software can blink the LED during execution.

The CLRCRU signal turns the LED ON upon power-up.

SECTION 7

OPTIONS

7.1 GENERAL

This section explains the various options available to the user of the TM 990/101M. These options include:

- Use of TMS 2716 EPROM's (2K x 8 bits each) instead of TMS 2708 EPROM's (1K x 8 bits each)

- On-board expansion of EPROM and RAM

- Asynchronous serial interrupt from one or both of the TMS 9902's

- RS-232-C/TTY/Multidrop interfaces with the Local Serial Port

- Use of slow access time EPROM's by insertion of one WAIT state.

- Use of TM 990/301 Microterminal

- External switch actuation of a RESET or RESTART signal

- Power-up RESET or LOAD

- Memory Map change by reprogramming of the PROM

- Line-by-Line Assembler in EPROM.

Figures 7-1 and 7-2 show board locations applicable to this section. Table 7-1 is a summary of jumpers and capacitors used with these options.

7.2 ON-BOARD MEMORY EXPANSION

7.2.1 EPROM EXPANSION
EPROM memory can be expanded on-board in two ways.

- Add two more TMS 2708 EPROM chips (1K x 8 bits each), for a total of four, to provide an additional 1K words of memory.

- Use two or four TMS 2716 EPROM chips (2K x 8 bits each) to provide 2K or 4K words of memory.

Figure 7-3 shows placement of EPROM chips and corresponding memory addresses (in bytes). The board silkscreen designators identify the necessary jumper placement at E9/E10/E11, E26-E30, and E31-E35.

NOTE
Check the jumper placements on your board against Table 7-2 for proper configuration of your board.

In general, for TMS 2708 use, jumpers are placed as shown in line 1 of Table 7-2; for TMS 2716, they are placed as shown in line 2. These jumpers switch the chip enable and A4 signals as required for the memory device used. Location of RAM and EPROM in opposite ends of memory can be reversed by jumpering E16 to E15 (instead of E16-E17); this starts RAM at M.A. 0000_{16} and EPROM starts in upper memory. In addition, EPROM

E8/E53
NO WAIT
STATE FOR
ON-BOARD
EPROM*

E54/E55
SELECT PORT P3
FOR USE WITH
A TERMINAL*

E20/E21; E22/E23; E24/E25
POWER TO TM 990/301
MICROTERMINAL**

E7/E8
ONE WAIT
STATE FOR
ON-BOARD
EPROM

E18/E19
CONNECTS
P3 PIN 1
TO GROUND*

E55/E56
SELECT PIN P3 FOR
USE WITH A MODEM

E41/E42, E45/E46,
E49/E50, E51/E52
MULTIDROP
INTERFACE
TERMINATION
RESISTORS***

*E16/E17
RAM IN HIGH
MEMORY, EPROM
IN LOW MEMORY

E15/E16
RAM IN LOW
MEMORY
EPROM IN HIGH

***E43/E44
MULTIDROP
INTERFACE

HALF DUPLEX
SELECTORS
E47/E48

E13/E14
SELECTS
ON-BOARD
EPROM*

E39/E40*
CONNECTS
EIA AND TTY
INTERFACE
TO TMS 9902

E12/E13
DESELECTS
ON-BOARD
EPROM

CONNECTS
MULTIDROP
INTERFACE
TO TMS 9902
E38/E39

ENABLES
TTY INTER-
FACE
E36/E37

E9/E10
SELECTS
2716 MODE
ADDRESS MAP

SELECTS
2708 MODE
ADDRESS MAP*
E10/E11

E32/E33
E34/E35

U43, U45
ARE
TMS 2708

CONNECTS INT5 TO P1-17*
E4/E5

CONNECTS INT5 TO
AUXILIARY TMS 9902
E5/E6

CONNECTS
INT4 TO P1-18*
E1/E2

CONNECTS INT4 TO
MAIN TMS 9902
E2/E3

U42, U44 ARE
TMS 2716
E26/E27
E28/E29

U42, U44
ARE TMS 2708*
E27/E28
E29/E30

U43, U45 ARE
TMS 2716
E31/E32
E33/E34

NOTES: *THIS POSITION IS THE NORMAL POSITION ON ALL BOARDS.
        **NORMAL POSITION FOR -1 AND -3 BOARDS ALSO.
        ***NORMAL POSITION FOR -2 BOARDS ALSO.

Figure 7-1. Jumper Placement
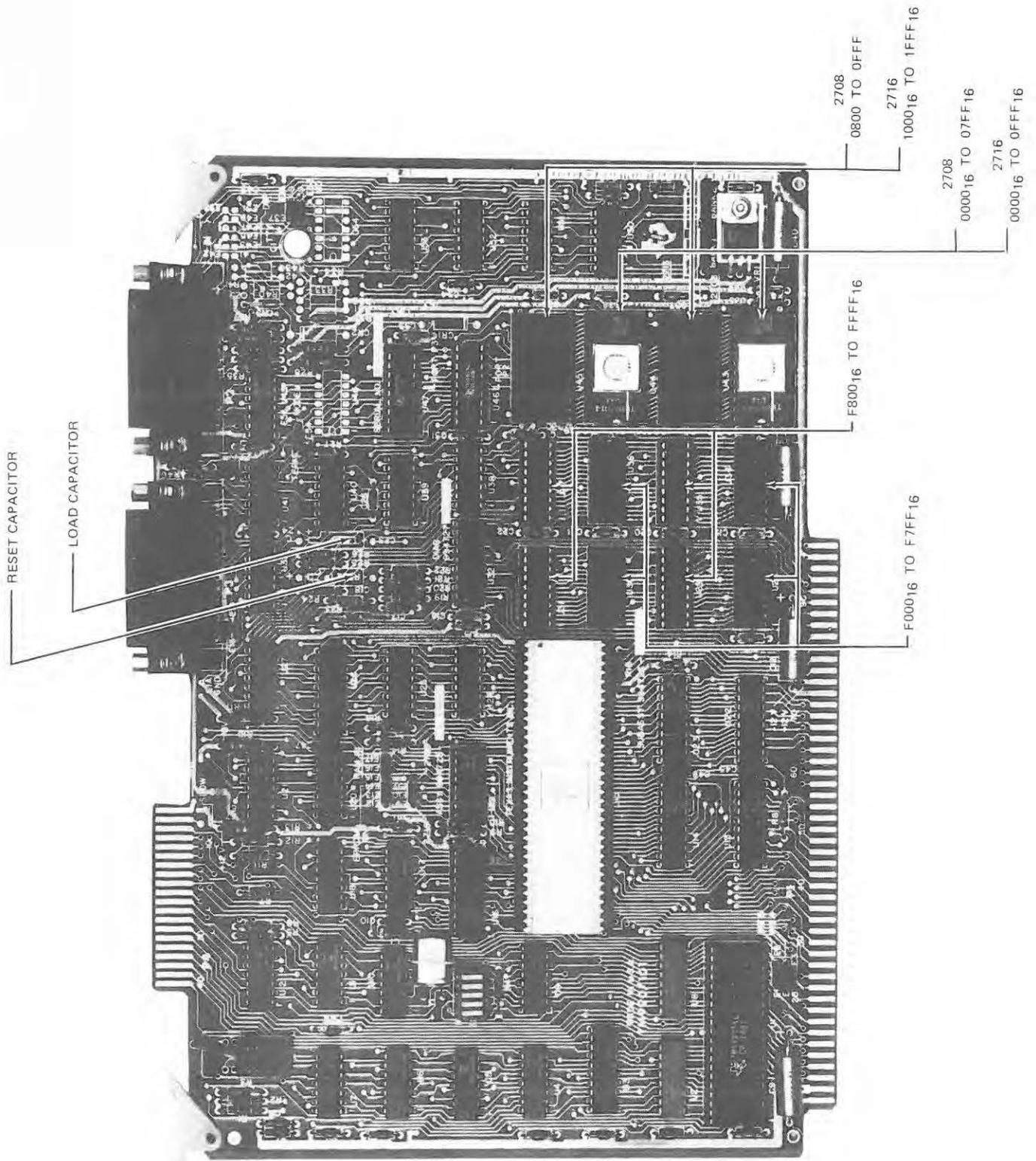
Figure 7-2. Memory and Capacitor Placement

Table 7-1. Master Jumper Table

No. Pins Staked	Pins Connected Together	Function When Connected
3	E1-E2	Connects INT 4 to pin 18 of P1 edge connector
	E2-E3	Connects INT4 to TMS 9902 of LOCAL I/O port
3	E4-E5	Connects INT5 to pin 17 of P1 edge connector
	E5-E6	Connects INT5 to TMS 9902 of REMOTE I/O port
3	E7-E8	Causes 1 WAIT state when on-board EPROM is accessed
	E8-E53	Causes no WAIT state: memory cycles are full speed
3	E9-E10	Selects memory map for TMS 2716 EPROM's
	E10-E11	Selects memory map for TMS 2708 EPROM's
3	E12-E13	On-board EPROM is disabled from memory map
	E13-E14	On-board EPROM is enabled into memory map
3	E15-E16	EPROM at high addresses, RAM in low
	E16-E17	EPROM at low addresses, RAM in high
2	E18-E19	Pin 1 of P3 is connected to GROUND
2	E20-E21	Microterminal: +5 volts to P2-14
2	E22-E23	Microterminal power: +12 volts to P2-12
2	E24-E25	Microterminal power: -12 volts to P2-13
5	E27-E28; E29-E30	Main EPROM is TMS 2708
	E26-E27; E28-E29	Main EPROM is TMS 2716
5	E32-E33; E34-E35	Expansion EPROM is TMS 2708
	E31-E32; E33-E34	Expansion EPROM is TMS 2716
2*	E36-E37	Teletype terminal connected to P2
3	E38-E39	Multidrop Interface in use with LOCAL I/O port
	E39-E40	EIA or TTY interface in use with LOCAL I/O port

Table 7-1. Master Jumper Table (Concluded)

No. Pins Staked	Pins Connected Together	Function When Connected
2 each**	E41-E42,E45-E46 E49-E50,E51-E52	Multidrop termination resistors connected
2 each**	E43-E44, E47-E48	Multidrop Half Duplex operation enabled
3	E54-E55	Connects TMS 9902 RTS to CTS for port P3 to communicate with an EIA compatible terminal.
	E55-E56	Connects TMS 9902 CTS to port P3 directly for communication with an EIA modem.

*On TM 990/101M-1 and -3 only
**On TM 990/101M-2 only

Table 7-2. Jumper Pins by Board Dash Number (Factory Installation)

Board Dash No.	Positions Staked	Jumper Installation at Factory (Positions)
-1, -3	E1-E40, E53-E56	E1-E2 E4-E5 E10-E11 E13-E14 E16-E17 E18-E19 E20-E21 E22-E23 E24-E25 E27-E28 E29-E30 E32-E33 E34-E35 E39-E40 E8-E53 E54-E55
-2	E1-E35, E38-E56	E1-E2 E4-E5 E10-E11 E13-E14 E16-E17 E18-E19 E27-E28 E29-E30 E32-E33 E34-E35 E39-E40 E41-E42 E43-E44 E45-E46 E47-E48 E49-E50 E51-E52 E8-E53 E54-E55

can be disabled from the memory map (in effect, it no longer exists) using jumper E12-E13 (jumper placement E13-E14 enables it onto the memory map).

7.2.2 RAM EXPANSION

Four additional TMS 4045-2 RAM chips can be added as shown in Figure 7-3. This will provide an additional 1K words of RAM. Location of RAM and EPROM at opposite ends of memory can be reversed by jumpering E16 to E15 (instead of E16-E17); this will place RAM starting at M.A. 0000_{16} and EPROM starting in upper memory.

M.A. (HEX)

		JUMPERS
0000		
U42, U44	BANK 1 2 TMS 2708'S (1K X 8 EACH)	E10/E11 E13/E14 E27/E28 E29/E30
0800		
U43, U45	BANK 2 2 TMS 2708'S (1K X 8 EACH) (EXPANSION)	E10/E11 E13/E14 E32/E33 E34/E35
OFFE		

M.A. (HEX)

		JUMPERS
0000		
U42, U44	BANK 1 2 TMS 2716'S (2K X 8 EACH)	E9/E10 E26/E27 E28/E29
1000		
U43, U45	BANK 2 2 TMS 2716'S (2K X 8 EACH) (EXPANSION)	E9/E10 E31/E32 E33/E34
1FFE		

(A) EPROM EXPANSION

M.A. (HEX)

U28, U30, U34, U36	F000 BANK 2 (EXPANSION)	TMS 4045 (EACH 1K X 4 WITH
U29, U31, U35, U37	F800 BANK 1 FFEE	4 IN EACH BANK. TOTAL EXPANSION TO 2K X 16 BITS)

(B) RAM EXPANSION

Figure 7-3. Memory Expansion Maps

7.3 SLOW EPROM
Slow EPROM's can be used with the TM 990/101M by using a jumper between pins E7 and E8. This connects WAIT to READY when on-board EPROM is addressed. Refer to Table 7-3.

Table 7-3. Slow EPROM Table

System Speed	EPROM Type	Access Time	Jumper E7-E8	E8-E53
3 MHz	TMS 2708	450 ns		Installed
3 MHz	TMS 2708	650 ns	Installed	
3 MHz	TMS 2716	450 ns		Installed
3 MHz	TMS 2716	650 ns	Installed	

7.4 SERIAL COMMUNICATION INTERRUPT
Either or both serial ports (TMS 9902's) can be interrupt driven.

* Main Communications Port (EIA/TTY/MD) at P2: interrupt 4.

* Auxiliary Communications Port (EIA) at P3: interrupt 5.

As shown in Figure 7-4, any of four conditions at either TMS 9902 can cause an interrupt condition (change in data set mode, character received, character transmitted, or TMS 9902 timer counted down to zero). An interrupt service routine can check the TMS 9902 bits through the CRU to establish cause of the interrupt, then take appropriate action. Further information is available in the TMS 9902 Asynchronous Controller Data Manual.

7.5 RS-232-C/TTY/MULTIDROP INTERFACES (MAIN PORT, P2)

7.5.1 TTY INTERFACE
Appendix A covers cabling for a Teletype Model 3320/5JE. To use this terminal (20 mA current loop), connect pins E36 and E37 with a jumper plug.

CAUTION
Verify correct voltage levels at connector P2 before attaching a teletypewriter type terminal.

Connect the cable to the terminal and to the microcomputer board. The EIA/MD jumper plug must be connected between pins E39 and E40.

7.5.2 RS-232-C INTERFACE
Appendix B covers cabling for an RS-232-C compatible terminal. To use this type of terminal, disconnect the TTY jumper and make sure the EIA/MD jumper is in the EIA position. Connect the cable to the terminal and to the microcomputer board.
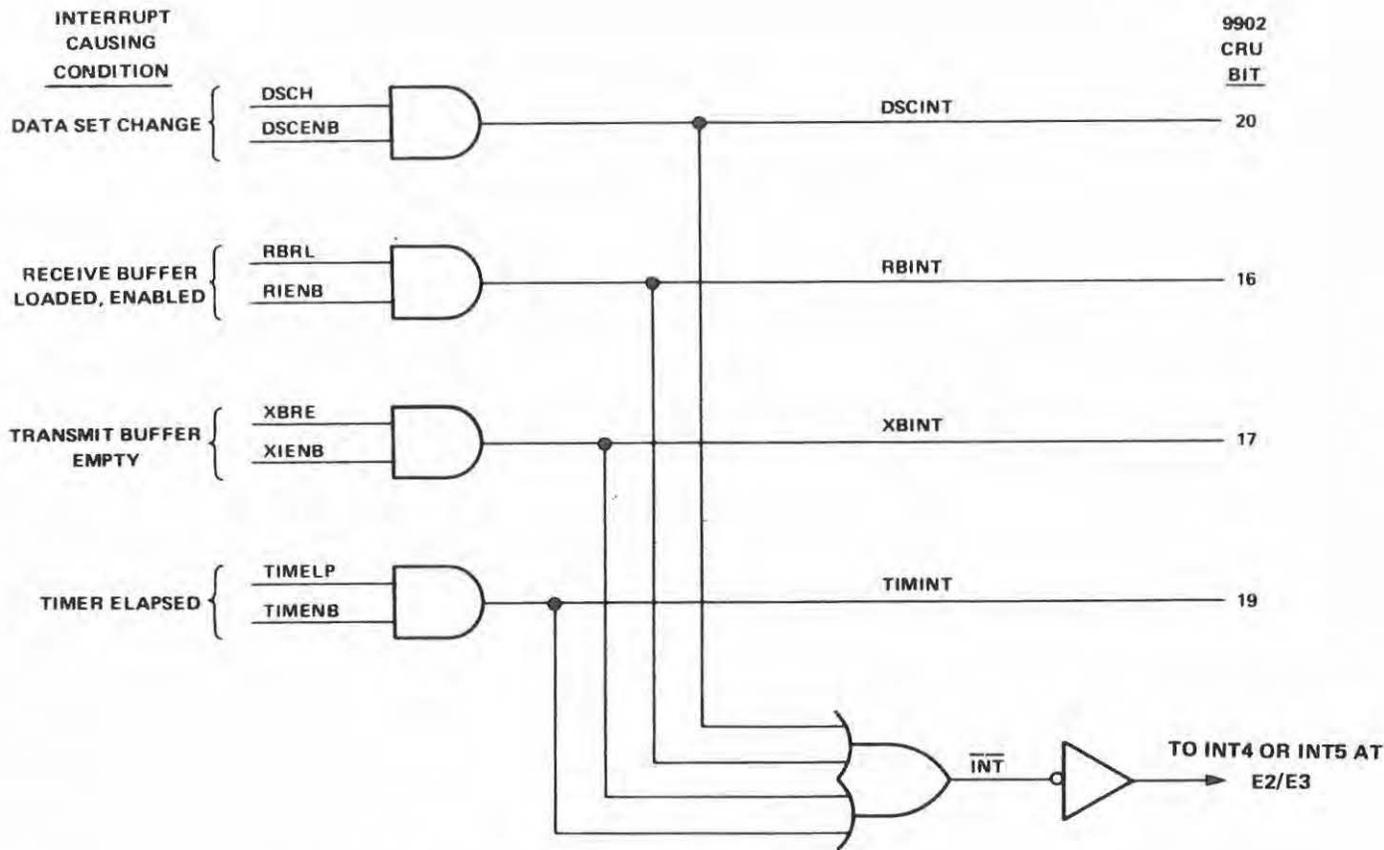
Figure 7-4. Four Interrupt-Causing Conditions At TMS 9902

7.5.3 MULTIDROP INTERFACE

Figure 7-5 shows the multidrop interface in use with a system of TM 990/100-series microcomputer boards. The two boards at the extreme ends of the lines are considered "terminating" boards; whereas, the boards in the middle are non-terminating. Half-duplex operation requires one twisted-pair line (i.e., two wires), and full-duplex operation requires two twisted pairs (i.e., four wires). Refer to Figure 7-6 for cabling.

Table 7-4 shows the jumper configuration for the various configurations. As an example, a common system requirement is for a full duplex board-to-board communication between only two boards. This requirement is fulfilled by the jumper configuration shown on line 4 of the table.

7.5.3.1 Full Duplex Master-Slave

This communications setup is used when there is only one master station and several slave stations. The system setup is shown in Figure 7-7. The advantage of this approach is that one station is in command and control of communication is thus centralized, and also each master-slave communication is full duplex. The half duplex jumpers are removed.
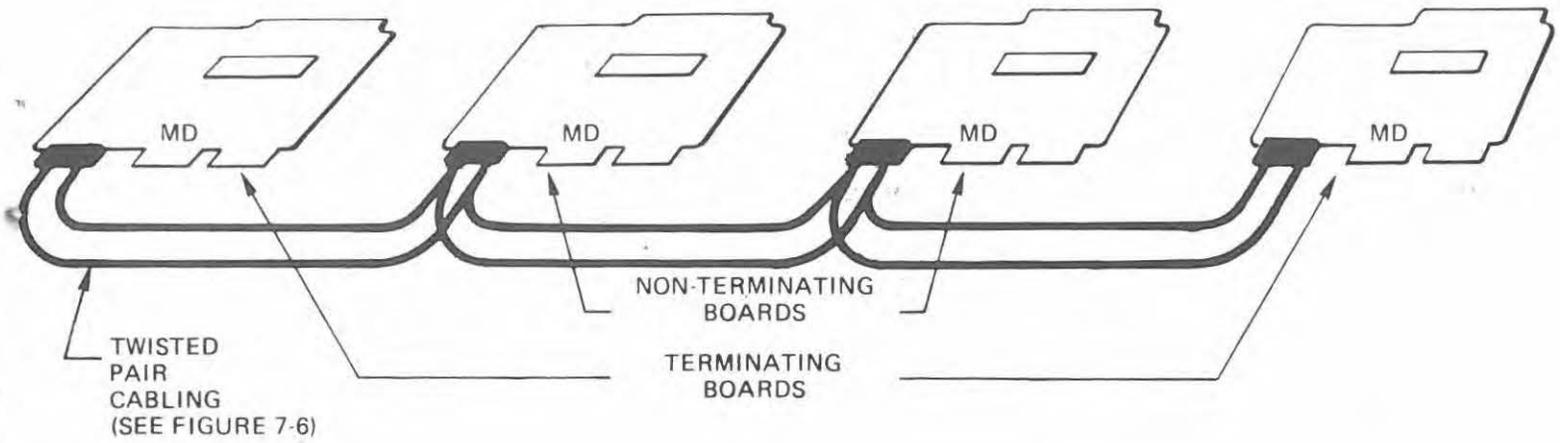
Figure 7-5. Multidrop System
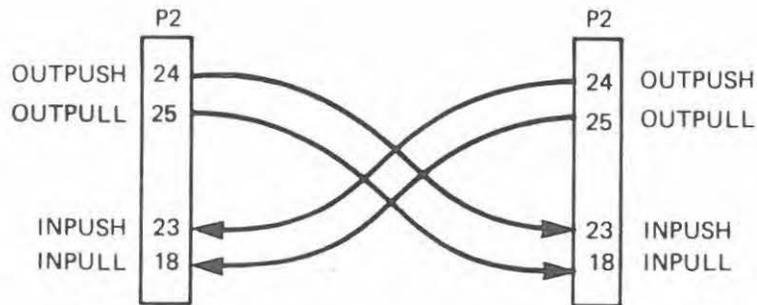
NOTE: ALWAYS CONNECT A "PUSH" LINE TO A "PUSH"
LINE AND A "PULL" LINE TO A "PULL" LINE

Figure 7-6. Multidrop Cabling

Table 7-4. Multidrop Jumper Table

Mode	Install	Remove
Half Duplex, non-terminating	E43-E44, E47-E48	E41-E42, E45-E46, E49-E50, E51-E52
Full Duplex, non-terminating	None	All E41-E52
Half Duplex, terminating	All E41-E52	None
Full Duplex, terminating	E41-E42, E45-E46, E49-E50, E51-E52	E43-E44, E47-E48
All	E38-E39	
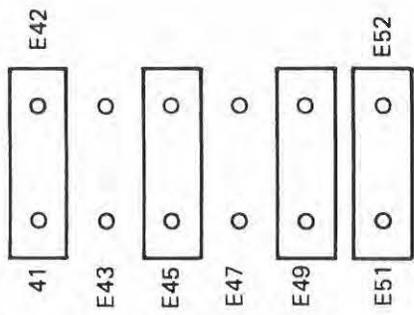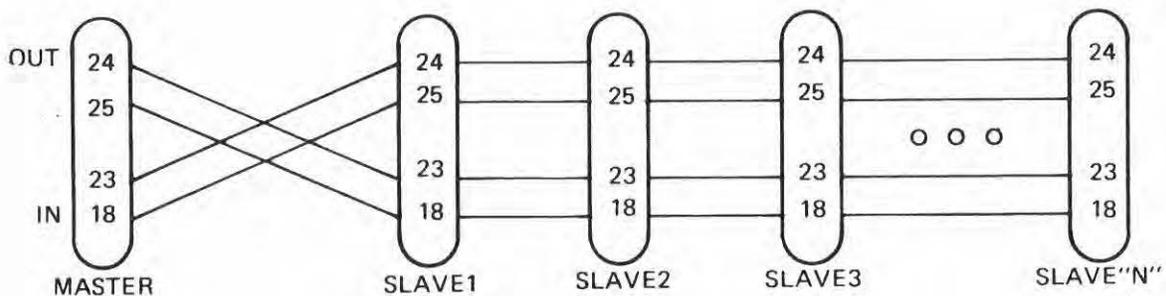
MASTER AND SLAVE "N"
JUMPER ARRANGEMENT.
(OTHERS HAVE NO JUMPERS)

Figure 7-7. Master-Slave Full Duplex Multidrop System

The output of the master station is routed to the input of each slave station. The output of each slave is routed together to the one input of the master. The control codes provided by the master should insure that only one slave transmits at one time. Note four wires total are needed: one pair receive and one pair transmit.

7.5.3.2 Half-Duplex Operation

This configuration is used when only two wires - one pair - is desired. The half duplex jumpers are installed and the one twisted pair is connected at either pins 18 and 23 or pins 24 and 25 of the P2 connector, on all stations. See Figure 7-8.

Protocol must be determined carefully for this configuration to prevent many stations becoming "live" on the lines at once. One station may be appointed master and send control codes, or a round robin technique may be used where control passed from one to another. Conversations are always half-duplex, so when a master station requests a message, it must wait for the addressed station to finish its transmission. This means that control is given up periodically, and a malfunctioning slave station can "hang up" the whole system. This approach does enjoy the advantage of two wires instead of four, though.
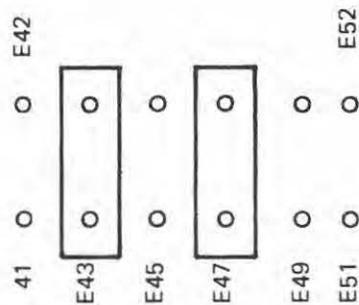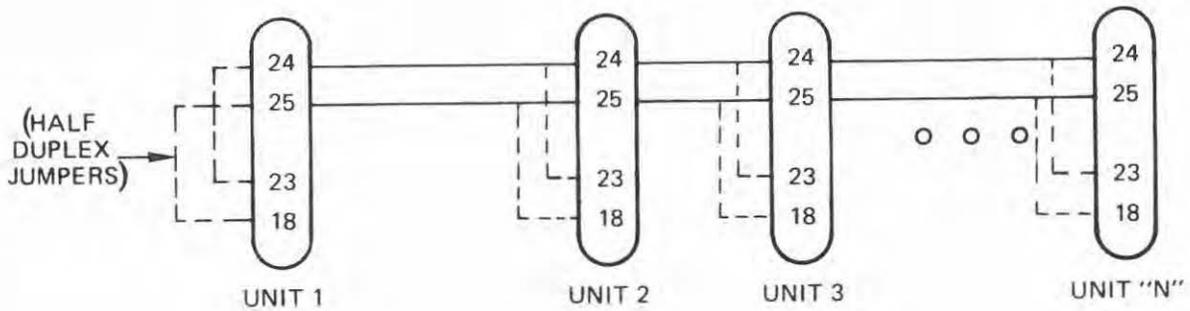
Figure 7-8. Half-Duplex Multidrop System

7.6 EXTERNAL SYSTEM RESET/LOAD

The RESET function is activated from off-board by the assertion of a low state on the PRES.B line, pin 94 on connector P1. An SPST pushbutton to ground can be connected to this line, and should be debounced by a 39 uf tantalum capacitor at C18.

The LOAD function can be activated by asserting a low state on the $\overline{\text{RESTART}}$.B line, pin 93 of connector P1. An SPST pushbutton to ground, with attendant C23 for debouncing, can be used for external actuation.

7.7 REMOTE COMMUNICATIONS

Jumpering pin E18 to E19 connects pins 1 and 7 of connector P2 to ground. Removing this jumper leaves only pin 7 at ground. In some applications, it is not desirable to have signal ground connected to chassis ground, to prevent ground loops or keep an isolated chassis isolated. In these cases, remove the jumper. In most cases, though, there is no special consideration needed, and the jumper may be left in place.

Serial Port P3 can be used to directly communicate with an EIA compatible terminal. This type of operation requires that a jumper plug be installed between E54 and E55, which connects RTS to CTS of the TMS 9902, enabling operation of this device. The terminal with its proper cable (see Appendix B) may be plugged directly into connector P3.

If communications with an EIA compatible modem (see Section 8, Applications, under EIA Serial Port Applications) is desired, insert the jumper plug between pins E55 and E56. This connects CTS of the TMS 9902 to the line receiver on the P3 connector. The TM 990/506 modem cable, or equivalent, must be used.

7.8 MEMORY MAP CHANGE

The entire system memory map is divided into two categories - on-board and off-board. This division as well as the enable lines to on-board blocks of memory, are controlled by a PROM, a 74S287.

Blank PROM's may be programmed by the user to reconfigure the memory map. For a discussion of the pattern generating process, refer to Section 6, Theory of Operations, under Addressing Decoding.

7.9 TM 990/402 LINE-BY-LINE ASSEMBLER

A line-by-line assembler is available, programmed on two TMS 2708 EPROM's. It will assemble each instruction as it is input by the user. The resulting machine code will be printed on the terminal and placed in continuous memory locations. The TIBUG monitor must be present to use the assembler.

No relocatable labels can be used. Jump instructions use dollar-sign plus or minus byte displacements, and symbolic addresses are input as absolute locations. Error codes identify syntax errors (illegal op code), displacement errors (jump instructions), and range errors (e.g., R33). Figure 7-9 is an example of assembly output using the line-by-line assembler.

7.10 TM 990/301 MICROTERMINAL

An alternate to a hard-copy terminal is a TM 990/301 microterminal for user communication to and from the TM 990/101M. The size of a hand-held calculator, the TM 990/301 uses its light-emitting diode (LED) display to show hexadecimal or decimal values. Features of the TM 990/301 include:

- Hexadecimal to signed decimal and signed decimal to hexadecimal conversion of displayed value.

- Display and change contents of Workspace Pointer, Program Counter, Status Register, or CRU ports.

- Increment through memory displaying contents.

- Display and change contents of memory addresses.

- Halt or single step user program execution.

- Begin program execution.

- Keyboard values 0 through F_{16}.

This microterminal comes with its own cable which attaches to the 25-pin connector P2. To supply power to the microterminal, place jumpers at E20/E21 E22/E23 and E24/E25. When the microterminal is not connected, make sure that these jumpers are disconnected. Jumper E39/E40 must be in (EIA position) for microterminal operation. See Figure 7-2.

Figure 7-9 shows the microterminal and cabling to the TM 990/101M.

7.11 OEM CHASSIS
An original equipment manufacturer (OEM) chassis is available. It features slots for four boards, a motherboard backplace interfacing to P1 on the board, and a terminal strip for power, $\overline{PRES}$.B, $\overline{INT1}$.B, and $\overline{RESTART}$.B. A dimensional drawing of the OEM chassis is shown in Figure 7-10. A schematic of the backplace is shown in Figure 7-11. P1 pin assignments are listed in Hable H-1 of Appendix H.

NOTE
Dimension between card slots is one inch.

```
                       ─ MEMORY ADDRESS

                       ─ ASSEMBLER MACHINE CODE

                       ─ USER INPUT SOURCE CODE

FD00        /FE00 ◄───────────── CHANGE MEMORY ADDRESS
FE00 2FA0 XOP @>FE0C,14
FE02 FE0C
FE04        V◆S ◄────────────── SYNTAX ERROR
FE04 0460 B @>0080
FE06 0080
FE08        /FE0C ◄───────────── CHANGE MEMORY ADDRESS
FE0C 434F $CONGRATULATIONS. YOUR PROGRAM WORKS! ◄────── TEXT STATEMENT
FE0E 4E47
FE10 5241
FE12 5455
FE14 4C41
FE16 5449
FE18 4F4E
FE1A 532E
FE1C 2059
FE1E 4F55
FE20 5220
FE22 5052
FE24 4F47
FE26 5241
FE28 4D20
FE2A 574F
FE2C 524B
FE2E 5321
FE30 0707 +>0707
FE32 0700 +>0700
```

Figure 7-9. Line-By-Line Assembler Output

Figure 7-10. TM 990/301 Microterminal

BACKPLANE

0.406
BARRIER
STRIP

1.876

1.876

5.0

0.627
2 PLACES

0.089
2 PLACES

0.128
4 PLACES

11.798

12.548

7.430
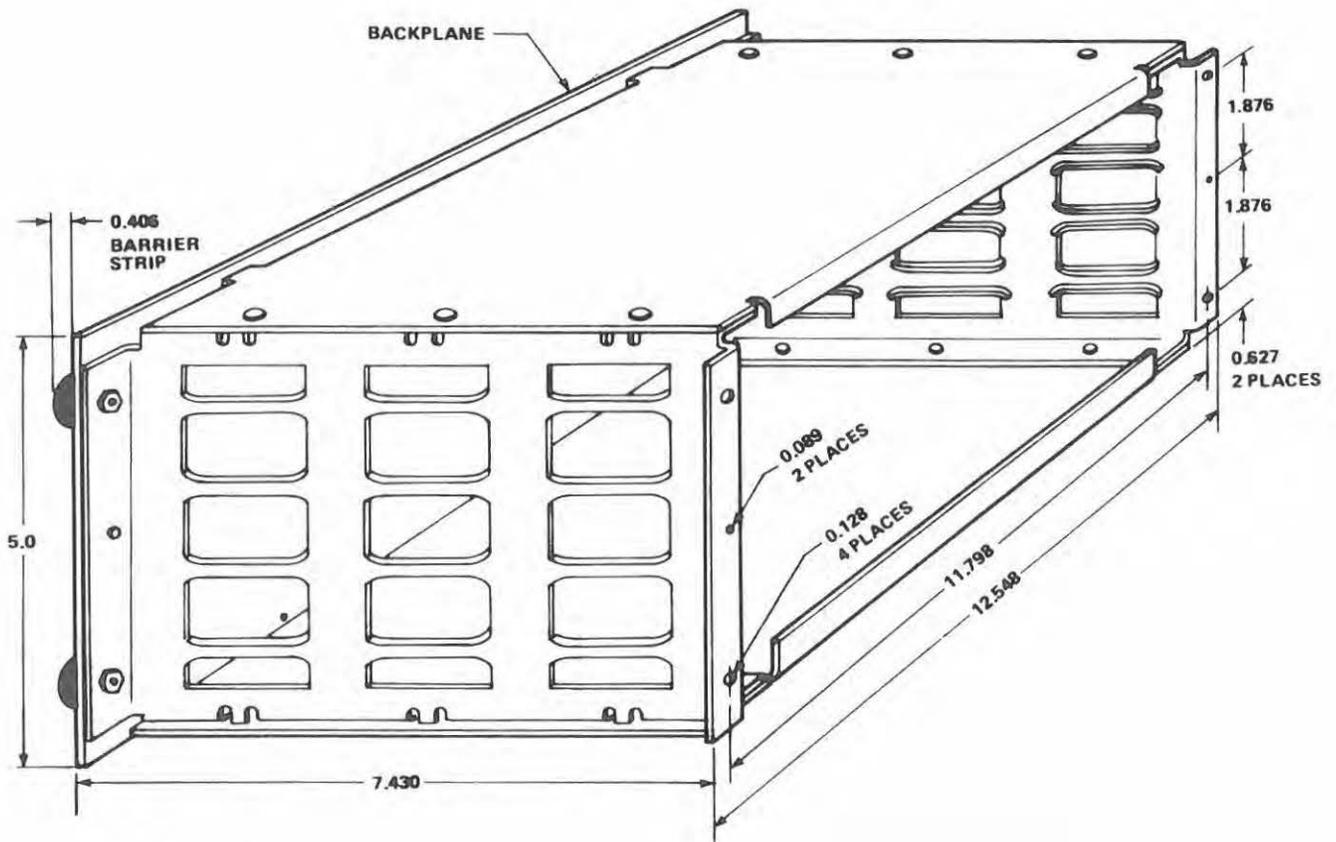
NOTES:
1. DIMENSIONS IN INCHES
2. DISTANCE BETWEEN SLOTS
   IS 1 INCH
3. ALL DIMENSIONS ± 0.010.

Figure 7-11.   TM 990/510 OEM Chassis

NOTE: BACKPLANE PIN ASSIGNMENTS LISTED
IN TABLE H-1 (APPENDIX H).
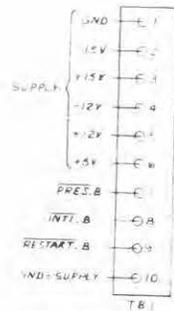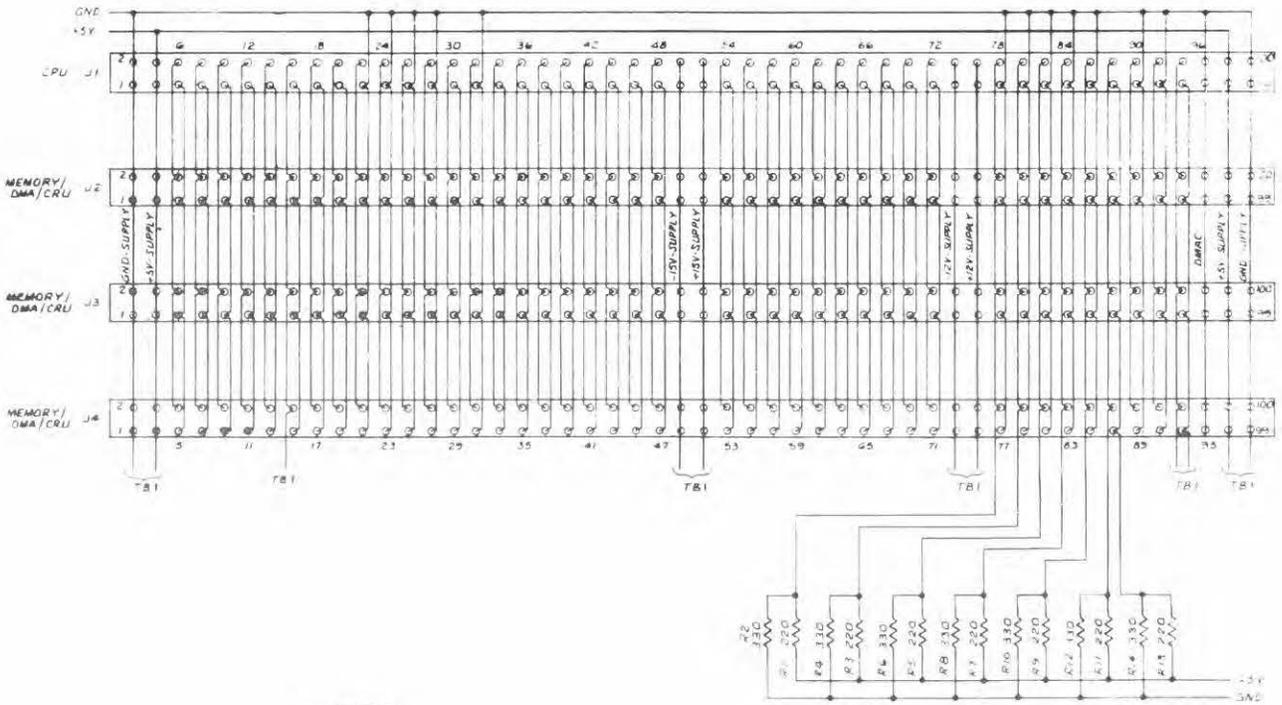
TERMINAL STRIP
IN BACK OF CHASSIS

Figure 7-12. OEM Chassis Backplane Schematic

APPLICATIONS

8.1 GENERAL

This section covers various methods of communicating to applications hardware external to the TM 990/101M. Figure 8-1 shows board locations applicable to this section.

8.2 OFF-BOARD RAM

Figure 8-2 shows a logic diagram for adding additional RAM off-board. The buffers are controlled by the same logic that is used on board the TM 990/101M. The dual flip-flops are used to generate one wait state whenever the memory is enabled. The 74LS155 decodes the five most significant address lines. The A0 and A1 lines select this memory board, and A2, A3 and A4 select one of six banks of expansion RAM. The outputs of the 74LS155 select 1K word banks, starting with the 1Y1 output, which corresponds to an address range of $E800_{16}$ to $EFFF_{16}$. Lines 1Y2 and 1Y3 are not used since they respond to the address range of $F000_{16}$ to $FFFF_{16}$, which are on-board the TM 990/101M. Additional 1K word banks connect to 1Y0, and so on up to 2Y0, which responds to the lowest address in this application, $C000_{16}$.

Alternatively, if the user wishes to address eight banks of RAM on this memory board, using 1Y2 and 1Y3, then the on-board memory can be moved to $B000_{16}$ to $BFFF_{16}$, or some other address, by reprogramming the Memory Address Decoder PROM on board the TM 990/101M.

The 74LS08 bringing $\emptyset 1B$ onto the memory board is used to buffer the system bus, in keeping with the practice that only one LS load per board should appear for a system bus signal. It may easily be omitted. The two 7438's with pull-up resistors attached are used instead of a 74LS04 and 74LS00 to keep down the parts count.

8.3 OFF-BOARD TMS 9901

Figure 8-3 shows the wiring of an off-board TMS 9901 at the CRU bit address $0FE0_{16}$. Only the programmable I/O section is used; the clock and interrupt section is ignored. The R12 bit address is $1FC0_{16}$.

Connection is made through the system bus, P1. The CRUIN, CRUOUT, and CRUCLKB signals are gated by the 1G signal. Chip enable is performed by one 74LS30. Other addresses are not so easy to decode; the use of the various decode chips would enable a bank of TMS 9901's.

8.4 OFF-BOARD EIGHT-BIT I/O PORT

Figure 8-4 shows the wiring of an I/O port with separate 8-bit inputs and outputs. The input is a 74LS251 selector, also known as a TIM 9905. The output is an addressable latch array, a 74LS259 (or a TIM 9906). Address decoding is done by random logic, and the R12 CRU address is 0200_{16}. Note that MEMEN is not used in adress decoding, so this circuit is active even during memory cycles. Again this does no harm since CRUCLKB is inactive and CRUIN is ignored by the processor.

TMS 9902 FOR MAIN PORT P2

TMS 9902 FOR AUXILIARY PORT P3

TMS 9900

CONNECTS INT4 TO MAIN TMS 9902 (E2/E3) OR TO P1-18 (E1/E2)

TMS 9901

CONNECTS INT5 TO AUXILIARY TMS 9902 (E5/E6) OR TO P1-17 (E4/E5)
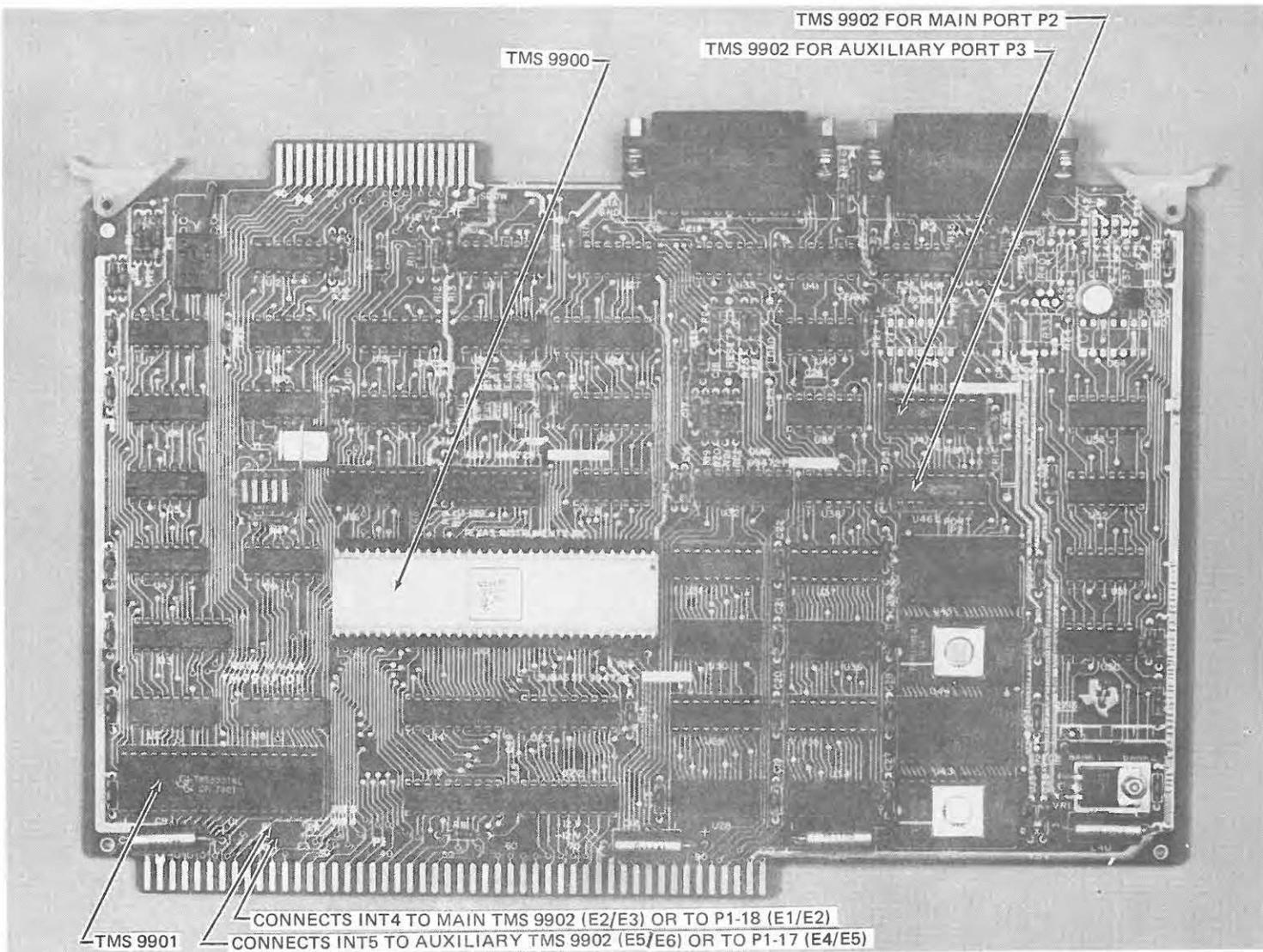
Figure 8-1. Major Components Used in I/O
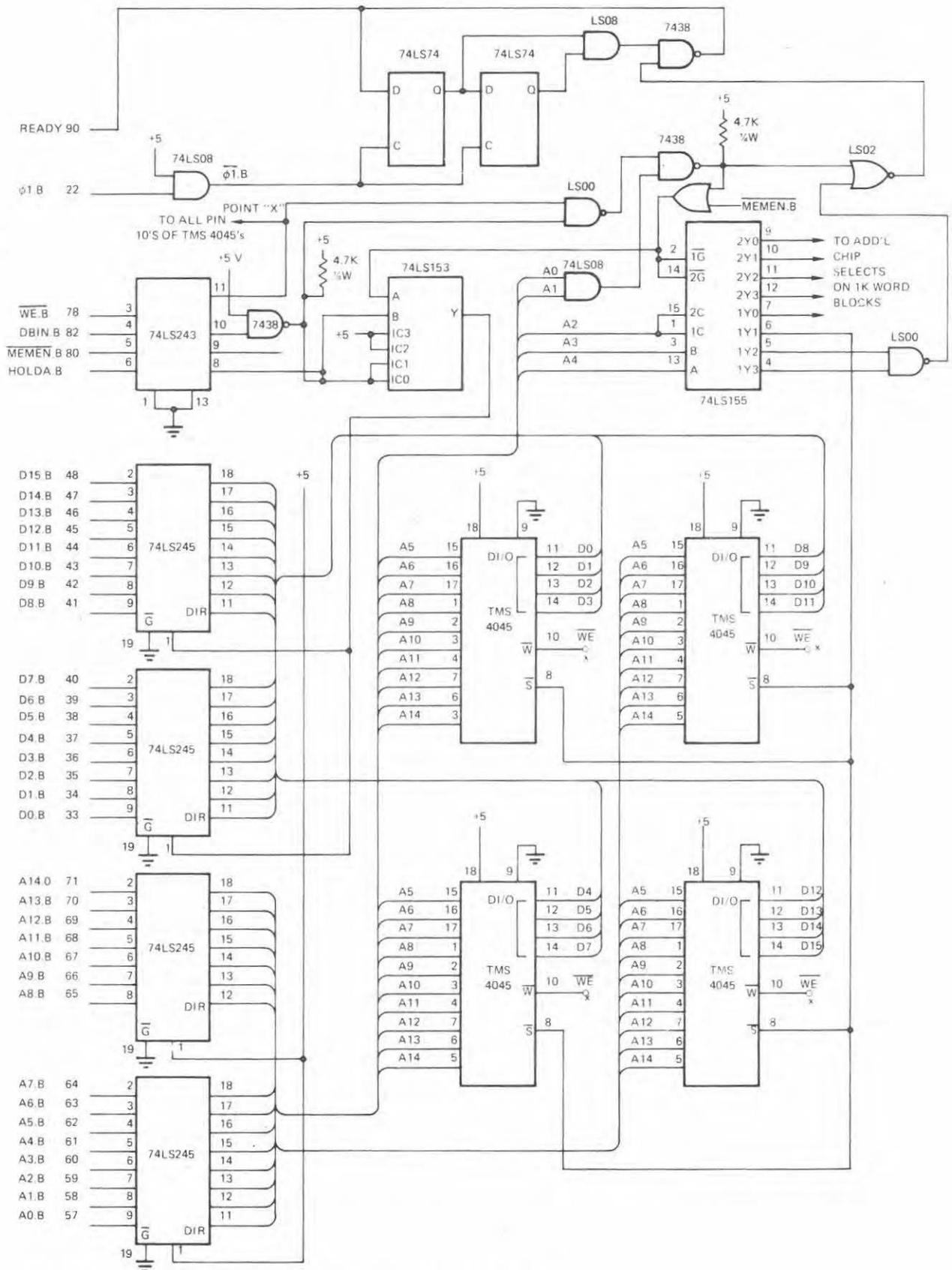
Figure 8-2. Off-Board Memory
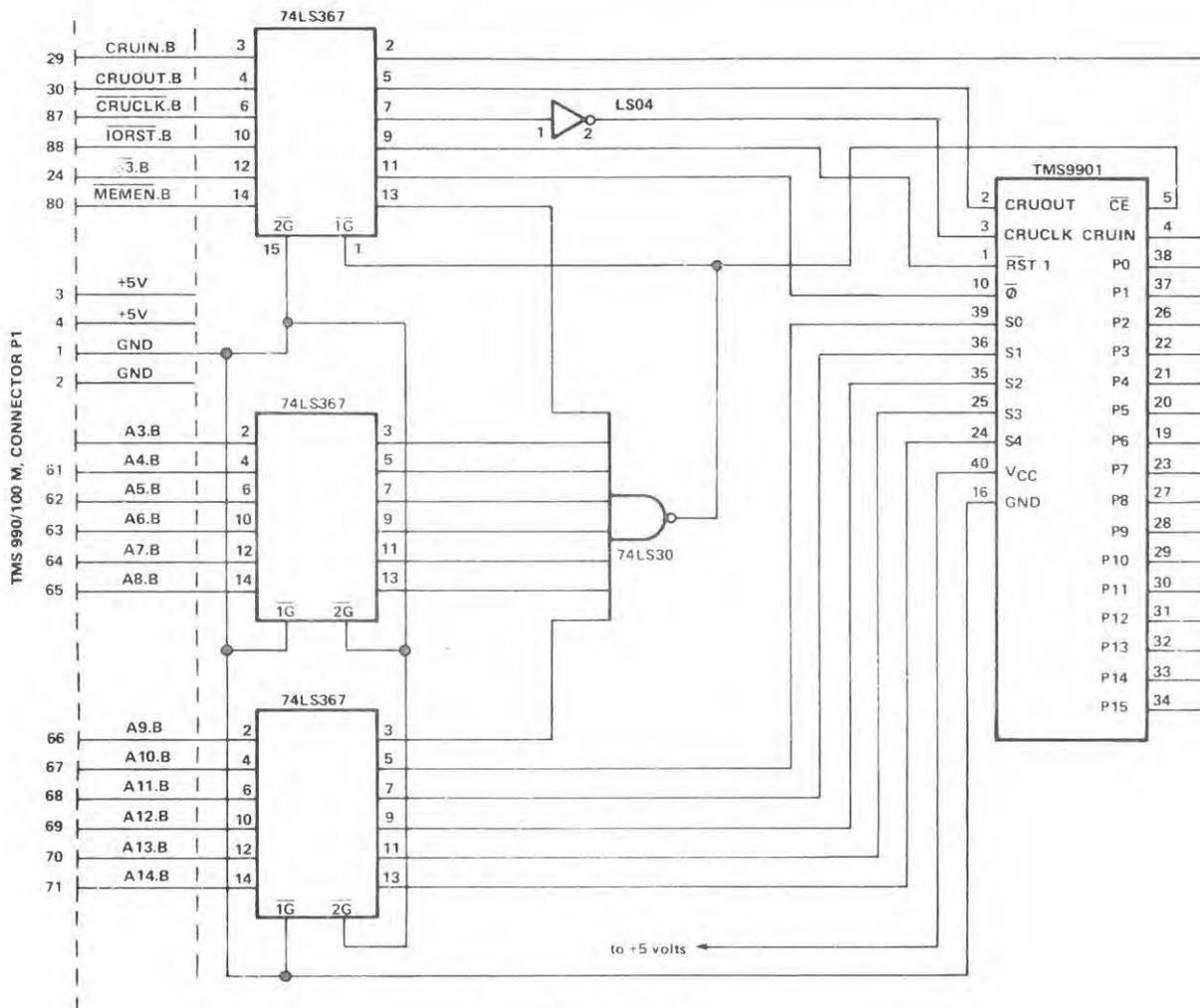
8-3

Figure 8-3. Circuitry To Add TMS 9901 Off-Board
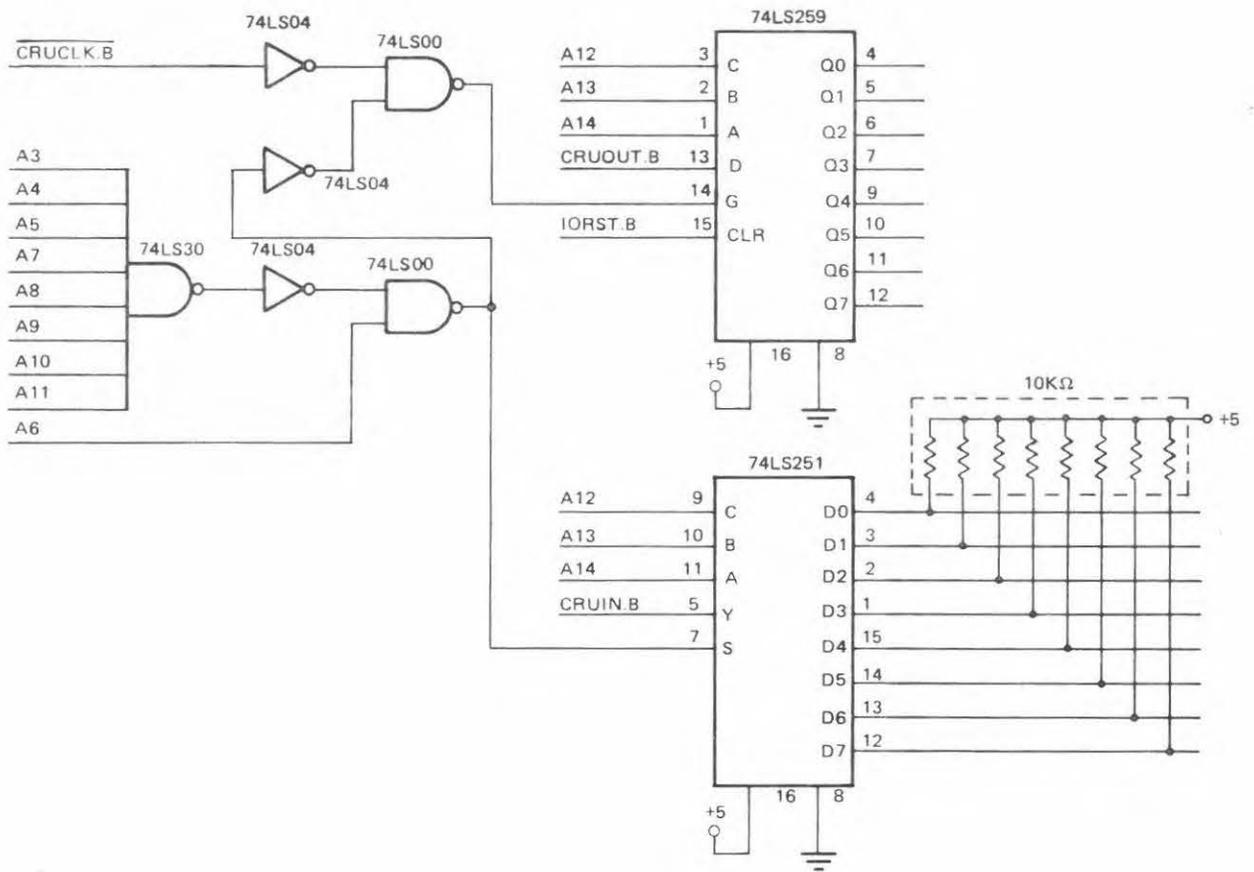
Figure 8-4. 8-Bit 9905/06 Port

8.5 EXTRA RS-232-C TERMINAL PORT

Figure 8-5 shows a diagram of a serial I/O port suitable for most RS-232-C terminals. The handshaking signals used are DATA CARRIER DETECT, which is generated from the REQUEST-TO-SEND tied back to CLEAR-TO-SEND on the TMS 9902, and DATA TERMINAL READY, which is brought into the TMS 9902 for program interrogation. The two 3.3K resistors supply a "fake" CLEAR-TO-SEND and DATA-SET-READY to those terminals requiring them.

Since only half of the packages are used on the 75188 and 75189 devices, another TMS 9902 may be added for an additional serial port. The R12 CRU address is $1FC0_{16}$.

75188: pin 1 = -12, pin 7 = GND, pin 14 = +12
75189: pin 7 = GND, pin 14 = +5
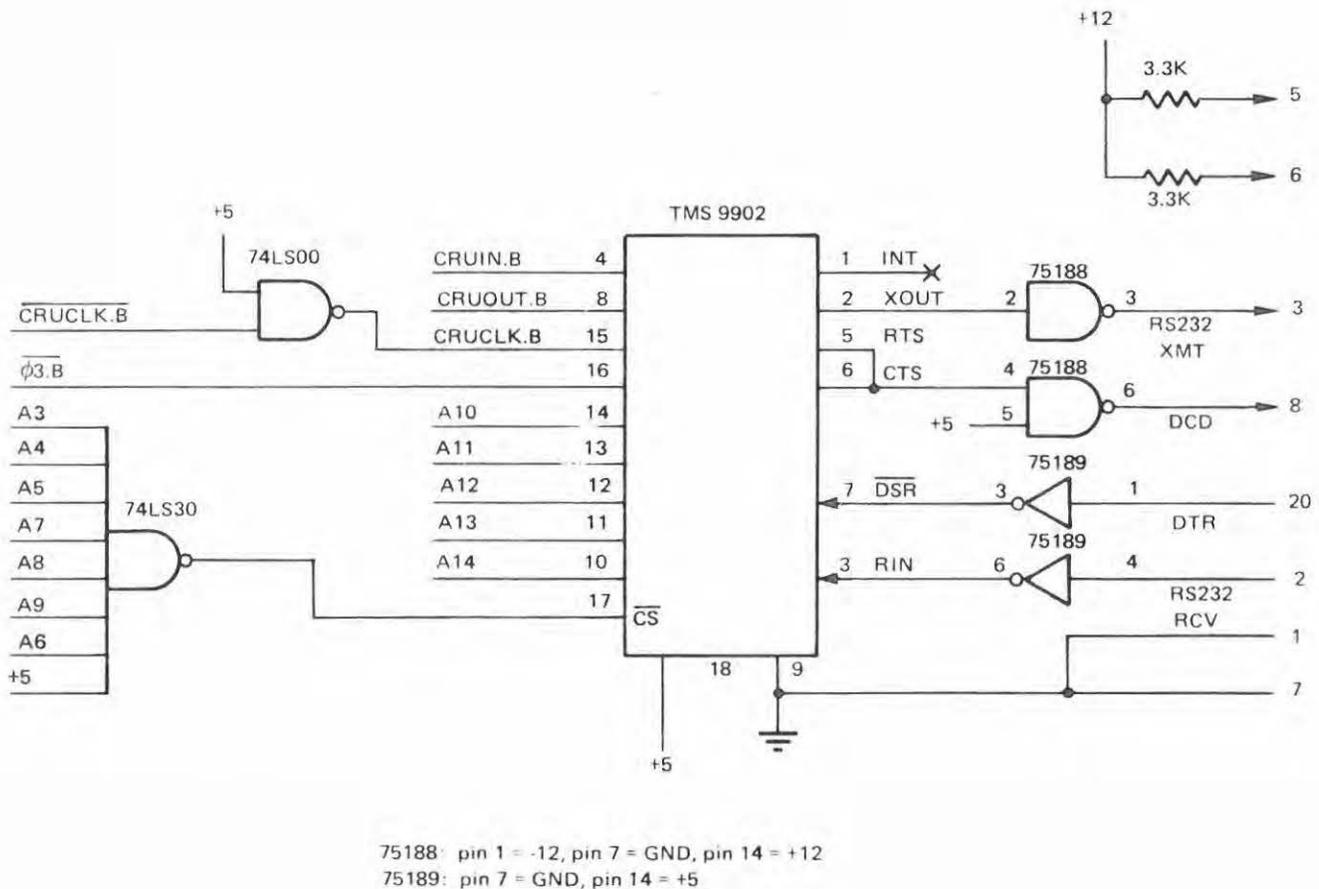
Figure 8-5. RS-232-C Port

8.6 DIRECT MEMORY ACCESS (DMA) APPLICATIONS (FIGURES 8-6 AND 8-7)

The microcomputer 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 result in 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 provide faster I/O response time and higher 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.

Microcomputer direct memory access occurs in block and cycle stealing modes, using the CPU hold capability. The I/O device drives $\overline{HOLD}$ 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 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-MHz 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 333 nanoseconds.

When HOLDA goes high, the CPU address bus, data bus, DBIN, $\overline{MEMEN}$, and $\overline{WE}$ are held 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 the proper timing to transfer data to or from the memory as shown in Figure 8-6. Thus the DMA device has control of the memory bus when the CPU enters the hold state (HOLDA = 1), and may perform memory accesses without intervention by the microprocessor. Because the lines shown in Figure 8-6 go into high impedance when HOLDA = 1, the DMA controller must drive 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 HOLD and normal CPU operation proceeds. TMS 9900 $\overline{HOLD}$ and HOLDA timing are shown in Figure 8-7.

8.6.1 DMA SYSTEM TIMING (FIGURE 8-8)

The Direct Memory Access (DMA) process can be divided into three distinct phases (shown in Figure 8-8):

- Acquisition of memory control from the system.

- Memory control by the DMA device, and

- Release of memory control to the system.

In systems with multiple DMA devices, the memory control phase can be shared by the devices on a priority basis; however, the acquisition and release phases must remain distinct in that the release phase must end before another acquisition phase beings. This is necessary to avoid any memory access conflict resulting from the hold acknowledge signal (HOLDA) delay which occurs when the hold signal ($\overline{HOLD}$) is released.

Figure 8-6. DMA Bus Control

8-8
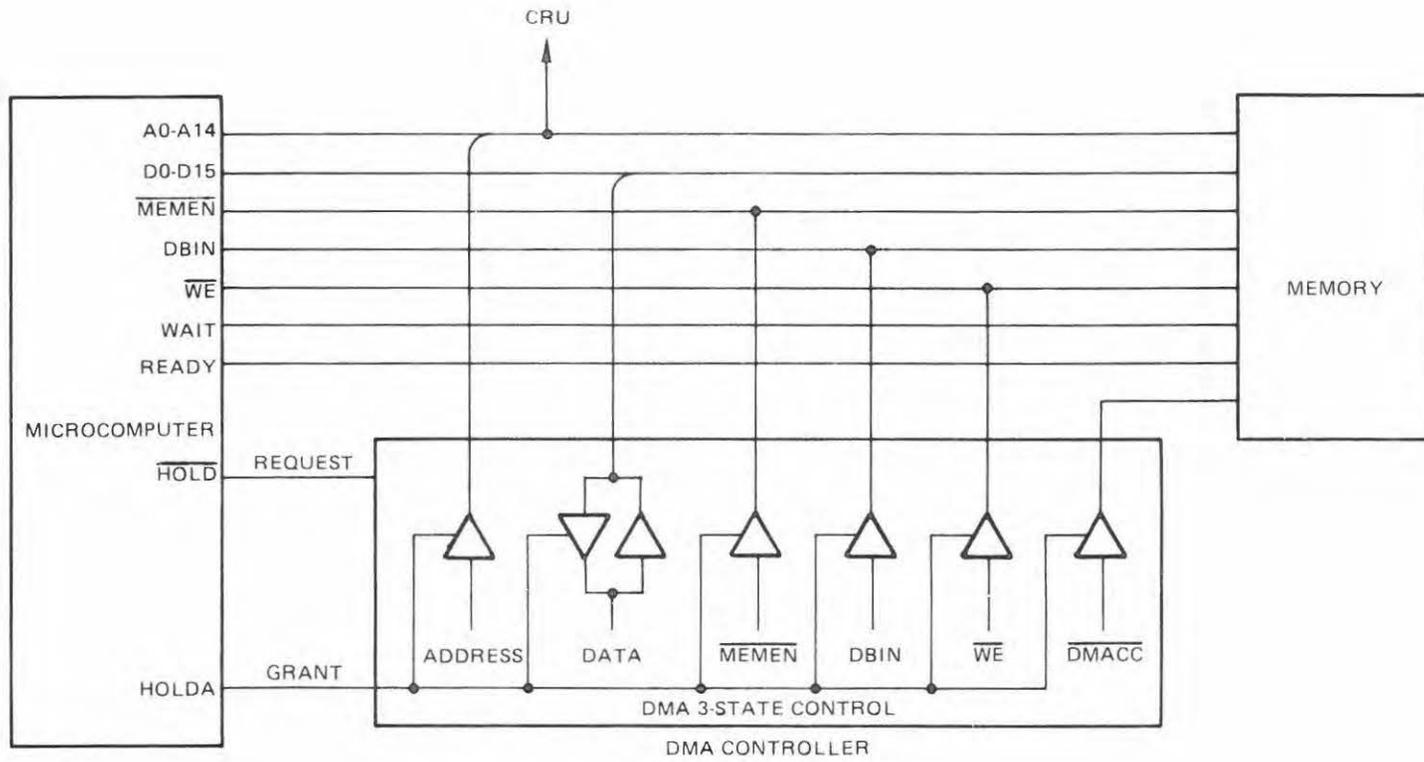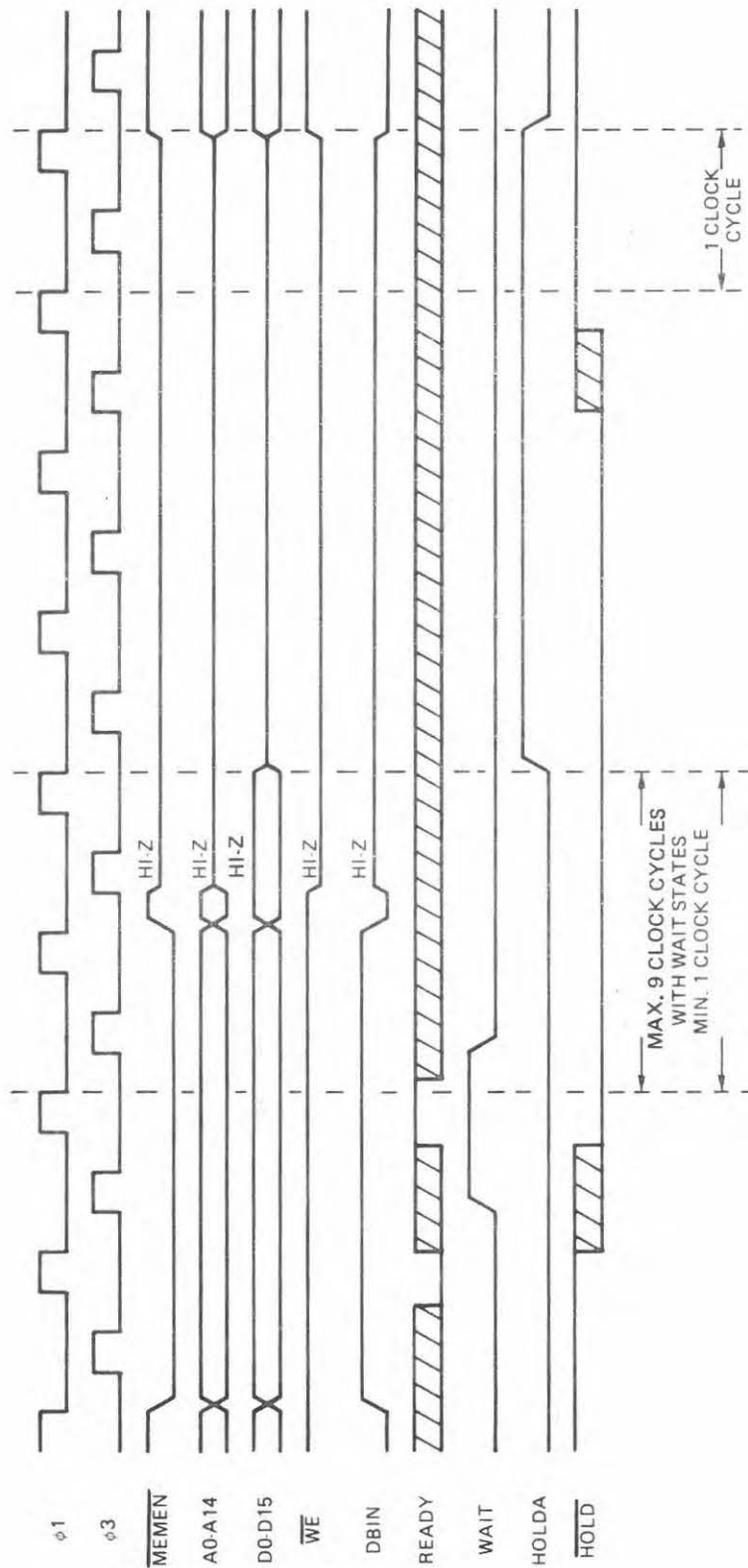
Figure 8-7.   CPU HOLD and HOLDA Timing

Figure 8-8. DMA System Timing
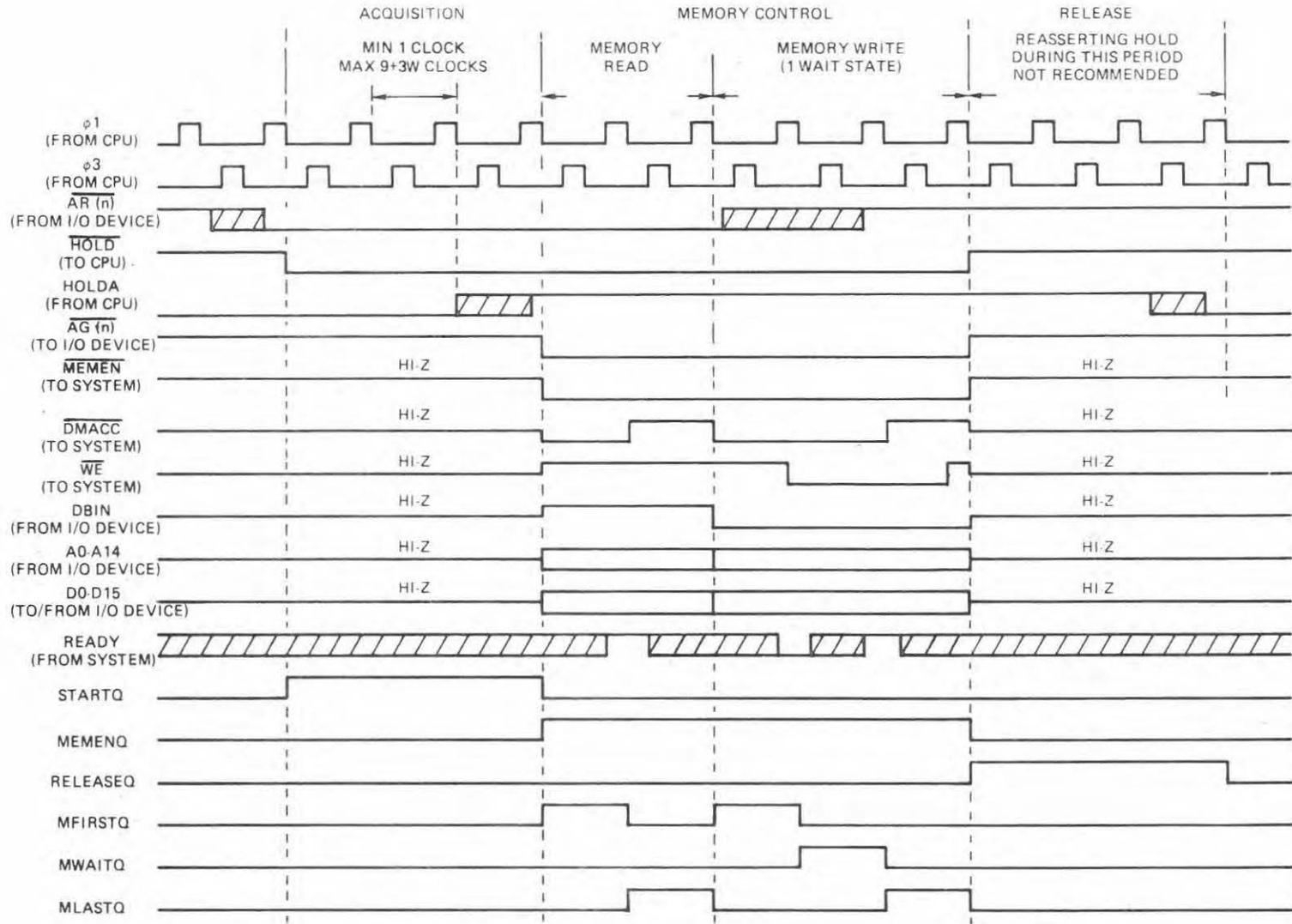
The acquisition of memory control from the system begins when the $\overline{\text{HOLD}}$ signal is asserted by the DMA device. This signal is driven by an open-collector circuit and must be synchronized to the trailing edge of clock phase one ($\emptyset$1). The acquisition phase ends at the first trailing edge of o1 following the receipt of HOLDA. Round-trip timing delays between the DMA device and the CPU must be considered during device controller design.

The control of memory by the DMA device begins at the completion of the acquisition and continues for as many memory cycles as required. The device controller must provide the memory cycle timing signals $\overline{\text{MEMEN}}$, DBIN, $\overline{\text{WE}}$, and $\overline{\text{DMACC}}$ (TM 990 bus signal) as well as the memory address and data signals. The memory cycle timing must duplicate the microcomputer memory cycle timing with respect to minimum setup and hold times and also to synchronization to o1 and o3 clocks. The device controller must monitor the READY signal and wait as required by the memory. The device controller must not require unnecessary wait states (wait states not required by the microcomputer) because of device controller setup timing; however, the device controller can delay the start of a memory cycle to allow setup time for the DBIN, DATA, and address signals.

The release of memory control to the system begins when $\overline{\text{HOLD}}$ is released by the DMA device and is complete when the CPU releases HOLDA. Since the CPU requires two $\emptyset$1 clock cycles for the release of HOLDA, resumption of memory access during the release phase can cause a memory access conflict when the DMA device responds to HOLDA just prior to HOLDA being released. This conflict will cause loss of data and possibly modification of random memory locations.

8.6.2 MEMORY CYCLE TIMING (FIGURE 8-9)
As shown in Figure 8-9, a memory cycle consists of two states, MFIRSTQ and MLASTQ, plus wait states MWAITQ as required by memory. Each state is one o1 clock cycle long. If additional DBIN, data or address setup time is required, a setup state can be inserted before the MFIRSTQ state. The MLASTQ states marks the end of a memory cycle. Read data will be stable at the end of MLASTQ. The control signals $\overline{\text{MEMEN}}$ and $\overline{\text{HOLD}}$ which are static during a memory cycle are allowed to change at the end of MLASTQ. In a multichannel-DMA controller, the device access granted signals are allowed to change at the end of MLASTQ.

8.6.3 DMA SYSTEM GUIDELINES
1. DMA and CPU memory cycle timing should be identical.

2. DMA memory cycles can include memory-dependent wait states.

3. DMA devices must not require memory to insert wait states.

4. DMA devices must allow HOLDA to drop after releasing $\overline{\text{HOLD}}$ prior to reasserting $\overline{\text{HOLD}}$.

5. Three-state bus conflicts must be avoided.

6. Multiple DMA devices must not attempt simultaneous memory access.

7. Sufficient data and address setup times prior to $\overline{\text{WE}}$ must be maintained.

8. Most DMA device timing problems will occur at the first and last memory accesses and at device to device changeover in systems with multiple devices.
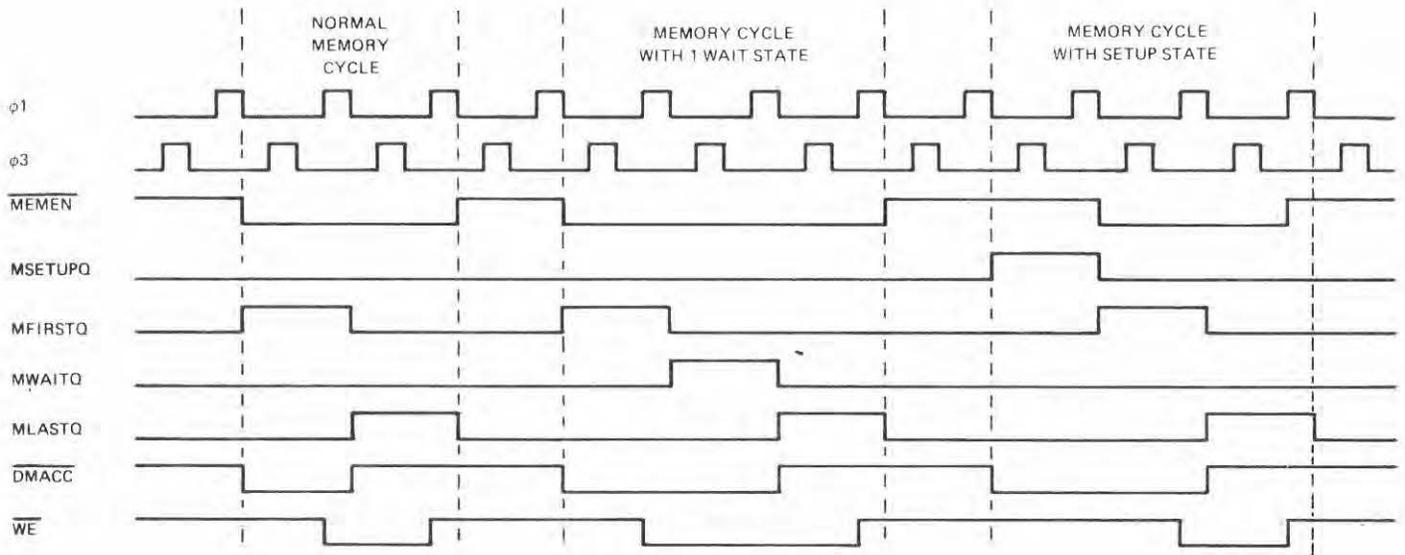
Figure 8-9. Memory Cycle Timing

8.6.4 MULTIPLE-DEVICE DIRECT MEMORY ACCESS CONTROLLER

This section outlines the design of an eight-device, priority-access controller for the direct memory access system shown in Figure 8-10. The controller accepts access requests from the device controllers, acquires memory from the CPU, grants memory access to the highest-priority device switching from device to device as required, and generates all necessary memory cycle timing signals.

The DMA controller interfaces with the device controllers (shown in Figure 8-11) through a DMA control bus consisting of access request ($\overline{AR0}$ through $\overline{AR7}$), access granted ($\overline{AG0}$ through $\overline{AG7}$), and memory cycle complete ($\overline{MCOMP}$) signals. To access memory a controller asserts access request and waits for access granted. The controller then drives the address bus (A0 through A15), and the data bus (D0 through D15) as required, and the DBIN signal. The $\overline{MCOMP}$ signal indicates that the memory cycle will be complete and read data will be stable on the data bus at the trailing edge of the o1 clock. A device can request multiple memory cycles by continuously asserting access request. Access request is released during the first clock cycle of the last required memory cycle.

8-12
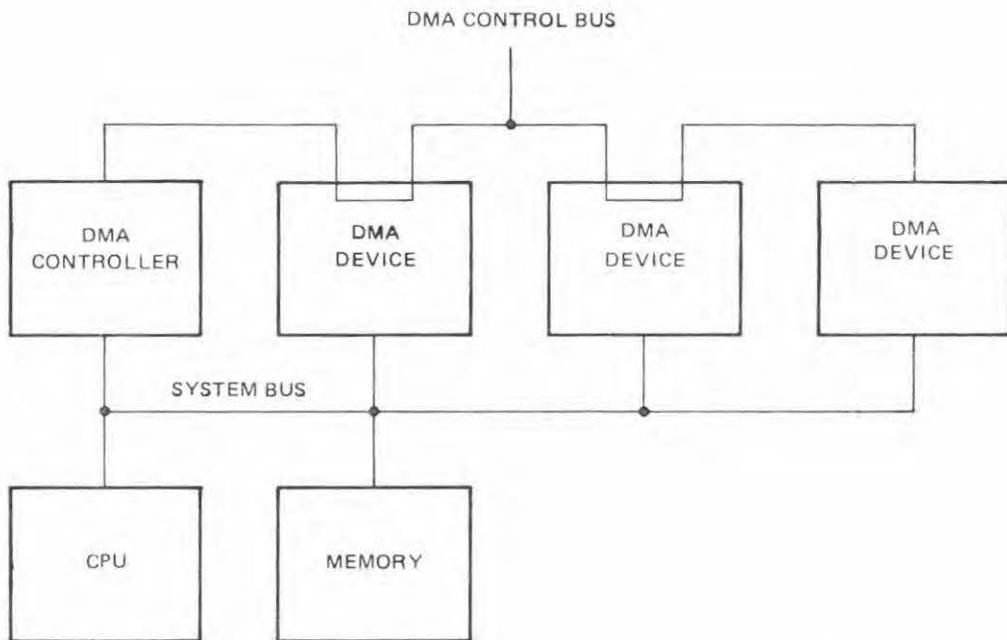
DMA CONTROL BUS

Figure 8-10. DMA System Block Diagram
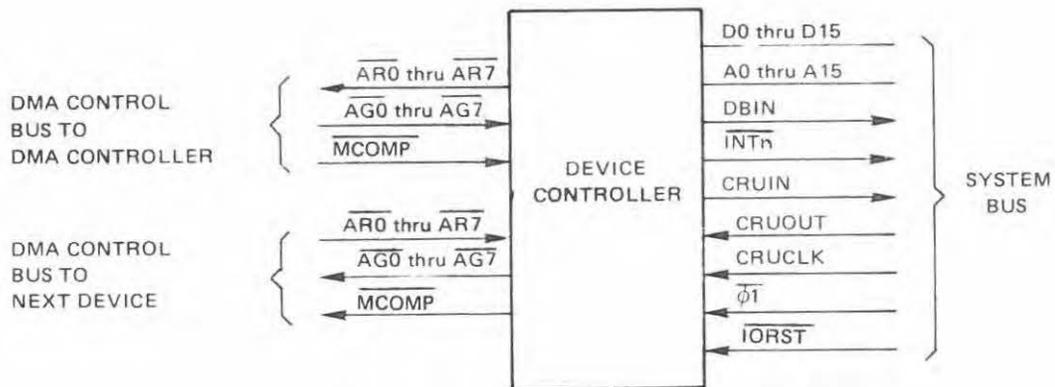
Figure 8-11. DMA Device Controller

The DMA controller (shown in Figure 8-12) provides memory access control, memory cycle timing, and priority-based access of memory by the device controllers. Access requests are synchronized to system clock, then prioritized using a priority encoder followed by a decoder. The priority encoder also provides the signal DMAR which indicates if any device is requesting access. Memory access is granted to the highest-priority device when HOLDA is received from the CPU and at the end of each memory cycle. This is done by loading a register with the decoder outputs. If no device is requesting access, the decoder is disabled and the register is loaded thus disabling all access granted signals. Loading of the register is inhibited from the time $\overline{HOLD}$ is released by the DMA controller until HOLDA is released by the CPU in order to avoid an access conflict between the DMA and the CPU due to the HOLDA response time.
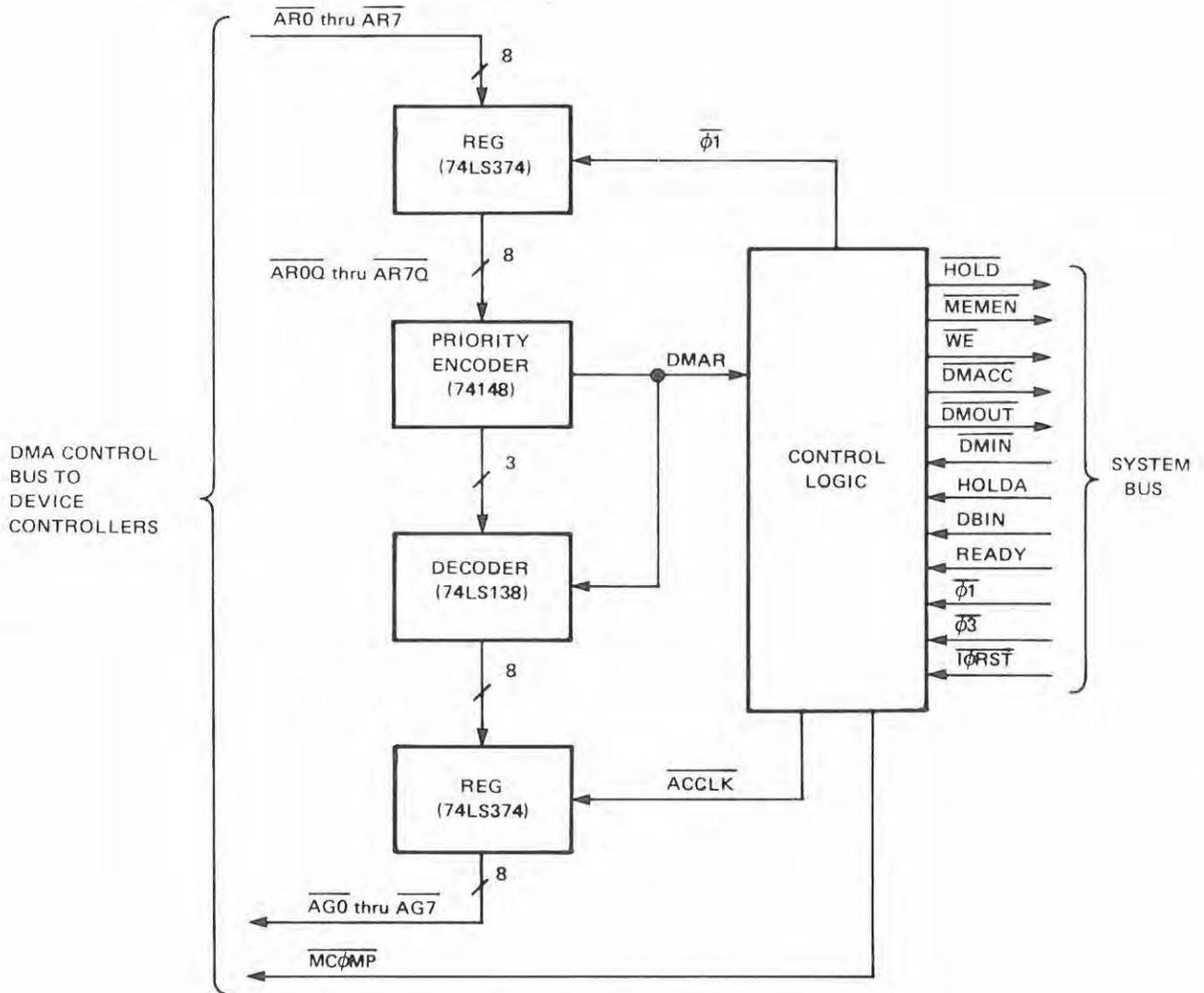
Figure 8-12. DMA Controller

The DMA controller timing with priority contention is shown in Figure 8-13. The logic equations for the DMA controller are:

$$\text{DMAR} = \text{AROQ} + \text{AR1Q} + \ldots + \text{AR7Q}$$

$$\text{STARTQ}_J = \text{DMARQ} \bullet \overline{\text{MEMENQ}} \bullet \overline{\text{RELEASEQ}}$$

$$\text{STARTQ}_K = \text{HOLDA} \bullet \text{STARTQ}$$

$$\text{MEMENQ}_J = \text{HOLDA} \bullet \text{STARTQ} = \text{STARTQ}$$

$$\text{MEMENQ}_K = \overline{\text{DMARQ}} \bullet \text{MLASTQ}$$

$$\text{RELEASEQ}_J = \overline{\text{DMARQ}} \bullet \text{MLASTQ} = \text{MEMENQ}$$

$$\text{RELEASEQ}_K = \overline{\text{HOLDA}} \bullet \text{RELEASEQ}$$

$$\text{HOLD} = \text{DMAR} \bullet \overline{\text{RELEASEQ}} + \text{STARTQ} + \text{MEMENQ}$$

$$\text{MFIRSTQ}_D = \text{HOLDA} \bullet \text{STARTQ} + \text{DMAR} \bullet \text{MLASTQ}$$

$$\text{MWAITQ}_D = \text{MFIRSTQ} \bullet \overline{\text{READY}} + \text{MWAITQ} \bullet \overline{\text{READY}}$$

$$\text{MLASTQ}_D = \text{MFIRSTQ} \quad \text{READY} + \text{MWAITQ} \bullet \text{READY}$$

$$\text{WEQ}_D = \overline{\text{DBIN}} \bullet \text{MFIRSTQ} + \text{WEQ} \quad \text{MWAITQ}$$

$$\text{DMACC} = \text{MFIRSTQ} + \text{MWAITQ}$$

$$\text{ACGATE} = \text{HOLDA} \bullet \text{STARTQ} + \text{MLASTQ}$$

$$\text{ACCLK} = \overline{\text{ACGATE} \bullet \emptyset 1}$$

$$\text{MCOMP} = \overline{\text{MLASTQ}}$$

where signals ending with the letter Q are flip-flop outputs and signals with subscripts are the corresponding flip-flop inputs. All flip-flops are code-triggered on the trailing edge of $\emptyset 1$ except WEQ ($\emptyset 1$ leading edge).

8-15
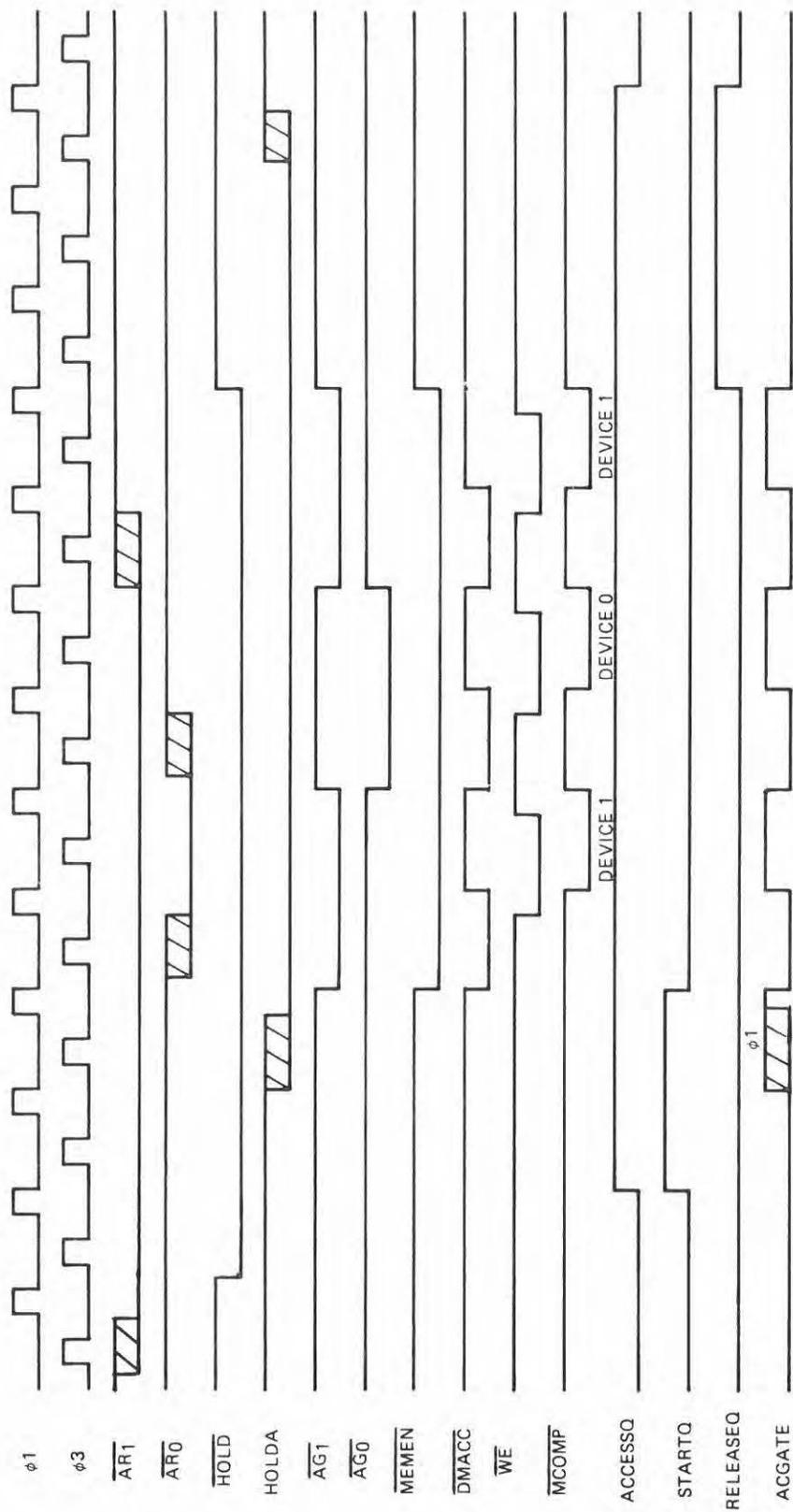
Figure 8-13. DMA Controller Timing

8-16

8.7 EIA SERIAL PORT APPLICATIONS

This section describes the cable configurations and connector pin assignments used with the microcomputer EIA serial port (connector P3). Interconnection information is included for 103-, 202-, and 201- series modems and EIA data terminals. A typical system configuration is shown in Figure 8-14. TI offers a ready-made cable for use with all of the above modems, the TM 990/506.
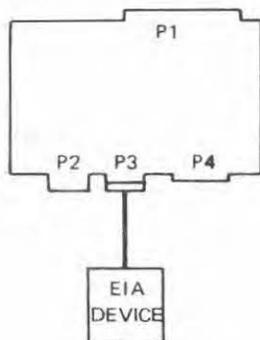
Figure 8-14. Cable Connections

8.7.1 CABLE PIN ASSIGNMENTS

Tables 8-1, 8-2, 8-3, and 8-4 provide pin assignment information for interface cables.

Table 8-1. 103/113 Data Set Cable

101 Pin On P3 (Male)	Modem Pin 103/113 (Male)	RS-232-C Circuit	Function
1	1	AA	Protective Ground
3	2	BA	Transmitter Data
2	3	BB	Receiver Data
8	4	CA	Request to Send
16	5	CB	Clear to Send
19	6	CC	Data Set Ready
7	7	AB	Signal Ground
20	8	CF	Received Line Signal Detector (DCD)
21	20	CD	Data Terminal Ready
22	22	CE	Ring Indicator

Table 8-2. 202/212 Data Set Cable

101 Pin On P3 (Male)	Modem Pin 202/212 (Male)	RS-232-C Circuit	Function
1	1	AA	Protective Ground
3	2	BA	Transmitter Data
2	3	BB	Receiver Data
8	4	CA	Request to Send
16	5	CB	Clear to Send
19	6	CC	Data Set Ready
7	7	AB	Signal Ground
20	8	CF	Received Ling Signal Detector (DCD)
21	20	CD	Data Terminal Ready
22	22	CE	Ring Indicator

Note: Pins 11 and 12 (reverse channel on 202) are not connected

Table 8-3. 201 Data Set Cable

101 Pin On P3 (Male)	Data Set Pin 201 (Male)	Circuit 201	Function
1	1	AA	Protective Ground
3	2	BA	Transmit Data
2	3	BB	Receive Data
8	4	CA	Request to Send
16	5	CB	Clear to Send
19	6	CC	Data Set Ready
7	7	AB	Signal Ground
20	8	CB	Data Carrier Detect
15	15	DB	Transmitter Signal Element Timing
17	17	DD	Receiver Signal Element Timing
21	20	CD	Data Terminal Ready
22	22	CE	Ring Indicator

Note: Pin 14 (new synchronization) is not connected

Table 8-4. Data Terminal Cable

101 Pin On P3	Data Terminal Pin (Female)	RS-232-C Circuit	Function
1	1	AA	Protective Ground
2	2	BA	Transmitter Data
3	3	BB	Receiver Data
4	4	CA	Request to Send
5	5	CB	Clear to Send
6	6	CC	Data Set Ready
7	7	AB	Signal Ground
8	8	CF	Data Carrier Detect
20	20	CD	Data Terminal Ready

8.7.2 MODEM (DATA SET) INTERFACE SIGNAL DEFINITIONS

8.7.2.1 Pin 1 (AA) Protective Ground
This interface lead is connected to signal ground of the microcomputer by connecting pin E18 to E19 with a jumper.

8.7.2.2 Pin 2 (BA) Transmitter Data
The interface lead provides the electrical connection from the microcomputer to the associated data set for the purpose of transferring a bit-by-bit serialization of the data which is to be transmitted across the communication channel. In the time domain, character information presented on this lead will appear least significant bit first through most significant data bit. In asynchronous systems, each character serialization will be preceded by a start bit and followed by one or more stop bits.

8.7.2.3 Pin 2 (BB) Receiver Data
This interface lead provides the electrical connection from the associated data set to the microcomputer for the purpose of transferring a bit-by-bit serialization of the data which has been received from the remote end of the associated communications channel. The received character format is the same as the format transmitted.

8.7.2.4 Pin 4 (CA) Request to Send
This circuit originates in the microcomputer and is utilized to condition the asociated data set into the transmit mode. In half-duplex facilities this interface signal is also utilized by the associated data set to control the direction of transmission and to aid in the performance of the call turnaround function. Some full-duplex facilities such as the Bell System 103- and 212-type data sets do not actually require this circuit for normal operation but it will continue to function as if it were required. Once the microcomputer has asserted the REQUEST TO SEND interface signal its transmit logic must remain in an idle state until the associated data set has responded with the CLEAR TO SEND interface signal described in the next paragraph.

8.7.2.5 Pin 5 (CB) Clear to Send

The CLEAR TO SEND interface signal originates on the associated data set and indicates to the microcomputer that serial data transmission may proceed across circuit BA on pin #2. Some full-duplex facilities such as the Bell System 103-type data sets actually hold this circuit asserted once the communications channel has been established but the microcomputer must ignore this constant status indication if circuit CA on pin #4 is not asserted.

8.7.2.6 Pin 6 (CC) Data Set Ready

This interface lead originates in the associated data set and indicates to the microcomputer that all prerequisite conditions are satisfied and therefore data communications may now proceed. It is to be noted that the DATA SET READY lead is indicative of the status of the local data set only and in no way can be used to infer anything about the status of the remote data set.

8.7.2.7 Pin 7 (AB) Signal Ground

This interface lead provides the common ground reference potential for all interchange circuits except circuit AA on pin #1. In addition, this circuit is electrically in common with the logic signal ground of the microcomputer. A jumper provides electrical commonality with circuit AA to minimize the introduction of noise into the electronic circuitry. The jumper may be removed at installation time if necessary.

8.7.2.8 Pin 8 (CF) Received Line Signal Detector

More commonly known as DATA CARRIER DETECT, this interface lead originates in the associated data set and is utilized to indicate to the microcomputer that a signal suitable for demodulation is being received on the communications channel. Communications interfaces utilize this signal to prepare for data reception and therefore all internal receiver logic must be held in an idle state until circuit CF is asserted.

8.7.2.9 Pins 9 to 14 Not Used

8.7.2.10 Pin 15 (DB) Transmission Signal Element Timing

The DB circuit originates on an associated synchronous data set and is utilized to provide the driving clock for all of the internal transmit logic on the microcomputer. The microcomputer will present serial data to circuit BA on pin #2 synchronously with the negative-to-positive transition of the clocking signal on circuit DB. An associated synchronous data set samples the data bit presented on circuit BA synchronously with the positive-to-negative transition of the clocking signal on circuit DB.

It is worthwhile to note at this point that most synchronous data set provide an external transmitter clock option by which the user can provide its own clock to the modem across circuit DA on pin #24 of the EIA standard RS-232-C. Under thes conditions the modem will synchronize circuit DB on pin #15 with the previousl mentioned external transmitter clock. This method of supplemental clocking is not supported by the microcomputer. Accordingly, the microcomputer is capable of interfacing only to synchronous data sets which have the standard factory-wirec internal transmitter clock circuit installed.

8.7.2.11 Pin 16 Not Used

8.7.2.12 Pin 17 (DD) Receiver Signal Element Timing
The DD circuit originates on an associated synchronous data set and is utilized to provide the driving clock for all of the internal receiver logic on the microcomputer. An associated synchronous data set will present serial data to circuit BB on pin #3 synchronously with the NEGATIVE-TO-POSITIVE transition of the clocking signal on the circuit DD. The microcomputer samples the data bit presented on circuit BB synchronously with the POSITIVE-TO-NEGATIVE transition of the clocking signal on circuit DD.

8.7.2.13 Pin 18 And 19 Not Used

8.7.2.14 Pin 20 (CD) Data Terminal Ready
This circuit originates in the microcomputer and is utilitzed to prepare the associated data set for connection once a call has been established. The actual connection can be initiated by either a manual or automatic answering procedure in addition to either a manual or automatic call origination procedure. Circuit CD is dropped to terminate a completed call and should not be raised again until the associated data set has responded by dropping circuit CC on pin #6.

8.7.2.15 Pin 21 Not Used

8.7.2.16 Pin 22 (CE) Ring Indicator
This interface signal originates on the associated data set and indicates to the microcomputer that an incoming call is pending on the communications channel. Note that the microcomputer incorporates an integrator circuit on the RING INDICATOR signal to protect against the spikes and false-rings normally associated with circuit CE due to the inductive coupling effects inherent in the cables used to connect the microcomputer with external data sets.

8.7.2.17 Pins 23 to 25 Not Used
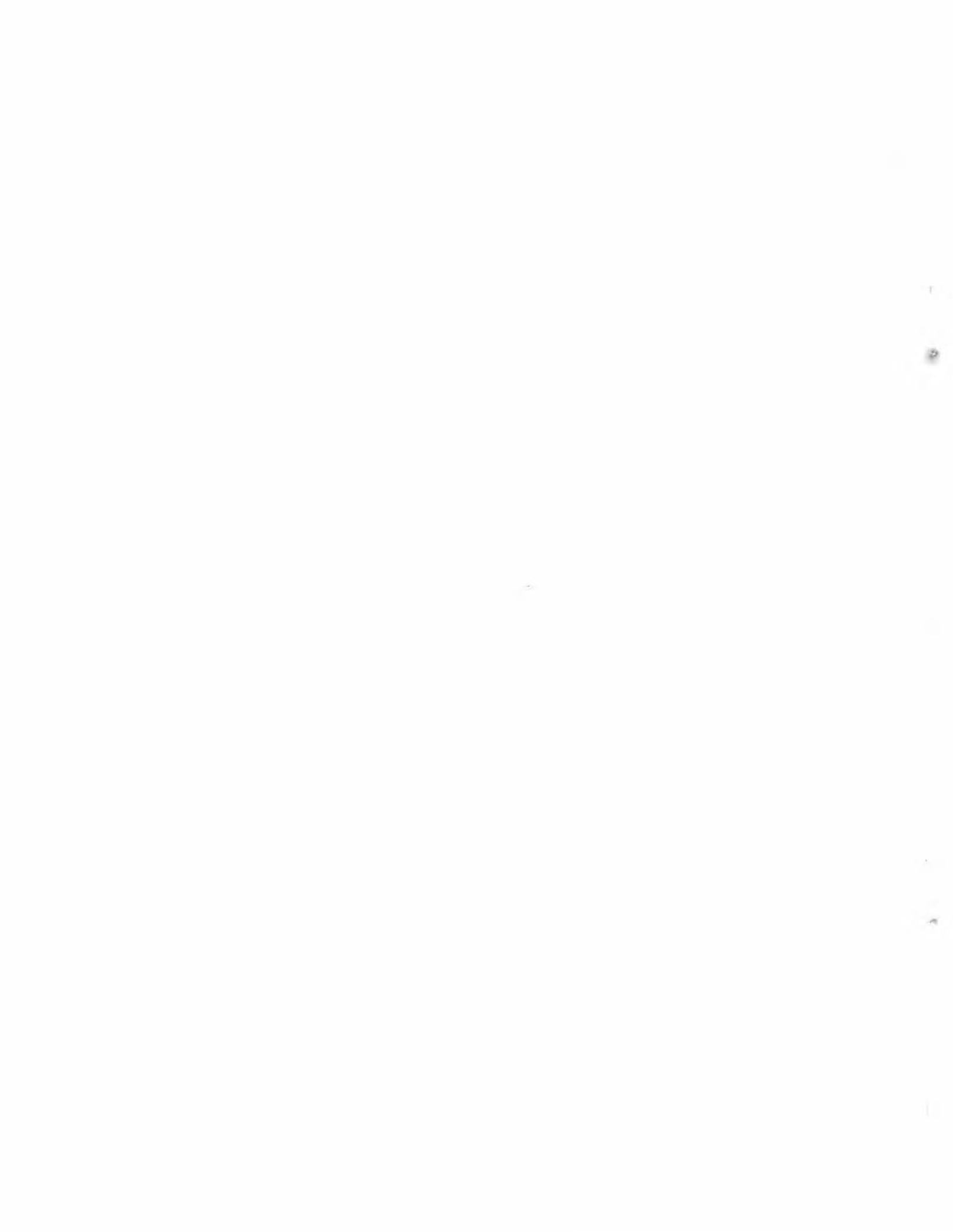
WIRING TELETYPE MODEL 3320/5JE FOR TM 990/101M

A-1 GENERAL
Figure A-1 shows the wiring configuration required to connect a 3320/5JE Teletype in a 20 mA current loop with a TM 990/101M. Other teletypewriter models may require different connections; therefore, consult the manufacturer for correct wiring of other models. Teletypewriters can be used with Assembly No. 999211-0001 only.

CAUTION
Note the 117 Vac connection at pins 1 and 2. Be sure that this voltage is not accidently wired to the TM 990/101M board.

A-2 CONNECTIONS
The following assumes that the teletypewriter is wired as it came from the factory.

(1) Locate the 151411 terminal block at the left rear (viewed from the rear) of the machine (Figure A-1).

(2) Move the white/blue wire from terminal 4 to terminal 5 on the terminal block.

(3) Move the brown/yellow wire from terminal 3 to terminal 5 on the terminal block.

(4) Move the purple wire from terminal 8 to terminal 9 on the terminal block (for 20 mA neutral signaling).

(5) Locate the power resistor behind the teletype power supply. Remove the blue wire from the 750 ohm tap and connect it to the 1450 ohm tap, as shown in Figure A-2.

(6) Check pins 3, 4, 6, and 7 at terminal strip 151411. Voltage to ground must be zero with power applied. If not, do not connect to the TM 990/101M.

NOTE
For teletypewriter operation jumper E36/E37 must be installed and E39/E40 must be in the EIA position.

A-3 TROUBLESHOOTING
If the printer continues to chatter after the RESET switch on the TM 990/101M has been activated, reverse connections 6 and 7 at the terminal strip.
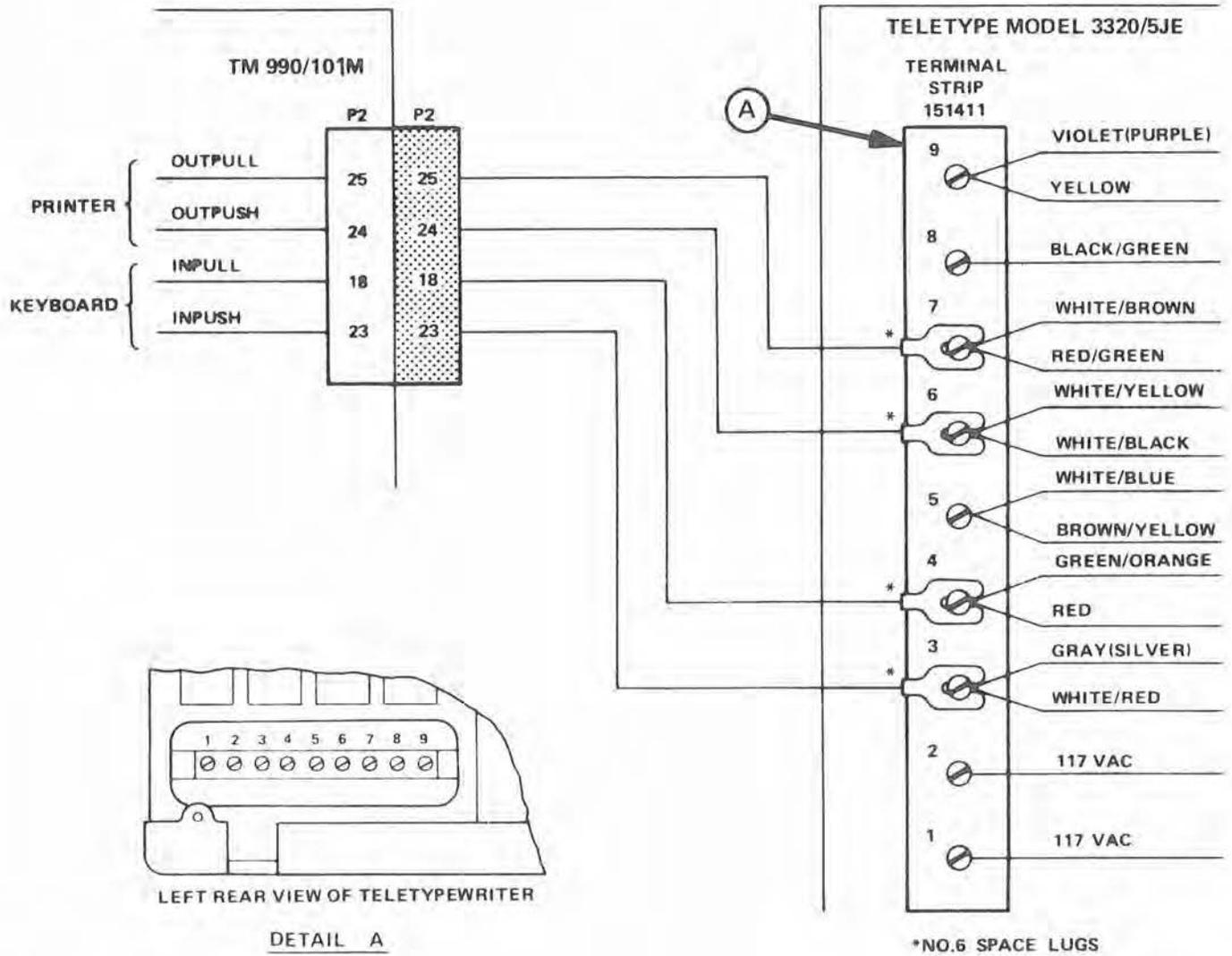
FIGURE A-1. TELETYPEWRITER TERMINAL STRIP CONNECTIONS

A0001412

FRONT

1450 OHM TAP
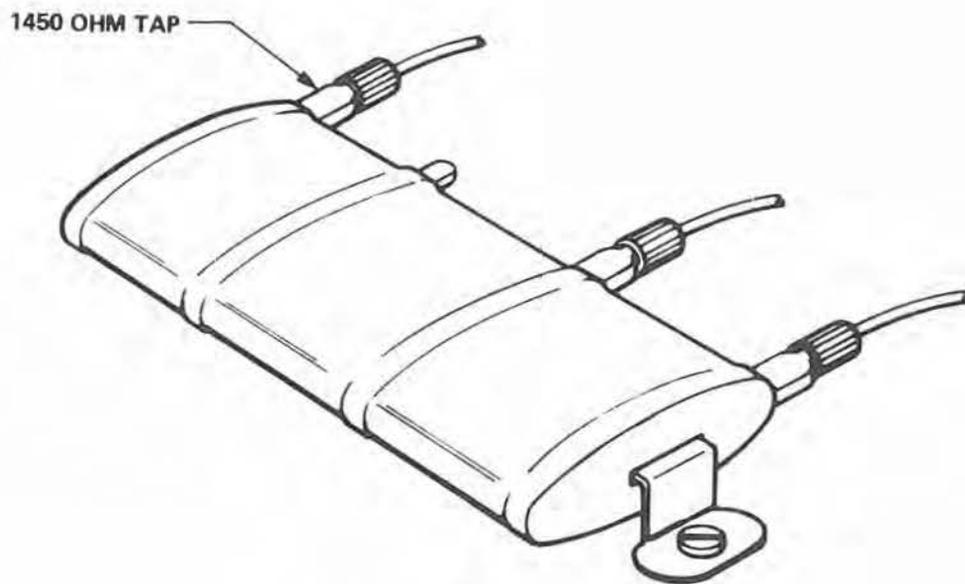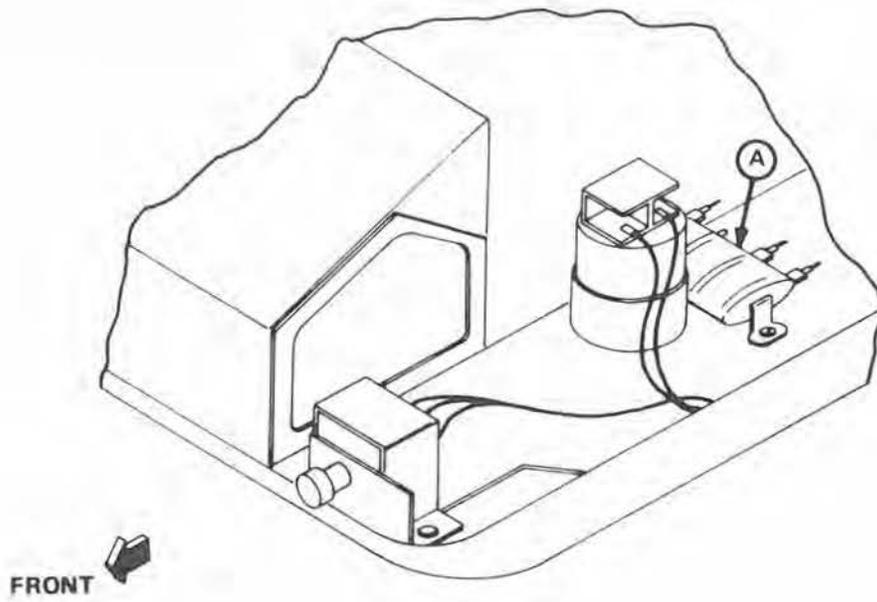
A0001413

DETAIL A

FIGURE A-2. TELETYPEWRITER RESISTOR CONNECTION

APPENDIX B

EIA RS-232-C CABLING

Figure B-1 shows the wiring for the 743 KSR cable attached between connector P2
on the TM 990/101M and a 743 KSR data terminal. Also shown is the relationship
between cable wires and signals to the serial interface, the TMS 9902. Figure B-2
shows the cable configuration for the 733 data terminal.

NOTE
When using an RS-232-C device, disconnect jumper E36/E37
and insert jumper E39/E40 (EIA position). See Figure
7-2.
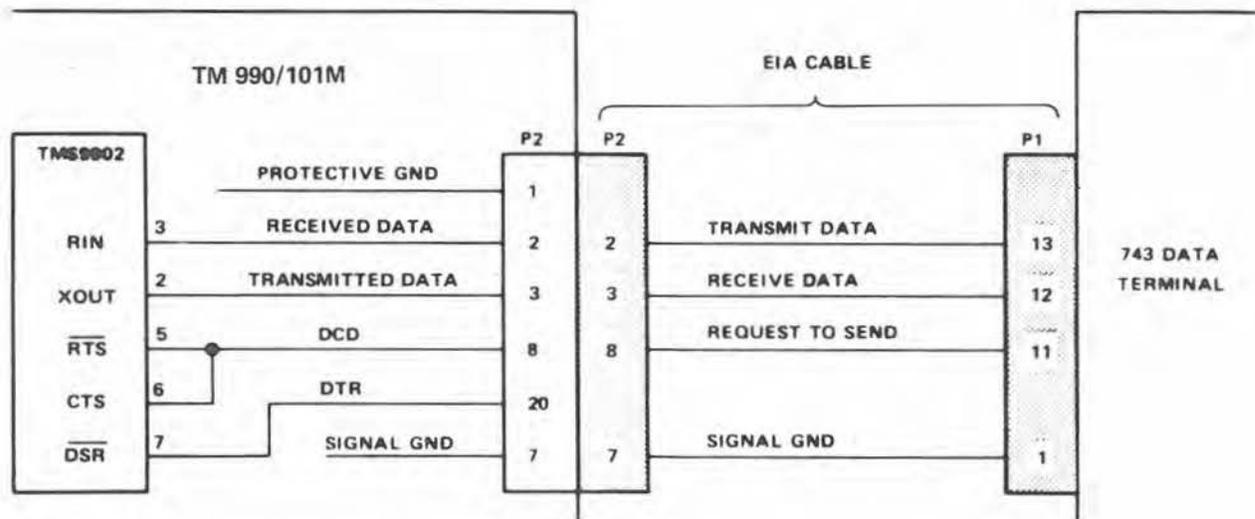
FIGURE B-1. EIA RS-232-C CABLING FOR 743 DATA TERMINAL
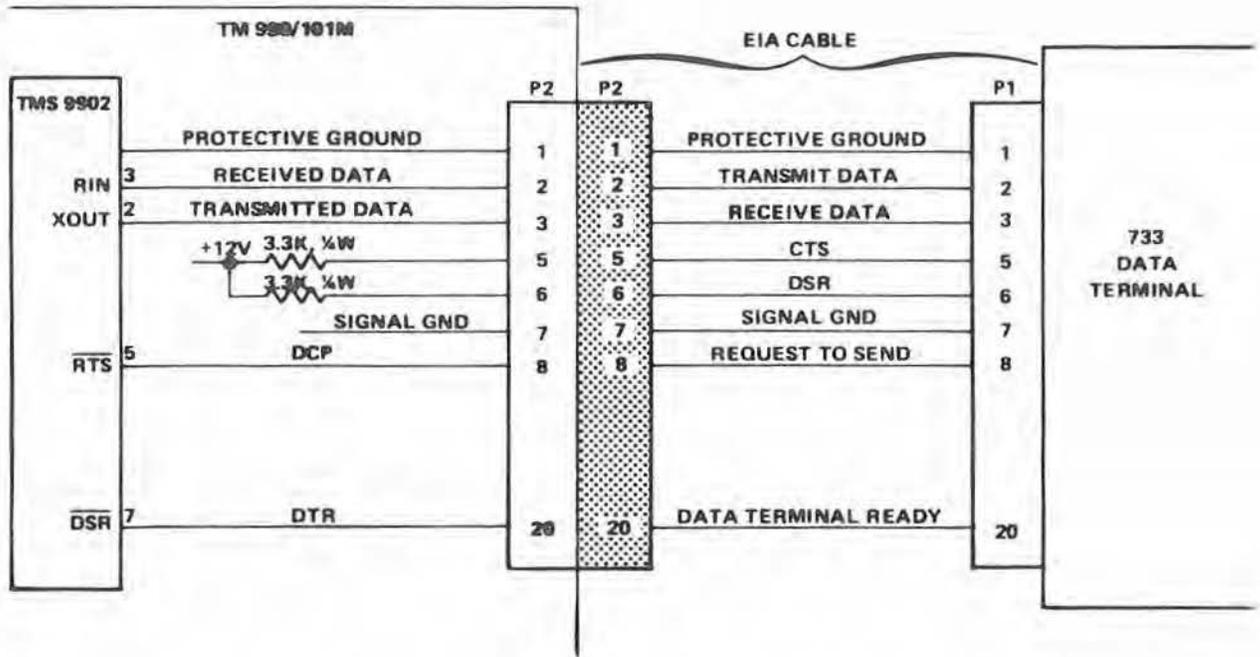
FIGURE B-2. EIA RS-232-C CABLING FOR 733 DATA TERMINAL

ASCII CODE

TABLE C-1. *ASCII CONTROL CODES

CONTROL	BINARY CODE	HEXADECIMAL CODE
NUL – Null	000 0000	00
SOH – Start of heading	000 0001	01
STX – Start of text	000 0010	02
ETX – End of text	000 0011	03
EOT – End of transmission	000 0100	04
ENQ – Enquiry	000 0101	05
ACK – Acknowledge	000 0110	06
BEL – Bell	000 0111	07
BS – Backspace	000 1000	08
HT – Horizontal tabulation	000 1001	09
LF – Line feed	000 1010	0A
VT – Vertical tab	000 1011	0B
FF – Form feed	000 1100	0C
CR – Carriage return	000 1101	0D
SO – Shift out	000 1110	0E
SI – Shift in	000 1111	0F
DLE – Data link escape	001 0000	10
DC1 – Device control 1	001 0001	11
DC2 – Device control 2	001 0010	12
DC3 – Device control 3	001 0011	13
DC4 – Device control 4 (stop)	001 0100	14
NAK – Negative acknowledge	001 0101	15
SYN – Synchronous idle	001 0110	16
ETB – End of transmission block	001 0111	17
CAN – Cancel	001 1000	18
EM – End of medium	001 1001	19
SUB – Substitute	001 1010	1A
ESC – Escape	001 1011	1B
FS – File separator	001 1100	1C
GS – Group separator	001 1101	1D
RS – Record separator	001 1110	1E
US – Unit separator	001 1111	1F
DEL – Delete, rubout	111 1111	7F

*American Standards Institute Publication X3.4-1968

TABLE C-2. *ASCII CHARACTER CODE

CHARACTER	BINARY CODE	HEXADECIMAL CODE	CHARACTER	BINARY CODE	HEXADECIMAL CODE
Space	010 0000	20	P	101 0000	50
!	010 0001	21	Q	101 0001	51
" (dbl. quote)	010 0010	22	R	101 0010	52
#	010 0011	23	S	101 0011	53
$	010 0100	24	T	101 0100	54
%	010 0101	25	U	101 0101	55
&	010 0110	26	V	101 0110	56
' (sgl. quote)	010 0111	27	W	101 0111	57
(	010 1000	28	X	101 1000	58
)	010 1001	29	Y	101 1001	59
* (asterisk)	010 1010	2A	Z	101 1010	5A
+	010 1011	2B	[	101 1011	5B
, (comma)	010 1100	2C	\	101 1100	5C
− (minus)	010 1101	2D	]	101 1101	5D
. (period)	010 1110	2E	Λ	101 1110	5E
/	010 1111	2F	_ (underline)	101 1111	5F
0	011 0000	30		110 0000	60
1	011 0001	31	a	110 0001	61
2	011 0010	32	b	110 0010	62
3	011 0011	33	c	110 0011	63
4	011 0100	34	d	110 0100	64
5	011 0101	35	e	110 0101	65
6	011 0110	36	f	110 0110	66
7	011 0111	37	g	110 0111	67
8	011 1000	38	h	110 1000	68
9	011 1001	39	i	110 1001	69
:	011 1010	3A	j	110 1010	6A
;	011 1011	3B	k	110 1011	6B
<	011 1100	3C	l	110 1100	6C
=	011 1101	3D	m	110 1101	6D
>	011 1110	3E	n	110 1110	6E
?	011 1111	3F	o	110 1111	6F
@	100 0000	40	p	111 0000	70
A	100 0001	41	q	111 0001	71
B	100 0010	42	r	111 0010	72
C	100 0011	43	s	111 0011	73
D	100 0100	44	t	111 0100	74
E	100 0101	45	u	111 0101	75
F	100 0110	46	v	111 0110	76
G	100 0111	47	w	111 0111	77
H	100 1000	48	x	111 1000	78
I	100 1001	49	y	111 1001	79
J	100 1010	4A	z	111 1010	7A
K	100 1011	4B	{	111 1011	7B
L	100 1100	4C			111 1100	7C
M	100 1101	4D	}	111 1101	7D
N	100 1110	4E	~	111 1110	7E
O	100 1111	4F			

APPENDIX D

BINARY, DECIMAL AND HEXADECIMAL NUMBERING

D-1 GENERAL
This appendix covers numbering systems to three bases (2, 10, and 16) which are used throughout this manual.

D-2 POSITIVE NUMBERS

D-2.1 DECIMAL (BASE 10). When a numerical quantity is viewed from right to left, the rightmost digit represents the base number to the exponent 0. The next digit represents the base number to the exponent 1, the next to the exponent 2, then exponent 3, etc. For example, using the base 10 (decimal):

$$10^6 \quad 10^5 \quad 10^4 \quad 10^3 \quad 10^2 \quad 10^1 \quad 10^0$$
$$X, \quad X \quad X \; X, \quad X \quad X \quad X$$

or

```
1,000,000
    100,000
      10,000
 |  | | 1000   100  10   1
 ▼  ▼ ▼
 X , X X X  ,   X    X    X
```

For example, 75,264 can be broken down as follows:

```
75, 264
 | || |
 | || └──── 4 x 10⁰ = 4 x 1              4
 | |└────── 6 x 10¹ = 6 x 10      =     60
 | └─────── 2 x 10² = 2 x 100     =    200
 └───────── 5 x 10³ = 5 x 1000    =   5000
 └───────── 7 x 10⁴ = 7 x 10,000  = +70000
                                      ──────
                                      75264₁₀
```

For example, 75,264 can be broken down as follows:

$4 \times 10^0 = 4 \times 1$ = 4

$6 \times 10^1 = 6 \times 10$ = 60

$2 \times 10^2 = 2 \times 100$ = 200

$5 \times 10^3 = 5 \times 1000$ = 5000

$7 \times 10^4 = 7 \times 10{,}000$ = +70000

75264_{10}

D-2.2 BINARY (BASE 2). As base 10 numbers use ten digits, base 2 numbers use only 0 and 1. When viewed from right to left, they each represent the number 2 to the powers 0, 1, 2, etc., respectively as shown below:

$$2^{15} \qquad 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$$

$$(32,768) \circ \circ \circ (64) \ (32) \ (16) \ (8) \ (4) \ (2) \ (1)$$

$$X \qquad \circ \circ \circ \ X \quad\ X \quad\ X \quad\ X \quad\ X \quad\ X \quad\ X$$

For example, 11011_2 can be translated into base 10 as follows:
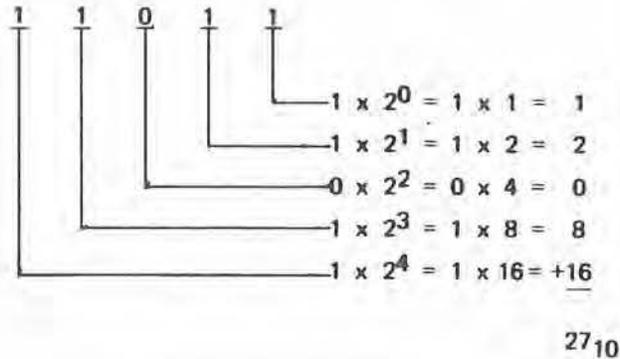
$$27_{10}$$

or 11011_2 equals 27_{10}.

Binary is the language of the digital computer. For example, to place the decimal quantity 23 (23_{10}) into a 16-bit memory cell, set the bits to the following:
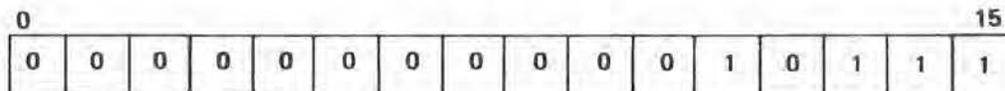
which is $1 + 2 + 4 + 16 = 23_{10}$.

D-2.3 HEXADECIMAL (BASE 16). Whereas binary uses two digits and decimal uses ten digits, hexadecimal uses 16 (0 to 9, A, B, C, D, E, and F).

The letters A through F are used to represent the decimal numbers 10 through 15 as shown on the following page.

N_{10}	N_{16}
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7

N_{10}	N_{16}
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

When viewed from right to left, each digit in a hexadecimal number is a multiplier of 16 to the powers 0, 1, 2, 3, etc., as shown below:

$$16^3 \quad 16^2 \quad 16^1 \quad 16^0$$

$$(4096) \quad (256) \quad (16) \quad (1)$$

$$X \quad\quad X \quad\quad X \quad\quad X$$

For example, $7\,B\,A\,5_{16}$ can be translated into base 10 as follows:

$$
\begin{array}{llll}
7 & B & A & 5 \\
\end{array}
$$

$$
\begin{aligned}
5 \times 16^0 &= 5 \times 1 & &= & 5 \\
10 \times 16^1 &= 10 \times 16 & &= & 160 \\
11 \times 16^2 &= 11 \times 256 & &= & 2\,816 \\
7 \times 16^3 &= 7 \times 4096 & &= & 28\,672 \\
& & & & \overline{31\,653_{10}}
\end{aligned}
$$

or $7\,B\,A\,5_{16}$ equals $31,653_{10}$.

Because it would be awkward to write out 16-digit binary numbers to show the contents of a 16-bit memory word, hexadecimal is used instead. Thus

$$003E_{16} \text{ or } > 003E \; (> \text{ indicates hexadecimal})$$

is used instead of

$$0000\ 0000\ 0011\ 1110_2$$

to represent 62_{10} as computed below:

BASE 2

```
1   1   1   1   1   0₂
                  └──0 X 2⁰  =   0
                └────1 X 2¹  =   2
              └──────1 X 2²  =   4
            └────────1 X 2³  =   8
          └──────────1 X 2⁴  =  16
        └────────────1 X 2⁵  =  32
```

$$0 \times 2^0 = 0$$
$$1 \times 2^1 = 2$$
$$1 \times 2^2 = 4$$
$$1 \times 2^3 = 8$$
$$1 \times 2^4 = 16$$
$$1 \times 2^5 = \underline{32}$$
$$62_{10}$$

BASE 10

```
6   2₁₀
    └──2 X 10⁰  =   2
  └────6 X 10¹  =  60
```

$$2 \times 10^0 = 2$$
$$6 \times 10^1 = \underline{60}$$
$$62_{10}$$

BASE 16

```
3   E₁₆
    └──14 X 16⁰  =  14
  └────3 X 16¹  =  48
```

$$14 \times 16^0 = 14$$
$$3 \times 16^1 = \underline{48}$$
$$62_{10}$$

Note that separating the 16 binary bits into four-bit parts facilitates recognition and translation into hexadecimal.

```
0000   0000   0011   1110₂              C     7     B     F₁₆
  ↓      ↓      ↓      ↓        or       ↓     ↓     ↓     ↓
  0      0      3      E₁₆            1100  0111  1011  1111₂
```

Table D-1 is a conversion chart for converting decimal to hexadecimal and vice versa. Table D-2 shows binary, decimal and hexadecimal equivalents for numbers 0 to 15. Note that Table D-1 is divided into four parts, each part representing four of the 16-bits of a memory cell or word (bits 0 to 15 with bit 0 being the most significant bit (MSB) and bit 15 being the least significant bit (LSB). Note that the MSB is on the left and represents the highest power of 2 and the LSB on the right represents the 0 power of 2 ($2^0 = 1$). As explained later, the MSB can also be used to signify number polarity (+ or −).

NOTE

To convert a binary number to decimal or hexadecimal, convert the *positive* binary value as described in Section D-4.

	MSB							LSB
	16^3		16^2		16^1		16^0	
BITS	0 1 2 3		4 5 6 7		8 7 8 11		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 hexadecimal to decimal, find the nearest decimal number in the above table less than or equal to the number being converted. Set down the hexadecimal equivalent then subtract this number from the nearest decimal number. Using the remainder(s), repeat this process. For example:

$$31{,}362_{10} = 7000_{16} + 2690_{10} \qquad 7000$$
$$2{,}690_{10} = A00_{16} + 130_{10} \qquad A00$$
$$130_{10} = 80_{16} + 2_{10} \qquad 80$$
$$2_{10} = 2_{16} \qquad \underline{2}$$
$$7A82_{16}$$

TABLE D-2. BINARY, DECIMAL, AND HEXADECIMAL EQUIVALENTS

BINARY (N_2)	DECIMAL (N_{10})	HEXADECIMAL (N_{16})
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
10000	16	10
10001	17	11
10010	18	12
10011	19	13
10100	20	14
10101	21	15
10110	22	16
10111	23	17
11000	24	18
11001	25	19
11010	26	1A
11011	27	1B
11100	28	1C
11101	29	1D
11110	30	1E
11111	31	1F
100000	32	20

D-3 ADDING AND SUBTRACTING BINARY

Adding and subtracting in binary uses the same conventions for decimal: carrying over in addition and borrowing in subtraction.

Basically,

```
  0          1                                              10
+ 1        + 1                                            -  1
----       ----                                           ----
  1         10    (the carry, 1, is carried to the left)    01    (1 is borrowed from
                                                                    top left)
```

```
   1 ⎫                                                         11
   1 ⎬  = 0 + carry 1                                           1
     ⎭                                                       +  1
 + 1      = 0 (from above) + 1 = 1                           ----
 ----                                                         101
  11
     ⎳carry                              carry 1 + 1 = 10 ─────┛
```

```
   1 ⎫                                                      1
   1 ⎬  = 0 + 1 carry              1000 ⎫              ⎧  0110
     ⎭                            -   1 ⎬ Borrow the 1 ⎨  -  1
   1 ⎫                             ----- ⎭              ⎩  -----
   1 ⎬  = 0 + 1 carry              0111                    0111
 + 1 ⎭
 -----
  100
     ⎳0 + 0 = 0
     ⎳carry 1 + carry 1
```

D-4 POSITIVE/NEGATIVE CONVERSION (BINARY). To compute the negative equivalent of a positive binary or hexadecimal number, or interpret a binary or hexadecimal negative number (determine its positive equivalent) use the two's complement of the binary number.

NOTE

To convert a binary number to decimal, convert the *positive* binary value (*not* the negative binary value) and add the sign.

Two's complementing a binary number includes two simple steps:

a. Obtain one's complement of the number (1's become 0's, 0's becomes 1's) (invert bits).

b. Add 1 to the one's complement.

For example, with the MSB (left-most bit) being a sign bit:

010 $(+2_2)$	111 (-1_2)	110 (-2_2)	101 (-3_2)
101 Invert	000 Invert	001 Invert	010 Invert
+ 1 Add 1	+ 1 Add 1	+ 1 Add 1	+ 1
110 (-2_2)	001 $(+1_2)$	010 $(+2_2)$	011 $(+3_2)$

This can be expanded to 16-bit positive numbers:

(=39F6$_{16}$)	0011	1001	1111	0110	($39F6_{16}$ = +14,838$_{10}$)
	1100	0110	0000	1001	Invert
				+1	Add 1
(=C60A$_{16}$)	1100	0110	0000	1010	($C60A_{16}$ = −14,838$_{10}$) Two's Complement

SIGN BIT(−)

And to 16-bit negative numbers:

(=C60A$_{16}$)	1100	0110	0000	1010	($C60A_{16}$ = −14,838$_{10}$)
	0011	1001	1111	0101	Invert
				+1	Add 1
(=39F6$_{16}$)	0011	1001	1111	0110	($39F6_{16}$ = +14,838$_{10}$) Two's Complement

SIGN BIT(+)

PARTS LIST

Table E-1.   Parts For all Dash Numbers

Symbol	Description	-0001	-0002	-0003
C1-C8, C11, C13-C17, C19-C22, C24 C26-C39, C41-C44	Capacitor, 0.047 μF	x	x	x
C9, C12, C25, C40	Capacitor, 22 mFd	x	x	x
C10	Capacitor, 18 pFd	x	x	x
C45-C48	Capacitor 0.047 mFd, 10%	x	x	x
CR1, CR2	Diode, IN5333B		x	
CR3	Diode, IN914B	x		x
DS1	Diode (L.E.D., CM 4-43)	x	x	x
E1-E40, E53-E56	Pin, Jumper (BEI 75481-002)	x		x
E1-E35, E38-E56	Pin, Jumper (BEI 75481-002)		x	
All Jumpers	Plug, Jumper (BEI 65474-004, R 530153-002)	x	x	x
L1	Coil, RF, 033 μH	x	x	x
P2, P3	Connector, 25 pin (AMP 206584-2)	x	x	x
Q1	Transistor, PNP	x		x
R1, R2, R4, R5, R7, R8, R11, R23, R26, R44, R45	Resistor, 4.7K ohm	x	x	x
R3, R12	Resistor, 2.2K ohm	x	x	x
R6	Resistor, 1.0K ohm	x	x	x
R9, R10, R14, R15	Resistor, 10.0 ohm	x	x	x
R13, R16, R17	Resistor, 2.2 ohm	x	x	x
R18, R24, R25	Resistor, 68.0 ohm	x	x	x

Symbol	Description	-0001	-0002	-0003
R19, R21, R39	Resistor, 330 ohm, 1/4 W	x	x	x
R34, R40, R41, R43	Resistor, 330 ohm, 1/4 W		x	
R20, R22	Resistor, 220 ohm	x	x	x
R27	Resistor, 3.9K ohm	x	x	x
R28	Resistor, 2.7K ohm	x		x
R29	Resistor, 330 ohm, 1/2 W	x		x
R30	Resistor 33K ohm	x		x
R31-R33, R42	Resistor, 27K ohm		x	
R35, R36, R46, R47	Resistor, 3.3K ohm	x	x	x
R37	Resistor, 3.3K ohm	x		x
R38	Resistor, 560 ohm	x		x
S1	Switch, toggle	x	x	x
S2	Switch, 5 position DIP	x	x	x
U1	IC, TMS 9901	x	x	x
U2, U$\overline{8}$	Resistor, 10.0K ohms pkg.	x	x	x
U3, U26, U32	IC, SN74LS241N, Line Drivers	x	x	x
U4, U18	Network, SN74LS08N	x	x	x
U5, U6, U10, U17, U20	Network, SN74LS74N	x	x	x
U7, U27	Network, SN74LS04N	x	x	x
U9, U39	Network, SN74LS251	x	x	x
U11	Network, SN74LS132N	x	x	x
U12	Network, SN74LS14N	x	x	x
U13, U14, U22, U23	IC, SN74LS245N, Octal Buffer	x	x	x

Table E-1. Parts For All Dash Numbers (Continued)

Symbol	Description	-0001	-0002	-0003
U15	TMS 9900	x	x	x
U16	TIM 9904, clock driver	x	x	x
U19	PROM, 74S287, memory decode	x	x	x
U21	Network, SN74LS02N	x	x	x
U24	Network, SN74LS153N	x	x	x
U25, U52	Network, SN74LS138N	x	x	x
U28, U30, U34, U36	TMS 4045 1024 x 4 RAM			x
U29, U31, U35, U37	TMS 4045 1024 x 4 RAM	x	x	
U33, U49	IC, SN75188N, Line Drivers	x	x	x
U38	Network, SN74LS10N	x	x	x
U40, U41	Network, SN75189AN	x	x	x
U42	TMS 2708, EPROM, TIBUG byte 1	x		
U44	TMS 2708, EPROM, TIBUG byte 0	x		
U42, U44	TMS 2716, 2048 x 8 EPROM		x	
U42-U45	TMS 2716, 2048 x 8 EPROM			x
U46, U47	TMS 9902 Asynchronous Communication Controller	x	x	x
U48	IC, SN75112N		x	
U50	Network, SN74LS00N	x	x	x
U51	IC, SN74LS259N, low power Schottky	x	x	x
U53	Network, SN75154N	x	x	x
U54	Network, SN751074N, Interface		x	

Table E-1. Parts For All Dash Numbers (Concluded)

Symbol	Description	-0001	-0002	-0003
VR1	IC, UA 7905C/MC 7905CP, Voltage Regulator	x	x	x
XU1	Socket, 40 pin	x	x	x
XU15	Socket, 64 pin	x	x	x
XU16, XU47	Socket, 20 pin	x	x	x
XU19	Socket, 16 pin	x	x	x
XU28-XU31, XU34-XU37 XU46	Socket, 18 pin	x	x	x
XU42-XU45	Socket, 24 pin	x	x	x
X1	Crystal, 48 MHz, 3rd overtone,5%, HC-18U	x	x	x

APPENDIX F
SCHEMATICS

NOTES UNLESS OTHERWISE SPECIFIED

1. CAPACITANCE VALUES ARE IN MICROFARADS

2. RESISTANCE VALUES ARE IN OHMS

3. ALL RESISTORS ARE ¼ W, 5%

[4] C18 AND C23 ARE USER'S OPTION

[5] PIN NUMBER ASSIGNMENTS FOR U47 APPLY TO THE 20 PIN SOCKET TO ENABLE USER TO INTERCHANGE THE TMS 9902 (18 PINS) WITH A TMS9903 (20 PIN) (USER'S OPTION)

[6] JUMPER PLUGS ARE INSTALLED ON E20-E21, E22-E23, E24-E25 AS SHOWN ON -0001 ONLY

[7] THESE COMPONENTS ARE INSTALLED ON -0001 ONLY

[8] THESE COMPONENTS ARE INSTALLED ON -0002 ONLY

9. NC DENOTES NO CONNECTION

Device	Supply Voltages to Pin Number				
	-12 V	-5 V	GND	+5 V	+12 V
TMS 9900		1	26	7	27
TMS 9901			16	40	
TMS 9902			4	18	
TMS 9902/OI socket			9	20	
TMS 9904			5,10	20	13
TMS 4045			9	18	
TMS 2708/2716		21	12	24	19
74LS241, 74LS245			10	20	
75188	1		7		14
75189			7	14	
75154			8	15	
75107		11	7	14	
75112		11	7	14	
74LS138, 153, 251, 259, 74C902			8	16	
74LS00			7	14	
74LS02			7	14	
74LS04			7	14	
74LS08			7	14	
74LS10			7	14	
74LS13			7	14	
74LS74			7	14	
74LS132			7	14	

+12V — P1-75, 76

C48 .047   C11, C35, C38 .047

+5V — P1-3, 4, 97, 98

C45-C47 .047   C25 22   C1-C8 C13 C14 C16 C17 C19-C22, C24, C26, C34, C41-C44 .047   VR1 UA7905   C40 22   C9 22   C15, C37 .047   -5V

C12 22   C36, C39 .047

GND — P1-1, 2, 21, 23, 25, 27, 77, 79, 81, 83, 85, 89, 91, 99, 100
P4-5, 7, 9, 11, 13, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39

-12V — P1-73, 74

SPARES

U2 2% +5V

U17 74LS04   U27 74LS04   U40 7589A   U41 7589A

U12 74LS14   U33 7588   U40 7589A   U49 7588

F-3

F-5

F-6

990 OBJECT CODE FORMAT

G.1 GENERAL

In order to correctly load a program into memory using a loader, the program in hexadecimal machine code must be in a particular format called object format. Such a format is required by the *TIBUG* loader (paragraph 3.2.7 explains loader execution). This object format has a tag character for each 16-bit word of coding which flags the loader to perform one of several operations. These operations include:

- Load the code at a user-specified absolute address and resolve relative addresses. (Most assemblers assemble a program as if it was loaded at memory address 0000_{16}; thus, relative addresses have to be resolved.)

- Load entire program at a specific address.

- Set the program counter to the entry address after loading.

- Check for checksum errors that would indicate a data error in an object record.

G.2 STANDARD 990 OBJECT CODE

Standard 990 object code consists of a string of hexadecimal digits, each representing four bits, as shown in Figure G-1.


```
                  TAG CHARACTER

                      LENGTH OF RELOCATABLE CODE          TAG CHARACTERS


0004CBLINK    A0000BFF70C0004B04CCBC060B000EBCC60C0004BB0202C002C7F1FCF    0001
A0012BCC42BCC60C004AB0200B3D09B0420B0000B0300B0003B0201B00037F30DF         0002
A00288B1D00B10FFBFF70C0030B020CB0100B1D03B04CCB0601B1604B1E007F2AFF        0003
A003EB0201B0003B0380B1D00B0380B0420B03807F760F                             0004
3001ETIME0150000BLINK 7FAABF                                               0005
:     BLINK                          TXMIRA                                 0006

        END OF OBJECT          CHECKSUM FIELD
```

FIGURE G-1. OBJECT CODE EXAMPLE

The object record consists of a number of tag characters, each followed by one or two fields as defined in Table G-1. The first character of a record is the first tag character, which tells the loader which field or pair of fields follows the tag. The next tag character follows the end of the field or pair of fields associated with the preceding tag character. When the assembler has no more data for the record, the assembler writes the tag character 7 followed by the checksum field, and the tag character F, which requires no fields. The assembler then fills the rest of the record with blanks, and begins a new record with the appropriate tag character.

Tag character 0 is followed by two fields. The first field contains the number of bytes of relocatable code, and the second field contains the program identifier assigned to the program by an IDT assembler directive. When no IDT directive is entered, the field contains blanks. The loader uses the program identifier to identify the program, and the number of bytes of relocatable code to determine the load bias for the next module or program. The PX9ASM assembler is unable to determine the value for the first field until the entire module has been assembled, so PX9ASM places a tag character 0 followed by a zero field and the program identifier at the beginning of the object code file. At the end of the file, PX9ASM places another tag character zero followed by the number of bytes of relocatable code and eight blanks.

Tag characters 1 and 2 are used with entry addresses. Tag character 1 is used when the entry address is absolute. Tag character 2 is used when the entry address is relocatable. The hexadecimal field contains the entry address. One of these tags may appear at the end of the object code file. The associated field is used by the loader to determine the entry point at which execution starts when the loading is complete.

Tag characters 3 and 4 are used for external references. Tag character 3 is used when the last appearance of the symbol in the second field is in relocatable code. Tag character 4 is used when the last appearance of the symbol is absolute code. The hexadecimal field contains the location of the last appearance. The symbol in the second field is the external reference. Both fields are used by the linking loader to provide the desired linking to the external reference.

For each external reference in a program, there is a tag character in the object code, with a location, or an absolute zero, and the symbol that is referenced. When the object code field contains absolute zero, no location in the program requires the address that corresponds to the reference (an IDT character string, for example). Otherwise, the address corresponding to the reference will be placed in the location specified in the object code by the linking loader. The location specified in the object code similarly contains absolute zero or another location. When it contains absolute zero, no further linking is required. When it contains a location, the address corresponding to the reference will be placed in that address by the linking loader. The location of each appearance of a reference in a program contains either an absolute zero or another location into which the linking loader will place the referenced address.

TABLE G-1. OBJECT OUTPUT TAGS SUPPLIED BY ASSEMBLERS

TAG CHARACTER	HEXADECIMAL FIELD (FOUR CHARACTERS)	SECOND FIELD	MEANING
0	Length of all relocatable code	8-character program identifier	Program start
1	Entry address	None	Absolute entry address
2	Entry address	None	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	None	Checksum
8	Ignore checksum	None	Do not checksum for error
9	Load address	None	Absolute load address
A	Load address	None	Relocatable load address
B	Data	None	Absolute data
C	Data	None	Relocatable data
D	Load bias value*	None	Load point specifier
F	None	None	End-of-record
G	Location	6-character symbol	Relocatable symbol definition
H	Location	6-character symbol	Absolute symbol definition

*Not supplied by assembler

Tag characters 5 and 6 are used for external definitions. Tag character 5 is used when the location is relocatable. Tag character 6 is used when the location is absolute. Both fields are used by the linking loader to provide the desired linking to the external definition. The second field contains the symbol of the external definition.

Tag character 7 precedes the checksum, which is an error detection word. The checksum is formed as the record is being written. It is the 2's complement of the sum of the 8-bit ASCII values of the characters of the record from the first tag of the record through the checksum tag 7. If the tag character 7 is replaced by an 8, the checksum will be ignored. The 8 tag can be used when object code is changed in editing and it is desired to ignore checksum.

Tag characters 9 and A are used with load addresses for data that follows. Tag character 9 is used when the load address is absolute. Tag character A is used when the load address is relocatable. The hexadecimal field contains the address at which the following data word is to be loaded. A load address is required for a data word that is to be placed in memory at some address other than the next address. The load address is used by the loader.

Tag characters B and C are used with data words. Tag character B is used when the data is absolute; an instruction word or a word that contains text characters or absolute constants, for example. Tag character C is used for a word that contains a relocatable address. The hexadecimal field contains the data word. The loader places the word in the memory location specified in the preceding load address field, or in the memory location that follows the preceding data word.

To have object code loaded at a specific memory address, precede the object program with the D tag followed by the desired memory address (e.g., DFD00).

Tag character F indicates the end of record. It may be followed by blanks.

Tag characters G and H are used when the symbol table option is specified with other 990 assemblers. Tag character G is used when the location or value of the symbol is relocatable, and tag character H is used when the location or value of the symbol is absolute. The first field contains the location or value of the symbol, and the second field contains the symbol to which the location is assigned.

The last record of an object code file has a colon (:) in the first character position of the record, followed by blanks. This record is referred to as an end-of-module separator record.

Figure G-2 is an example of an assembler source listing and corresponding object code. A comparison of the object tag characters and fields with the machine code in the source listing will show how object code is constructed for use by the loader.

SOURCE STATEMENT NO.

LOCATION COUNTER (ADDRESS RELATIVE TO FIRST OBJECT BYTE)

MACHINE CODE

SAMPLE   SDSMAC 945278 **

PAGE 0001

```
0001                        IDT  'SAMPLE'
  02 0000 0006'             DATA WSPACE
  03 0002 008A'             DATA START
0004 0004 0000             DATA 0
0005 0006         WSPACE BSS  32
0006 0026         TABLE  BSS  100
0007 008A         START
0008 008A 04CC             CLR  12
0009 008C 04C0             CLR  0
0010 008E 0202             LI   2, TABLE
     0090 0026'
0011 0092 C800             MOV  0, @TABLE+2
     0094 0028'
0012 0096 1001             JMP  $+4
0013 0098         LOOP
0014 0098 0204             LI   4, >1234
     009A 1234
0015 009C 0244             ANDI 4, >FEED
     009E FEED
0016 00A0 DC84             MOVB 4, *2+
0017 00A2 0205             LI   5, >5555
     00A4 5555
0018 00A6 C805             MOV  5, @TABLE
     00A8 0026'
0019                        END
NO ERRORS
```

```
000AASAMPLE   A0000C0006C008AB0000A008AB04CCB04C0B     0026BC8007F200F        000
C0028B1001B0204B1234B0244BFEEDBDC84B0205B5555BC805C0026 7F3C1F                000
:     SAMPLE    00/00/00  08:14:23            SDSMAC  945278  **
```

FIGURE G-2. SOURCE CODE AND CORRESPONDING OBJECT CODE

G-5

APPENDIX H

P1, P2, AND P4 PIN ASSIGNMENTS

TABLE H-1. CHASSIS INTERFACE CONNECTOR (P1) SIGNAL ASSIGNMENTS

P1 PIN	SIGNAL	P1 PIN	SIGNAL	P1 PIN	SIGNAL
33	D0.B	71	A14.B	12	$\overline{\text{INT13}}$.B
34	D1.B	72	A15.B	11	$\overline{\text{INT14}}$.B
35	D2.B	22	$\overline{\text{Ø1}}$.B	14	$\overline{\text{INT15}}$.B
36	D3.B	24	$\overline{\text{Ø3}}$.B	28	EXTCLK.B
37	D4.B	92	$\overline{\text{HOLD}}$.B	3	+5V
38	D5.B	86	HOLDA.B	4	+5V
39	D6.B	82	DBIN.B	97	+5V
40	D7.B	26	$\overline{\text{CLK}}$.B	98	+5V
41	D8.B	80	$\overline{\text{MEMEN}}$.B	75	+12V
42	D9.B	84	$\overline{\text{MEMCYC}}$.B	76	+12V
43	D10.B	78	$\overline{\text{WE}}$.B	73	−12V
44	D11.B	90	READY.B	74	−12V
45	D12.B	87	$\overline{\text{CRUCLK}}$.B	1	GND
46	D13.B	30	CRUOUT.B	2	GND
47	D14.B	29	CRUIN.B	21	GND
48	D15.B	19	IAQ.B	23	GND
57	A0.B	94	$\overline{\text{PRES}}$.B	25	GND
58	A1.B	88	$\overline{\text{IORST}}$.B	27	GND
59	A2.B	16	$\overline{\text{INT1}}$.B	31	GND
60	A3.B	13	$\overline{\text{INT2}}$.B	77	GND
61	A4.B	15	$\overline{\text{INT3}}$.B	79	GND
62	A5.B	18	$\overline{\text{INT4}}$.B	81	GND
63	A6.B	17	$\overline{\text{INT5}}$.B	83	GND
64	A7.B	20	$\overline{\text{INT6}}$.B	85	GND
65	A8.B	6	$\overline{\text{INT7}}$.B	89	GND
66	A9.B	5	$\overline{\text{INT8}}$.B	91	GND
67	A10.B	8	$\overline{\text{INT9}}$.B	99	GND
68	A11.B	7	$\overline{\text{INT10}}$.B	100	GND
69	A12.B	10	$\overline{\text{INT11}}$.B	93	$\overline{\text{RESTART}}$.B
70	A13.B	9	$\overline{\text{INT12}}$.B		

TABLE H-2. SERIAL I/O INTERFACE (P2) PIN ASSIGNMENTS

P2 PIN	SIGNAL	DESCRIPTION
1	GND	
7	GND	
3	RS232 XMT	RS232 Serial Data Out
2	RS232 RCV	RS232 Serial Data In
5	CTS	Clear to Send (3.3KΩ pull-up to +12 V)
6	DSR	Data Set Ready (3.3KΩ pull-up to +12 V)
8	DCD	Carrier Detect
20	DTR	Data Terminal Ready
18,23	TTY XMT	TTY Receive Loop/Private Wire Receive Pair
24,25	TTY RCV	TTY Transmit Loop/Private Wire Transmit Pair
17	RCV CLK	Receive Clock
15	XMT CLK	Transmit Clock
12*	+12 V	Jumper Option for Microterminal
13*	−12 V	Jumper Option for Microterminal
14*	+5 V	Jumper Option for Microterminal
16	RESTART	Invokes the Load Interrupt to the TMS 9900 CPU

*When using the Microterminal, these voltages are jumpered to the corresponding pin in connector P2. Else, the voltages are not connected

TABLE H.3 SERIAL I/O INTERFACE (P3) PIN ASSIGNMENTS

P3 PIN	SIGNAL	DESCRIPTION
1	OPTIONAL GND	GROUND IF JUMPER AT E18, E19
7	GND	GROUND
2	RS232 RCV	RS232 Serial Data In
3	RS232 XMT	RS232 Serial Data Out
5	CTS-Terminal	Terminal Clear to Send (3.3 kΩ pull-up to +12 V)
6	DSR-Terminal	Terminal Data Set Ready (3.3 kΩ pull-up to +12 V)
8	DCD-Terminal	Terminal Data Carrier Detect (activated by TMS 9902 Request to Send)
16	CTS-Modem	Modem Clear to Send*
19	DSR-Modem	Modem Data Set Ready*
20	DTR-Terminal	Terminal Data Terminal Ready
	DCD-Modem	Modem Data Carrier Detect*
21	DTR-Modem	Modem Data Terminal Ready*
15	$\overline{\text{SCT}}$	Synchronous Transmit Clock
17	SCR	Synchronous Receive Clock
22	RI	Ring Indicator

*Used with TM 990/506 Modem Cable Only.

TABLE H-4. PARALLEL I/O INTERFACE (P4) SIGNAL ASSIGNMENT

P4 PIN	SIGNAL	P4 PIN	SIGNAL
20	P0	17	GND
22	P1	15	GND
14	P2	13	GND
16	P3	11	GND
18	P4	9	GND
10	P5	39	GND
12	P6	37	GND
24	$\overline{INT15}$ or P7	35	GND
26	$\overline{INT14}$ or P8	33	GND
28	$\overline{INT13}$ or P9	31	GND
30	$\overline{INT12}$ or P10	29	GND
32	$\overline{INT11}$ or P11	27	GND
34	$\overline{INT10}$ or P12	25	GND
36	$\overline{INT9}$ or P13	23	GND
38	$\overline{INT8}$ or P14	21	GND
40	$\overline{INT7}$ or P15	19	GND
7	GND	1	+12 V
8	POSITIVE EDGE TRIGGER $\overline{INT6}$	2	−12 V
		3	+5 V
		4	SPARE
		5	GND
		6	NEGATIVE EDGE TRIGGER $\overline{INT6}$

H-4

APPENDIX I

TM 990/301 MICROTERMINAL

I.1 GENERAL

The Texas Instruments Microterminal offers all of the features of a minicomputer front panel at reduced cost. The Microterminal, intended primarily to support the Texas Instruments TM 990/1XXM microcomputers, allows the user to do the following:

- Read from ROM or read/write to RAM

- Enter/display Program Counter

- Execute user program in free running mode or in single instruction mode

- Halt user program execution

- Enter/display Status Register

- Enter/display Workspace Pointer (this term is unique to the Texas Instruments 9900 microprocessor)

- Enter/display CRU data (this term is unique to the Texas Instruments 9900 microprocessor)

- Convert hexadecimal quantity to signed decimal quantity

- Convert signed decimal quantity to hexadecimal quantity

I.2 SPECIFICATIONS

- Power Requirements
  +12V (±3%), 50 mA
  −12V (±3%), 50 mA
  +5V (±3%), 150 mA

- Operating Temperature: 0°C to 50°C (+32° to +122°F)

- Operating Humidity: 0 to 95 percent, non-condensing

- Shock: Withstand 2 foot vertical drop

I.3 INSTALLATION AND STARTUP

To install the Microterminal onto a TM 990/1XX microcomputer, do the following:

- Attach jumpers to:
  - On TM 990/100M: J13, J14, and J15, and set J7 to EIA position
  - On TM 990/101M: E20-E21, E22-E23, and E24-E25
  - On TM 990/180M: J4, J5, and J6, and set J13 to EIA position.

- Attach the EIA cable from the Microterminal to connector P2. Signals between the Microterminal and the microcomputer are listed as in Table 1.

- To initialize the system, actuate the microcomputer RESET switch, then press the microterminal [CLR] key.

NOTE

If the user has installed the *optional* filter capacitor on the $\overline{RESTART}$ input, this capacitor must be removed for proper operation (e.g., if C5 is installed on the TM 990/100M or TM 990/180M microcomputer, this capacitor must be removed).

FIGURE I-1. TM 990/301 MICROTERMINAL

TABLE I-1. EIA CABLE SIGNALS

EIA Connector Pin	Interface Signal	AT TM 990/100M/180M/101M	
		P2 Pin	Signal
2	TERMINAL DATA OUT	−2	RS232 RCV
3	TERMINAL DATA IN	−3	RS232 XMT
7	GND	−7	GND
12	+12V	−12	+12V
13	−12V	−13	−12V
14	+ 5V	−14	+ 5V
16	HALT	−16	RESTART

CAUTION

Before attaching the Microterminal to a power source, verify voltage
levels between ground and EIA connector pins 12, 13, and 14
at connector P2 on the board. Voltage should not exceed values in
Table I-1.

I.4   KEY DEFINITIONS

I.4.1   DATA KEYS

CLR   Clear Key — Depressing this key blanks display, initializes and sends initialization message (ASCII code
for A and ASCII code for Z) to host microcomputer.

0
1
·
·
F/—

Hexadecimal Data Keys — Depressing any one of these keys shifts that value into the right-hand display
digit. All digits already in the data display are left shifted. For all operations other than decimal to
hexadecimal conversion, the fourth digit from the right is shifted off the end of the right-hand display
field when a data key is depressed. For a decimal to hexadecimal conversion, the fifth display digit from
the right, rather than the fourth, is shifted off the end of the data field.

I.4.2   INSTRUCTION EXECUTION

H/S   Pressing this key while a program is running (run displayed) will halt program execution. The address of
the next instruction will be displayed in the four left-hand display digits, and the contents of that
address will be displayed in the four right-hand digits. Pressing this key while the program is halted, will
execute a single instruction using the values in the Workspace Pointer (WP), Program Counter (PC), and
Status Register (ST), and the displays will be updated to the next memory address and contents at that
address.

RUN   Pressing this key initiates program execution at the current values in the WP, PC; run is displayed in the
three right-hand display digits.

I.4.3   ARITHMETIC

H→D   The signed hexadecimal data contained in the four right-hand display digits is converted to signed
decimal data. Note that the fourth display digit from the right is the sign bit ($1 =$ negative). The
conversion limits are minus $32,768_{10}$ (8000_{16}) to plus $32,767$ ($7FFF_{16}$). Two H→D key depressions are
required. The sequence is:

1   Depress H→D .
2   Enter data via hex data key depressions.
3   Depress H→D . The results of the conversion are displayed in the five right-hand display
digits.

D→H   The decimal data contained in the five right-hand display digits is converted to hexadecimal. The
conversion limits are the same as for hexadecimal to decimal conversion. The sequence is:

1   Depress D→H .
2   Enter data via hex data key depressions.
3   Depress D→H . The results of the conversion are displayed in the four right-hand display
digits.

I.4.4   REGISTER ENTER/DISPLAY

EWP   Pressing this key causes the value displayed in the four right-hand digits to be entered into the WP.

DWP   Pressing this key causes the WP contents to be displayed in the four right-hand display digits.

EPC   Pressing this key causes the value displayed in the four right-hand digits to be entered into the PC.

DPC   Pressing this key causes the PC contents to be displayed in the four right-hand display digits.

EST   Pressing this key causes the value displayed in the four right-hand digits to be entered into the ST.

DST   Pressing this key causes the ST contents to be displayed in the four right-hand display digits.

I.4.5   CRU DISPLAY/ENTER

DCRU   Pressing this key causes the data at the designated Communications Register Unit (CRU) addresses to be displayed. Designate from one to 16 CRU bits at a specified CRU address by using four hexadecimal digits. The first digit is the count of bits to be displayed. The next three digits are the CRU address (equal to bits 3 to 14 in register 12 for CRU addressing). When **DCRU** is depressed, the bit count and address are shifted to the left-hand display, and the right-hand display will contain the values at the selected CRU output addresses. The output value will be zero-filled on the left, depending upon bit count entered. If less than nine bits, the value will be contained in the left two hexadecimal digits. If nine or more, the value will be right justified in all four hexadecimal digits.

ECRU   Pressing this key enters a new value at the CRU addresses and bit count shown in the left display after depressing **DCRU**. The new value is entered from the keyboard and displayed in the right-hand display. Pressing **ECRU** enters this value onto the CRU at the address shown in the left display.

<div align="center">CAUTION</div>

Avoid setting new values at the TMS 9902 on the TM 990/100M/180M/101M through the CRU (TMS 9902 is at CRU address 0040_{16}), as this device controls I/O functions.

I.4.6   MEMORY ENTER, DISPLAY, INCREMENT

EMA   Pressing this key will cause (1) the memory address (MA) in the right-hand display to be shifted to the left-hand display and (2) the contents of that memory address to be displayed in the right-hand display.

EMD   Pressing this key causes the value in the right-hand display to be entered into the memory address contained in the left-hand display. The contents of that location will then be displayed in the four right-hand display digits (entered then read back).

EMDI   Pressing this key causes the same action as described for the **EMD** key; it also increments the memory address by two and displays the contents at that new address. The memory address is displayed on the left and the contents at that address is displayed on the right.

I.5   EXAMPLES

I.5.1   EXAMPLE 1, ENTER PROGRAM INTO MEMORY

Enter the following program starting at RAM location $FE00_{16}$. Set the workspace pointer to $FF00_{16}$ and the status register to 2000_{16}. Single step through the program and verify execution. Then execute the program in free run mode and verify execution. Then halt program execution.

In the following examples, XXXX indicates memory contents at current value in Memory Address Register.

OPCODE	INSTRUCTIONS		
04C0	CLR	R0	CLEAR WORKSPACE REGISTER 0
0580	INC	R0	INCREMENT WORKSPACE REGISTER 0
0280	CI	R0, >00FF	CHECK FOR COUNT 255
00FF			
16FC	JNE	$-6	JUMP TO INC R0 IF NOT DONE
10FF	JMP	$-0	STAY HERE WHEN FINISHED

	KEY ENTRIES		DISPLAY
Clear Display	Depress	CLR	
Enter PC Value	Depress	F/− E 0 0	FE00
Enter into PC	Depress	EPC	FE00
Display PC	Depress	DPC	FE00
Enter ST Value	Depress	2 0 0 0	2000
Enter into ST	Depress	EST	2000
Display ST	Depress	DST	2000
Enter WP Value	Depress	F/− F/− 0 0	FF00
Enter Into WP	Depress	EWP	FF00
Display WP	Depress	DWP	FF00
Enter MA Value	Depress	F/− E 0 0	FE00
Enter Into MA	Depress	EMA	FE00 xxxx
Enter CLR 0 Opcode	Depress	0 4 C 0	FE00 04C0
Enter data, increment MA	Depress	EMDI	FE02 xxxx
Enter INC 0 Opcode	Depress	0 5 8 0	FE02 0580
Enter Data, Increment MA	Depress	EMDI	FE04 xxxx
Enter CI Opcode	Depress	0 2 8 0	FE04 0280
Enter Data, Increment MA	Depress	EMDI	FE06 xxxx

		KEY ENTRIES	DISPLAY
Enter CI			
Immediate Operand	Depress	0 0 F F	FE06 00FF
Enter Data,			
Increment MA	Depress	EMDI	FE08 xxxx
Enter JNE $-6			
Opcode	Depress	1 6 F C	FE08 16FC
Enter Data,			
Increment MA	Depress	EMDI	FE0A xxxx
Enter			
JMP $-0 Opcode	Depress	1 0 F F	FE0A 10FF
Enter Data,			
Increment MA	Depress	EMDI	FE0C xxxx

The program has now been entered into RAM. Since the PC, ST and WP values have been previously set, the program can be executed in single step mode by depressing the H/S key.

			DISPLAY (AFTER)	EXECUTES INSTRUCTION
	Depress	H/S	FE02 0580	CLR R0
	Depress	H/S	FE04 0280	INC R0
	Depress	H/S	FE08 16FC	CI RO,>00FF
	Depress	H/S	FE02 0580	JNE S-6

This cycle will continue until R0 reaches the count of 255 at which point the program will continuously execute at location FE0A$_{16}$ because it is a jump to itself.

To verify this, depress: DISPLAY

RUN   [run]

The program should now be "looping to self" at location FE0A$_{16}$. To verify this, depress:

H/S   FE0A 10FF

Now examine the memory location corresponding to Register 0.

Depress	F F 0 0	FE0A FF00
Depress	EMA	FF00 00FF

This illustrates that FF$_{16}$ did become the final contents of WP0. Note that, when the program was being entered into RAM, EMDI was used rather than EMD because of the rather desirable feature of automatic address incrementing. The advantage of using EMD is that the actual contents of the addressed memory location are displayed after key depression (echoed back after being entered).

I.5.2 EXAMPLE 2, HEXADECIMAL TO DECIMAL CONVERSIONS

Convert 8000_{16} to a decimal number

Depress	CLR		
Depress	H→D		
Depress	8 0 0 0		8000
Depress	H→D		−3 2768

Convert 0020_{16} to a decimal number

Depress	CLR		
Depress	H→D		
Depress	2 0		20
Depress	H→D		32

I.5.3 EXAMPLE 3, DECIMAL TO HEXADECIMAL CONVERSIONS

Convert 45_{10} to hex

Depress	CLR		
Depress	D→H		
Depress	4 5		45
Depress	D→H		2D

Convert -1024_{10} to hex

Depress	CLR		
Depress	D→H		
Depress	F/− 1 0 2 4	−	1024
Depress	D→H		FC00

I.5.4 EXAMPLE 4, ENTER VALUE ON CRU

Send a bit pattern to the CRU at CRU address (bits 3 to 14 of R12) $0E0_{16}$ with a bit count of 9 containing a value of 5 (000000101_2).

Depress	CLR					
Depress	9	0	E	0		90E0
Depress	DCRU					90E0 YYYY
Depress	0	0	0	5		90E0 0005
Depress	ECRU					

YYYY indicates value at the current CRU address. Note that a DCRU operation is always required to specify bit count/CRU address.

I.5.5 EXAMPLE 5. ENTER, VERIFY VALUE AT MEMORY ADDRESS

Enter 0040_{16} into location FE20 and verify that it got there.

Depress	CLR					
Depress	F	E	2	0		FE20
Depress	EMA					FE20 xxxx
Depress	0	0	4	0		FE20 0040
Depress	EMD					FE20 0040

The contents of address FE20 are verified by an echo of data from memory to display following the pressing of EMD. If it is desired to view and enter data at address FE22, depress EMD.

CRU INSTRUCTION AND ADDRESSING EXAMPLES USING TMS 9901

The following figures, J-1 to J-6, are examples of addressing the TMS 9901 through the CRU, pointing out in graphic form:

- External I/O in parallel (multibit) and serial (single bit) forms,

- The relationship between the CRU bits addressed and the bits in the source operand of the STCR instructions,

- The relationship between the CRU bit addressed and the displacement in single-bit instructions.

The TMS 9901 occupies 32 bit positions of CRU space with the low 16 bits at CRU software base address 0100_{16} and the high 16 bits at CRU software base address 0120_{16}. To access the low 16 bits of the TMS 9901 through the CRU, load 0100 into register 12.

The high 16 bits at CRU software base address 0120_{16} are the parallel I/O bits, shown in the accompanying figures. These may be set, reset, or read in any order or in any combination of 1 to 16 bits. Since CRU operations are serial, data from the microprocessor (either serial or parallel) is transmitted serially to the TMS 9901, which outputs it in parallel. Likewise, during input, data present at the TMS 9901 I/O pins (in parallel) is shifted serially to the microprocessor using the CRU. It is necessary only to load register 12 with 0120_{16} and use either the LDCR or STCR instructions. Bear in mind that the CRU operations of 1 to 8 bits affect the left byte (more significant half) of a word (registers take up a full memory word).

The lower 16 bits of the TMS 9901 at CRU software base address 0100_{16} are used for control of interrupts and the timer function, and to restore the I/O lines to the input mode with output buffers disabled and floating. Interrupt requests are presented to the TMS 9901, each on its own line, and are compared against an internal mask. If the internal interrupt mask allows, the particular interrupt request is encoded into TMS 9901 output lines IC0 to IC3 (going to interrupt input lines IC0 to IC3 at the TMS 9900) as explained on page 6 of the TMS 9900 data manual and page 8 of the TMS 9901 data manual. The TMS 9901 also pulls the INTREQ- line low on interrupt requests (not during RESET), which goes to INTREQ- at the TMS 9900.

(1) ASSEMBLY LANGUAGE:

        LI   R12, > 0120
        LDCR   RO,15

(2) SOURCE ADDRESS IN MEMORY:
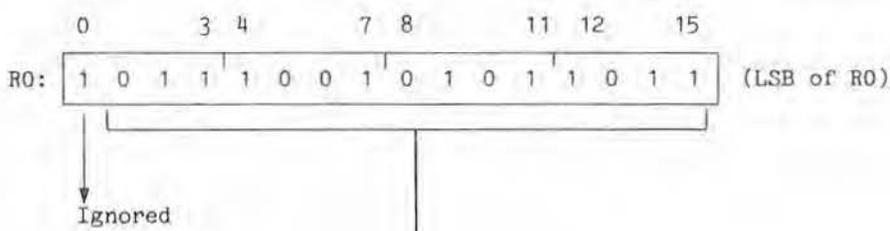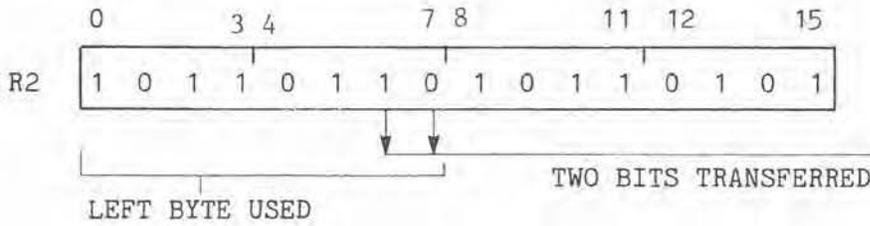

(3) ADDRESSING:

Address lines at operation start
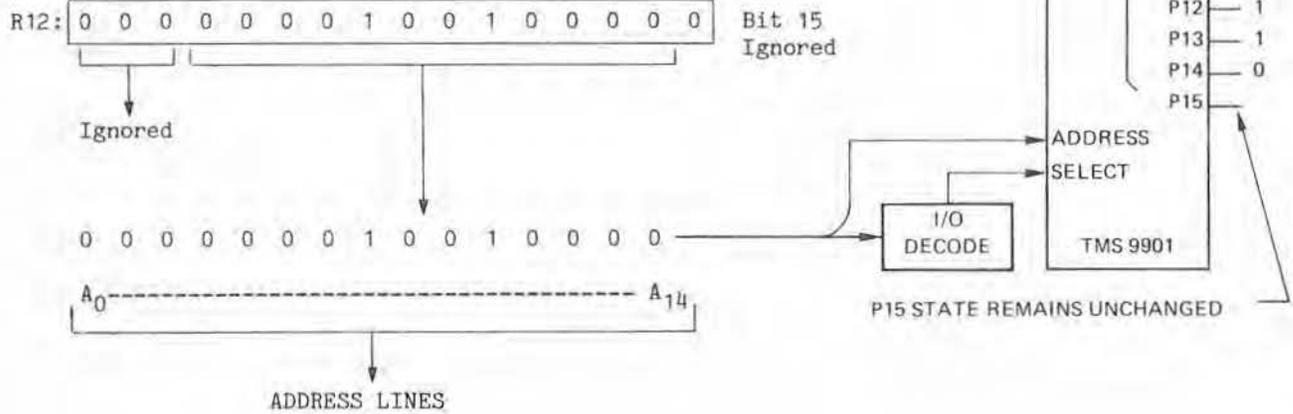
Figure J-1. LDCR Word Execution To TMS 9901

(1) ASSEMBLY LANGUAGE:

       LI   R12,> 0128
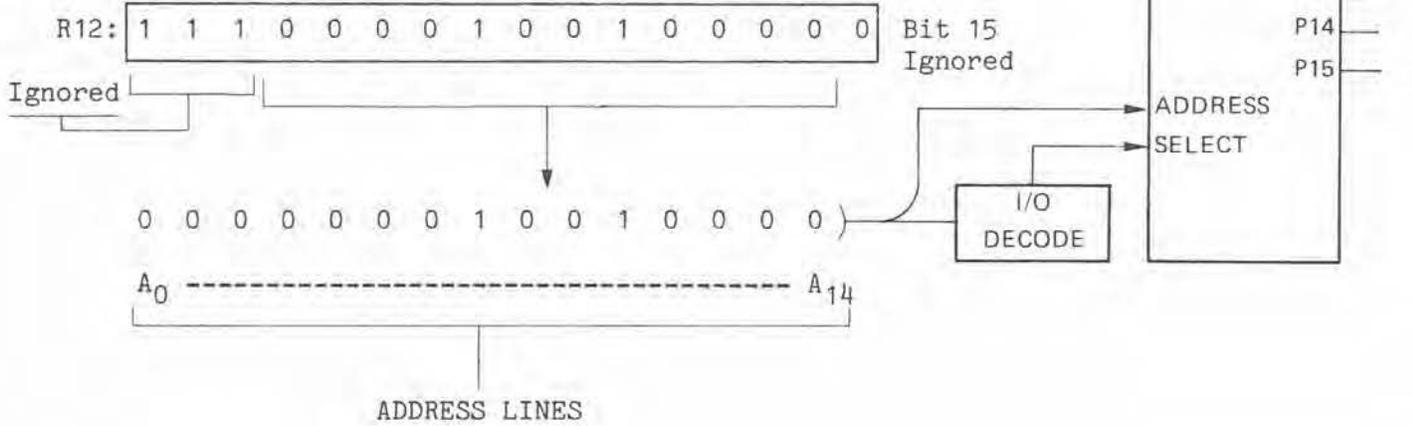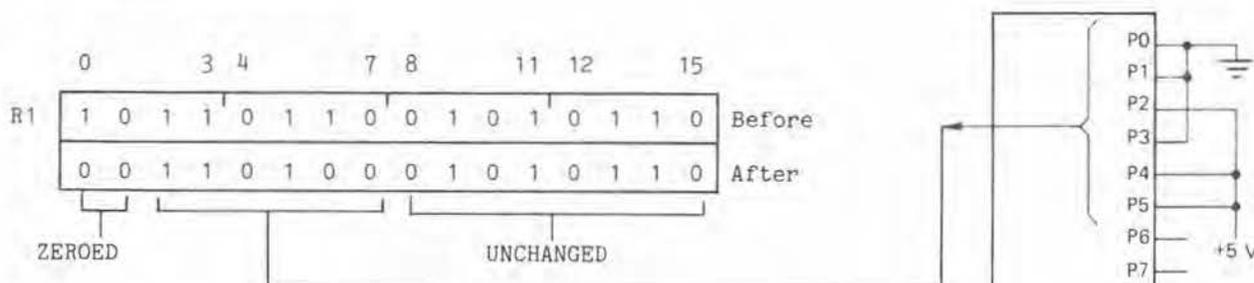   LDCR   R2,2

(2) SOURCE ADDRESS IN MEMORY:

```
        0       3 4       7 8      11 12      15
R2    | 1 0 1 1 0 1 1 0 1 0 1 1 0 1 0 1 |
```

LEFT BYTE USED   TWO BITS TRANSFERRED

(3) ADDRESSING:

```
R12: | 1 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 |   Bit 15
                                            Ignored
```

Ignored

```
      0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 )
      A0 ---------------------------------- A14
```

ADDRESS LINES

ADDRESS
SELECT

I/O
DECODE

PO
P1
P2
P3
P4
P5
P6
P7
P8 0
P9 1
P10
P11
P12
P13
P14
P15

Figure J-2. LDCR Byte Execution To TMS 9901

J-3

(1) ASSEMBLY LANGUAGE:

```
LI      R12, >120
STCR    R3,11
```

(2) SOURCE ADDRESS IN MEMORY:
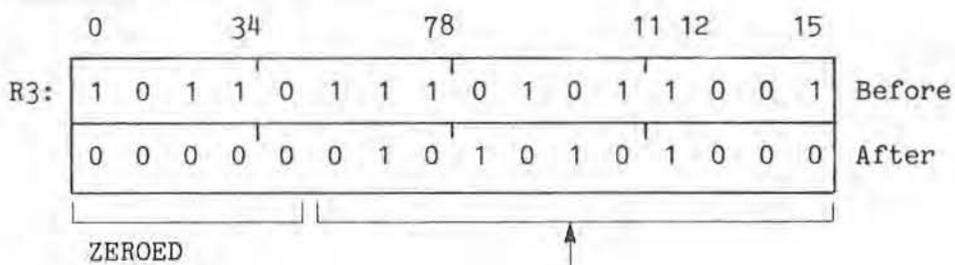
(3) ADDRESSING:

Address lines at operation start
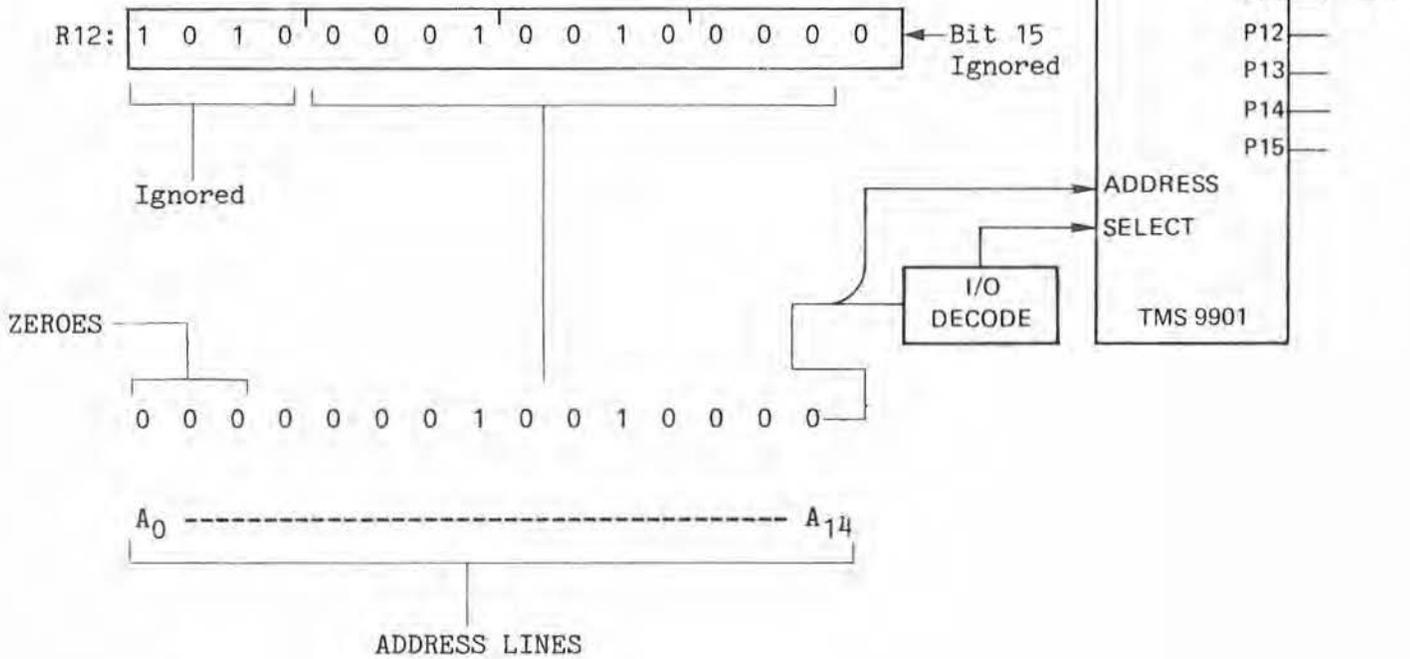
Figure J-3. STCR Word Execution To TMS 9901

J-4

(1) ASSEMBLY LANGUAGE:

```
LI      R12,> 120
STCR    R1,6
```

(2) SOURCE ADDRESS IN MEMORY:


```
        0       3  4       7 8      11 12      15
R1    1 0 1 1 0 1 1 0 0 1 0 1 0 1 1 0   Before
      0 0 1 1 0 1 0 0 0 1 0 1 0 1 1 0   After

      ZEROED                 UNCHANGED
```

(3) ADDRESSING:

```
R12: 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0   Bit 15
                                        Ignored
Ignored

     0 0 0 0 0 0 0 1 0 0 1 0 0 0 0

A0                               A14

         ADDRESS LINES
```

TMS 9901

P0
P1
P2
P3
P4
P5
P6   +5 V
P7
P8
P9
P10
P11
P12
P13
P14
P15

ADDRESS
SELECT

I/O
DECODE
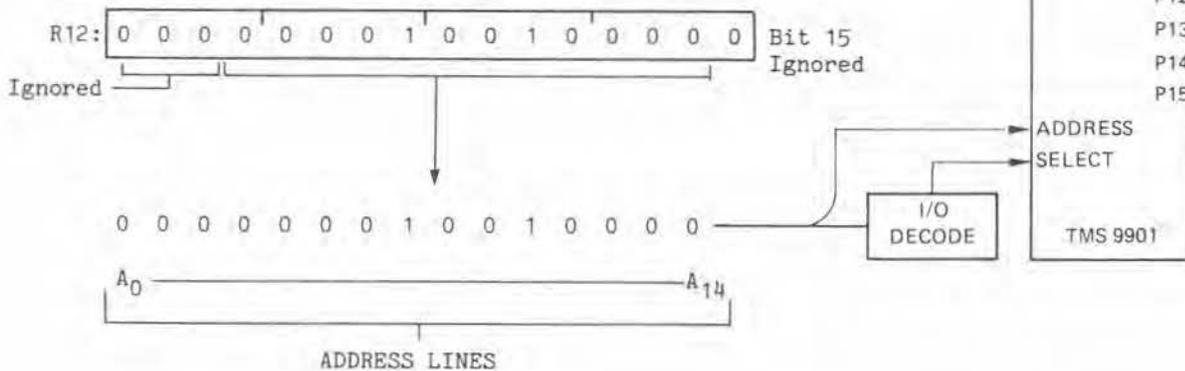
Figure J-4.   STCR Byte Execution To TMS 9901

J-5

(1) ASSEMBLY LANGUAGE:

        LI   R12,>140
        TB   -3

(2) ADDRESSING:

R12 `0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0` Bit 15 ← is ignored

ignored

`1 1 1 1 1 1 1 1 1 1 0 1` ← -3 Displacement Added to Address

sign extend

ZEROES

`0 0 0 0 0 0 0 1 0 0 1 1 1 0 1`

A_0 —————————————— A_{14}

ADDRESS LINES

ADDRESS

SELECT

I/O DECODE

TMS 9901

F
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13 — 1
P14
P15

(3) STATUS REGISTER:

BIT NO. 0   3   15

`1`
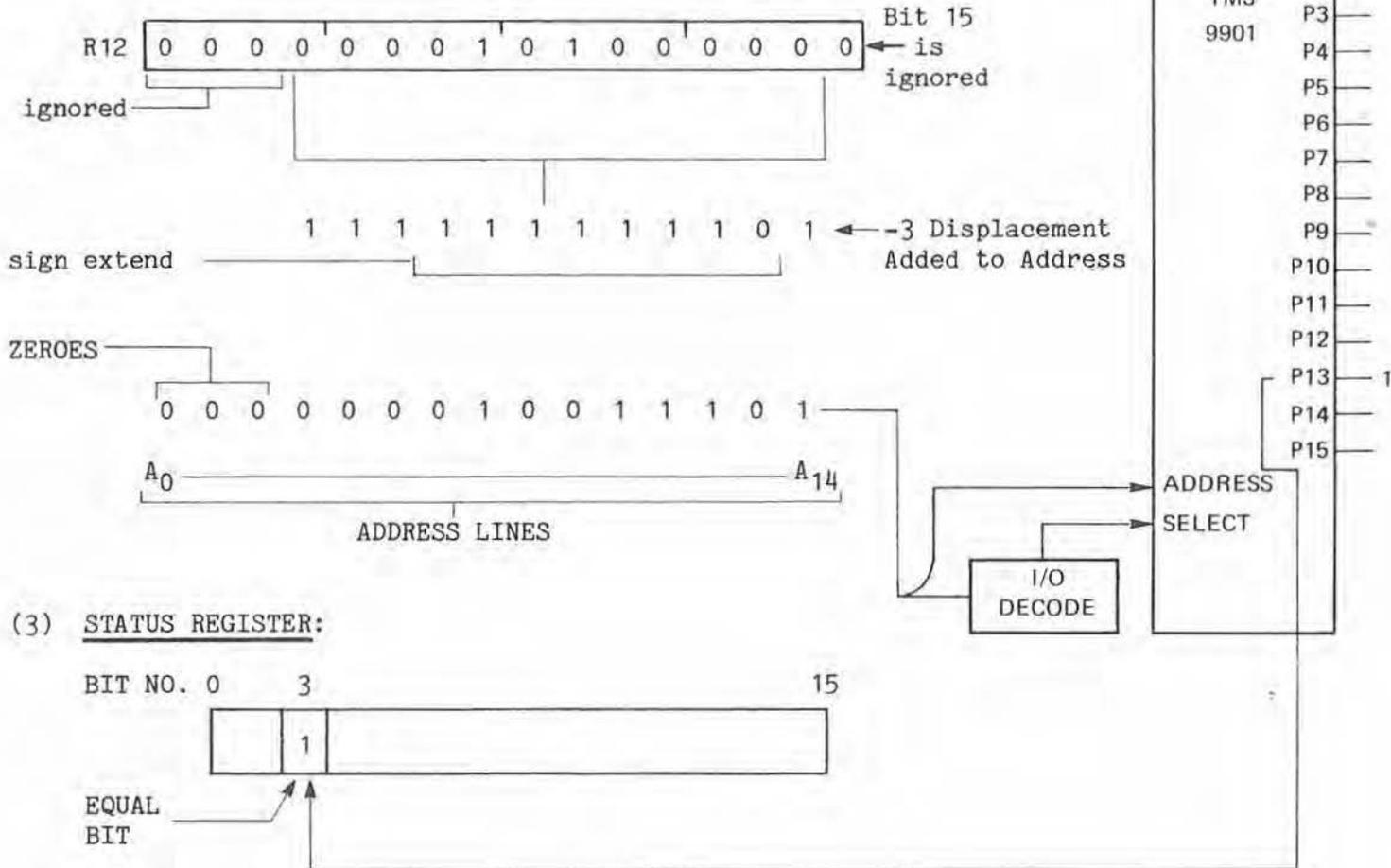
EQUAL
BIT

NOTE
If a JEQ (jump on equal) instruction follows a TB
instruction, a 1 found will cause a jump, and a 0 found
will not cause a jump (1 = EQUAL state).

Figure J-5.   Test CRU Bit At TMS 9901

(1) ASSEMBLY LANGUAGE:

```
LI   R12,> 0120
SBZ  7
```

(2) ADDRESSING:
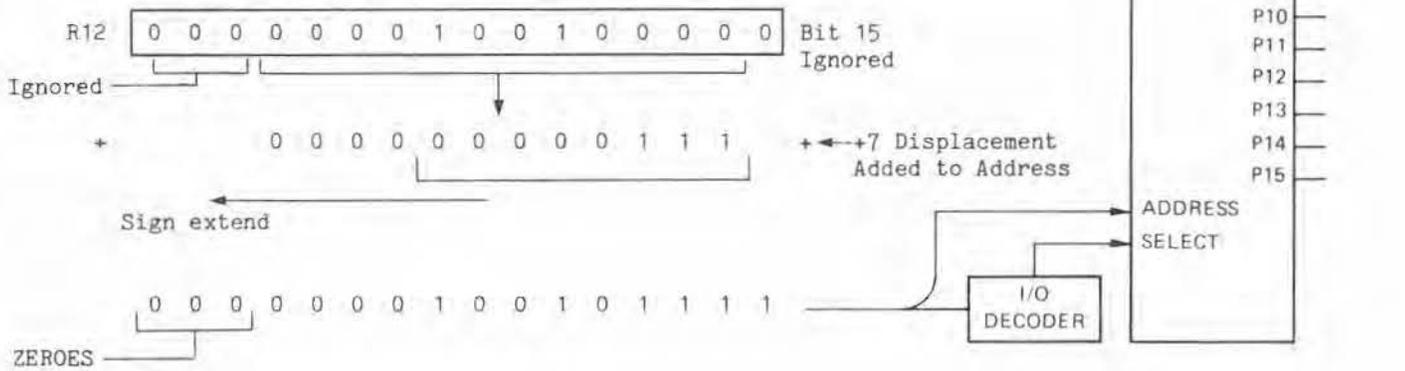
Figure J-6. Set CRU Bit At TMS 9901

APPENDIX K

EXAMPLE PROGRAMS

EXAMPLE PROGRAMS

This appendix contains listings of programs that can be loaded into memory or reassembled into memory for demonstration or entertainment purposes. These listings are commented to provide ancillary data and explain the individual programming techniques. Assembly listing format is as follows:

———————————— SOURCE STATEMENT NUMBER (DECIMAL)

——————————— RELATIVE MEMORY ADDRESS

————————— ASSEMBLED OBJECT CODE (HEXADECIMAL)

——————— LABEL FIELD

—————— OP CODE MNEMONIC

————— OPERAND

COMMENT FIELD ———

```
0029 FE00 02E0   START   LWPI WSP          SET WORKSPACE POINTER
     FE02 FF96
0030 FE04 0200           LI    R0,10       R0 = TENS MULTIPLIER
     FE06 000A
0031 FE08 04C9           CLR  R9           R9 = NO. OF TRIES
0032 FE0A 04CA           CLR R10           R10 = NO. OF TRIES
0033 FE0C 020C           LI    R12,>80     TMS 9902 CRU ADDR.
     FE0E 0080
0034                  *  OUTPUT OPENING MESSAGE
0035 FE10 2FA0           XOP  @MESS1,14    OPENING MESSAGE
     FE12 FEDA
```

——————— FULL-LINE COMMENT BEGINS WITH ASTERISK

—— ADDRESS OF LABEL MESS1 IS M.A. FEDA$_{16}$

The code can be reassembled and loaded with the L TIBUG command, or the change memory command (M) can be used to insert assembled object code at the memory addresses shown in the listing (beginning at $FE00_{16}$, program start). The assembled object code is listed in column 3 of the listing, opposite the corresponding memory address in column 2. It is important that the programs be entered at the addresses noted, or that proper consideration be given to the labelled addresses which have been assembled into absolute addresses relative to the beginning of the program (address $FE00_{16}$). This consideration is important when entering the code using the enter memory (M) command with program start not at address $FE00_{16}$.

If the code is to be loaded beginning at an address other than $FE00_{16}$ as a program start address, it must be refigured to the new program bias. For example, if the program was to be loaded beginning at $FC00_{16}$, labelled addresses must be decreased by 200_{16} ($FE00_{16}$ - $FC00_{16}$ = 200_{16}). Note that jump instructions create a displacement value and not a memory address; thus, jump instructions using labels are not affected by a new program start address.

At the back of each listing is a cross-reference of labels and number of the source statement in which they are used (column one of the listing contains source statement numbers).

If the Line-By-Line Assembler (LBLA) is used, an absolute address must be substituted for labelled addresses. These hexadecimal values are in the first column of the cross-reference table of labels.

K.1   MASTERMIND GAME

The printout of this game in execution (below) illustrates game rules and
objective. The program generates a five-digit number. To win, you must deduct which
five digits make up the number, and their correct order. Only digits 1 to 8 are
used. After each guess, the program prints the letters X and 0 for each correct
digit entered. In addition, each X indicates a digit is in the correct column. You
are given only 12 tries to win.


```
MASTERMIND..GUESS NNNNN N=1-8 12 TRIES
YOU GET X FOR A MATCH, 0 FOR A HIT

    1..11111  X
    2..12222  0
    3..31333  0
    4..41          ◄─────────────────── CONTROL-H CAUSES ENTRY TO BE IGNORED, ALLOWS ENTRY REPEAT
    4..44144  X0
    5..55415  00
    6..64166  XX0
    7..46177  0000
    8..64718  XXX00
    9..64781  XXXXX    WINNER!   N=64781

    1..11111
    2..22222  X
    3..23333  XX0
    4..32434  000
    5..25353  XXX00
    6..           ◄──────────── CR RESTARTS PROGRAM
MASTERMIND..GUESS NNNNN N=1-8 12 TRIES
YOU GET X FOR A MATCH, 0 FOR A HIT

    1..11111
    2..22222  X
    3..23333  00
    4..32444  XXX
    5..34255  X00
    6..           ◄──────────── ESC KEY RETURNS CONTROL TO MONITOR
?
```

K-3

```
0001                              IDT   'MMIND'
0003          *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
0004          *  THIS PROGRAM PLAYS MASTERMIND ON THE TM 990/1XX MICRO-
0005          *  COMPUTERS. THE OBJECT OF THE GAME IS TO GUESS, BY
0006          *  LOGICAL DEDUCTION, A 5-DIGIT NUMBER GENERATED BY THE
0007          *  COMPUTER. THE COMPUTER USES ONLY THE DIGITS 1 TO 8. YOU
0008          *  HAVE 12 GUESSES TO ACCOMPLISH THIS. THE COMPUTER WILL
0009          *  INDICATE A CORRECT DIGIT GUESSED BY A LETTER O AND
0010          *  INDICATE THE DIGIT IS CORRECTLY PLACED WITHIN THE
0011          *  5-DIGIT NUMBER WITH THE LETTER X. OTHER RULES THAT APPLY:
0012          *      - A CARRIAGE RETURN RESTARTS THE GAME
0013          *      - AN ESCAPE KEY INPUT RETURNS YOU TO THE MONITOR
0014          *      - CONTROL H KEY ALLOWS YOU TO SCRAP PRESENT LINE OF
0015          *          ENTRIES AND REENTER NEW LINE
0016          *  THIS GAME IS ASSEMBLED TO BE LOADED AT M.A. >FE00 BY
0017          *  USE OF THE AORG ASSEMBLER DIRECTIVE. THIS PROGRAM CAN BE
0018          *  ASSEMBLED BY THE LBLA AT THE ADDRESSES SHOWN IN COLUMN
0019          *  TWO OF THE LISTING. CORRESPONDING OBJECT CODE FOR THOSE
0020          *  ADDRESSES IS SHOWN IN COLUMN THREE. GOOD LUCK!
0021          *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
0022      0000  R0     EQU   0                    NO. OF GUESSES
0023      0001  R1     EQU   1                    RANDOM NO. ARRAY ADDRESS
0024      0002  R2     EQU   2                    RANDOM NO. COMPUTATION USE
0025      0003  R3     EQU   3                    RANDOM NO. COMPUTATION USE
0026      0004  R4     EQU   4                    10 CONSTANT FOR DECIMAL COMPUT
0027      0005  R5     EQU   5                    CONTAINS ASCII 'X'
0028      0006  R6     EQU   6                    CONTAINS ASCII 'O'
0029      0007  R7     EQU   7                    ADDRESS OF X'S & O'S BUFFER
0030      0008  R8     EQU   8
0031      0009  R9     EQU   9                    RANDOM NO. ARRAY ADDRESS
0032      000A  R10    EQU   10                   RANDOM NO. ARRAY ADDRESS+5
0033      000B  R11    EQU   11                   RANDOM NO. SEED
0034      000C  R12    EQU   12                   ASCII '1' (>3100)
0035      000D  R13    EQU   13                   CAST OUT CHARACTER MAP
0036 FE00        AORG  >FE00                      LOAD AT M.A. >FE00
0037          *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
0038          *
0039          *  PROCEDURE AREA OF EXECUTABLE CODE
0040          *
0041          *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
0042                START
0043 FE00 02E0       LWPI  WS                     SET WORKSPACE POINTER
     FE02 FED6
0044 FE04 2FA0       XOP   @RULES,14              PRINT RULES
     FE06 FF0C
0045                M005
0046 FE08 2FA0       XOP   @CRLF,14               PRINT CR-LF
     FE0A FF72
0047 FE0C 04C0       CLR   R0                     COUNTS 12 GUESSES
0048 FE0E C049       MOV   R9,R1                  R1 POINTS TO RANDOM ARRAY
```

```
0050                        * COMPUTE RANDOM NUMBER, MOVE TO LOCATION MM
0051                        M010
0052 FE10 0202                  LI   R2,509        COMPUTE RANDOM NUMBER
     FE12 01FD
0053 FE14 388B                  MPY  R11,R2
0054 FE16 0223                  AI   R3,291
     FE18 0123
0055 FE1A C2C3                  MOV  R3,R11
0056                        * CAUSE RANDOM DIGITS TO BE IN RANGE 1-8
0057 FE1C 0953                  SRL  R3,5
0058 FE1E B0CC                  AB   R12,R3        MAKE ASCII, RANGE 1-8
0059 FE20 DC43                  MOVB R3,*R1+       PUT IN RANDOM ARRAY
0060 FE22 8281                  C    R1,R10        TEST FOR END OF LOOP
0061 FE24 1AF5                  JL   M010         DO UNTIL R1-R10
0062                        *
0063                        * DETERMINE NUMBER OF UPCOMING GUESS
0064                        * PRINT UPCOMING GUESS NUMBER TO PROMPT USER
0065                        *
0066                        M015
0067 FE26 0580                  INC  R0           GUESS GUESS+1
0068                        * CLEAR ARRAY THAT HOLDS ASCII X'S AND O'S
0069                        * IF CONTROL H PRESSED, START HERE
0070 FE28 C087              RESTRT MOV  R7,R2      XOB ADDR TO R2
0071 FE2A 04F2                  CLR  *R2+          *
0072 FE2C 04F2                  CLR  *R2+          *
0073 FE2E 04D2                  CLR  *R2           *
0074                        * CONVERT GUESS NUMBER FOR OUTPUT
0075 FE30 C080                  MOV  R0,R2         GUESS NO. TO R2
0076 FE32 04C1                  CLR  R1
0077 FE34 3C44                  DIV  R4,R1         DIVIDE R1R2 BY 10
0078 FE36 06C1                  SWPB R1            QUOTIENT IN LEFT BYTE
0079 FE38 F081                  SOCB R1,R2         MERGE QUOTIENT & REMAINDER
0080 FE3A 1302                  JEQ  M020          PUT IN SPACE IF FIRST DIGIT=0
0081 FE3C 0262                  ORI  R2,>3030      MAKE ASCII DIGITS
     FE3E 3030
0082                        M020
0083 FE40 0262                  ORI  R2,>2030      MAKE ASCII SPACE & DIGIT
     FE42 2030
0084 FE44 C802                  MOV  R2,@GCD       PUT IN PRINT BUFFER
     FE46 FEF4
0085 FE48 2FA0                  XOP  @GUESNO,14    PRINT GUESS NUMBER
     FE4A FEF2
```

```
0087                   *
0088                   * INPUT CHARACTER & TEST FOR COLUMN MATCH
0089                   *
0090 FE4C C209              MOV   R9,R8          RANDOM NUMBER ADDR IN R8
0091 FE4E C047              MOV   R7,R1          X & O BUFF ADDR IN R1
0092 FE50 0202              LI    R2,INPUT       INPUT BUFFER ADDR IN R2
     FE52 FF5A
0093 FE54 04CD              CLR   R13            CLEAR BIT MAP OF CAST OUT CHAR
0094                   M030
0095 FE56 2F43              XOP   R3,13          READ DIGIT
0096                   * WAS CR, ESCAPE, OR CONTROL-H KEY PRESSED?
0097 FE58 0283              CI    R3,>0D00       CAR. RET. ENTERED?
     FE5A 0D00
0098 FE5C 13D1              JEQ   START          YES, RESTART GAME
0099 FE5E 0283              CI    R3,>1B00       ESCAPE KEY ENTERED?
     FE60 1B00
0100 FE62 131C              JEQ   MONITR         YES, RETURN TO MONITOR
0101 FE64 0283              CI    R3,>0800       CONTROL-H PRESSED?
     FE66 0800
0102 FE68 13DF              JEQ   RESTRT         YES, RESTART THIS ENTRY
0103 FE6A 9303              CB    R3,R12         IS NO. LESS THAN 1?
0104 FE6C 1AF4              JL    M030           YES, READ ANOTHER
0105 FE6E 0283              CI    R3,>3800       IS NO. GREATER THAN 8?
     FE70 3800
0106 FE72 1BF1              JH    M030           YES, READ ANOTHER
0107 FE74 2F03              XOP   R3,12          NO, IN RANGE, ECHO
0108                   * IS DIGIT A MATCH AND IN RIGHT COLUMN?
0109 FE76 9E03              CB    R3,*R8+        DIGIT IN RIGHT COLUMN?
0110 FE78 1603              JNE   M040           NO, PUT CHAR IN CHAR BUFFER
0111 FE7A 06C3              SWPB  R3             YES, PUT BINARY 0 IN MSB OF '
0112 FE7C DC45              MOVB  R5,*R1+        PUT AN X IN THE XO BUFFER
0113 FE7E 058D              INC   R13            MAP CAST OUT CHAR
0114                   M040
0115 FE80 DC83              MOVB  R3,*R2+        ZERO OR CHAR TO INPUT BUFFER
0116 FE82 0B1D              SRC   R13,1          PUT BIT IN MAP
0117 FE84 8288              C     R8,R10         FIFTH NUMBER INPUT?
0118 FE86 1AE7              JL    M030           NO, READ ANOTHER GUESS
0119 FE88 0281              CI    R1,XOB+5       YES, IS XO BUFFER FULL?
     FE8A FF0B
0120 FE8C 1A09              JL    M050           NO, NO WINNER YET
0121 FE8E 2FA0              XOP   @XOBP,14       YES, PRINT XO BUFF (ALL X'S)
     FE90 FF04
0122                   *
0123 FE92 2FA0              XOP   @WINNER,14     PRINT WINNER
     FE94 FF60
0124                   *
0125                   M045
0126 FE96 2FA0              XOP   @NUMBER,14     PRINT NUMBER
     FE98 FEFA
0127                   *
0128 FE9A 10B6              JMP   M005           PLAY ANOTHER GAME
0129 FE9C 0460  MONITR B    @>0080              RETURN TO MONITOR
     FE9E 0080
```

```
0131                     *
0132                     *
0133                     * TEST FOR 0'S...
0134                     *
0135                MO50
0136  FEA0 0202          LI   R2,INPUT          INPUT BUFFER START IN R2
      FEA2 FF5A
0137                MO52
0138  FEA4 D0F2          MOVB *R2+,R3           TEST BYTE FROM INPUT BUFFER
0139  FEA6 130C          JEQ  MO60              BYTE CAST OUT IF EQUAL TO ZERO
0140  FEA8 C209          MOV  R9,R8             R8 POINTS TO WORK ARRAY
0141  FEAA 09BD          SRL  R13,11            POSITION CAST OUT CH MAP
0142                MO55
0143  FEAC 0B1D          SRC  R13,1             TEST FOR CAST OUT CHAR
0144  FEAE 9E03          CB   R3,*R8+           DOES BYTE MATCH WORK ARRAY ?
0145  FEB0 1805          JOC  MO57              IF CAST OUT, MO57
0146  FEB2 1604          JNE  MO57              IF NOT EQUAL, MO57
0147  FEB4 DC46          MOVB R6,*R1+           ON HIT, PUT 0 IN XO BUFFER
0148  FEB6 026D          ORI  R13,>8000         MAP CAST OUT CHAR
      FEB8 8000
0149  FEBA B0C3          AB   R3,R3             SPOIL COMPARISON, FINISH LOOP
0150                MO57
0151  FEBC 8298          C    R8,R10            TEST FOR LAST DIGIT
0152  FEBE 1AF6          JL   MO55              IF LOW, DO ANOTHER DIGIT
0153                MO60
0154  FEC0 0282          CI   R2,INPUT+5        LAST DIGIT IN INPUT BUFFER?
      FEC2 FF5F
0155  FEC4 1AEF          JL   MO52              NO, DO NEXT DIGIT
0156  FEC6 2FA0          XOP  @XOBF,14          YES, PRINT XO BUFF
      FEC8 FF04
0157  FECA 0280          CI   R0,12             TWELVE GUESSES MADE?
      FECC 000C
0158  FECE 1A0B          JL   MO15              NO, MORE GUESSES REMAIN
0159  FED0 2FA0          XOP  @SORRY,14         YES, PRINT SORRY
      FED2 FF6A
0160  FED4 10E0          JMP  MO45              PRINT NUMBER FOR PLAYER
```

```
0162                 *   *   *   *   *   *   *   *   *   *   *   *   *   *   *
0163                 *
0164                 * DATA SECTION
0165                 *
0166                 *   *   *   *   *   *   *   *   *   *   *   *   *   *   *
0167                 * WORKSPACE
0168 FED6 0000  WS      DATA 0,0,0,0                    R0-R3
     FED8 0000
     FEDA 0000
     FEDC 0000
0169 FEDE 000A          DATA 10                         R4   CONVERSION CONSTANT
0170 FEE0   58          TEXT 'X '                       R5
     FEE1   20
0171 FEE2   4F          TEXT 'O '                       R6
     FEE3   20
0172 FEE4 FF06          DATA XOB                        R7
0173 FEE6 0000          DATA 0                          R8
0174 FEE8 FEFE          DATA NN                         R9
0175 FEEA FF03          DATA NN+5                        R10
0176 FEEC 5555          DATA >5555                      R11-RANDOM NUMBER SEED
0177 FEEE 3100          DATA >3100                      R12
0178 FEF0 0000          DATA 0                          R13-CAST OUT CHAR MAP
0179                 *
0180                 * TEXT STATEMENTS
0181                 *
0182                 * LINE NUMBER OF THIS GUESS
0183 FEF2 0D0A  GUESNO DATA >0D0A                       CR, LINE FEED
0184 FEF4 0000  GCD     DATA $-$                        CONVERTED GUESS NUMBER
0185 FEF6   2E          TEXT '..'
     FEF7   2E
0186 FEF8   07          BYTE 7,0                        BELL/STOP
     FEF9   00
0187                 * RANDOM NUMBER OF COMPUTER IN ASCII
0188 FEFA   20  NUMBER TEXT '  N='
     FEFB   20
     FEFC   4E
     FEFD   3D
0189 FEFE 0000  NN      DATA 0,0,0
     FF00 0000
     FF02 0000
0190                 * X'S AND O'S BUFFER SHOWING HITS & MISSES
0191 FF04   20  XOBP    TEXT '  '                       SPACES FOR PRINTING
     FF05   20
0192 FF06 0000  XOB     DATA 0,0,0
     FF08 0000
     FF0A 0000
0193                 * RULES OUTPUT AT BEGINNING OF GAME
0194             RULES
0195 FF0C 0D0A          DATA >0D0A
0196 FF0E   4D          TEXT 'MASTERMIND'
     FF0F   41
     FF10   53
     FF11   54
     FF12   45
```

```
          FF13  52
          FF14  4D
          FF15  49
          FF16  4E
          FF17  44
0197 FF18  2E              TEXT '..GUESS NNNNN N=1-8 12 TRIES'
          FF19  2E
          FF1A  47
          FF1B  55
          FF1C  45
          FF1D  53
          FF1E  53
          FF1F  20
          FF20  4E
          FF21  4E
          FF22  4E
          FF23  4E
          FF24  4E
          FF25  20
          FF26  4E
          FF27  3D
          FF28  31
          FF29  2D
          FF2A  38
          FF2B  20
          FF2C  31
          FF2D  32
          FF2E  20
          FF2F  54
          FF30  52
          FF31  49
          FF32  45
          FF33  53
0198 FF34  0D0A            DATA >0D0A
0199 FF36  59              TEXT 'YOU GET X FOR A MATCH, O FOR A HIT'
          FF37  4F
          FF38  55
          FF39  20
          FF3A  47
          FF3B  45
          FF3C  54
          FF3D  20
          FF3E  58
          FF3F  20
          FF40  46
          FF41  4F
          FF42  52
          FF43  20
          FF44  41
          FF45  20
          FF46  4D
          FF47  41
          FF48  54
          FF49  43
```

K-9

```
          FF4A    48
          FF4B    2C
          FF4C    20
          FF4D    4F
          FF4E    20
          FF4F    46
          FF50    4F
          FF51    52
          FF52    20
          FF53    41
          FF54    20
          FF55    48
          FF56    49
          FF57    54
0200      FF58    00              BYTE 0
```

```
0202                    * BUFFER OF NUMBERS INPUT
0203 FF5A 0000  INPUT   DATA 0,0,0
     FF5C 0000
     FF5E 0000
0204                    *
0205 FF60   20  WINNER  TEXT '  WINNER'
     FF61   20
     FF62   57
     FF63   49
     FF64   4E
     FF65   4E
     FF66   45
     FF67   52
0206 FF68   21          BYTE >21,0
     FF69   00
0207 FF6A   20  SORRY   TEXT ' SORRY'
     FF6B   53
     FF6C   4F
     FF6D   52
     FF6E   52
     FF6F   59
0208 FF70   00          BYTE 0,0
     FF71   00
0209 FF72   0D  CRLF    BYTE >D,>A,0,0
     FF73   0A
     FF74   00
     FF75   00
0210                    *
0211                    END   START

0000 ERRORS
```

```
CRLF     0209     0046
GCD      0184     0084
GUESNO   0183     0085
INPUT    0203     0092  0136  0154
M005     0045     0128
M010     0051     0061
M015     0066     0158
M020     0082     0080
M030     0094     0104  0106  0118
M040     0114     0110
M045     0125     0160
M050     0135     0120
M052     0137     0155
M055     0142     0152
M057     0150     0145  0146
M060     0153     0139
MONITR   0129     0100
NN       0189     0174  0175
NUMBER   0188     0126
R0       0022     0047  0067  0075  0157
R1       0023     0048  0059  0060  0076  0077  0078  0079  0091  0112
                  0119  0147
R10      0032     0060  0117  0151
R11      0033     0053  0055
R12      0034     0058  0103
R13      0035     0093  0113  0116  0141  0143  0148
R2       0024     0052  0053  0070  0071  0072  0073  0075  0079  0081
                  0083  0084  0092  0115  0136  0138  0154
R3       0025     0054  0055  0057  0058  0059  0095  0097  0099  0101
                  0103  0105  0107  0109  0111  0115  0138  0144  0149
                  0149
R4       0026     0077
R5       0027     0112
R6       0028     0147
R7       0029     0070  0091
R8       0030     0090  0109  0117  0140  0144  0151
R9       0031     0048  0090  0140
RESTRT   0070     0102
RULES    0194     0044
SORRY    0207     0159
START    0042     0098  0211
WINNER   0205     0123
WS       0168     0043
XOB      0192     0119  0172
XOBP     0191     0121  0156
```

THERE ARE 0041 SYMBOLS

K.2 HI-LO GAME

The printout of this game in execution (below) illustrates game rules and objectives. The program generates a number between 0 and 999. You have unlimited guesses to find the number, but you can be an expert, above average, average, or a turkey depending upon how many guesses used.

```
?L FE00
GUESS          ⎫
?R             ⎬  LOAD AND EXECUTE PROGRAM
W=FFB0         ⎪
P=0182  FE00   ⎪
?E             ⎭
```

```
CAN YOU GUESS MY NUMBER (0 TO 999)?
INPUT A NUMBER & PRESS THE SPACE BAR.
500    TOO LOW, TRY AGAIN!!
700    TOO LOW, TRY AGAIN!!
900    TOO HIGH, TRY AGAIN!
850    TOO LOW, TRY AGAIN!!
875    TOO HIGH, TRY AGAIN!
88                                       ←——————— CONTROL H PRESSED TO IGNORE ENTRY
 860    TOO HIGH, TRY AGAIN!
857    TOO HIGH, TRY AGAIN!
854    CORRECT! YOU'RE ABOVE AVERAGE  BECAUSE IT TOOK YOU 08 TRIES!


CAN YOU GUESS MY NUMBER (0 TO 999)?
INPUT A NUMBER & PRESS THE SPACE BAR.
500    TOO LOW, TRY AGAIN!!
700    TOO HIGH, TRY AGAIN!
650    TOO HIGH, TRY AGAIN!
575    CORRECT! YOU'RE AN EXPERT  BECAUSE IT TOOK YOU 04 TRIES!


CAN YOU GUESS MY NUMBER (0 TO 999)?
INPUT A NUMBER & PRESS THE SPACE BAR.
900    TOO HIGH, TRY AGAIN!
800    TOO HIGH, TRY AGAIN! ←——————— CR PRESSED TO START NEW GAME


CAN YOU GUESS MY NUMBER (0 TO 999)?
INPUT A NUMBER & PRESS THE SPACE BAR.
500    TOO HIGH, TRY AGAIN!
400    TOO HIGH, TRY AGAIN!
300    TOO HIGH, TRY AGAIN!
200    TOO HIGH, TRY AGAIN! ←——————— ESC PRESSED TO RETURN TO MONITOR

?
```

```
0001            *    *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *
0002            *   THIS GUESSING GAME CAN BE RUN ON A TM 990/1XX MICRO-
0003            *   COMPUTER WITH 432 (>1B0) WORDS OF USER AVAILABLE
0004            *   RAM MEMORY. IT IS WRITTEN TO BE LOADED AT M.A. >FE00
0005            *   AND CAN BE ASSEMBLED AT THAT ADDRESS USING THE LSLA
0006            *   OR BY LOADING THE OBJECT (COLUMN 3) AT THE MEMORY
0007            *   ADDRESSES (COLUMN 2). THE OBJECT OF THIS PROGRAM IS TO
0008            *   GUESS WHICH NUMBER THE COMPUTER HAS GENERATED, AND TO
0009            *   DO THIS WITHOUT BECOMING A TURKEY. FOLLOWING RULES APPLY:
0010            *      - CARRIAGE RETURN BRINGS YOU TO PROGRAM RESTART
0011            *      - ESCAPE KEY BRINGS YOU TO MONITOR
0012            *      - CONTROL-H KEY IGNORES THIS ENTRY
0013            *      - SPACE KEY CONTINUES GAME
0014            *   GOOD LUCK. J. WALSH
0015            *    *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *
0016                      IDT   'GUESS'
0018            * REGISTER EQUATES
0019      0000  R0        EQU   0                TENS MULTIPLIER
0020      0001  R1        EQU   1                GUESS NO. ACCUMULATOR
0021      0002  R2        EQU   2                MULTIPLY ANSWER
0022      0003  R3        EQU   3                ENTERED DIGIT
0023      0008  R8        EQU   8                CONTAINS COMPUTER'S NUMBER
0024      0009  R9        EQU   9                NO. TRIES/10
0025      000A  R10       EQU   10               NO. TRIES
0026      000C  R12       EQU   12               CRU ADDRESS   (TMS 9902 )
0027            * OBJECT CODE AT ABSOLUTE ADDRESS BEGINNING WITH >FE00
0028 FE00                 AORG  >FE00
0029            *    *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *
0030            *   PROCEDURE AREA: EXECUTABLE CODE
0031            *    *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *
0032            * INITIALIZE REGISTERS
0033 FE00 02E0  START LWPI WSP                   SET WORKSPACE POINTER
     FE02 FFA0
0034 FE04 0200        LI    R0,10                R0 = TENS MULTIPLIER
     FE06 000A
0035 FE08 04C9        CLR   R9                   R9 = NO. OF TRIES
0036 FE0A 04CA        CLR   R10                  R10 = NO. OF TRIES
0037 FE0C 020C        LI    R12,>80              TMS 9902 CRU ADDR.
     FE0E 0080
0038            * OUTPUT OPENING MESSAGE
0039 FE10 2FA0        XOP   @MESS1,14            OPENING MESSAGE
     FE12 FEB0
0040            * THIS ROUTINE IS A NUMBER GENERATOR THAT GENERATES
0041            * A NUMBER FROM 0 TO 999 BASED ON THE TIME TO RESPOND TO THE
0042            * OPENING MESSAGE. IT CHECKS A BIT AT THE TMS 9902 SERIAL
0043            * INTERFACE THAT SIGNIFIES THAT A DIGIT HAS BEEN RECEIVED FR
0044            * THE TERMINAL IN RESPONSE TO THE OPENING MESSAGE. RECEIPT O
0045            * THIS DIGIT MEANS A NUMBER IS BEING GUESSED. WHILE WAITING
0046            * FOR THIS FIRST NUMBER, R8 IS CONTINUOUSLY INCREMENTED FROM
0047            * 0 TO 999.
0048 FE14 04C8  NEWNO CLR   R8                   R8 TO CONTAIN COMPUTER'S NO.
0049 FE16 1F15  INCNO TB    21                   DIGIT RECEIVED?
0050 FE18 1307        JEQ   ECHO2                YES, ECHO CHARACTER
0051 FE1A 0288        CI    R8,999               NO. INCREMENTED TO 999?
```

```
        FE1C 03E7
0052 FE1E 13FA          JEQ    NEWNO            YES, CLEAR TO 0, RESTART
0053 FE20 0588          INC    R8               NO, INCREMENT NO. IN R8
0054 FE22 10F9          JMP    INCNO            LOOP, RECHECK FOR DIGIT INPUT
0055             * AFTER FIRST DIGIT IS ENTERED, COMPUTER'S NO. IS IN R8.
0056             * READ IN GUESSES AND CONVERT THESE TO HEXADECIMAL. SUM
0057             * FOR COMPARISON TO COMPUTER'S NO. IN R8. AS NEW NUMBER
0058             * IS READ, OLD VALUE IS MULTIPLIED BY 10 AND NEW VALUE
0059             * ADDED TO PRODUCT TO KEEP CUMULATIVE TOTAL OF DIGITS
0060             * ENTERED.
0061 FE24 2F20   ECHO0  XOP    @LFCR,12         DO LINE-FEED, CR
        FE26 FF34
0062 FE28 04C1   ECHO2  CLR    R1               CLEAR ACCUMMULATOR
0063 FE2A 2EC3   ECHO1  XOP    R3,11            ECHO CHAR., PLACE IT IN R3
0064 FE2C 06C3          SWPB   R3               PLACE VALUE IN RIGHT BYTE
0065             * WAS SPACE, CR, ESCAPE OR CONTROL-H PRESSED?
0066 FE2E 0283          CI     R3,>0020         SPACE BAR PRESSED?
        FE30 0020
0067 FE32 1311          JEQ    COMPRE           YES, COMPARE VALUES
0068 FE34 0283          CI     R3,>000D         CARRIAGE RET. PRESSED?
        FE36 000D
0069 FE38 13E3          JEQ    START            YES, RESTART PROGRAM
0070 FE3A 0283          CI     R3,>001B         ESCAPE PRESSED?
        FE3C 001B
0071 FE3E 1309          JEQ    MONITR           YES, RETURN TO MONITOR
0072 FE40 0283          CI     R3,>0008         WAS CONTROL-H PRESSED?
        FE42 0008
0073 FE44 13EF          JEQ    ECHO0            DO LFCR, RESTART GUESS
0074 FE46 0243          ANDI   R3,>000F         NO, SAVE 0-9 DIGIT ONLY
        FE48 000F
0075 FE4A 3840          MPY    R0,R1            PREVIOUS NO. X10
0076 FE4C A0C2          A      R2,R3            NEW NO. + ABOVE PRODUCT
0077 FE4E C043          MOV    R3,R1            ANSWR TO ACCUMMULATOR
0078 FE50 10EC          JMP    ECHO1            GET NEXT DIGIT
0079 FE52 0460   MONITR B      @>0080           GO TO MONITOR
        FE54 0080
0080             * COMPARE NUMBERS INPUT TO COMPUTER'S NUMBER
0081 FE56 058A   COMPRE INC    R10              INCREMENT NOS. GUESSED
0082 FE58 8201          C      R1,R8            COMPARE TO COMPUTER'S NO.
0083 FE5A 1102          JLT    LOW              NO. IS LESS THAN COMPUTER'S
0084 FE5C 1504          JGT    HIGH             NO. IS MORE THAN COMPUTER'S
0085 FE5E 1306          JEQ    EQUAL            NO. IS CORRECT VALUE
0086             * MESAGES FOR TOO HIGH, TOO LOW
0087 FE60 2FA0   LOW    XOP    @LOWM,14         TOO-LOW MESSAGE
        FE62 FF00
0088 FE64 10E1          JMP    ECHO2            GET NEXT NUMBER
0089 FE66 2FA0   HIGH   XOP    @HIGHM,14        TOO-HIGH MESSAGE
        FE68 FF1A
0090 FE6A 10DE          JMP    ECHO2            GET NEXT NUMBER
```

```
0092                     * CORRECT NUMBER WAS GUESSED
0093                     * FIND OUT HOW MANY TRIES WAS USED AND OUTPUT MESSAGE
0094 FE6C 2FA0   EQUAL   XOP   @CORECT,14      CORRECT GUESS MESSAGE
     FE6E FF38
0095 FE70 028A           CI    R10,7           TRY COUNT GREATER THAN 7?
     FE72 0007
0096 FE74 1503           JGT   $+8             YES, CHECK AGAIN
0097 FE76 2FA0           XOP   @SEVEN,14       NO, DO 0-7 TRIES MESSAGE
     FE78 FF4F
0098 FE7A 100E           JMP   COUNT           GO GET COUNT
0099 FE7C 028A           CI    R10,9           TRY-COUNT GREATER THAN 9?
     FE7E 0009
0100 FE80 1503           JGT   $+8             YES, CHECK AGAIN
0101 FE82 2FA0           XOP   @NINE,14        NO, DO 8-9 TRIES MESSAGE
     FE84 FF5A
0102 FE86 1008           JMP   COUNT           GO GET COUNT
0103 FE88 028A           CI    R10,13          TRY-COUNTER GREATER THAN 13?
     FE8A 000D
0104 FE8C 1503           JGT   $+8             YES, OUTPUT TURKEY MESSAGE
0105 FE8E 2FA0           XOP   @THIRTN,14      NO, DO 10-13 TRIES MESSAGE
     FE90 FF69
0106 FE92 1002           JMP   COUNT           GO GET COUNT
0107 FE94 2FA0           XOP   @TURKEY,14      OUTPUT >13 (TURKEY) MESSAGE
     FE96 FF72
0108                     * IF CORRECT NUMBER FOUND, OUTPUT NO. OF TRIES
0109 FE98 3E40   COUNT   DIV   R0,R9           DIVIDE TRY-NO. BY 10
0110 FE9A 0269           ORI   R9,>0030        OR IN >30 FOR ASCII NO.
     FE9C 0030
0111 FE9E 026A           ORI   R10,>0030       OR IN >30 FOR ASCII NO.
     FEA0 0030
0112 FEA2 06C9           SWPB  R9              REMAINDER IN LEFT BYTE
0113 FEA4 A289           A     R9,R10          2-DIGIT DECIMAL IN R10
0114 FEA6 C80A           MOV   R10,@NUMBR      MOVE QTY TO MESSAGE
     FEA8 FF92
0115 FEAA 2FA0           XOP   @CNT,14         OUTPUT NO. OF TRIES
     FEAC FF7D
0116 FEAE 10A8           JMP   START           GO TO BEGINNING OF PROGRAM
```

```
0118                    *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
0119                    *  DATA AREA: DATA STATEMENTS, TEXT STATEMENTS  ETC.
0120                    *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
0121                    * MESSAGES
0122 FEB0 0A0D   MESS1  DATA >0A0D,>0A0A
     FEB2 0A0A
0123 FEB4   43                  TEXT 'CAN YOU GUESS MY NUMBER (0 TO 999)? '
     FEB5   41
     FEB6   4E
     FEB7   20
     FEB8   59
     FEB9   4F
     FEBA   55
     FEBB   20
     FEBC   47
     FEBD   55
     FEBE   45
     FEBF   53
     FEC0   53
     FEC1   20
     FEC2   4D
     FEC3   59
     FEC4   20
     FEC5   4E
     FEC6   55
     FEC7   4D
     FEC8   42
     FEC9   45
     FECA   52
     FECB   20
     FECC   28
     FECD   30
     FECE   20
     FECF   54
     FED0   4F
     FED1   20
     FED2   39
     FED3   39
     FED4   39
     FED5   29
     FED6   3F
     FED7   20
0124 FED8 0A0D          DATA >0A0D              LINE FEED, CR
0125 FEDA   49          TEXT 'INPUT A NUMBER & PRESS THE SPACE BAR. '
     FEDB   4E
     FEDC   50
     FEDD   55
     FEDE   54
     FEDF   20
     FEE0   41
     FEE1   20
     FEE2   4E
     FEE3   55
     FEE4   4D
```

```
            FEE5    42
            FEE6    45
            FEE7    52
            FEE8    20
            FEE9    26
            FEEA    20
            FEEB    50
            FEEC    52
            FEED    45
            FEEE    53
            FEEF    53
            FEF0    20
            FEF1    54
            FEF2    48
            FEF3    45
            FEF4    20
            FEF5    53
            FEF6    50
            FEF7    41
            FEF8    43
            FEF9    45
            FEFA    20
            FEFB    42
            FEFC    41
            FEFD    52
            FEFE    2E
            FEFF    20
0126 FF00  2020   LOWM    DATA  >2020               DOUBLE SPACE
0127 FF02  54             TEXT  'TOO LOW, TRY AGAIN!!'
     FF03  4F
     FF04  4F
     FF05  20
     FF06  4C
     FF07  4F
     FF08  37
     FF09  2C
     FF0A  20
     FF0B  54
     FF0C  52
     FF0D  59
     FF0E  20
     FF0F  41
     FF10  47
     FF11  41
     FF12  49
     FF13  4E
     FF14  21
     FF15  21
0128 FF16  0A0D           DATA  >0A0D,0              LINE FEED, CR, END MSG
     FF18  0000
0129 FF1A  2020   HIGHM   DATA  >2020               TWO SPACES
0130 FF1C  54             TEXT  'TOO HIGH, TRY AGAIN!'
     FF1D  4F
     FF1E  4F
```

```
              FF1F    20
              FF20    48
              FF21    49
              FF22    47
              FF23    48
              FF24    2C
              FF25    20
              FF26    54
              FF27    52
              FF28    59
              FF29    20
              FF2A    41
              FF2B    47
              FF2C    41
              FF2D    49
              FF2E    4E
              FF2F    21
0131 FF30    0A0D           DATA >0A0D,0        LINE FEED, CR, END MSG
     FF32    0000
0132 FF34    0A0D    LFCR   DATA >0A0D          LINE FEED, CR
0133 FF36    00             BYTE 0              END OF MESSAGE
0134 FF38    0707    CORECT DATA >0707,>0707    BELLS
     FF3A    0707
0135 FF3C    2020           DATA >2020          SPACES
0136 FF3E    43             TEXT 'CORRECT! YOU''RE '
     FF3F    4F
     FF40    52
     FF41    52
     FF42    45
     FF43    43
     FF44    54
     FF45    21
     FF46    20
     FF47    59
     FF48    4F
     FF49    55
     FF4A    27
     FF4B    52
     FF4C    45
     FF4D    20
0137 FF4E    00             BYTE 0              END OF MESSAGE
0138 FF4F    41     SEVEN   TEXT 'AN EXPERT '
     FF50    4E
     FF51    20
     FF52    45
     FF53    58
     FF54    50
     FF55    45
     FF56    52
     FF57    54
     FF58    20
0139 FF59    00             BYTE 0
0140 FF5A    41     NINE    TEXT 'ABOVE AVERAGE
     FF5B    42
```

```
                FF5C     4F
                FF5D     56
                FF5E     45
                FF5F     20
                FF60     41
                FF61     56
                FF62     45
                FF63     52
                FF64     41
                FF65     47
                FF66     45
                FF67     20
     0141       FF68     00            BYTE 0
     0142       FF69     41    THIRTN   TEXT  'AVERAGE '
                FF6A     56
                FF6B     45
                FF6C     52
                FF6D     41
                FF6E     47
                FF6F     45
                FF70     20
     0143       FF71     00            BYTE 0
     0144       FF72     41    TURKEY   TEXT  'A TURKEY '
                FF73     20
                FF74     54
                FF75     55
                FF76     52
                FF77     4B
                FF78     45
                FF79     59
                FF7A     20
                FF7B     20
     0145       FF7C     00            BYTE 0
     0146       FF7D     20    CNT      TEXT ' BECAUSE IT TOOK YOU '
                FF7E     42
                FF7F     45
                FF80     43
                FF81     41
                FF82     55
                FF83     53
                FF84     45
                FF85     20
                FF86     49
                FF87     54
                FF88     20
                FF89     54
                FF8A     4F
                FF8B     4F
                FF8C     4B
                FF8D     20
                FF8E     59
                FF8F     4F
                FF90     55
                FF91     20
```

HI-LO GAME FOR TM 990/1XX MICROCOMPUTERS

```
0147 FF92 0000   NUMBR   DATA 0                    PLACE ASCII NO. HERE
0148 FF94 20             BYTE >20
0149 FF95 54             TEXT 'TRIES!'
     FF96 52
     FF97 49
     FF98 45
     FF99 53
     FF9A 21
0150 FF9B 07             BYTE 7,7,7,0              BELLS (ASCII 07)
     FF9C 07
     FF9D 07
     FF9E 00
0151            WSP      EVEN                      WORKSPACE START (R0 LOC)
0152            END
```

0000 ERRORS

```
CNT      0146       0115
COMPRE   0081       0067
CORECT   0134       0094
COUNT    0109       0098  0102  0106
ECHO0    0061       0073
ECHO1    0063       0078
ECHO2    0062       0050  0088  0090
EQUAL    0094       0085
HIGH     0089       0084
HIGHM    0129       0089
INCNO    0049       0054
LFCR     0132       0061
LOW      0087       0083
LOWM     0126       0087
MESS1    0122       0039
MONITR   0079       0071
NEWNO    0048       0052
NINE     0140       0101
NUMBR    0147       0114
R0       0019       0034  0075  0109
R1       0020       0062  0075  0077  0082
R10      0025       0036  0081  0095  0099  0103  0111  0113  0114
R12      0026       0037
R2       0021       0076
R3       0022       0063  0064  0066  0068  0070  0072  0074  0076  0077
R8       0023       0048  0051  0053  0082
R9       0024       0035  0109  0110  0112  0113
SEVEN    0138       0097
START    0033       0069  0116
THIRTN   0142       0105
TURKEY   0144       0107
WSP      0151       0033
```

THERE ARE 0032 SYMBOLS

INDEX

TM 990/101M MICROCOMPUTER
USER RESPONSE SHEET

It is our desire to provide our customers with the best documentation possible. After using this manual, please complete this sheet and mail it, postpaid, to us. Your comments will be given every consideration.

1. Is the manual well organized? Yes _____ No _____ Comments: _____

2. Is text clearly presented and adequately illustrated? Yes _____ No _____

Comments: _____

3. What subject matter could be expanded or clarified? _____

4. Is the instruction set adequately covered? Yes _____ No _____

Comments: _____

5. Do you wish more data that would clarify an instruction? Yes _____ No _____

Comments: _____

6. Do you wish more data to clarify an application? Yes _____ No _____

Comments: _____

7. Please explain the application intended for your board:

School Course _____ Home _____ Evaluation _____ OEM Application _____ Other _____

If OEM Application, please describe: _____

_____

8. Other comments concerning the TM 990/101M and this manual: _____

_____

_____

_____

_____

Name: _____

Address _____ State _____ ZIP _____

School (if applicable) _____ Major _____ Year _____

MP337 REV. D
1602001-9701

Printed in U.S.A.