As you are now the owner of this document which should have come to you for free, please consider making a donation of £1 or more for the upkeep of the (Radar) website which holds this document. I give my time for free, but it costs me money to bring this document to you. You can donate here https://blunham.com/Misc/Texas

Many thanks.

Please do not upload this copyright pdf document to any other website. Breach of copyright may result in a criminal conviction.

This Acrobat document was generated by me, Colin Hinson, from a document held by me. I requested permission to publish this from Texas Instruments (twice) but received no reply. It is presented here (for free) and this pdf version of the document is my copyright in much the same way as a photograph would be. If you believe the document to be under other copyright, please contact me.

The document should have been downloaded from my website https://blunham.com/, or any mirror site named on that site. If you downloaded it from elsewhere, please let me know (particularly if you were charged for it). You can contact me via my Genuki email page: https://www.genuki.org.uk/big/eng/YKS/various?recipient=colin

You may not copy the file for onward transmission of the data nor attempt to make monetary gain by the use of these files. If you want someone else to have a copy of the file, point them at the website. (https://blunham.com/Misc/Texas). **Please do not point them at the file itself as it may move or the site may be updated.**

It should be noted that most of the pages are identifiable as having been processed by me.

I put a lot of time into producing these files which is why you are met with this page when you open the file.

If you find missing pages, pages in the wrong order, anything else wrong with the file or simply want to make a comment, please drop me a line (see above).

It is my hope that you find the file of use to you.

Colin Hinson
In the village of Blunham, Bedfordshire.

TEXAS INSTRUMENTS

TM 990

TM 990/1481
High—Performance CPU Modules

Volume I

MICROPROCESSOR SERIES™

October 1980

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

TABLE OF CONTENTS

LIST OF TABLES

SECTION 1

INTRODUCTION

1.1 GENERAL

The TM 990/1481 is a high speed general purpose central processing unit implemented on two multilayer printed circuit boards. Utilizing Schottky and low-power Schottky TTL logic, including the 54/74S481 LSI Processor Bit-Slice, it offers a performance improvement of up to 39 X over the TM 990/101MA single board microcomputer. Appendix I contains benchmark data. Because the TM 990/1481 processor generates a bus clock of up to 5 MHz, it will interface only to TM 990 modules designed for 5 MHz or higher operation. A typical system configuration is shown in Figure 1-1. Figures 1-2 and 1-3 show the principal components of the TM 990/1481 processor and controller boards, respectively; Figure 1-4 gives the dimensions for the processor and controller boards; and Figure 1-5 is a block diagram of a TM 990/1481 system.

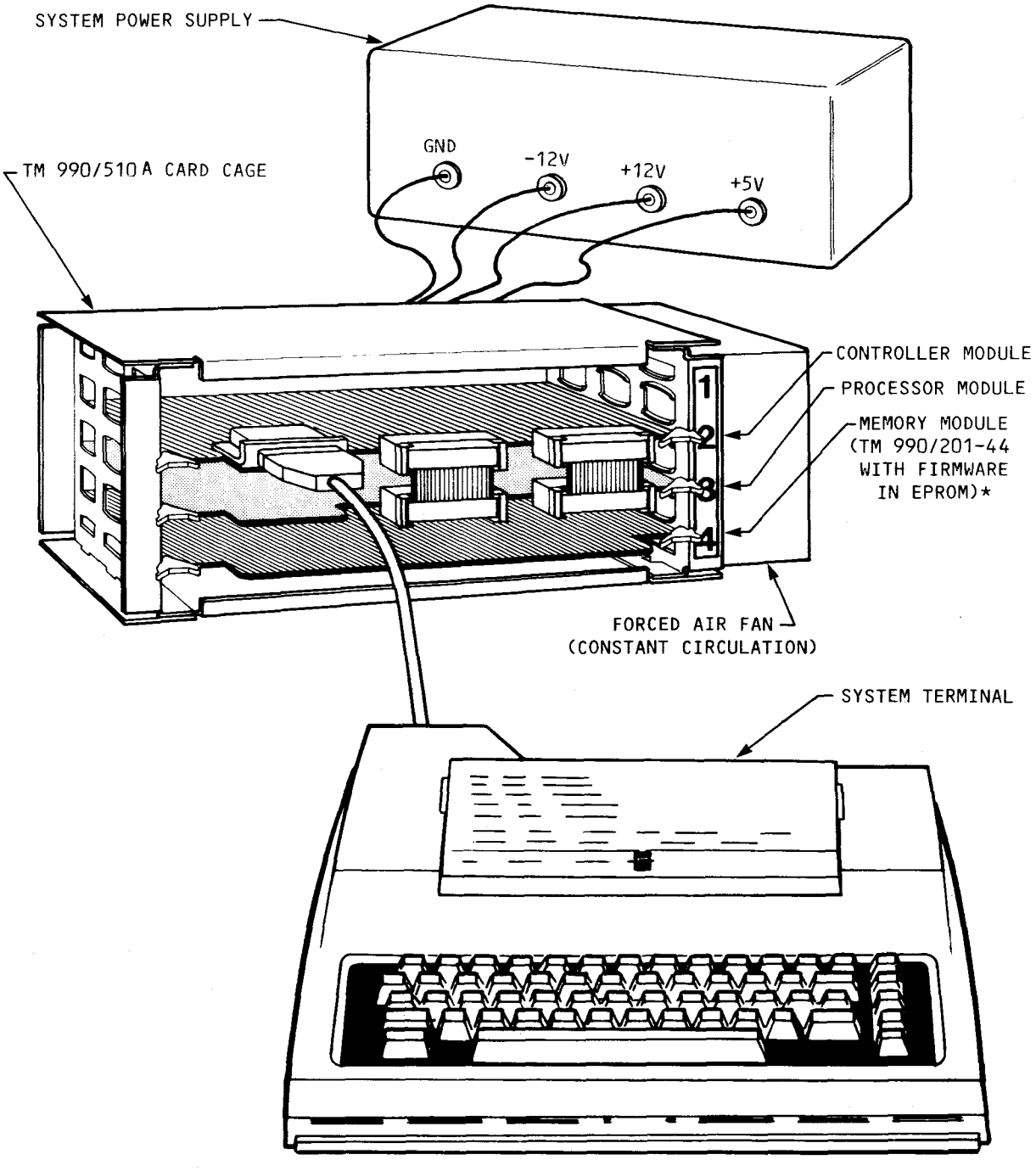
Some important features of the TM 990/1481 include:

- Software compatible with the 990 family of computers
- Incorporates floating point arithmetic instructions, signed multiply and divide, and single register LST and LWP instructions (floating point numbers are truncated rather than rounded to be compatable with other Texas Instruments computers)
- Meets TM 990 bus specification requirements for 5 MHz bus clock operation
- Provides 15 levels of prioritized and maskable interrupts
- Provides 2 programmable interval timers
- Interfaces to RS-232 terminals such as the TI Microterminal or Silent 700 terminals
- Provides special ARITHMETIC OVERFLOW interrupt
- Use of instruction "look ahead," a variable clock period, and a high degree of parallelism in the architecture produces up to 39 X speed improvement over the TM 990/101MA (See benchmarks in Appendix I).

1.2 MANUAL ORGANIZATION

This manual is organized as follows:

- Section 1 covers TM 990/1481 characteristics and specifications.
- Section 2 shows how to install, powerup, and operate the TM 990/1481.
- Section 3 explains the TM 990/403 TIBUG monitor. TIBUG commands, XOPs, and error indicators are topics that are included.



* Note, if a TM 990/203 dynamic RAM module is used, software could be loaded from cassette using an ASR terminal such as the TI 733 ASR.

FIGURE 1-1. TYPICAL SYSTEM CONFUGURATION WITH TM 990/1481

- Section 4 covers the TM 990/1481 instruction set. Instruction formats, addressing modes, instruction execution times, user-defined instructions, use of floating point instructions, programming aids, and interrupts are described in this section.

- Section 5 covers the software applications support for the TM 990/1481.

- Section 6 covers I/O programming using the communication register unit (CRU).

- Section 7 covers the theory of operation with circuit descriptions keyed to schematic diagrams.

- Section 8 describes the microinstruction word format used in the TM 990/1481

- Section 9 provides a description of the TM 990/1481's three interfaces including: (1) TM 990 bus interface, (2) processor/controller interface, and (3) terminal interface.

1.3 GENERAL SPECIFICATIONS

- System Power Requirements:

		Current (Amps)	
Voltage	Regulation	Typ	Max
+5V	±3%	9.00	12.00
+12V	±3%	.02	.03
-12V	±3%	.02	.03

- Operating Temperature: 0°C to 70°C ambient at the board

- Module Dimensions: The processor and controller dimensions are given in Figure 1-4.

1.4 APPLICABLE DOCUMENTS

The following is a list of documents that provide supplementary information for the TM 990/1481 user.

- SN74S481, SN54LS/74LS481 4-Bit-Slice Schottky Processor Elements Data Manual

- TMS 9901 Programmable Systems Interface Data Manual

- TMS 9902 Asynchronous Communication Controller Data Manual

- Model 990 Computer, TMS 9900 Microprocessor Assembly Language Programmer's Guide (P/N 943441-9701)

- Model 990/12 Computer Assembly Language Programmer's Guide (P/N 2250077-9701 *A)

- TM 990/201 and TM 990/206 Expansion Memory Boards Data Manual (includes TM 990/201-44)

- TM 990/203 Dynamic RAM Memory Expansion Module.

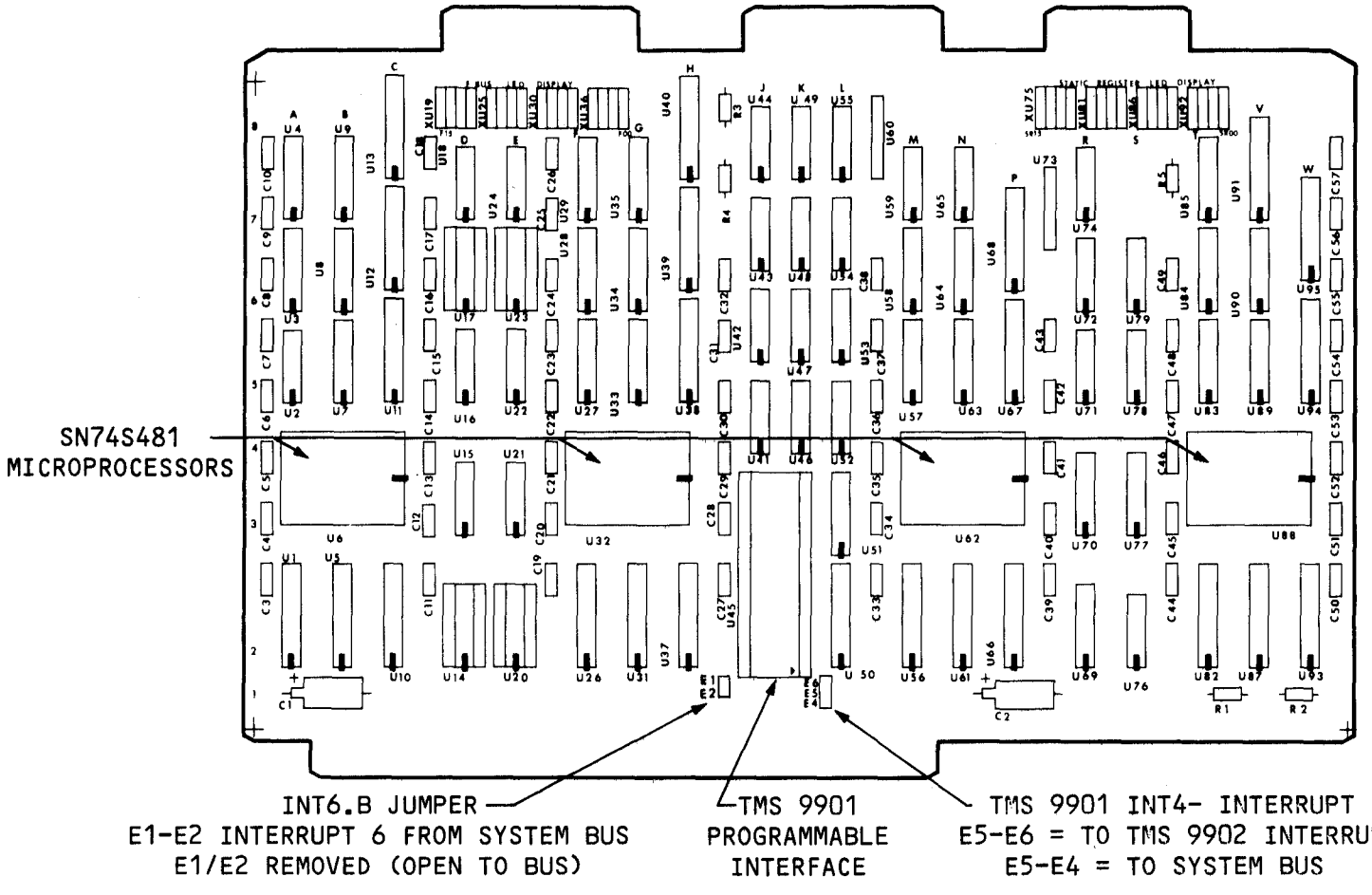


FIGURE 1-2. TM 990/1481 PRINCIPAL PROCESSOR COMPONENTS

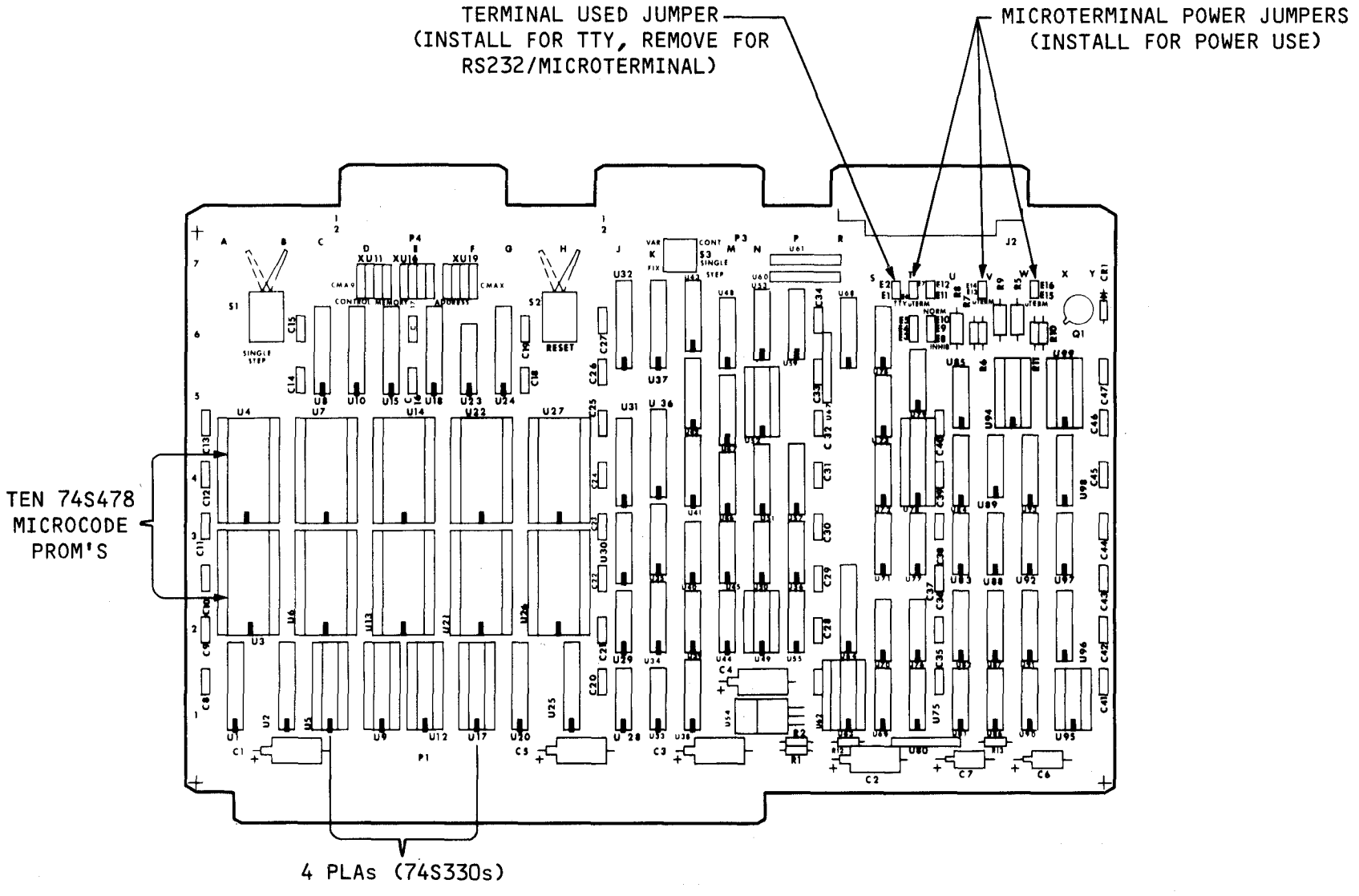


FIGURE 1-3. TM 990/1481 PRINCIPAL CONTROLLER COMPONENTS

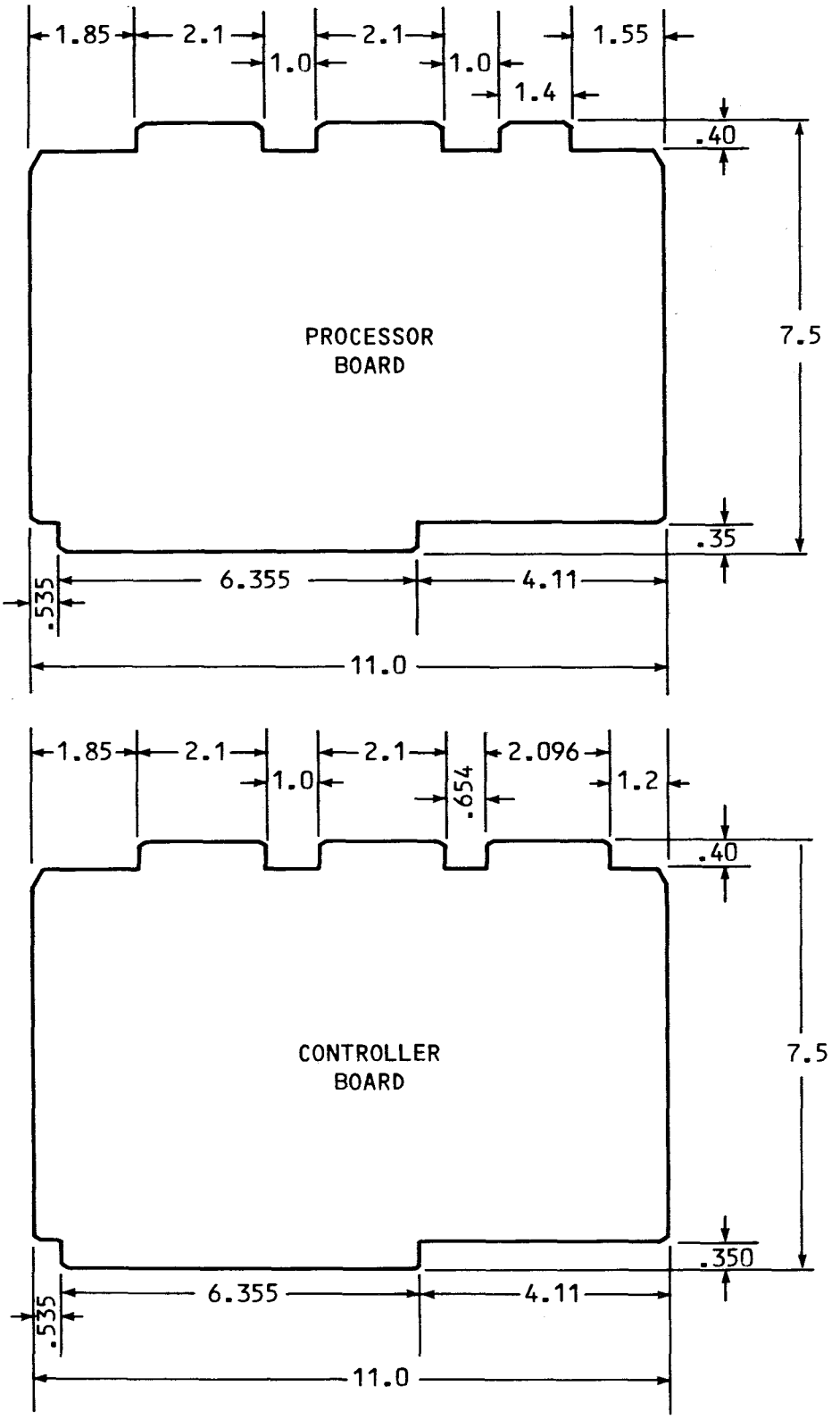


FIGURE 1-4. TM 990/1481 DIMENSIONS (IN INCHES)

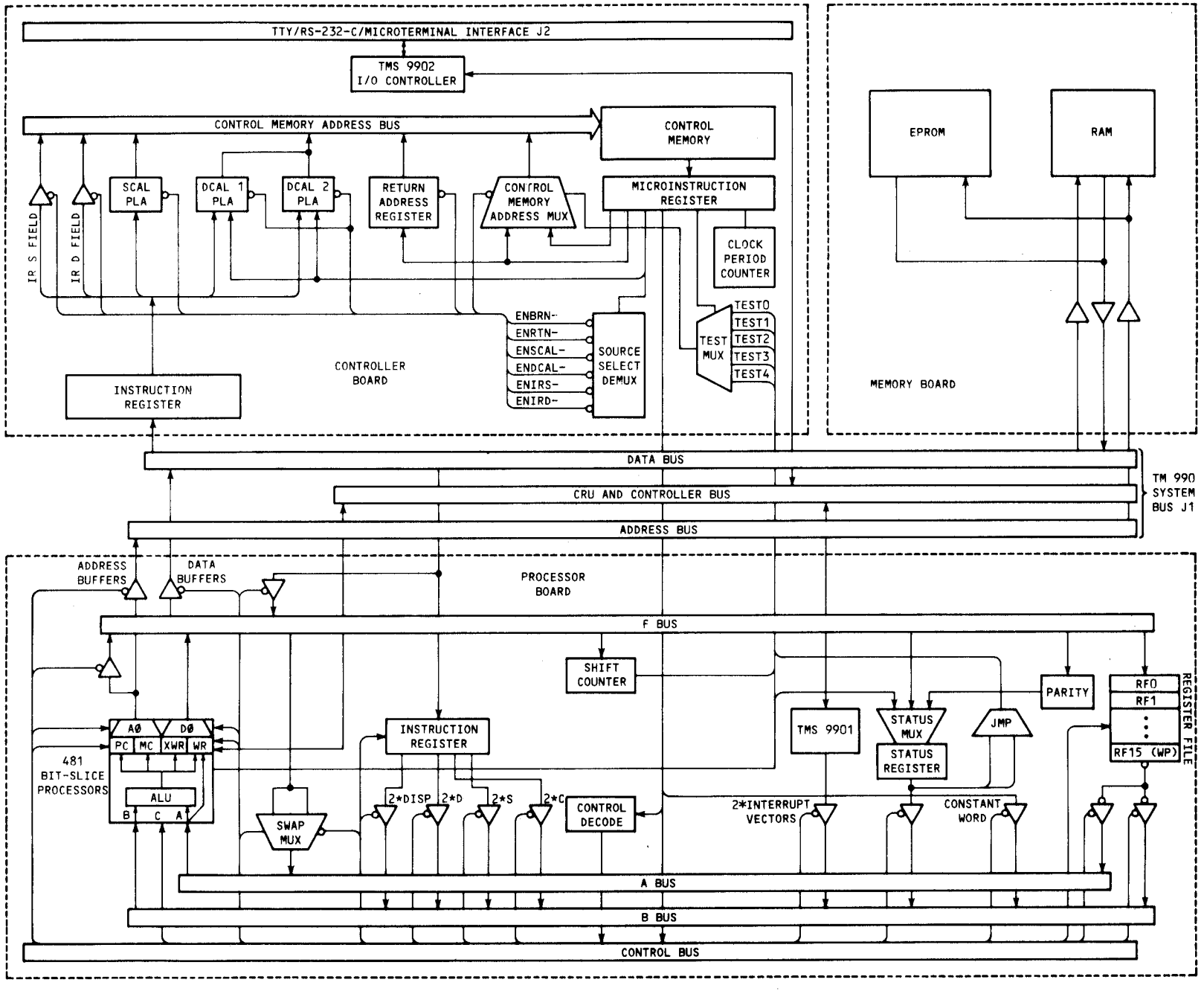


FIGURE 1-5. TM 9[illegible]1481 SYSTEM DIAGRAM

SECTION 2

INSTALLATION AND OPERATION

2.1 GENERAL

The procedures for unpacking and setting up the TM 990/1481 for operation are given in this section along with a test routine that can be used to check out the system.

2.2 UNPACKING AND INSPECTION

Remove the TM 990/1481 processor and controller boards and cables from their cartons and discard any protective wrapping.

Inspect both boards for any damage that could have occurred in shipping. Report any damage to your TI supplier.

2.3 PROCESSOR MODULE OPTIONS

The TM 990/1481 processor module provides jumper selection for the following functions:

- External interrupt line (INT6.B-)
- TMS 9901 INT4- interrupt input option

2.3.1 External Interrupt Line (INT6.B-) Select (E1-E2). This option is for factory use. Installing a jumper at E1-E2 connects INT6- at the TMS 9901 to the system bus, signal INT6.B- at P1-20. Removing this jumper opens this line.

2.3.2 TMS 9901 INT4- Interrupt Option (E4-E6). A jumper allows the user to connect the INT4- input to the TMS 9901 from either the interrupt output (TINT-) of the TMS 9902 on the Controller Module or from INT4.B- of the system bus at pin P1-18.

- E5-E4: TMS 9901 INT4- input connects to system bus at P1-18.
- E5-E6: TMS 9901 INT4- input connects to TINT- from TMS 9902.
- Unjumpered: No input to TMS 9901 INT4-

NOTE

The TMS 9901 and TMS 9902, both of which contain programmable clocks, are addressable through the CRU. The TMS 9902 is located at software base address 0080_{16} and the TMS 9901 is located at software base address 0100_{16}. A CRU map is shown in Appendix B.

2.4 CONTROLLER MODULE OPTIONS

The TM 990/1481 controller module provides options via:

- Three jumper settings
  - TTY or RS232C terminal select
  - Microterminal voltages
  - Map mode memory timing
- Memory configuration using wiring platform

2.4.1 Controller Module Jumpers

There are three jumper selectable options on the controller module: the terminal select option, the TM 990/301 microterminal supply voltage option, and the map mode memory timing option.

2.4.1.1 Terminal Select Option (E1-E2). The terminal select option allows the user to configure the TM 990/1481 so that either a TTY or RS-232-C/TM 990/301 terminal can be used. This jumper should be connected between E1 and E2 for operation with a TTY terminal; it should be removed for operation with either an RS-232-C terminal or a TM 990/301 microterminal.

2.4.1.2 TM 990/301 Microterminal Supply Voltage Option (E6-E7, E13-E16). Jumpers can be used to provide +12V, +5V, and -5V for the TM 990/301 microterminal. Install jumpers between:

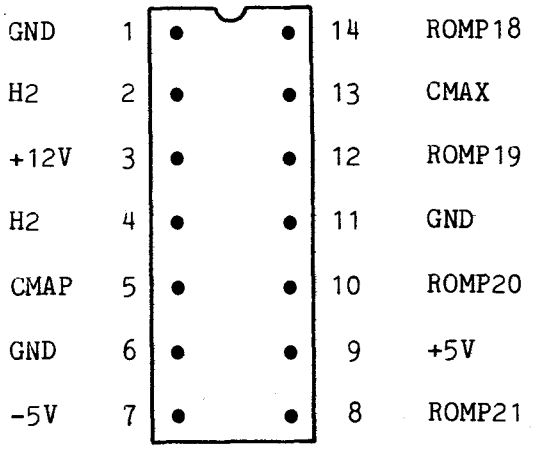
- E6 and E7
- E13 and E14
- E15 and E16

Installing these jumpers will couple the necessary supply voltage to the microterminal. These jumpers should be removed if a terminal other than the microterminal is used.

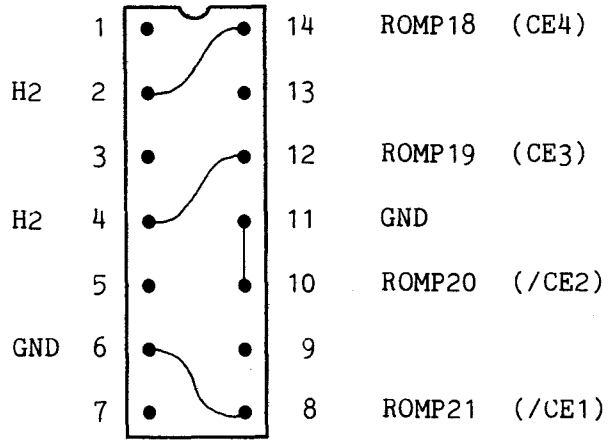
2.4.1.3 Map Mode Memory Timing Option (E11-E12). The normal position for this jumper is from E12 to E11. Removal of this jumper is currently reserved for future use.

2.4.2 ROM Wiring Platform at U49

The ROM wiring platform is wired for operation with the SN74S478N Schottky Bipolar PROM that is supplied with the TM 990/1481 controller board. Figure 2-1 shows the wiring of this platform as shipped from the factory.



(a) Pin nomenclature for ROM wiring platform at U49

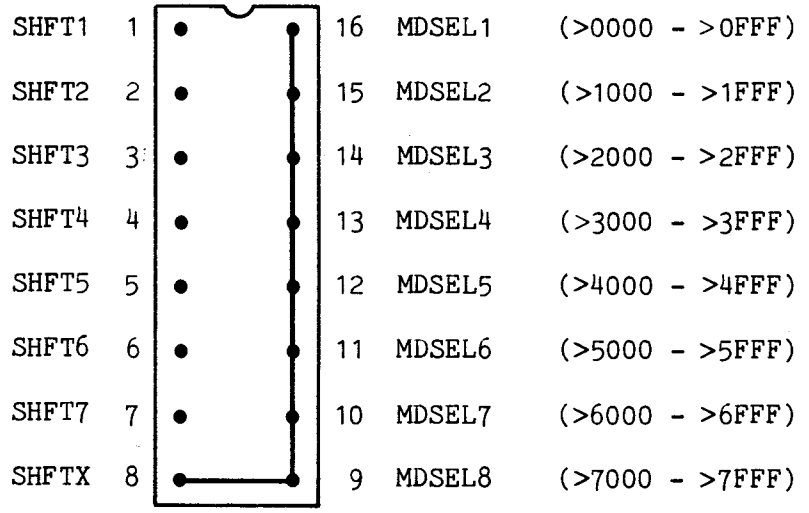


(b) Wiring of U49 for ROM SN74S478N 1K x 8 Schottky bipolar PROM

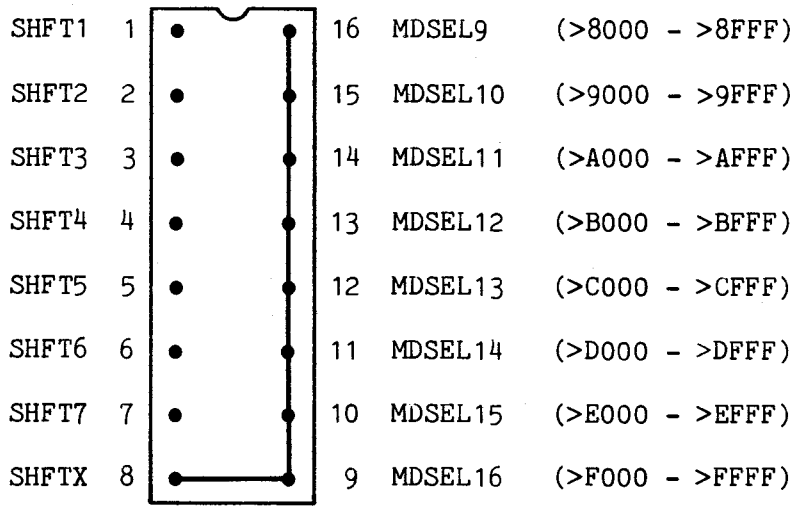
FIGURE 2-1. ROM ADAPTER PLUG WIRING

2.4.3 Controller Module Wiring Platforms for Memory Devices

There are two memory plugs located on the controller board which selects the proper waits states required for different memories and memory modules. These plugs are configured (wired) depending on the type of memory device and memory module used. Figure 2-2 identifies the various pin definitions of the plugs as well as how the plugs are wired at the factory. Plug 1 at U94 is used to program the first eight blocks of memory and plug 2 at U99 is used to program the second eight blocks of memory. MDSEL1 (pin 16 of plug 1) identifies the first 1K block of memory and MDSEL16 (pin 9 of plug 2) identifies the block of memory from $F000_{16}$ through $FFFF_{16}$. SHFTN (N = 1 to 7 or X) selects the number



(a) Plug 1 (U94/W5A) Top View



(b) Plug 2 (U99/X5A) Top View

NOTE: Wiring is as shipped from the factory

FIGURE 2-2. MEMORY DEVICE WIRING PLATFORMS AT U49 AND U99

of delays per access required for a particular board and memory device as shown in Table 2-1. A jumper wire is connected from the proper SHFTN pin to the applicable memory pin(s). Memory speed delay logic is further explained in detail in section 7.4.6.

TABLE 2-1. MEMORY PLUG WIRING

MEMORY BOARD	MEMORY DEVICE	MEMORY TYPE	PLUG WIRING	
			FROM	TO
TM 990/H201-44	TMS 2716	EPROM	SHFT7	See Note
	TMS 4045-15	RAM	SHFT3	See Note
	TMS 4045-20	RAM	SHFT4	See Note
	TMS 4045-30	RAM	SHFT5	See Note
	TMS 4045-45	RAM	SHFT7	See Note
TM 990/H203-13	TMS 4115	RAM	SHFTX	See Note

NOTE: As shown in Figure 2-2, connect the SHFTN pin of platforms U94 and U99 to to MDSELN pins on the opposite side of the platform that reflect the memory configuration. Note in the figure that as shipped from the factory, SHFTX is wired to all sixteen 4K byte memory combinations that make up the entire 32 K word addressing.

2.4.4 Controller Module DIP and Toggle Switches

2.4.4.1 RESET Toggle Switch. The RESET switch can be toggled by the user from the front edge of the controller module. When toggled, the RESET switch produces the RESET- signal which forces the TM 990/1481 to perform a context switch to the WP and PC vectors at interrupt trap zero (vectors at 0000_{16} and 0002_{16}). This action also causes IORST- on the system bus (via pin J1-88) to be active a minimum of two REFLCK periods.

2.4.4.2 Fixed Period (Slow Clock) Mode DIP Switch (S3, VAR/FIX). One half of switch S3 is the fixed period mode switch. In the FIX position, the user selects a fixed period of 666.6 nanoseconds for the microinstruction clock cycle that is independent of the clock control field. In the VAR position, the user selects a variable period (200 ns - 666.6 ns) high-speed clock with period controlled by the clock control field of the microinstruction. The normal position of the VAR/FIX DIP switch is the VAR position.

2.4.4.3 SINGLE STEP Mode DIP Switch (S3, CONT/SINGLE STEP). The single step mode switch is the second half of switch S3. In the SINGLE STEP position, the microinstruction clock is turned off allowing single-step through microcode using the SINGLE STEP toggle switch. For normal operation, the CONT/SINGLE STEP switch should be in the CONT position.

2.4.4.4 SINGLE STEP Toggle Switch. The SINGLE STEP toggle switch is accessible at the front edge of the Controller Module; however, it is used for factory test purposes only. When used for test, LEDs (e.g., Dialight 547-2007) are populated in sockets XU11, XU16, and XU19 on the controller module and in sockets XU19, XU25, XU36, XU75, XU81, XU86, and XU92 on the processor module. These lights show test status.

NOTE

The SINGLE STEP toggle switch is for factory test purposes only. When toggled, the TM 990/1481 executes one micro-instruction step, if in the single step mode.

2.5 REQUIRED EQUIPMENT

The following items are required for a system using the TM 990/1481:

1) TM 990/510A, TM 990/520A, or TM 990/530 card cage

2) DC power supply capable of meeting the power requirements given in section 1.3

3) Suitable terminal (and cable assembly) such as either a TI Silent 700 Terminal or a TM 990/301 Microterminal.

4) Memory module such as the TM 990/201-44 or TM 990/203

5) Adequate fan forced cooling (NOTE: The TM 990/1481 dissipates 45 watts, typical. Appendix E of the TM 990/530 manual covers criteria to determine cooling requirements.)

2.6 POWER SUPPLY AND CARD CAGE CONNECTIONS

Figure 2-3 shows the necessary connections between a suitable dc power supply and a TM 990/510A card cage. Either a TM 990/520A or a TM 990/530 card cage could be used in lieu of the TM 990/510A. Power requirements are listed in section 1.3.

2.7 TM 990/1481 PROCESSOR/CONTROLLER PLACEMENT AND INTERCONNECTIONS

Adequate ventilation is a necessity for the TM 990/1481. If possible, the card cage should be placed in the vertical plane. Fan forced cooling will probably be mandatory to maintain the ambient air temperature at less than or equal to 70 degrees centigrade at the hottest point above the boards.

The 40-pin edge connector cables are provided for interconnections between the processor and controller modules. Figure 2-4 shows the proper positions for these connectors (processor J3 to controller J3 and processor J4 to controller J4).

The J2 connector on the controller allows connection to a TTY or any RS-232-C device such as the TI Silent 700 Terminal or the TM 990/301 Microterminal.

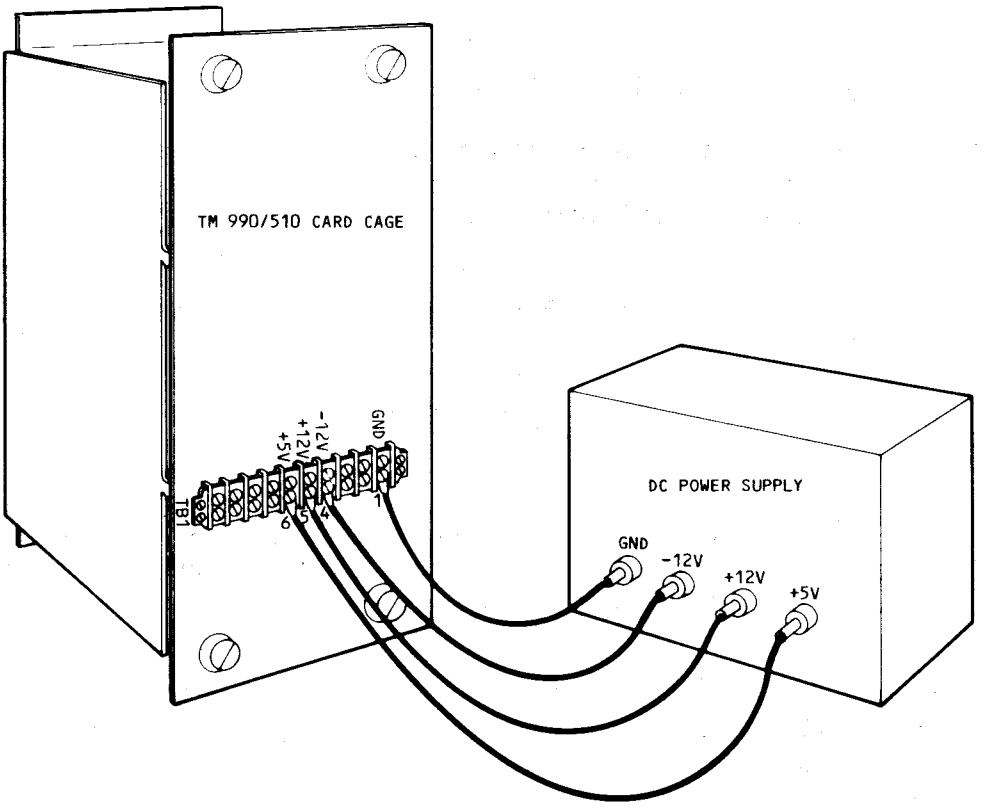


FIGURE 2-3. POWER SUPPLY CONNECTIONS

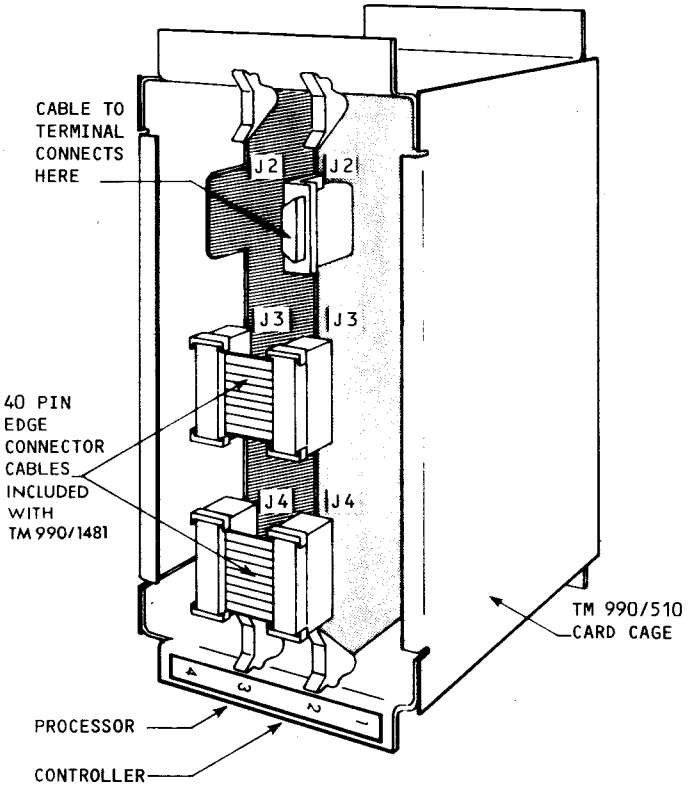


FIGURE 2-4. TM 990/1481 CONTROLLER/PROCESSOR INTERCONNECTIONS

There are several cable assemblies that can be used to provide the necessary interface between various terminals and the TM 990/1481. Table 2-2 provides a listing of these cables.

TABLE 2-2. CABLE ASSEMBLIES

MODEL NUMBER	DESCRIPTION
TM 990/502	Connects TM 990/1481 to an RS-232-C terminal except for those mentioned below
TM 990/503A	Connects TM 990/1481 to a TI 743 or 745
TM 990/504A	Connects TM 990/1481 to a Model ASR 33 teletype modified for 20 mA current loop operation
TM 990/505	Connects TM 990/1481 to a TI 733 ASR data terminal

A cable can be fabricated using a 25-pin RS-232-C plug, type DB25P, and a suitable plug for the terminal. Figure 2-5 shows the necessary connections between the TM 990/1481 and a RS-232-C terminal. Figure 2-6 shows the connections between a TM 990/1481 and a TI 743 or 745. Figure 2-7 shows the connections between a TM 990/1481 and a TI 733 ASR data terminal.

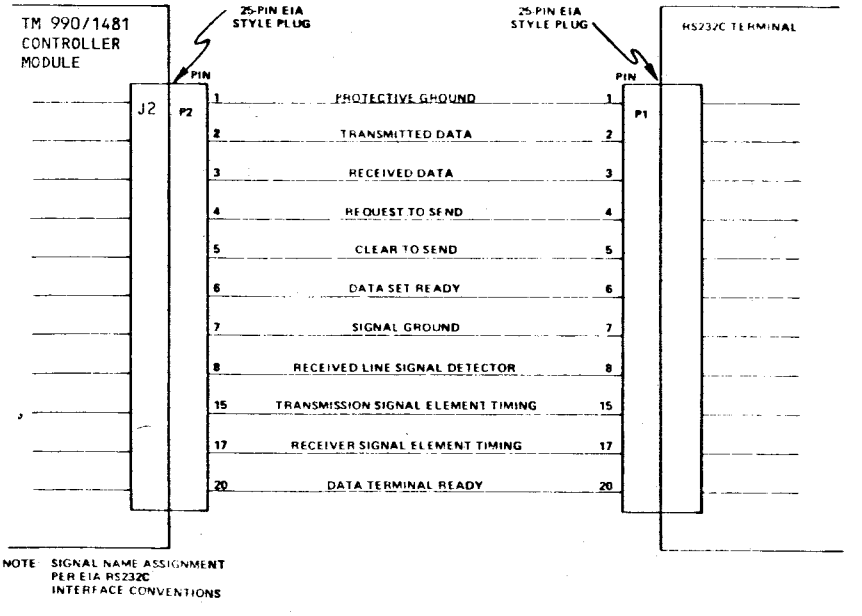


FIGURE 2-5. TM 990/1481 AND RS-232-C TERMINAL CONNECTIONS

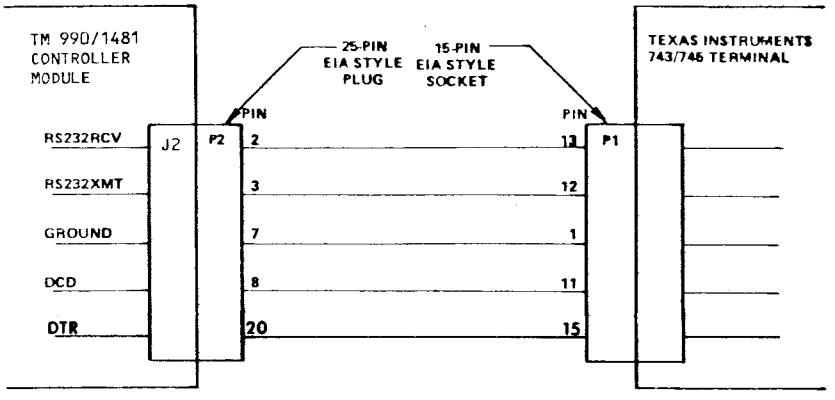


FIGURE 2-6. TM 990/1481 AND TI 743 OR 745 TERMINAL CONNECTIONS

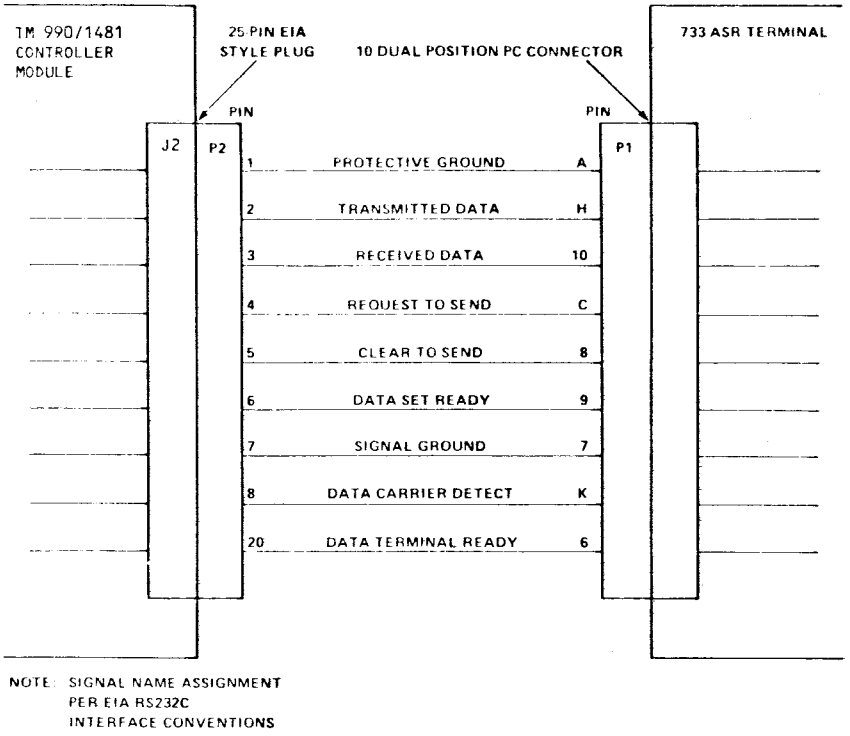


FIGURE 2-7. TM 990/1481 AND TI 733 ASR DATA TERMINAL

2.8 TYPICAL INSTALLATION AND INITIALIZATION SEQUENCE

2.8.1 Example of TM 990/201 Board Setup

Install the TM 990/403 TIBUG monitor on a TM 990/201-44 memory board in sockets U56 and U64. Set the switches on the TM 990/201 as follows:

	EPROM at 0000-7FFF				RAM at C000-FFFF			
	S1	S2	S3	S4	S5	S6	S7	S8
	ON	ON	ON	ON	ON	OFF	OFF	ON

Set the jumpers on the TM990/201 board according to the TM990/201 manual.

Install EPROMs containing user firmware on the board. If available, the TM 990/433 Floating Point Demonstration Software could be installed in sockets U57 and U65 on the TM 990/201 board. Switch settings are explained in your memory board user's manual for EPROM and in 403 TIBUG for the new RAM maps.

2.8.2 Board Installation

CAUTION

> Always disconnect power before inserting or removing a board from the card cage.

Install the TM 990/1481 Processor and Controller cards in the chassis with connectors J3 and J4 cabled to the corresponding connectors on each card (using the supplied cables). Connect a terminal to J2 of the controller. Check that the TM 990/201 card (or other memory board) is correctly populated and installed.

Verify that adaquate cooling is provided, optimally by forced air means.

Apply power to the boards and press the RESET toggle switch on the controller board. Now press the RETURN key on your terminal; the TIBUG banner should be printed, along with a prompt.

The TIBUG monitor is now executing, and all of the monitor commands may be executed. Commands for the TM 990/403 TIBUG are explained in the next section, Section 3.

SECTION 3

TM 990/403 TIBUG INTERACTIVE DEBUG MONITOR

3.1 GENERAL

The TM 990/403 TIBUG is debug monitor which provides an interactive interface between the user and the TM 990/1481. It is available as an option, supplied on two TMS 2716 EPROMs.

TIBUG occupies EPROM memory space from memory address (M.A.) 0000_{16} to $0FFF_{16}$ 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 eight 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 4.6.9.

All communications 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.
- Press carriage return on the terminal keyboard. 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 Cassett/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 Workspaces	3.2.12
X	Move ALU test to RAM and Execute	3.2.13
G	Start Execution At Location >1000	3.2.14

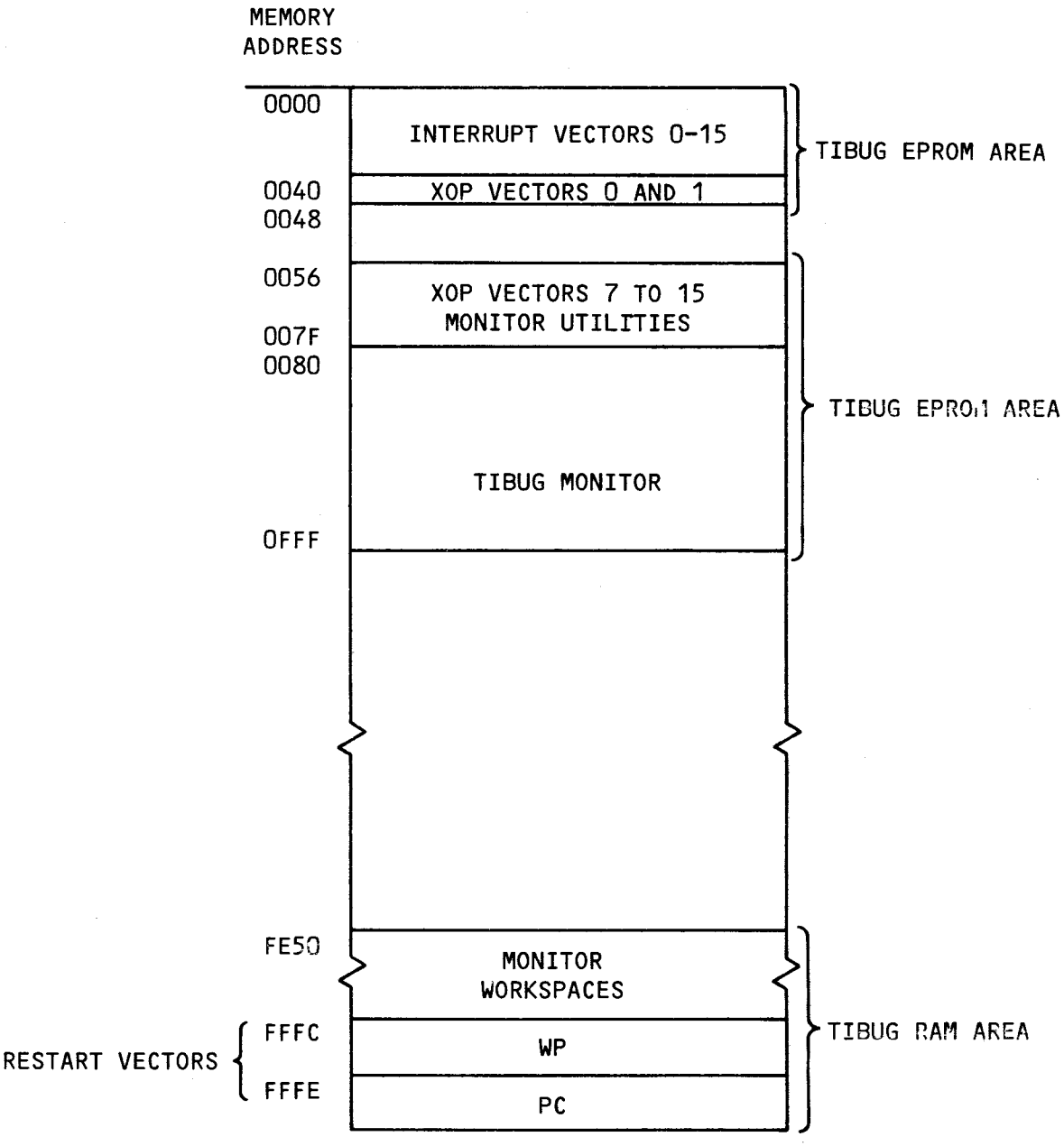


FIGURE 3-1. MEMORY REQUIREMENTS FOR TIBUG

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 otherwise indicated, 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 instruction. The breakpoint is controlled by a software interrupt, XOP 15.

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

EXAMPLE:

```
?B 3E06
BP    3A00    3E06    1400
?
```

3.2.2 CRU Inspect/Change (C)

3.2.2.1 Syntax

C <CRU Software Base Address> $\left\{ \begin{matrix} \wedge \\ , \end{matrix} \right\}$ <Count> <(CR)>

3.2.2.2 Description. The Communication Register Unit (CRU) input bits from "CRU software base address" to ("CRU software base address" + 2("count") -2) are displayed right justified in a 16-bit hexadecimal representation. "CRU software base address" is a 16-bit value in R12 bits 0 to 15; this is the same as the contents of register 12 as used by CRU instructions (Section 5). Up to 16 CRU bits may be displayed. The corresponding CRU output bits may be altered following input bit display by keying in desired hexadecimal data, right justified. A carriage return following data output forces a return to the command scanner. A minus sign (-) or a space causes the same CRU input bits to be displayed again. Defaults for "base address" and "count" are 0 (M.A. 0000) and 0 (count of 16) respectively. "Count" is a hexadecimal value of 0 to F_{16} with 0 indicating 16_{10}.

The CRU inspect/change monitor command displays from 1 to 16 CRU bits, right justified. The command syntax includes the CRU address and the number of CRU bits to be displayed. The CRU address is the 16-bit contents of R12 as explained in Section 5 (vs. the CRU bit address ("base address") in bits 4 to 14 of R12); thus the user must use 2 X CRU bit 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}.

```
? C 100,7
0100=007F
```

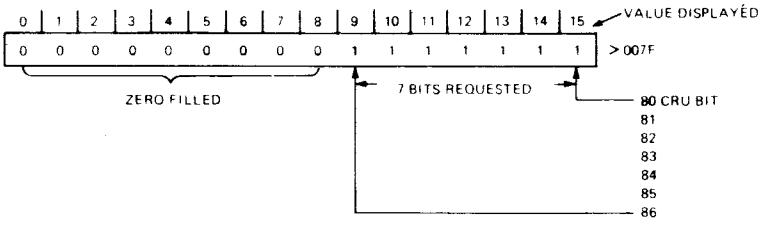


FIGURE 3-2. CRU BITS INSPECTED BY C COMMAND

EXAMPLES:

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

```
?C 20,8
0020=00FF <-- CARRIAGE RETURN ENTERED
?
```

(2) Set value of eight CRU output bits at CRU 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 0,1
0000=0001  -
0000=0001  -
0000=0001  -     <-- MINUS SIGN ENTERED
0000=0001  -
0000=00FF  -
0000=0001 <-- CARRIAGE RETURN ENTERED
?
```

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

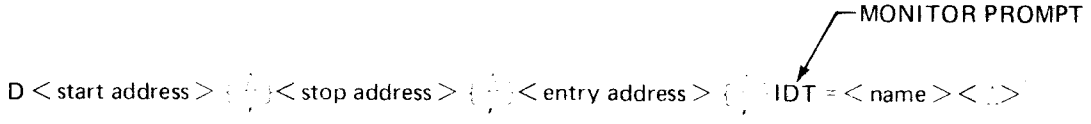
```
?C 100
0100=FFFE <-- ZERO INDICATES INTERRUPT MODE
```

(5) Check the contents of the clock register on the TMS 9901 (bits 1 to 14)

```
?C 102  E
0102=000E
?
```

3.2.3 Dump Memory To Cassette/Paper Tape (D)

3.2.3.1 Syntax



3.2.3.2 Description. 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. 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 inthe LINE position.

(7) Answer the "READY Y/N" query with a "Y"; the "Y" will not be echoed.

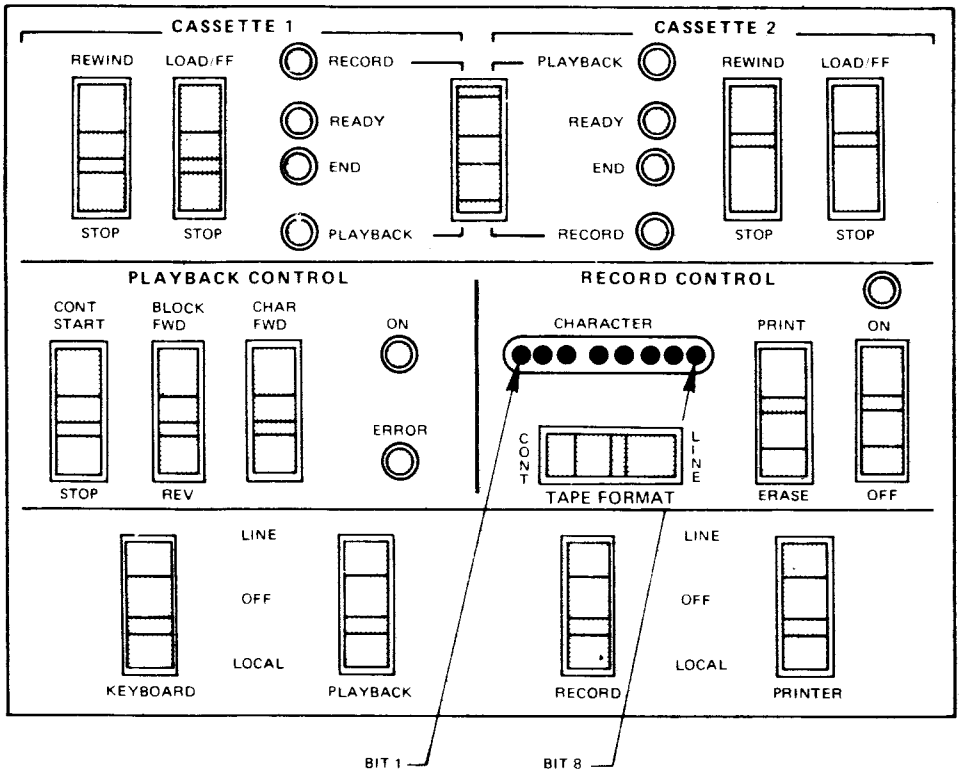


FIGURE 3-3. 733 ASR MODULE ASSEMBLY (UPPER UNIT) SWITCH PANEL

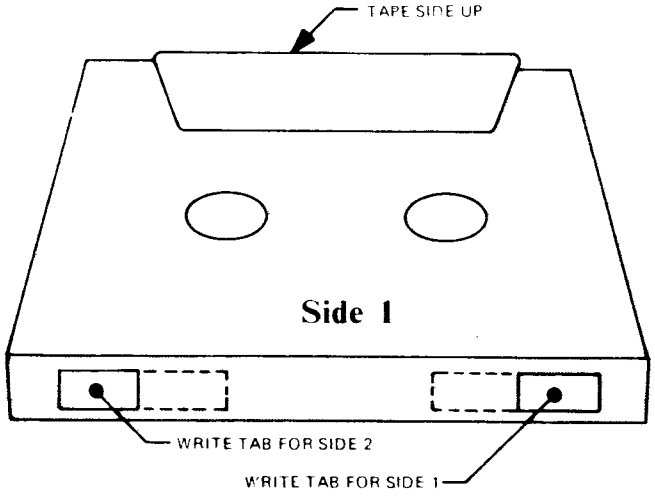


FIGURE 3-4. TAPE TABS

3.2.3.4 Dump To Paper Tape

The terminal is assumed to be an ASR 33 teletypwriter. 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 program 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 > { , } < stop address > { , } < value > { (CR) }

3.2.5.2 Description. The contents of memory locations from "start adddress" 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 > { , } < 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 tage 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 ASR 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 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

3.2.8.1 Syntax

- Memory Inspect/Change Syntax

  M < address > < (CR) >

- Memory Dump Syntax

  M < start address > { ^ } < stop 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 3E00 <-------------- CARRIAGE RETURN ENTERED
3E00=FF0F
3E02=0012  FFFF <------ NEW CONTENTS ENTERED
3E04=0311  - <--------- MINUS SIGN ENTERED
3E02=FFFF <------------ NEW CONTENTS
3E04=0311
3E06=0032  EEAA <------ NEW CONTENTS AND
?                       CARRIAGE 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 W, P, and S identify the values of the three principal hardware registers passed to the processor 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=2345   3F00 )
P=FF45   3E00 )  <-- SPACES ENTERED
S=860D   0000 )  <-- WORKSPACE POINTER
?S       3F00    3E02    0000   <-- PROGRAM COUNTER
?S       3F00    3E04    0000   <-- STATUS REGISTER
?S       3F00    3E06    0000
?S       3F00    3E08    0000
?S       3F00    3E0A    0000
?
```

NOTE

Incorrect results are obtained when the S instruction causes execution of an XOP instruction (see paragraph 4.6.9) in a user program. To avoid these problems the B command should be used to execute any XOP's in a program (rather than the S command).

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 990/1481 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:

```
?W<-------------------- CARRIAGE RETURN ENTERED
R0=F942  R1=0084  R2=FA2A  R3=002U  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
```

CAUTION

The following commands, X and G, are for factory test purposes only. Do not enter these commands to the monitor.

3.2.13 Move ALU Test to RAM and Execute (X)

3.2.13.1 Syntax

X <CR>

3.2.13.2 Description

This command is for factory test purposes only. It moves the ALU test from EPROM locations to RAM, then executes the test.

3.2.14 Start Execution at Address 1000_{16} (G)

3.2.14.1 Syntax

G <CR>

3.2.14.2 Description

This command is for factory test purposes only. It causes the start of execution at memory address 1000_{16}.

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 5.6.9. 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
7	Time Delay Via TMS 9901 Clock	3.3.1
8	Write 1 Hexadecimal Character to Terminal	3.3.2
9	Read Hexadecimal Word from Terminal	3.3.3
10	Write 4 Hexadecimal Characters to Terminal	3.3.4
11	Echo Character	3.3.5
12	Write 1 Character to Terminal	3.3.6
13	Read 1 Character from Terminal	3.3.7
14	Write Message to Terminal	3.3.8

NOTE

All characters are in ASCII code.

NOTE

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 Time Delay Via TMS 9901 Clock

Format: XOP Rn,7

The value in Rn represents the increments of 21.33 us delays desired. The delay range is from 106.6 us to 349.525 ms. The clock interrupt (3) is utilized by this XOP. Control returns to the instruction following the extended operation.

EXAMPLE: For a delay of 25 ms, the count of 1172 (25ms/21.35 us = 1172) will be used.

```
LI R1,1172              Set Up Delay Count
XOP R1,7                Do Extended Operation
(Next Instruction)      Execution Continued Here
```

NOTE

TIBUG uses XOP Rn,7 when determining Baud rates and when communicating with a terminal.

3.3.2 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 equivalent (0 to F) and output on a 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 follow.

```
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

3.3.3 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. IS 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 in previous example).

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

EXAMPLE:

```
A.L.    XOP     R6,9    READ HEXADECIMAL WORD INTO R6
        DATA    >3F80   RETURN ADDRESS, IF NO NUMBER
        DATA    >3F86   RETURN ADDRESS, IF ERROR
```

M.L. / M.A.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
3F00	0	0	1	0	1	1	1	0	0	1	0	0	0	1	1	0	>2E46
3F02	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	>3F80
3F04	0	0	1	1	1	1	1	1	1	0	0	0	0	1	1	0	>3F86

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. The plus sign (+) 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 $FFC0_{16}$.

3.3.4 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 HEXNUMBER

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

Terminal Output: 2C46

3.3.5 Echo Character (XOP 11)

Format: XOP Rn,11

This is a combination of XOP's 13 (read character) and 12 (write character). A charcter 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,>3E00        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
```

3.3.6 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.

3.3.7 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.

3.3.8 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          OP CODE          A.L. MNEMONIC
 (Hex)            (Hex)

 3E00             2FA0            XOP @> 3EE0,14
 3E02             3EE0
 3E04
  .                .
  .                .
  .                .
 3EE0             5445            TEXT 'TEST'
 3EE2             5354
 3EE4             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 733ASR, 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.

SECTION 4

TM 990/1481 INSTRUCTION SET

4.1 GENERAL

This section covers the instruction set used with the TM 990/1481 including machine and assembly language. This instruction set includes the standard TM 990 instruction set plus twenty-two floating point instructions. These latter include signed multiply and divide, double-precision multiply and divide, and real/integer conversions. Other topics covered in this section include:

- User memory
- Workspace concept
- Status register
- Instruction formats and addressing modes
- Instructions
- Comparison of jumps, branches, and XOPs
- Instruction execution times
- User defined instructions
- Use of floating point instructions
- Programming aids
- Interrupts.

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) and the Model 990/12 Computer Assembly Language Programmer's Guide (P/N 2250077-9701*A).

4.2 USER MEMORY

The amount of available user RAM space in memory for program execution depends on the memory module used and how it is configured. This information can be found in the user's guide for the memory module that is used.

4.3 WORKSPACE CONCEPT

Figure 4-1 shows the TM 990/1481 with RAM and EPROM memory. The RAM memory section will be used for user workspaces. Workspaces are blocks of memory that are used as data registers. The location of the starting address of a workspace is defined by a single hardware register called the workspace pointer. The workspace consists of sixteen 16-bit registers in memory. The LWPI (load workspace pointer immediate) instruction is used to define the starting address for the user workspace; if additional registers are needed, the program simply reloads the workspace pointer with the starting address of the new workspace. The number of 16-register workspaces is limited only by the amount of memory in the system.

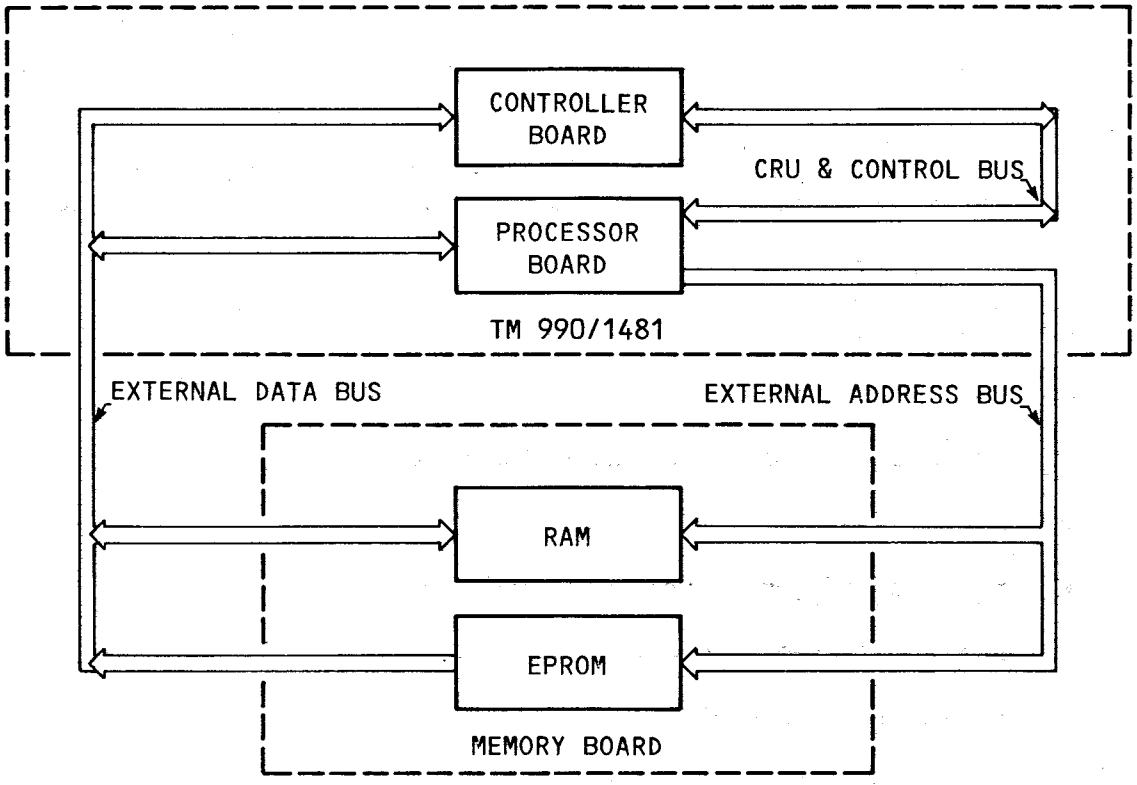


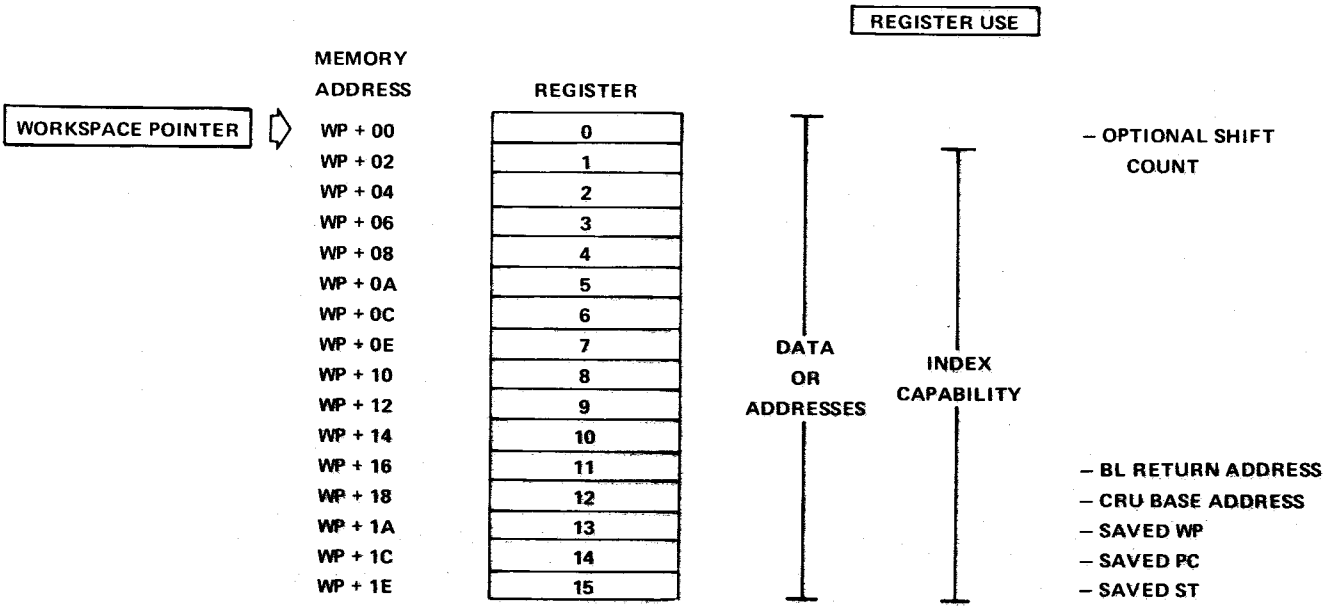
FIGURE 4-1. TM 990/1481 WITH RAM/ROM MEMORY

The uses of the workspace registers are shown in Table 4-1. All 16 registers (R0-R15) may be used for storage of addresses, temporary data, and accumulated results. R1-R15 may be used as index registers to specify a bias from a fixed-memory location to address an instruction operand. Register 0 may contain the number of bit positions an operand is shifted by the shift instructions (SLA, SRA, SRC, and SRL). R11 will contain one of the following:

1. the return address when the branch and link (BL) instruction is executed or,
2. the address parameter of an XOP instruction.

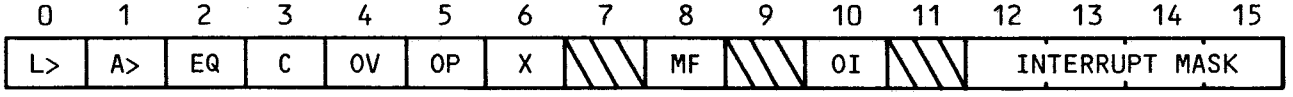
Bits 3-14 of R12 contain the CRU base address for CRU instructions. R13-R15 will contain the internal register values which are reloaded when the return to workspace (RTWP) instruction is executed.

TABLE 4-1. WORKSPACE REGISTERS



4.4 STATUS REGISTER

The status register is similar to that of other microprocessors in that it contains flag bits which indicate results of the most recent arithmetic or logical operation performed. It also contains a 4-bit interrupt mask which defines the lowest-priority level interrupt which will be recognized by the microprocessor. The bit structure of the status register is shown in Figure 4-2. A description of the information conveyed by the bits in the status register follows.



L>	LOGICALLY GREATER THAN
A>	ARITHMETICALLY GREATER THAN
EQ	EQUAL
C	CARRY
OV	OVREFLOW
▨	RESERVED

OP	ODD PARITY
X	XOP BEING EXECUTED
MF	MAP FILE SELECT
OI	OVERFLOW INTERRUPT ENABLE (ALLOW INTERRUPT ON OVERFLOW)

FIGURE 4-2. STATUS REGISTER BIT STRUCTURE

4.4.1 Logical Greater Than

This bit contains the result of a comparison of words or bytes as unsigned binary numbers. The most significant bit (MSB) of words being logically compared represents 2^{15} (32,768), and the MSB of bytes being logically compared represents 2^{7} (128).

4.4.2 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.

4.4.3 Equal

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

4.4.4 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 instructions to store the value of the last bit shifted out of the workspace register being shifted.

4.4.5 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} (See Section 4.10).

4.4.6 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.

4.4.7 Extended Operation

The extended operation bit of the status register is set to one when a software implemented extended operation (XOP) is initiated.

4.4.8 Status Bit Summary

Table 4-2 describes each of the status bits individually and lists those instructions which can alter each status bit.

Section 4.6 describes each instruction and identifies which status bits are affected by each instruction. Those bits that are not indicated as affected by an instruction will remain unchanged after execution of that instruction.

All status register bits are physically implemented on the TM 990/1481 . ST7, ST8, and ST9 are, however, not functionally implemented; instead they are routed to connector J2 of the Processor board to allow implementation of their associated functions external to the TM 990/1481. ST11 is not accessible external to the TM 990/1481 Processor board.

TABLE 4-2. STATUS REGISTER BIT DEFINITIONS

BIT	NAME	INSTRUCTION	CONDITION TO SET BIT TO 1 OTHERWISE SET BIT TO ZERO *NOTE 1
ST0	LOGICAL GREATER THAN (LGT)	C, CB	If MSB(SA) = 1 and MSB(DA) = 0, or if MSB(SA) = MSB(DA) and MSB[(DA)-(SA)] = 1
		CI	If MSB(W) = 1 and MSB(IOP) = 0, or if MSB(W) = MSB(IOP) and MSB[IOP-(W)] = 1
		ABS	If (SA) <> 0
	all others except DIV, MPY, XOP, CZC, COC, LIMI, CLR, SBZ, SBO, TB, JOP, JH, JL, LWPI, X, B, JNO, JOC, JNC, JNE, JGT, STST, LWP, JHE, JEQ, JLE, JLT, JMP, STWP, SWPB, BLWP, LREX, SETO, CKOF, CKON, RSET, IDLE, SWPB, BL, LST, RTWP and NOP		If RESULT <> 0
ST1	ARITHMETIC GREATER THAN (AGT)	C, CB	If MSB(SA) = 0 and MSB(DA) = 1, or if MSB(SA) = MSB(DA) and MSB[(DA)-(SA)] = 1
		CI	If MSB(W) = 0 and MSB(IOP) = 1, or if MSB(W) = MSB(IOP) and MSB[IOP-(W)] = 1
		ABS	If MSB(SA) = 0 and (SA) <> 0
	all others except DIV, MPY, XOP, CZC, COC, BL, BLWP, TB, SBZ, SBO, JOP, JH, JL, X, IDLE, JNO, JOC, JNC, JNE, JGT, CLR, LREX, JHE, JEQ, JLE, JLT, JMP, STWP, LWP, SETO, SWPB, B, CKOF, CKON, RSET, LIMI, LWPI, STST, LST, RTWP and NOP		If MSB(RESULT) = 0 and RESULT <> 0
ST2	EQUAL (EQ)	C, CB	If (SA) = (DA)
		CI	If (W) = IOP
		COC	If (SA)AND(/DA) = 0
		CZC	If (SA)AND(DA) = 0
		TB	If CRUIN = 1
		ABS	If (SA) = 0
	all others except DIV, MPY, XOP, SBZ, SBO, JOP, JH, JL, JNO, JOC, JNC, JNE, JGT, JHE, JEQ, JLE, JLT, JMP, SETO, SWPB, BL, CLR, X, B, BLWP, LREX, CKOF, CKON, RSET, IDLE, LIMI, LWPI, STST, STWP, LWP, LST, RTWP, and NOP		If RESULT = 0
ST3	CARRY (CO)	A, AB, ABS, AI, DEC, DECT, INC, INCT, NEG, S, SB	If CARRY OUT = 1
		SLA, SRA, SRC, SRL	If the last bit shifted out is a 1
		AR, SR, MR, DR, AD, SD, MD, DD	Set to 1 on OVERFLOW and set to 0 on UNDERFLOW (only valid if ST4 = 1)
		CDE, CRE, CDI, CRI	If number too big to represent as an integer (only valid if ST4 = 1)

TABLE 4-2. STATUS REGISTER BIT DEFINITIONS (continued)

BIT	NAME	INSTRUCTION	CONDITION TO SET BIT TO 1 OTHERWISE SET BIT TI ZERO
ST4	OVERFLOW (OV)	A, AB	If MSB(SA) = MSB(DA) and MSB(RESULT) <> MSB(DA)
		AI	If MSB(W) = MSB(IOP) and MSB(RESULT) <> MSB(W)
		S, SB	If MSB(SA) <> MSB(DA) and MSB(RESULT) <> MSB(DA)
		DEC, DECT	If MSB(SA) = 1 and MSB(RESULT) = 0
		INC, INCT	If MSB(SA) = 0 and MSB(RESULT) = 1
		SLA	If MSB changes during shift
		DIV	If MSB(SA) =0 and MSB(DA) = 1 , or if MSB(SA) = MSB(DA) and MSB[(DA)-(SA)] = 0
		DIVS	If (SA) = 0, or if overflow occurs
		ABS, NEG	If (SA) = >8000
		AR, SR, MR, DR, AD, SD, MD, DD, CDE, CRE, CDI, CRI	If floating point overflow or underflow or conversion overflow occurs
ST5	PARITY (OP)	CB, MOVB	If (SA) has an odd number of 1's
		LDCR, STCR	If 0 < C < 9 and (SA) has an odd number of 1's
		AB, SB, SOCB, SZCB	If RESULT has an odd number of 1's
ST6	XOP	XOP	If XOP instruction is executed via the software trap
ST10	OVERFLOW INTERRUPT ENABLE (OI)		*NOTE 2
ST12-ST15	INTERRUPT MASK	LIMI	If corresponding bit of IOP is 1
		RSET	Resets ST12, ST13, ST14, and ST15
		INTERRUPTS	*NOTE 3
ST0-ST15		LST	If corresponding bit of WR(S) is 1
		RTWP	If corresponding bit of WR(15) is 1
		RESET INTERRUPT	All status bits are cleared

*NOTE 1: When floating point operations and the signed divide instruction result in an overflow condition, only ST3 and ST4 are affected; all other status bits will reflect the value present previous to the execution of the instruction which resulted in the overflow.

*NOTE 2: When an interrupt or XOP occurs, status register bits seven through eleven are reset during the context switch. However, their original values are stored in WR (15) and can be restored to the status register with an RTWP. Status bits seven through eleven can only be set using the LST and RTWP instructions as directed above.

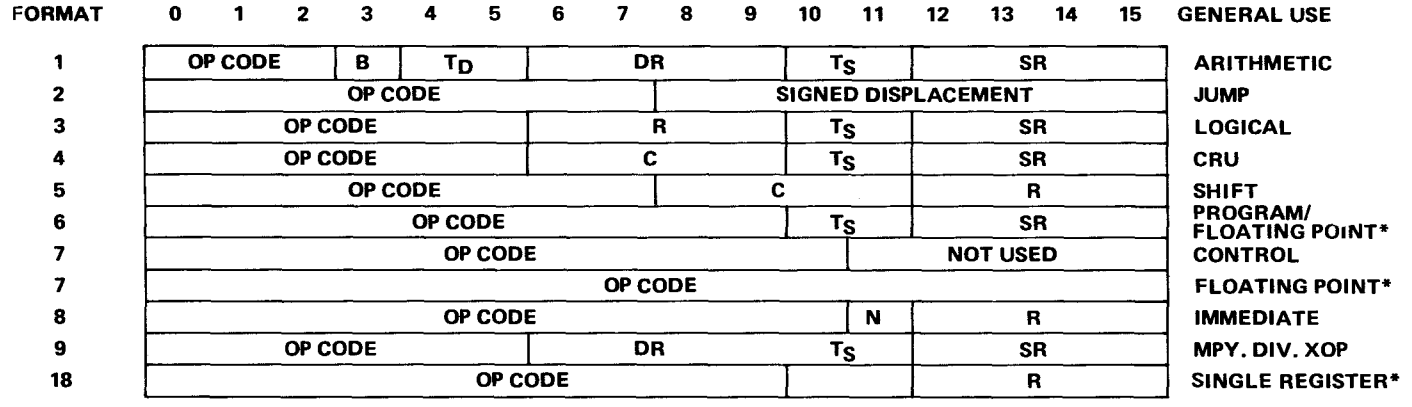
*NOTE 3: When a maskable (level one through fifteen) interrupt occurs, status register bits 12 through 15 are set so that the interrupt mask's contents are equal to the next higher priority interrupt level than the interrupt currently being executed. The LOAD interrupt dose not affect these status bits.

4.5 INSTRUCTION FORMATS AND ADDRESSING MODES

There are 95 instructions in the TM 990/1481 instruction set. In order to implement this instruction set nine instruction formats are used. The various instructions require from one to three words for full definition. The first word will contain the op code (operation code). The op code is the operation specified by the instruction converted into binary code. Other information that might be included in the bit fields of the first word include:

- T field - this field identifies what type of addressing mode is used.
- R field - this field identifies the workspace register number that is being affected.
- C field - this field specifies the number of bits affected by a CRU instruction or the number of bits to be shifted in a shift instruction.
- B field - this field identifies an instruction as either byte or word oriented. A one indicates that a byte will be addressed, while a zero indicates that a word will be affected.

Figure 4-3 shows the format for the first word of an instruction.



KEY

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

* Floating point, signed multiply and divide, and single register operand instructions are only implemented on the TM 990/1481 CPU module

FIGURE 4-3. INSTRUCTION FORMATS

In computers that operate on the stored program concept, the program consisting of instructions and data is first stored in memory prior to executing it. In order to retrieve the instructions and data for processing and manipulation some means of generating the addresses needed for locating the instructions/data is needed. The ways that addresses can be generated during program execution are called addressing modes. The TM 990/1481 provides 8 addressing modes as listed below:

1) Direct register addressing
2) Indirect register addressing
3) Indirect register autoincrement addressing
4) Symbolic memory addressing, not indexed
5) Symbolic memory addressing, indexed
6) Immediate addressing
7) Program counter relative addressing
8) CRU bit addressing.

The first three modes (direct register, indirect register, indirect register autoincrement) involve the workspace registers. In direct register addressing a workspace register contains the operand while in indirect register addressing the workspace register contains the address where the operand is located. Indirect register autoincrement is the same as indirect register except that the register contents are automatically incremented after the address (contents) has been obtained. The increment is one for byte instructions and two for word instructions.

Symbolic memory addressing specifies a memory address that contains the operand. Symbolic memory addressing can be indexed. In indexed symbolic memory addressing, the address where the operand is located is the sum of the contents of the workspace register and a symbolic address. Symbolic memory addressing allows direct access to instructions and data located in user RAM memory.

The immediate addressing mode use the contents of the word following the instruction word as an operand of the instruction. This mode is used to define the starting address for the workspace registers or when an absolute value is to be specified as an operand.

Program counter relative addressing is used by the jump instructions. This mode allows a change in program counter contents, either an unconditional change or a change conditional on status register contents.

CRU bit addressing is used with the five CRU instructions (SBO, SBZ, TB, STCR, and LDCR). This mode allows for the setting of a specific CRU bit to either a one or zero, testing a specific bit and setting the equal status bit (status register) to the logic value read, loading or storing various bit patterns. This mode is especially useful in industrial control applications.

It should be noted that the first 5 addressing modes can be used by formats having a T field (Formats 1, 3, 4, 6, and 9). The immediate addressing mode is used with Format 8 and the program counter relative addressing mode is used with Format 2. CRU bit addressing is used with Formats 2 and 4.

Now that the TM 990/1481 addressing modes have been listed and briefly described, additional information regarding each mode with specific examples will be given.

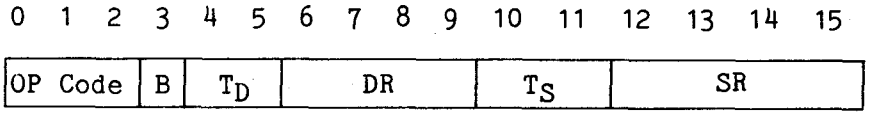
4.5.1 Direct Register Addressing

The direct register addressing mode specifies a workspace register that contains the operand. A workspace register is written as a term having a value of 0 through 15. In the example given in Figure 4-4, both the source and destination operands are registers. The T fields contain 00_2 to denote direct register addressing and their associated registers contain the binary value of the number of the register affected. The op code for add words (A) is 101_2. B = 0_2, because words are involved instead of bytes. As seen in Figure 4-5, the instruction A 1, 0 when coded is A001.

ASSEMBLY LANGUAGE:

A 1,0	Add a copy of the source operand (word) to the destination operand (word) and replace the destination operand with the sum (i.e., add contents of register 1 to register 0, place the sum in register 0).

FORMAT 1:



B = 1 for bytes and B = 0 for words.

MACHINE LANGUAGE:

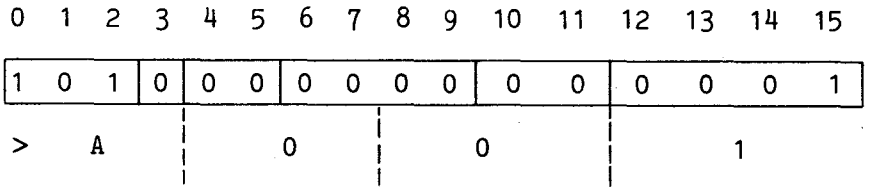


FIGURE 4-4. DIRECT REGISTER ADDRESSING EXAMPLE

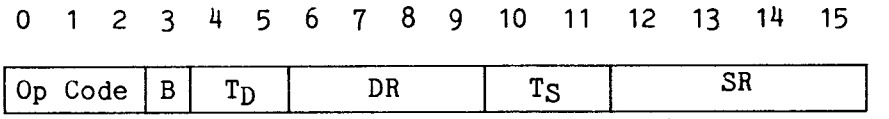
4.5.2 Indirect Register Addressing

Indirect register addressing specifies a workspace register that contains the address of the operand. An indirect workspace register is written as a term preceded by an asterisk (*). In the example given in Figure 4-6, both source and destination registers contain addresses specifying where the operands are located. The T fields contain 01_2 to denote indirect register addressing and their associated register fields contain the binary value of the number of the register affected. The op code for subtract bytes (SB) is 011_2. B = 1_2, because bytes are involved instead of words (it should be noted that the bytes involved are the leftmost or most significant bytes). As seen in Figure 4-5, the instruction SB *2,*3 when coded equals 74D2.

ASSEMBLY LANGUAGE:

SB *2,*3	Subtract a copy of the source operand (byte) from the destination operand (byte) and replace the destination operand with the difference (i.e., subtract the byte value at the address held in register 2 from the byte value at the address held in register 3; place the result in the byte at the address held in register 3).

FORMAT 1:



B = 1 for bytes and B = 0 for words.

MACHINE LANGUAGE:

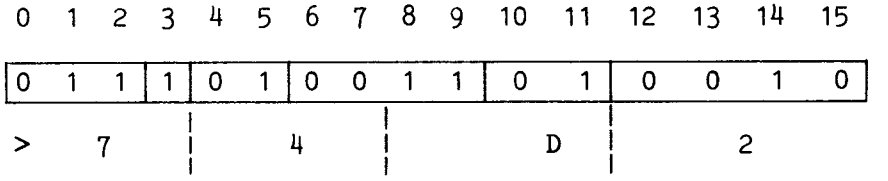


FIGURE 4-5. INDIRECT REGISTER ADDRESSING EXAMPLE

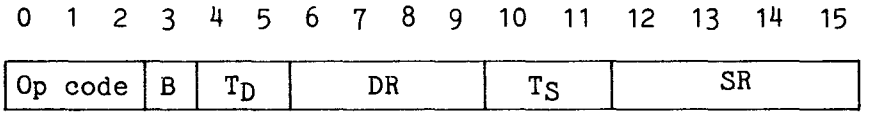
4.5.3 Indirect Register Autoincrement Addressing

The indirect register autoincrement addressing mode specifies a workspace register that contains the address of the operand. After the address is obtained from the workspace register, the workspace register is incremented. The workspace register increment is one for byte operations and two for word operations. An indirect register autoincrement is written as a term preceded by an asterisk (*) and followed by a plus (+) sign. In the example given in Figure 4-6, the source register contains an operand (direct register addressing) while the destination register contains the address where the other operand is located. Therefore the value of $T_S = 00_2$ (direct register addressing) while $T_D = 11_2$ (indirect register autoincrement addressing). The associated register fields contain the binary values of the numbers of the registers affected. The op code for add words (A) is 101_2. $B = 0_2$, because words are used instead of bytes. As seen in Figure 4-6, the instruction A 1,*0+ when coded equals AC01.

ASSEMBLY LANGUAGE:

A 1,*0+ — Add a copy of the source operand (word) to the destination operand (word) and replace the destination operand with the sum. After the addition is completed, the value in R0 will be incremented by 2 (i.e., add the contents of register 1 to the contents at the address found in register 0 and replace the contents of the address in register 0 with the sum; then increment by two the address in register 0).

FORMAT 1:



B = 1 for bytes and B = 0 for words.

MACHINE LANGUAGE:

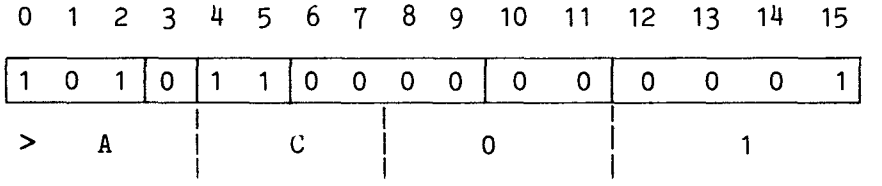


FIGURE 4-6. INDIRECT REGISTER AUTOINCREMENT ADDRESSING EXAMPLE

4.5.4 Symbolic Memory Addressing, Not Indexed

Symbolic memory addressing specifies a memory address that contains the operand. A symbolic memory address is written as an expression preceded by an at (@) sign. In the example given in Figure 4-7, both DR and SR are set to 0_2 as memory locations are being used instead of workspace registers. Both T fields contain 10_2 to denote symbolic memory addressing. The second word of the instruction contains the memory address for the source operand and the third word of the instruction contains the memory address for the destination operand. The op code for move words (MOV) is 110_2. B = 0_2, because words are involved instead of bytes. As seen in Figure 4-7, the instruction MOV @ 0200, @ 0202 when coded is:

```
>C820 (1st word)
>0200 (2nd word)
>0202 (3rd word)
```

ASSEMBLY LANGUAGE:

MOV @>200,@>202 — Replace the destination operand with a copy of the source operand (i.e., add the contents of memory address 0200_{16} to the contents of memory address 0202_{16} and replace the contents of memory address 0202_{16} with the sum).

FORMAT 1:

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			

B = 1 for bytes and B = 0 for words.

MACHINE LANGUAGE:

	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	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	>0200	Address of Source Operand
3rd Word	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	>0202	Address of Dest. Operand

FIGURE 4-7. SYMBOLIC MEMORY ADDRESSING EXAMPLE

4.5.5 Symbolic Memory Addressing, Indexed

Symbolic memory addressing can be indexed. This mode of addressing specifies a memory address that contains the operand. The memory address is determined by summing the contents of a workspace register and a symbolic address. An indexed memory address is written as an expression preceded by an at (@) sign and followed by a term enclosed in parenthesis. The term within the parenthesis is the index register. An example illustrating this addressing mode follows:

```
A @>0200(7),6
```

This instruction will sum the operand determined by indexed memory addressing with the operand in a direct workspace register. The first operand address is obtained by summing symbolic address 0200_{16} and the contents of workspace register R7. Assuming that R7 = 4, then the contents of >0204 (0200_{16} plus the contents (4) of R7) is added to the contents of R6 and then the sum is placed in R6. Indexing utilizes the contents of a workspace register to modify a symbolic address.

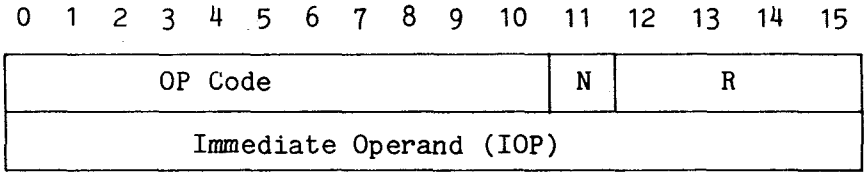
4.5.6 Immediate Addressing

Immediate addressing is used by the immediate instructions. Immediate instructions use the contents of the word following the instruction word as an operand of the instruction. This mode allows an absolute value to be specified as an operand. It can be used to load the workspace pointer, workspace registers, or the status register interrupt mask. Examples using this addressing mode are given in Figures 4-8 and 4-9. The first example uses this mode to define the starting address for the workspace registers at M.A. 0200_{16}. The value of R is set to 0 as no workspace register is involved. The resulting code is given in the figure. The second example uses this mode to load workspace register R5 with 15_{10} (Note: $15_{10} = F_{16}$). As this instruction involves a workspace register (R5), a value for R (0101_2) is given.

ASSEMBLY LANGUAGE:

```
LWPI  >0200          Load the workspace pointer with >0200.
```

FORMAT 8:



MACHINE LANGUAGE:

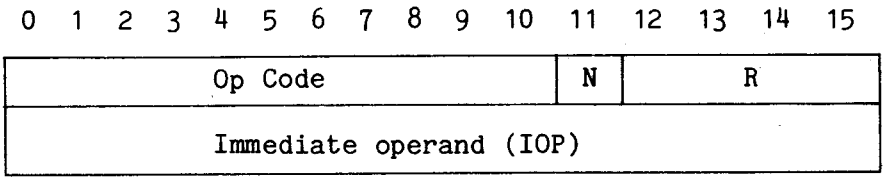
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1st Word	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	>02E0
2nd Word	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	>0200

FIGURE 4-8. IMMEDIATE ADDRESSING EXAMPLE 1

ASSEMBLY LANGUAGE:

LI 5,15 Load workspace register (R5) with 15_{10}.

FORMAT 8:



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	0	0	0	1	0	1	>0205
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	>000F

FIGURE 4-9. IMMEDIATE ADDRESSING EXAMPLE 2

4.5.7 Program Counter Relative Addressing

Program counter relative addressing is used by the jump instructions. This mode allows a change in program counter contents, either an unconditional change or a change conditional on status register contents. Examples using this mode follow:

JMP $ + 6 Jump to location, 6 bytes forward

JEQ $ + 4 If status register's equal bit = 1, jump 4 bytes (MA + 4)

JEQ $ If status register's equal bit = 1, stay in loop

4.5.8 CRU Bit Addressing

CRU bit addressing is used with the five CRU instructions (SBO, SBZ, TB, LDCR, STCR) to control I/O operations. There are two CRU bit addressing forms: CRU single-bit and CRU multibit. The CRU instructions SBO, SBZ, and TB use the single-bit form while CRU instructions LDCR and STCR use the multibit form. Prior to describing CRU bit addressing, the CRU interface will be reviewed.

The CRU interface uses three dedicated lines (CRUIN, CRUOUT, and CRUCLK) and the address bus. CRUIN and CRUOUT are used for serial data input and output, respectively. CRUCLK provides data timing strobes and is used in conjunction with CRUOUT. These lines are used in accordance with the address bus to transfer data to and from the microprocessor or to test the logic level of an addressed CRU input bit. Possibly the best way to envision the CRU is in terms of an addressable latch. The outputs from the latch are a function of the signal on the address lines and the logic level fed to the data input line. The output lines could be thought of as CRU bits and the address lines could be thought of as address lines emanating from the microprocessor. Depending on the address on the address bus and the level of the logic input signal, various outputs could be obtained. The CRUOUT line is similar to the

data input line for the latch and uses the SBO (set bit to one) and SBZ (set bit to zero) instructions to control the level on the output line (CRU bit).

The CRU interface is actually much more sophisticated than the simple concepts presented in the previous discussion. The CRU can not only set a specific line (CRU bit) to a one or a zero, it can also test the logic level of these lines (CRU bits). The CRU can also be used to transfer a bit pattern from a memory location to a specific output device or it can store a bit pattern presented to it by an input device. CRU I/O operations are implemented by a TMS 9901 Programmable Systems Interface located on the processor board.

Figure 4-10 shows the signals and lines used in CRU bit addressing.

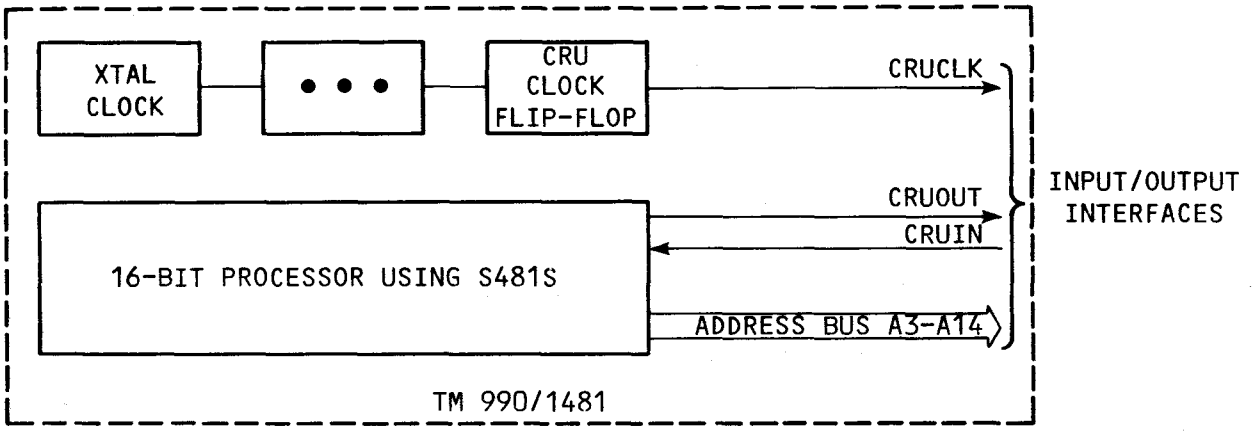


FIGURE 4-10. CRU INTERFACE

The CRU bit selected by single-bit instructions is determined by the value in bits 3-14 of workspace register 12 (R12) plus the value of the signed dispacement from the single-bit instruction (See Figure 4-11).

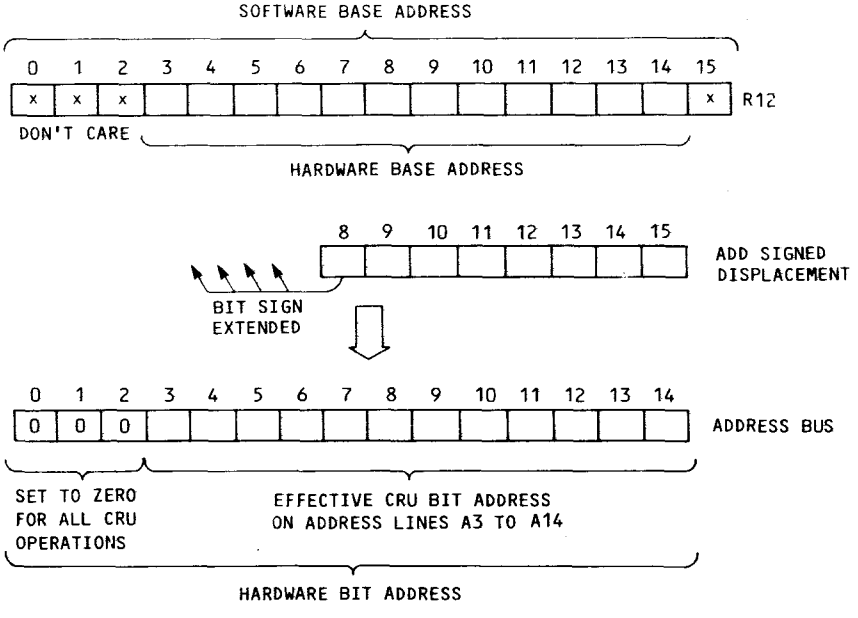


FIGURE 4-11. CRU BIT ADDRESS DEVELOPMENT

In order to determine the required CRU bit address, consult the CRU map for the TM 990/1481 located in Appendix B. The CRU address for output P0 of the TMS 9901 is 90 and P4 is displaced from P0 by four bits. The required code would be:

```
LI 12,>120      NOTE: 2 X >90 = >120
SBO 4
```

It should be noted that R12 was loaded with 2 X the desired value (90). This is necessary because bit 15 of R12 is not used in CRU base address determination. In order to place the correct CRU base address in R12, either a value equal to 2 X the required value may be loaded into R12 (as in the previous case) or the desired value may be loaded into R12 and then shifted left one bit using the following code:

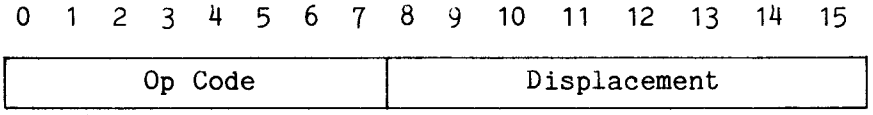
```
LI 12,>90
SLA 12,1
```

Figure 4-12 shows the machine and assembly language for the instruction SBO 4.

ASSEMBLY LANGUAGE:

SBO 4 — Set the digital output bit to a logic one on the CRU at the address derived by adding 4 to the hardware base address.

FORMAT :



MACHINE LANGUAGE:

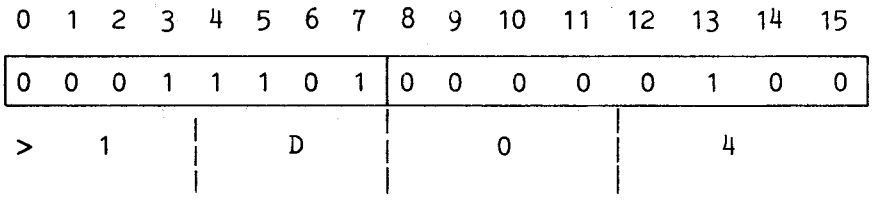


FIGURE 4-12. CRU BIT ADDRESSING EXAMPLE 1

CRU multibit instructions (LDCR and STCR) are used to transfer a specific number of bits from memory to the CRU at the address contained in bits 3-14 of workspace register 12 (R12) or to transfer a specified number of CRU bits from the CRU to the memory location supplied by the user as the source operand. The format for CRU multibit instructions is given in Figure 4-13. The C field specifies the number of bits to be transferred. If C = 0, 16 bits will be transferred. The CRU base register (R12, bits 3-14) defines the starting CRU bit address. Figure 4-13 shows the LDCR instruction being used to transfer 16 bits from MA >0200 to the CRU.

ASSEMBLY LANGUAGE:

LDCR @>0200,0 — Transfer the number of bits specified in the C field from the source operand to the CRU; note, If C = 0, 16 bits will be transferred (i.e., place the CRU hardware base address on the address bus, place the LSB value of memory address 0200_{16} on the CRUOUT line, increment the CRU hardware base address by one, and repeat this process until all 16 bits at M.A. 0200_{16} are transferred using the next bit to the left each time).

FORMAT 4:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Op Code						C				T_S		SR			

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	>3020
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	>0200

FIGURE 4-13. CRU BIT ADDRESSING EXAMPLE 2

4.6 INSTRUCTIONS

Table 4-3 lists terms used in describing the instructions of the TM 990/1481. Table 4-4 is an alphabetical list of instructions. Table 4-5 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 4-3. 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
n	Logical complement of n
>	Hexadecimal value

TABLE 4-4. INSTRUCTION SET, ALPHABETICAL INDEX

ASSEMBLY LANGUAGE MNEMONIC	MACHINE LANGUAGE OP CODE	FORMAT	STATUS REG. BITS AFFECTED	RESULT COMPARED TO ZERO	INSTRUCTION
A	A000	1	0–4	X	Add (Word)
AB	B000	1	0–5	X	Add (Byte)
ABS	0740	6	0–2	X	Absolute Value
AD	0E40	6	0–4	X	Add Double Precision Real
AI	0220	8	0–4	X	Add Immediate
ANDI	0240	8	0–2	X	And Immediate
AR	0C40	6	0–4	X	Add Real
B	0440	6	–	–	Branch
BL	0680	6	–	–	Branch and Link (R11)
BLWP	0400	6	–	–	Branch; New Workspace Pointer
C	8000	1	0–2	–	Compare (Word)
CB	9000	1	0–2,5	–	Compare (Byte)
CDE	0C05	7	0–4	X	Convert Double Precision Real to Extended Integer
CDI	0C01	7	0–4	X	Convert Double Precision Real to Integer
CED	0C07	7	0–4	X	Convert Extended Integer to Double Precision Real
CER	0C06	7	0–4	X	Convert Extended Integer to Real
CI	0280	8	0–2	–	Compare Immediate
CID	0E80	6	0–4	X	Convert Integer to Double Precision Real
CIR	0C80	6	0–4	X	Convert Integer to Real
CKOF	03C0	7	–	–	User Defined
CKON	03A0	7	–	–	User Defined
CLR	04C0	6	–	–	Clear Operand
COC	2000	3	2	–	Compare Ones Corresponding
CRE	0C04	7	0–4	X	Convert Real to Extended Integer
CRI	0C00	7	0–4	X	Convert Real to Integer
CZC	2400	3	2	–	Compare Zeroes Corresponding
DD	0F40	6	0–4	X	Divide Double Precision Real
DEC	0600	6	0–4	X	Decrement (by one)
DECT	0640	6	0–4	X	Decrement (by two)
DIV	3C00	9	4	–	Divide
DIVS	0180	6	0–2,4	X	Divide Signed
DR	0D40	6	0–4	X	Divide Real
IDLE	0340	7	–	–	Computer Idle
INC	0580	6	0–4	X	Increment (by one)
INCT	05C0	6	0–4	X	Increment (by two)
INV	0540	6	0–2	X	Invert (One's Complement)
JEQ	1300	2	–	–	Jump Equal (ST2 = 1)
JGT	1500	2	–	–	Jump Greater Than (ST1 = 1), Arithmetic
JH	1B00	2	–	–	Jump High (ST0 = 1 and ST2 = 0), Logical
JHE	1400	2	–	–	Jump High or Equal (ST0 or ST2 = 1), Logical
JL	1A00	2	–	–	Jump Low (ST0 and ST2 = 0), Logical
JLE	1200	2	–	–	Jump Low or Equal (ST0 = 0 or ST2 = 1), Logical
JLT	1100	2	–	–	Jump Less Than (ST1 and ST2 = 0), Arithmetic
JMP	1000	2	–	–	Jump Unconditional
JNC	1700	2	–	–	Jump No Carry (ST3 = 0)
JNE	1600	2	–	–	Jump Not Equal (ST2 = 0)
JNO	1900	2	–	–	Jump No Overflow (ST4 = 0)
JOC	1800	2	–	–	Jump On Carry (ST 3 = 1)

TABLE 4-4. INSTRUCTION SET, ALPHABETICAL INDEX (CONCLUDED)

ASSEMBLY LANGUAGE MNEMONIC	MACHINE LANGUAGE OP CODE	FORMAT	STATUS REG. BITS AFFECTED	RESULT COMPARED TO ZERO	INSTRUCTION
JOP	1C00	2	–	–	Jump Odd Parity (ST5 = 1)
LD	0F80	6	0–2	X	Load Double Precision Real
LDCR	3000	4	0–2,5	X	Load CRU
LI	0200	8	–	X	Load Immediate
LIMI	0300	8	12–15	–	Load Interrupt Mask Immediate
LR	0D80	6	0–2	X	Load Real
LREX	03E0	7	12–15	–	Load and Execute
LST	0080	18	0–15	–	Load Status Register
LWP	0090	18	–	–	Load Workspace Pointer Register
LWPI	02E0	8	–	–	Load Immediate to Workspace Pointer
MD	0F00	6	0–4	X	Multiply Double Precision Real
MOV	C000	1	0–2	X	Move (Word)
MOVB	D000	1	0–2,5	X	Move (Byte)
MPY	3800	9	–	–	Multiply
MPYS	01C0	6	0–2	X	Multiply Signed
MR	0D00	6	0–4	X	Multiply Real
NEG	0500	6	0–2	X	Negate (Two's Complement)
NEGD	0C03	7	0–2	X	Negate Double Precision Real
NEGR	0C02	7	0–2	X	Negate Real
ORI	0260	8	0–2	X	OR Immediate
RSET	0360	7	12–15	–	Reset AU
RTWP	0380	7	0–15	–	Return From Context Switch
S	6000	1	0–4	X	Subtract (Word)
SB	7000	1	0–5	X	Subtract (Byte)
SBO	1D00	2	–	–	Set CRU Bit to One
SBZ	1E00	2	–	–	Set CRU Bit to Zero
SD	0EC0	6	0–4	X	Subtract Double Precision Real
SETO	0700	6	–	–	Set Ones
SLA	0A00	5	0–4	X	Shift Left Arithmetic
SOC	E000	1	0–2	X	Set Ones Corresponding (Word)
SOCB	F000	1	0–2,5	X	Set Ones Corresponding (Byte)
SR	0CC0	6	0–4	X	Subtract Real
SRA	0800	5	0–3	X	Shift Right (sign extended)
SRC	0B00	5	0–3	X	Shift Right Circular
SRL	0900	5	0–3	X	Shift Right Logical
STCR	3400	4	0–2,5	X	Store From CRU
STD	0FC0	6	0–2	X	Store Double Precision Real
STR	0DC0	6	0–2	X	Store Real
STST	02C0	8	–	–	Store Status Register
STWP	02A0	8	–	–	Store Workspace Pointer
SWPB	06C0	6	–	–	Swap Bytes
SZC	4000	1	0–2	X	Set Zeroes Corresponding (Word)
SZCB	5000	1	0–2,5	X	Set Zeroes Corresponding (Byte)
TB	1F00	2	2	–	Test CRU Bit
X	0480	6	–	–	Execute
XOP	2C00	9	6	–	Extended Operation
XOR	2800	3	0–2	X	Exclusive OR

TABLE 4-5. INSTRUCTION SET, NUMERICAL INDEX

MACHINE LANGUAGE OP CODE (HEXADECIMAL)	ASSEMBLY LANGUAGE MNEMONIC	INSTRUCTION
0080	LST	Load Status Register
0090	LWP	Load Workspace Pointer
0180	DIVS	Divide Signed
0160	MPYS	Multiply Signed
0200	LI	Load Immediate
0220	AI	Add Immediate
0240	ANDI	And Immediate
0260	ORI	Or Immediate
0280	CI	Compare Immediate
02A0	STWP	Store WP
02C0	STST	Store ST
02E0	LWPI	Load WP Immediate
0300	LIMI	Load Int. Mask
0340	IDLE	Idle
0360	RSET	Reset AU
0380	RTWP	Return from Context Sw.
03A0	CKON	User Defined
03C0	CKOF	User Defined
03E0	LREX	Load & Execute
0400	BLWP	Branch; New WP
0440	B	Branch
0480	X	Execute
04C0	CLR	Clear to Zeroes
0500	NEG	Negate to Ones
0540	INV	Invert
0580	INC	Increment by 1
05C0	INCT	Increment by 2
0600	DEC	Decrement by 1
0640	DECT	Decrement by 2
0680	BL	Branch and Link
06C0	SWPB	Swap Bytes
0700	SETO	Set to Ones
0740	ABS	Absolute Value
0800	SRA	Shift Right Arithmetic
0900	SRL	Shift Right Logical
0A00	SLA	Shift Left Arithmetic
0B00	SRC	Shift Right Circular
0C00	CRI	Convert Real to Integer
0C01	CDI	Convert Double Precision Real to Integer
0C02	NEGR	Negate Real
0C03	NEGD	Negate Double Precision Real
0C04	CRE	Convert Real to Extended Integer
0C05	CDE	Convert Double Precision Real to Extended Integer
0C06	CER	Convert Extended Integer to Real
0C07	CED	Convert Extended Integer to Double Precision Real
0CC0	SR	Subtract Real
0C40	AR	Add Real
0C80	CIR	Convert Integer to Real
0D00	MR	Multiply Real

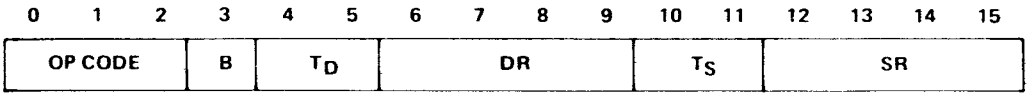
TABLE 4-5. INSTRUCTION SET, NUMERICAL INDEX (CONCLUDED)

MACHINE LANGUAGE OP CODE (HEXADECIMAL)	ASSEMBLY LANGUAGE MNEMONIC	INSTRUCTION
0D40	DR	Divide Real
0D80	LR	Load Real
0DC0	STR	Store Real
0E40	AD	Add Double Precision Real
0E80	CID	Convert Integer to Double Precision Real
0EC0	SD	Subtract Double Precision Real
0F00	MD	Multiply Double Precision Real
0F40	DD	Divide Double Precision Real
0F80	LD	Load Double Precision Real
0FC0	STD	Store Double Precision Real
1000	JMP	Unconditional Jump
1100	JLT	Jump on Less Than
1200	JLE	Jump on Less Than or Equal
1300	JEQ	Jump on Equal
1400	JHE	Jump on High or Equal
1500	JGT	Jump on Greater Than
1600	JNE	Jump on Not Equal
1700	JNC	Jump on No Carry
1800	JOC	Jump on Carry
1900	JNO	Jump on No Overflow
1A00	JL	Jump on Low
1B00	JH	Jump on High
1C00	JOP	Jump on Odd Parity
1D00	SBO	Set CRU Bits to Ones
1E00	SBZ	Set CRU Bits to Zeroes
1F00	TB	Test CRU Bit
2000	COC	Compare Ones Corresponding
2400	CZC	Compare Zeroes Corresponding
2800	XOR	Exclusive Or
2C00	XOP	Extended Operation
3000	LDCR	Load CRU
3400	STCR	Store CRU
3800	MPY	Multiply
3C00	DIV	Divide
4000	SZC	Set Zeroes Corresponding (Word)
5000	SZCB	Set Zeroes Corresponding (Byte)
6000	S	Subtract Word
7000	SB	Subtract Byte
8000	C	Compare Word
9000	CB	Compare Byte
A000	A	Add Word
B000	AB	Add Byte
C000	MOV	Move Word
D000	MOVB	Move Byte
E000	SOC	Set Ones Corresponding (Word)
F000	SOCB	Set Ones Corresponding (Byte)

4.6.1 Format 1 Instructions

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

GENERAL FORMAT:



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.

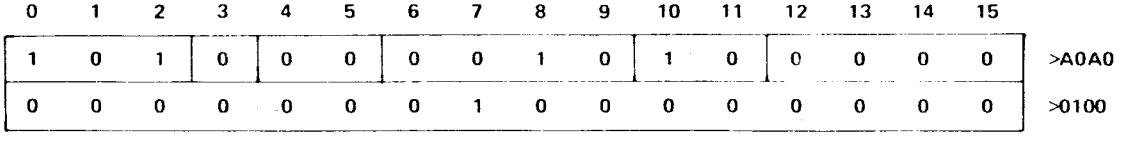
MNEMONIC	OP CODE 0 1 2	B 3	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
A	1 0 1	0	Add	Yes	0-4	(SA)+(DA) → (DA)
AB	1 0 1	1	Add bytes	Yes	0-5	(SA)+(DA) → (DA)
C	1 0 0	0	Compare	No	0-2	Compare (SA) to (DA) and set appropriate status bits
CB	1 0 0	1	Compare bytes	No	0-2,5	Compare (SA) to (DA) and set appropriate status bits
MOV	1 1 0	0	Move	Yes	0-2	(SA) → (DA)
MOVB	1 1 0	1	Move bytes	Yes	0-2,5	(SA) → (DA)
S	0 1 1	0	Subtract	Yes	0-4	(DA) – (SA) → (DA)
SB	0 1 1	1	Subtract bytes	Yes	0-5	(DA) – (SA) → (DA)
SOC	1 1 1	0	Set ones corresponding	Yes	0-2	(DA) OR (SA) → (DA)
SOCB	1 1 1	1	Set ones corresponding bytes	Yes	0-2,5	(DA) OR (SA) → (DA)
SZC	0 1 0	0	Set zeroes corresponding	Yes	0-2	(DA) AND ($\overline{SA}$) → (DA)
SZCB	0 1 0	1	Set zeroes corresponding bytes	Yes	0-2,5	(DA) AND ($\overline{SA}$) → (DA)

EXAMPLES

(1) ASSEMBLY LANGUAGE:

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

MACHINE LANGUAGE:



(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								MEANING	ST CONDITION TO CHANGE PC
	0	1	2	3	4	5	6	7		
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

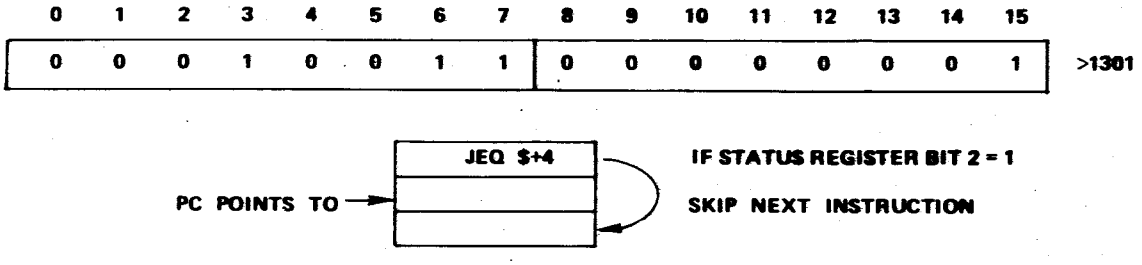
In assembly language, $ in the operand indicates "at this instruction". Essentially JMP $ causes an unconditional loop to the same instruction location, and JMP $+2 is essentially a no-op ($+2 means "here plus two bytes"). Note that the number following the $ is a byte count while displacement in machine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

EXAMPLES

(1) ASSEMBLY LANGUAGE:

```
JEQ   $+4      IF EQ BIT SET, SKIP 1 INSTRUCTION
```

MACHINE LANGUAGE:

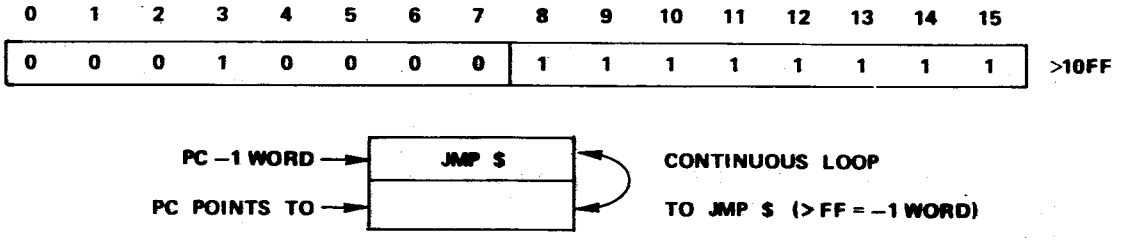


The above instruction continues execution 4 bytes (2 words) from the instruction location or, in other words, two bytes (one word) from the Program Counter value (incremented by 2 and now pointing to next instruction while JEQ executes). Thus, the signed displacement of 1 word (2 bytes) is the value to be added to the PC.

(2) ASSEMBLY LANGUAGE:

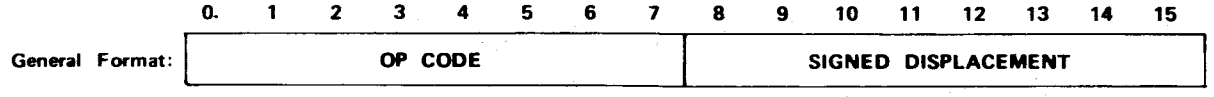
```
JMP   $        REMAIN AT THIS LOCATION
```

MACHINE LANGUAGE:



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.



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.

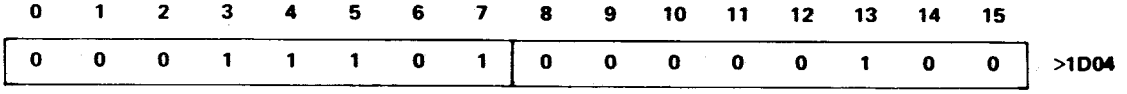
EXAMPLE

R12, BITS 3 TO 14 = >100

ASSEMBLY LANGUAGE:

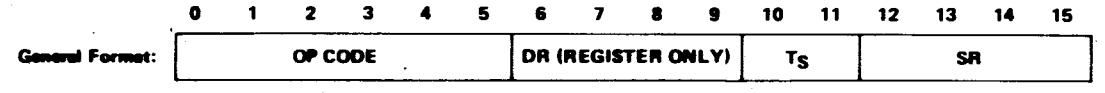
```
SBO   4     SET CRU ADDRESS >104 TO ONE
```

MACHINE LANGUAGE:



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.



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 0'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) ⊕ (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.

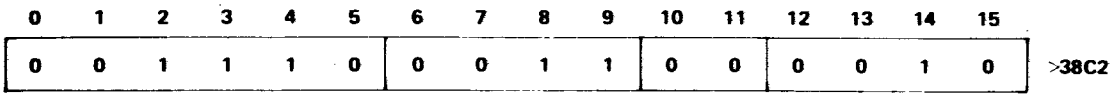
Exclusive OR Logic 1⊕0 = 1
0⊕0 = 0
1⊕1 = 0

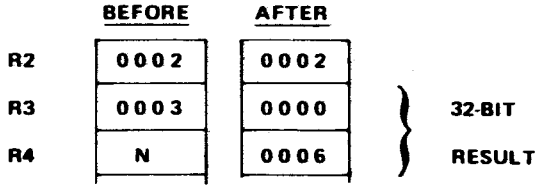
EXAMPLES

(1) ASSEMBLY LANGUAGE:

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

MACHINE LANGUAGE:





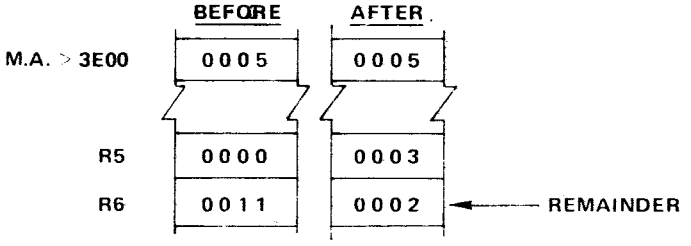
The destination operand is always a register, and the values multiplied are 16-bits, unsigned. The 32-bit result is placed in the destination register and destination register +1, zero filled on the left.

(2) ASSEMBLY LANGUAGE:

```
DIV   @>3E00, R5      DIVIDE CONTENTS OF R5 AND R6 BY VALUE AT M.A. >3E00
```

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
0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	>3E00



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 remainder 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

General Format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
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 R12 are not affected. Ts and SA 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.

MNEMONIC	OP CODE 0 1 2 3 4 5	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
LDCR	0 0 1 1 0 0	Load communcation register	Yes	0-2,5†	Beginning with LSB of (SA), transfer the specified number of bits from (SA) to the CRU.
STCR	0 0 1 1 0 1	Store communcation register	Yes	0-2,5†	Beginning with LSB of (SA), transfer the specified number of bits from the CRU to (SA). Load unfilled bit positions with 0.

†ST5 is affected only if $1 \leq C \leq 8$.

EXAMPLE

ASSEMBLY LANGUAGE:

```
LDCR   @>FE00,8      LOAD 8 BITS ON CRU FROM M.A. >FE00
```

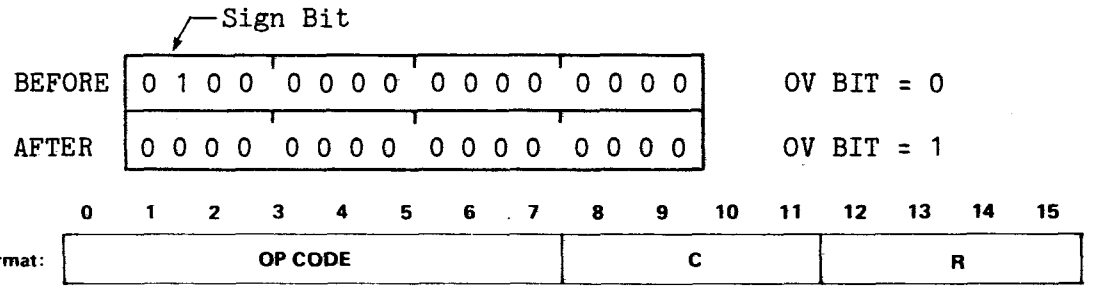
MACHINE LANGUAGE:

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

4.6.5 Format 5 (SHIFT) Instructions

These instructions shift (left, right, or circular) the bit patterns in a workspace register. The C field contains the number of bits to shift. The last bit value shifted out lis 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. For example:

SLA R1,3



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.

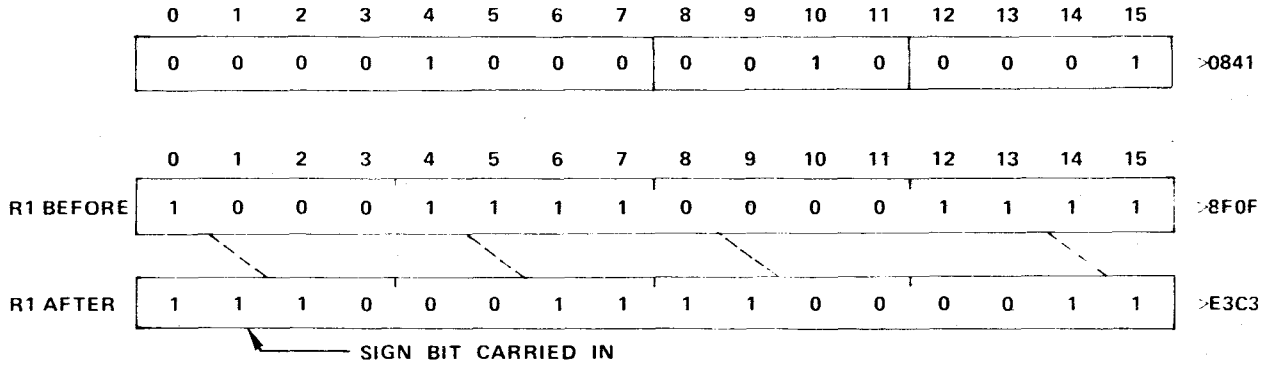
MNEMONIC	OP CODE 0 1 2 3 4 5 6 7	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
SLA	0 0 0 0 1 0 1 0	Shift left arithmetic	Yes	0-4	Shift (R) left. Fill vacated bit positions with 0.
SRA	0 0 0 0 1 0 0 0	Shift right arithmetic	Yes	0-3	Shift (R) right. Fill vacated bit positions with original 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
```

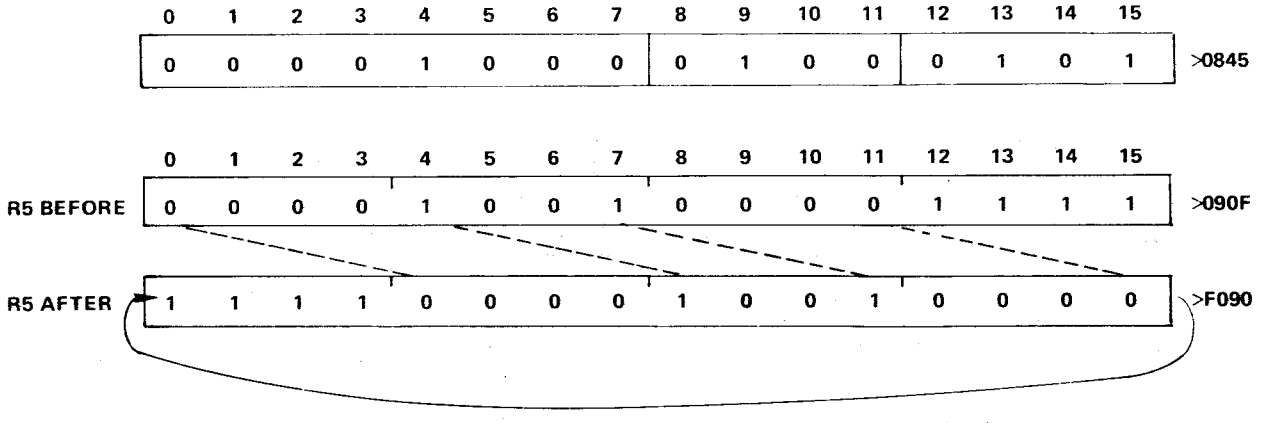
MACHINE LANGUAGE:



(2) ASSEMBLY LANGUAGE:

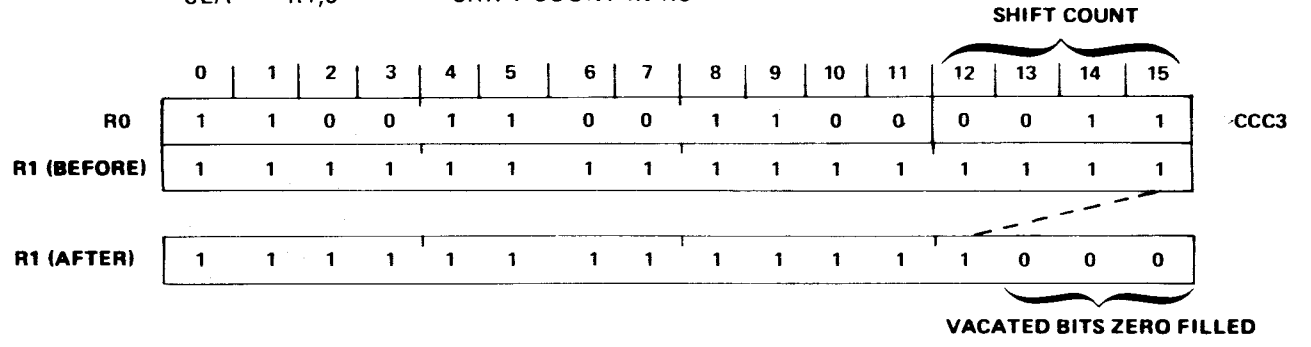
```
SRC    R5,4        CIRCULAR SHIFT R5 4 POSITIONS
```

MACHINE LANGUAGE:



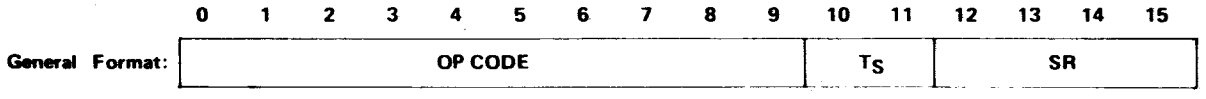
(3) ASSEMBLY LANGUAGE:

```
SLA    R1,0        SHIFT COUNT IN R0
```



4.6.6 Format 6 Instructions

These are single operand instructions. Examples illustrating the use of the floating point instructions and signed multiply and divide can be found in Appendix H.



The T_S and SR 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	(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.
STD	0	0	0	0	0	1	1	1	1	1	Start DP real	Yes	0–2	FPA → (SA)
LD	0	0	0	0	1	1	1	1	1	0	Load DP real	Yes	0–2	(SA) → FPA
DD	0	0	0	0	1	1	1	1	0	1	Divide DP real	Yes	0–4	FPA ÷ (SA) → FPA
MD	0	0	0	0	1	1	1	1	0	0	MPY DP real	Yes	0–4	PFA X (SA) → FPA
SD	0	0	0	0	1	1	1	0	1	1	Sub DP real	Yes	0–4	(FPA) – (SA) → FPA
CID	0	0	0	0	1	1	1	0	1	0	Convert integer to DP real	Yes	0–4	(SA) → FPA
AD	0	0	0	0	1	1	1	0	0	1	Add DP real	Yes	0–4	(SA) + FPA → FPA
STR	0	0	0	0	1	1	0	1	1	1	Store real	Yes	0–2	FPA → (SA)
LR	0	0	0	0	1	1	0	1	1	0	Load real	Yes	0–2	(SA) → FPA
DR	0	0	0	0	1	1	0	1	0	1	Divide real	Yes	0–4	FPA : (SA) → FPA
MR	0	0	0	0	1	1	0	1	0	0	Multiply real	Yes	0–4	FPA X (SA) → FPA
SR	0	0	0	0	1	1	0	0	1	1	Subtract real	Yes	0–4	FPA – (SA) → FPA
CIR	0	0	0	0	1	1	0	0	1	0	Convert integer to real	Yes	0–4	(SA) → FPA
AR	0	0	0	0	1	1	0	0	0	1	Add real	Yes	0–4	FPA + (SA) → FPA
DIVS	0	0	0	0	0	0	0	0	1	1	Divide signed	Yes	0–2,4	R0, R1 ÷ (SA) = reminder in R1 = quotient in R0
MPYS	0	0	0	0	0	0	0	1	1	1	Multiply signed	Yes	0–2	(R0) X (SA) → (R0 and R1)

NOTE

Jumps, branches, and XOP's are compared in Table 4-6.

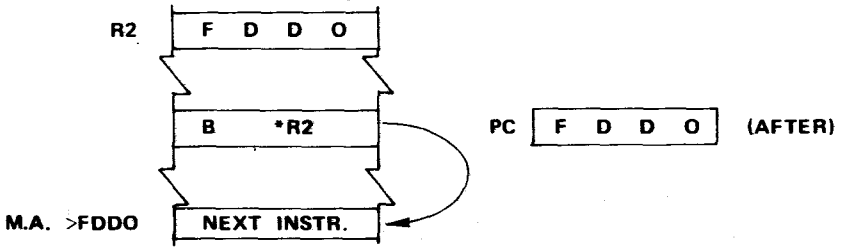
EXAMPLES

(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

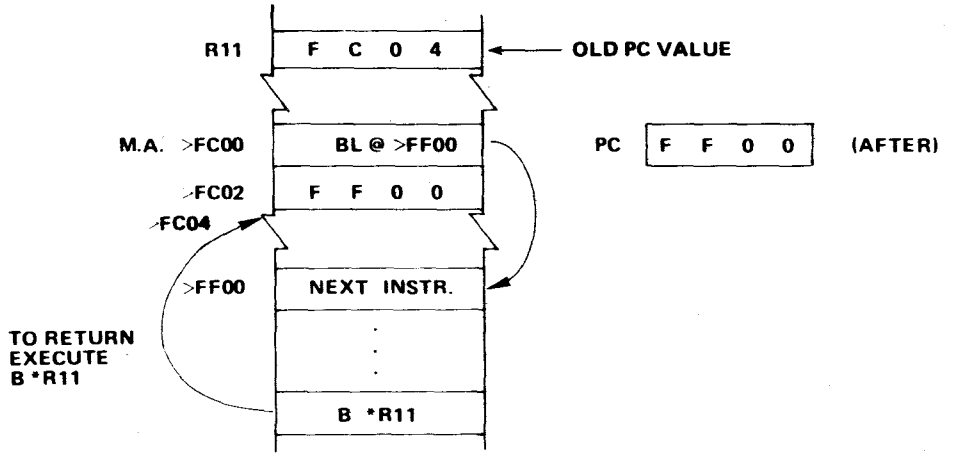


(2) ASSEMBLY LANGUAGE:

```
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



(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

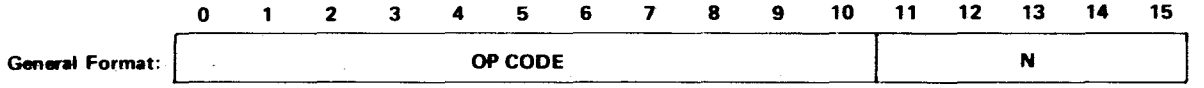
TABLE 4-6. COMPARISON OF JUMPS, BRANCHES, XOP'S

MNEMONIC	PARAGRAPH	DEFINITION SUMMARY
JMP	4.6.2	One-word instruction, destination restricted to +127, -128 words from program counter.
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.

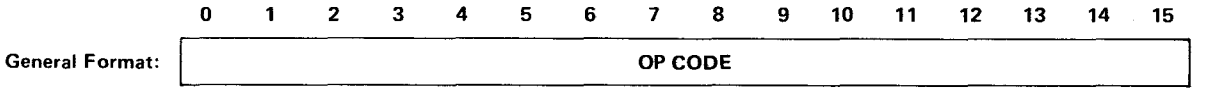
4.6.7 Format 7 RTWP/Control and Floating Point Instructions

Format 7 is used for 6 RTWP/control and 8 floating point instructions. The op code for RTWP/control instructions occupies bits 0-10 with bits 11-15 not used while the opcode for floating point instructions occupies all 16 bits. The formats for RTWP/control and floating point instructions are given below. Examples illustrating the use of the floating point instructions can be found in Appendix H.

CONTROL/RTWP INSTRUCTIONS



FLOATING POINT INSTRUCTIONS



MNEMONIC	OP CODE 0 1 2 3 4 5 6 7 8 9 10	MEANING	STATUS BITS AFFECTED	DESCRIPTION
IDLE	0 0 0 0 0 0 1 1 0 1 0	Idle	–	Suspend instruction execution until an interrupt, LOAD, or RESET occurs
RSET	0 0 0 0 0 0 1 1 0 1 1	Reset I/O & SR	12–15	0 → ST12 thru ST15
CKOF	0 0 0 0 0 0 1 1 1 1 0	User defined		---
CKON	0 0 0 0 0 0 1 1 1 0 1	User defined		---
LREX	0 0 0 0 0 0 1 1 1 1 1	Load interrupt		Control to *TIBUG*
RTWP	0 0 0 0 0 0 1 1 1 0 0	Return from Subroutine	0–15	(R13) → (WP) (R14) → (PC) (R15) → (ST)

MNEMONIC	OP CODE																MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15				
CED	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	1	Convert extended integer to DP real	Yes	0-4	FPA → FPA
CER	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	Convert extended integer to real	Yes	0-4	FPA → FPA
CDE	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	Convert DP real to extended integer	Yes	0-4	FFA → FPA
CRE	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	Convert real to extended integer	Yes	0-4	FPA → FPA
NEGD	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1	Negate DP real	Yes	0-2	−FPA → FPA
NEGR	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	Negate real	Yes	0-2	−(FPA) → FPA
CDI	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	Convert DP real to integer	Yes	0-4	FPA → FPA
CRI	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	Convert real to integer	Yes	0-4	FPA → FPA

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 by causing a context switch using the LOAD (NMI) vectors in the last two words of upper memory.

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 4-14). This context switch provides a new workspace register file and stores return values in the new workspace (See Figure 4-14). The operand (>FD00 above) is the M.A. of a two-word transfer vector, where the first word is the new WP value and the second word is the new PC value.

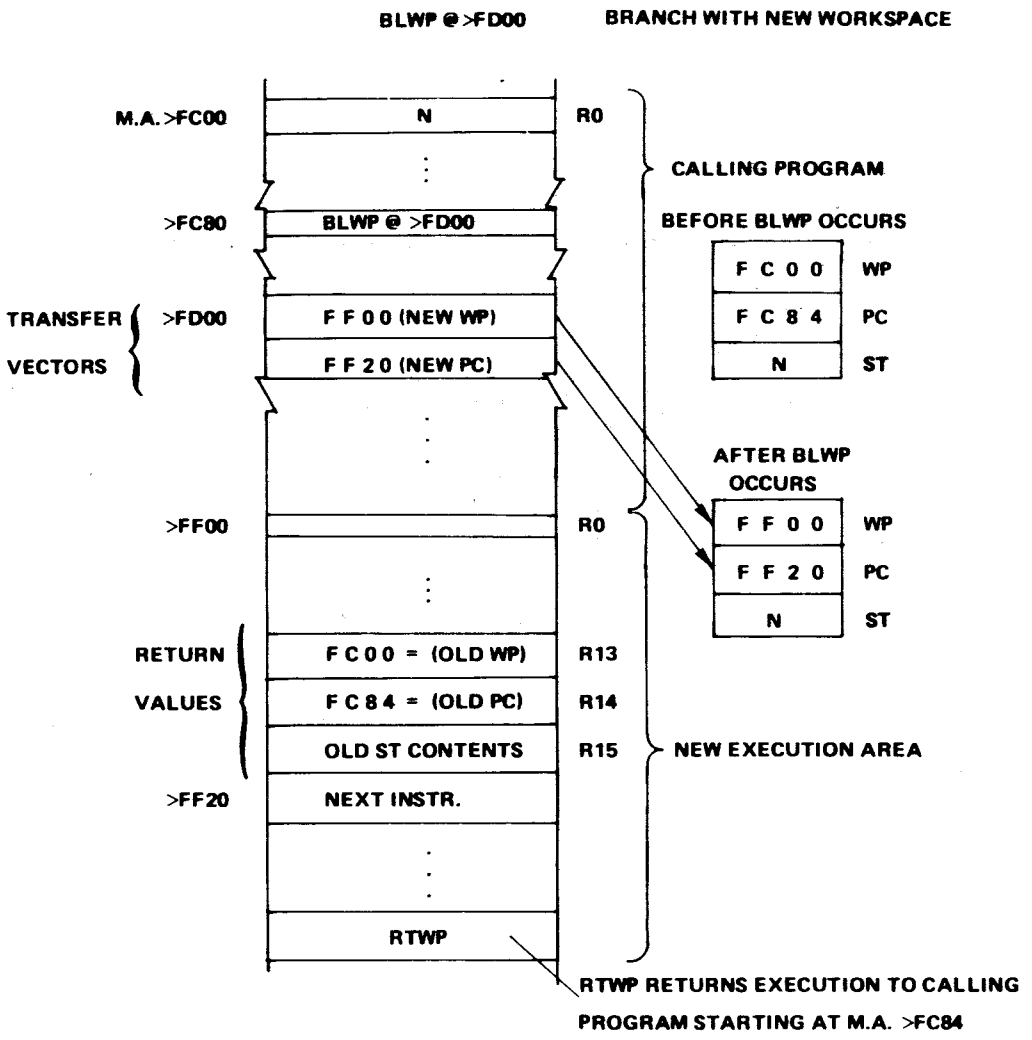


FIGURE 4-14. BLWP EXAMPLE

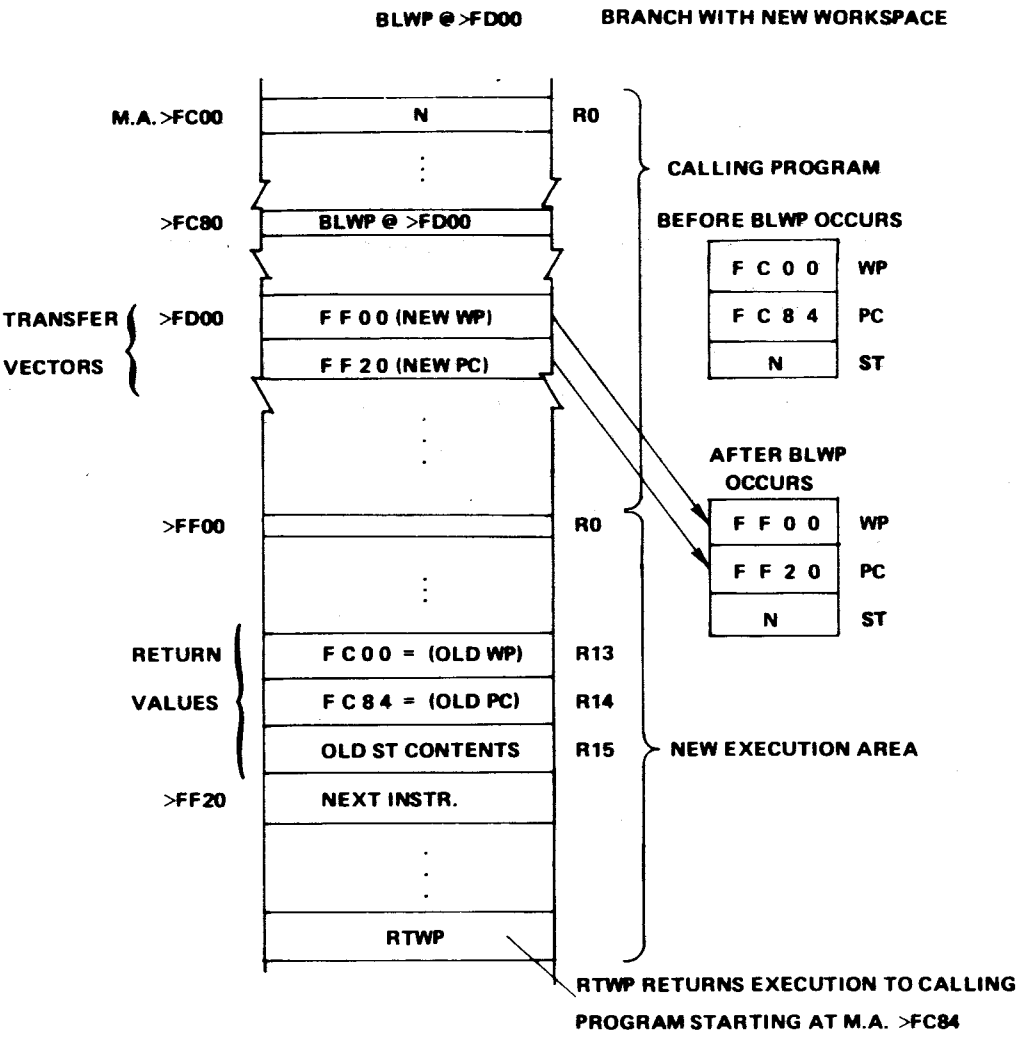
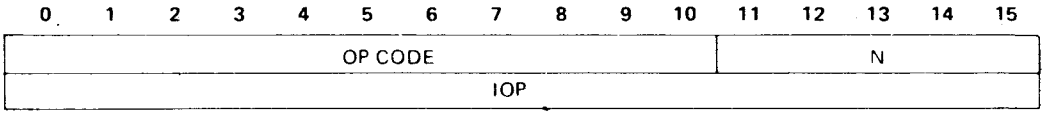


FIGURE 4-14. BLWP EXAMPLE

4.6.8.2 Internal Register Load Immediate Instructions

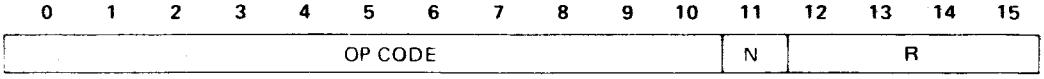
General format:



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

General format:



No ST bits are affected.

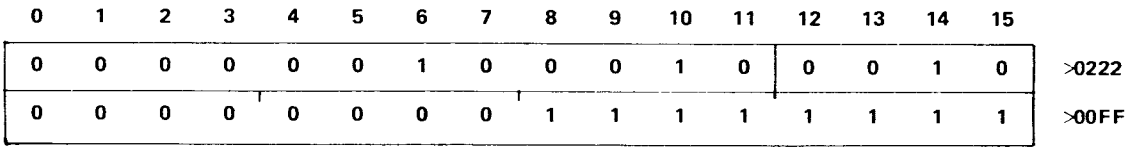
MNEMONIC	OP CODE 0 1 2 3 4 5 6 7 8 9 10	MEANING	DESCRIPTION
STST	0 0 0 0 0 0 1 0 1 1 0	Store status register	(ST) → (R)
STWP	0 0 0 0 0 0 1 0 1 0 1	Store workspace pointer	(WP) → (R)

EXAMPLES

(1) ASSEMBLY LANGUAGE:

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

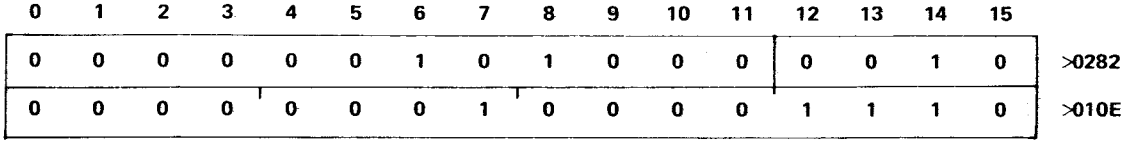
MACHINE LANGUAGE:



(2) ASSEMBLY LANGUAGE:

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

MACHINE LANGUAGE:



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

(3) ASSEMBLY LANGUAGE:

LWPI > 3E00 WP SET AT > 3E00 (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
0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	>3E00

This is used to define the workspace area in a task, usually placed at the beginning of a task.

(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 T_S 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 → (new R11)	
(old WP) → (new WR13)	
(old PC) → (new WR14)	
(old ST) → (new WR15)	

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 $7F_{16}$ is reserved for the transfer vectors of XOP's 0 to 15 (see Figure 3-1). 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 4-15 depicts calling an XOP to process a table of data; the data begins at M.A. $FF00_{16}$.

XOP's 0, 1 and 7 to 15 are used by the TM 990/403 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.

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

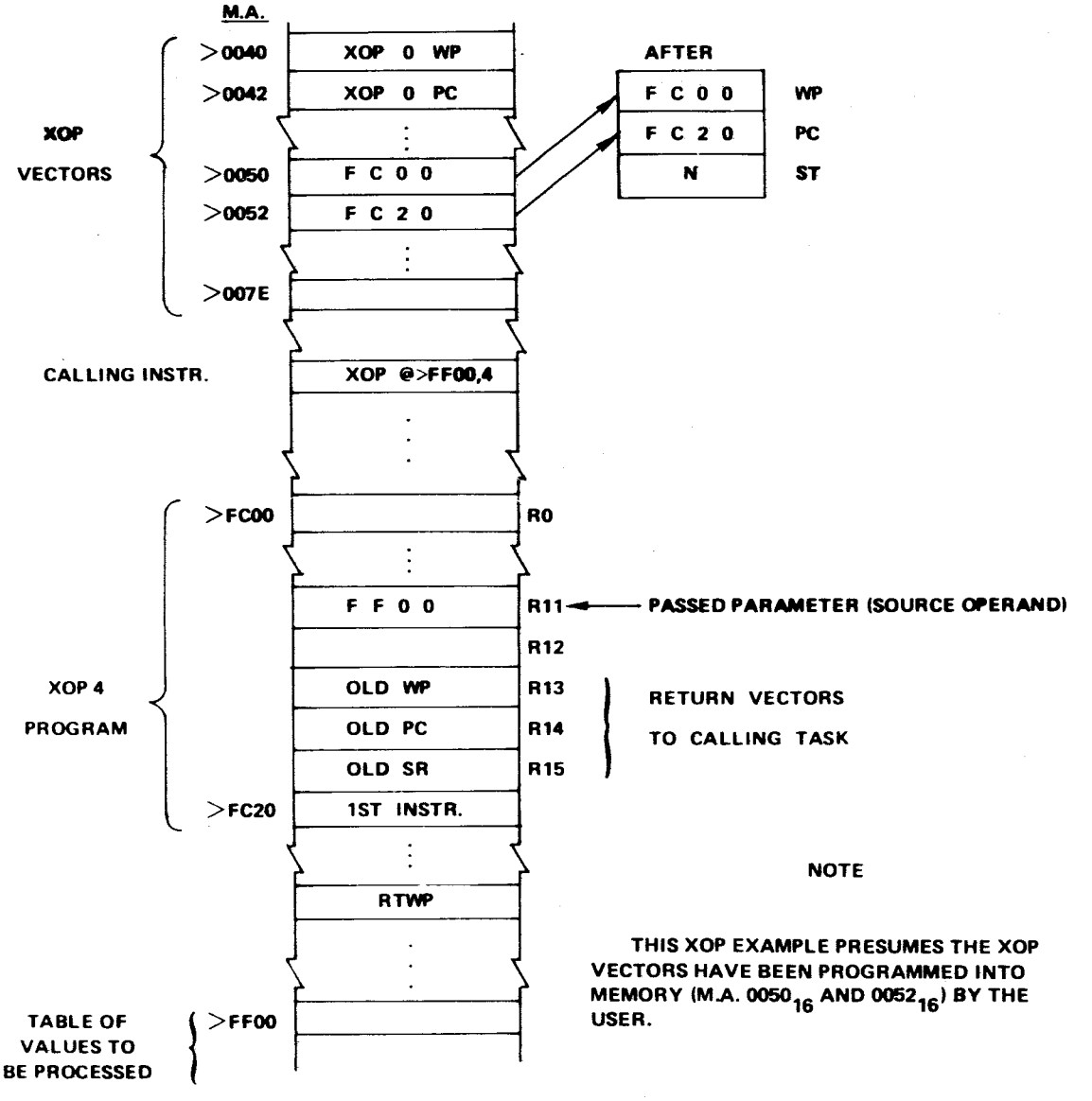


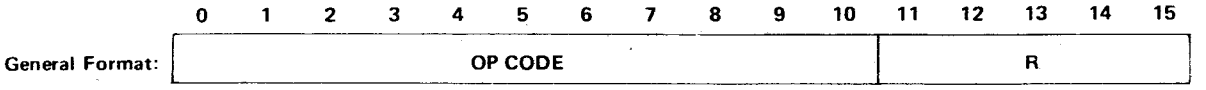
FIGURE 4-15. XOP EXAMPLE

4.6.10 Formats 10 Through 17 Instructions

Instructions using formats 10 through 17 are not implemented on the TM 990/1481.

4.6.11 Format 18 Single Register Operand Instructions

The operand field for format 18 instructions contains a workspace register address. Load status register (LST) and load workspace pointer (LWP) instructions comprise format 18 instructions that are implemented by the TM 990/1481. Examples illustrating the use of these instructions can be found in Appendix H.



MNEMONIC	OP CODE												MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
	0	1	2	3	4	5	6	7	8	9	10	11				
LST	0	0	0	0	0	0	0	0	1	0	0	0	Load status register	No	0-15	(R) → ST
LWP	0	0	0	0	0	0	0	0	1	0	0	1	Load workspace	No	None	(R) → WP

4.7 INSTRUCTION EXECUTION TIMES

Tables 4-7, 4-8, and 4-9 list data to derive execution times for TM990/1481 instructions under a variety of system memory configurations. Since the TM990/1481 has a variable period clock and the microinstruction cycles are not all the same length, the execution time must be determined as a multiple of the master clock period of 66.6 ns. In the examples in this presentation, actual execution time in nanoseconds will be computed (number of multiples times master clock period). Under the BASIC CYCLES column of Table 4-7 is the multiple for the basic execution time of the instruction exclusive of the standard operand derivation. (If the instruction requires special operand derivation, that time is included in the basic instruction execution time). Table 4-8 is used to derive the standard operand derivation.

Instruction execution time is determined in the following steps:

Step 1: Determine the number of base cycles from the BASE CYCLES column in Table 4-7 Base Cycles:_____

Step 2: a) Determine the memory delays per access from Table 4-9. Memory Delays per Access:_____

b) Determine the amount of memory accesses from the MEMORY FETCH CYCLES and MEMORY STORE CYCLES columns in Table 4-7. Memory Accesses:_____

c) Multiply Memory Accesses by Memory Delays per Access to find total Memory Delays (this must be done for each memory type in the system, if different):

P Memory: MA x MD/A_____
W Memory: MA x MD/A_____
G Memory: MA x MD/A_____

Step 3: Determine the extra base cycles from the BASE CYCLES of Table 4-8 Extra Base Cycles:_____

Step 4: a) Determine the Extra memory accesses from the MEMORY FETCH CYCLES, MEMORY STORE CYCLES of Table 4-8. Extra Memory Accesses:_____

b) Multiply Extra Memory Accesses by the number of Memory Delays per Access in Step 2a to find Extra Memory Delays (this must be done for each memory type in the system, if different):

P Memory: EMA x MD/A_____
W Memory: EMA x MD/A_____
G Memory: EMA x MD/A_____

Step 5: a) Add up the far-right column results of steps 1 through 4 above. Add results of steps 1 to 4_____

b) Multiply the sum in a) above by master clock period of 66.67 ns to determine instruction execution time: Instruction Execution Time:_____

Following the BASIC CYCLES column in Table 4-7 are columns listing the number of memory fetches and stores for each instruction. This is broken down into Program Memory (P), Workspace Memory (W), and General Memory (G). If slower memories are used, times must be added equal to the number of delays added to each fetch or store. Table 4-9 should be used to determine how many delays to use based on the type of memory used. The cycles are divided into P, W, and G

to allow for the fact that different speed memories might be used for each of these three, and the program and workspace areas might be separated to optimize the resulting ROM/RAM mix or the low cost vs. high speed cost/performance tradeoff. If applicable, this factor is used in steps 2 and 4. The General Memory (G) category indicates that the fetch or store may address any memory location or memory type (i.e., program, workspace or general data) for the purpose of accessing instruction operand data.

EXAMPLE 1

INSTRUCTION: A R1,@TABLE

MEMORY: TMS990/203-13 with TMS 4116 RAMs

STEP 1: Base Cycles
Find the basic execution time for an add instruction (A) from Table 4-7.
Base Cycles = 7 cycles

STEP 2: Memory Delays
Check Table 4-9 to determine how much delay to add due to memory speed.
DELAY = 3 delays/access x 2 accesses = 6 delays

STEP 3: Extra Base Cycles
Check columns SA, SO, and DO in Table 4-7 to see if other processing is required ("X" or note present).
Other processing required for SO and DO.
SO = R1 = REGISTER = 4 cycles
DO = @TABLE = DIRECT = 11 cycles

STEP 4: Extra Memory Delays
SO = R1 = REGISTER
= 1 access x 3 delays/access = 3 delays
DO = @TABLE = DIRECT
= 2 accesses x 3 delays/access = 6 delays

STEP 5: Total Cycles
Total = 7+6+4+11+3+6 = 37

Execution Time = 37 x 66.67 ns = 2.47 us

EXAMPLE 2

INSTRUCTION: SRA R1,4

MEMORY: TMS990/201-43A with TMS 4045-45 RAMs

STEP 1: Base Cycles
Find the basic execution time for a shift right arithmetic instruction (SRA) from Table 4-7.
Base Cycles = 14 cycles

STEP 2: Memory Delays
Check Table 4-9 to determine how much delay to add due to memory speed.
DELAY = 6 delays/access x 3 accesses = 18 delays

STEP 3: Extra Cycles
Check columns SA, SO, and DO in Table 4-7 see if other processing is required ("X" or note present).
Other processing required as specified in Table 4-7, Note 4.
Add C x 3 where C (the shift count) = 4
4 x 3 = 12 cycles

STEP 4: Extra Memory Delays
None

STEP 5: Total Cycles
Total = 14+18+12 = 44 cycles

Execution Time = 44 x 66.67 ns = 2.93 us

EXAMPLE 3

INSTRUCTION: DIV *R1+,R2

MEMORIES:
Program Memory (P)
TM 990/201-43A with TMS 2708 EPROMs
Workspace Memory (W)
TM 990/203-13 with TMS 2147 RAMs
General Memory (G)
TM 990/201-43A with TMS 4045-45 RAMs

STEP 1: Base Cycles
Find the basic execution time for a divide instruction (DIV) from Table 4-7.
BASE = 80 cycles

STEP 2: Memory Delays
Check Table 4-9 to determine how much delay to add due to memory speed.
Program Memory (P)
Delay = 6 delays/access x 1 access = 6 delays
Workspace Memory (W)
Delay = 3 delays/access x 4 accesses = 12 delays
General Memory
Delay = 3 delays/access x 0 accesses = 0 delays

STEP 3: Extra Base Cycles
Check columns SA, SO, and DO in Table 4-7 to see if other processing is required ("X" or note present).
Other processing required for SO.
SO = *R1+ = AUTO INC = 14 cycles

STEP 4: Extra Memory Delayx
SO = *R1+ = AUTO INC
Program Memory (P)
= 0 accesses x 6 delays/access = 0 delays
Workspace Memory (W)
= 2 accesses x 3 delays/access = 6 delays
*General Memory (G)
= 1 access x 6 delays/access = 6 delays
*NOTE: The (G) catagory means that it is not known wheter the memory access will involve the workspace area or the program area without further information. Here we assume the programmer will address the TM 990/201 for anything other that workspace or program.

STEP 5: Total Cycles
Total = 80+6+12+0+14+0+6+6 = 124 cycles

Execution Time = 124 x 66.67 ns = 8.27 us

TABLE 4-7. DATA TO DETERMINE TM990/1481 EXECUTION TIMES

INSTRUCTION MNEMONIC	CONDITION	BASE CYCLES	MEMORY FETCH CYCLES P W G	MEMORY STORE CYCLES W G	SA	SO	DO	OTHER
A		7	1 - -	- 1	-	X	X	
AB		10	1 - -	- 1	-	X	X	
ABS	SOURCE POS	16	1 - 1	- -	X	-	-	
	SOURCE NEG	23	1 - 1	- 1	X	-	-	
AD	MIN	416	1 4 4	4 -	X	-	-	*NOTE 1
	MAX	1328	1 4 4	4 -	X	-	-	
AI		15	2 1 -	1 -	-	-	-	
ANDI		15	2 1 -	1 -	-	-	-	
AR	MIN	211	1 2 2	2 -	X	-	-	*NOTE 1
	MAX	367	1 2 2	2 -	X	-	-	
B		6	1 - -	- -	X	-	-	
BL		9	1 - -	1 -	X	-	-	
BLWP		29	1 - 2	3 -	X	-	-	
C		7	1 - -	- -	-	X	X	
CB		7	1 - -	- -	-	X	X	
CDE	MAX	153	1 3 -	2 -	-	-	-	*NOTE 1
CDI	MAX	153	1 3 -	1 -	-	-	-	*NOTE 1
CED	MAX	201	1 - 2	4 -	-	-	-	*NOTE 1
CER	MAX	195	1 - 2	2 -	-	-	-	*NOTE 1
CI		15	2 1 -	- -	-	-	-	
CID	MAX	200	1 - 1	4 -	X	-	-	*NOTE 1
CIR	MAX	194	1 - 1	2 -	X	-	-	*NOTE 1
CKOF		9	1 - -	- -	-	-	-	
CKON		9	1 - -	- -	-	-	-	
CLR		6	1 - -	- 1	X	-	-	
COC		11	1 - -	1 -	-	X	-	
CRE	MAX	146	1 2 -	2 -	-	-	-	*NOTE 1
CRI	MAX	146	1 2 -	1 -	-	-	-	*NOTE 1
CZC		11	1 - -	1 -	-	X	-	
DD		2036	1 4 4	4 -	X	-	-	*NOTE 1
DEC		7	1 - -	- 1	-	X	-	
DECT		10	1 - -	- 1	-	X	-	
DIV		80	1 2 -	2 -	-	X	-	
DIVS	MAX	124	1 2 -	2 -	-	X	-	
DR		440	1 2 2	2 -	X	-	-	*NOTE 1
IDLE	MIN	5	1 - -	- -	-	-	-	

TABLE 4-7. DATA TO DETERMINE TM990/1481 EXECUTION TIMES (CONTINUED)

INSTRUCTION MNEMONIC	CONDITION	BASE CYCLES	MEMORY FETCH CYCLES P W G	MEMORY STORE CYCLES W G	SA	SO	DO	OTHER
INC		7	1 - -	- 1	-	X		
INCT		10	1 - -	- 1	-	X	-	
INV		7	1 - -	- 1	-	X	-	
JEQ		9	1 - -	- -	-	-	-	
JGT		9	1 - -	- -	-	-	-	
JH		9	1 - -	- -	-	-	-	
JHE		9	1 - -	- -	-	-	-	
JL		9	1 - -	- -	-	-	-	
JLE		9	1 - -	- -	-	-	-	
JLT		9	1 - -	- -	-	-	-	
JMP		9	1 - -	- -	-	-	-	
JNC		9	1 - -	- -	-	-	-	
JNE		9	1 - -	- -	-	-	-	
JNO		9	1 - -	- -	-	-	-	
JOC		9	1 - -	- -	-	-	-	
JOP		9	1 - -	- -	-	-	-	
LD		60	1 - 4	4 -	X	-	-	
LDCR		15*	1 1 -	- -	-	X	-	*NOTE 2
LI		11	2 - -	1 -	-	-	-	
LIMI		10	2 - -	- -	-	-	-	
LR		40	1 - 2	2 -	X	-	-	*NOTE 1
LREX		9	1 - -	- -	-	-	-	
LST		17	1 1 -	- -	-	-	-	
LWP		21	1 1 -	- -	-	-	-	
LWPI		10	2 - -	- -	-	-	-	
MD		892	1 4 4	4 -	X	-	-	*NOTE 1
MOV		7	1 - -	- 1	-	X	X*	*NOTE 3
MOVB		10	1 - -	- 1	-	X	X	
MPY		67	1 1 -	2 -	-	X	-	
MPYS	MAX	82	1 1 -	2 -	-	X	-	
MR		305	1 2 2	2 -	X	-	-	*NOTE 1
NEG		7	1 - -	- 1	-	X	-	
NEGD		42	1 4 -	1 -	-	-	-	*NOTE 1
NEGR		28	1 2 -	1 -	-	-	-	*NOTE 1
ORI		15	2 1 -	1 -	-	-	-	
RSET		9	1 - -	- -	-	-	-	
RTWP		21	1 3 -	- -	-	-	-	
S		7	1 - -	- 1	-	X	X	

TABLE 4-7. DATA TO DETERMINE TM990/1481 EXECUTION TIMES (CONTINUED)

INSTRUCTION MNEMONIC	CONDITION	BASE CYCLES	MEMORY FETCH CYCLES P W G	MEMORY STORE CYCLES W G	SA	SO	DO	OTHER
SB		10	1 - -	- 1	-	X	X	
SBO		23	1 1 -	- -	-	-	-	
SBZ		23	1 1 -	- -	-	-	-	
SD	MIN	416	1 4 4	4 -	X	-	-	*NOTE 1
	MAX	1328	1 4 4	4 -	X	-	-	
SETO		6	1 - -	- 1	X	-	-	
SLA	C <> 0	14*	1 1 -	1 -	-	-	-	*NOTE 4
	C = 0	69	1 2 -	1 -	-	-	-	
SOC		7	1 - -	- 1	-	X	X	
SOCB		10	1 - -	- 1	-	X	X	
SR	MIN	211	1 2 2	2 -	X	-	-	*NOTE 1
	MAX	367	1 2 2	2 -	X	-	-	
SRA	C <> 0	14*	1 1 -	1 -	-	-	-	*NOTE 4
	C = 0	69	1 2 -	1 -	-	-	-	
SRC	C <> 0	14*	1 1 -	1 -	-	-	-	*NOTE 4
	C = 0	69	1 2 -	1 -	-	-	-	
SRL	C <> 0	14*	1 1 -	1 -	-	-	-	*NOTE 4
	C = 0	69	1 2 -	1 -	-	-	-	
STCR	C = 0	184	1 2 -	- -	-	X	-	
	0 < C < 9	81*	1 2 -	- -	-	X	-	*NOTE 5
	8 < C < 16	72*	1 2 -	- -	-	X	-	*NOTE 5
STD		60	1 4 -	- 4	X	-	-	*NOTE 1
STR		40	1 2 -	- 2	X	-	-	*NOTE 1
STST		6	1 - -	1 -	-	-	-	
STWP		6	1 - -	1 -	-	-	-	
SWPB		10	1 - 1	- 1	X	-	-	
SZC		7	1 - -	- 1	-	X	X	
SZCB		10	1 - -	- 1	-	X	X	
TB		23	1 1 -	- -	-	-	-	
X		6	1 - -	- -	X	-	-	

TABLE 4-7. DATA TO DETERMINE TM990/1481 EXECUTION TIMES (CONCLUDED)

INSTRUCTION MNEMONIC	CONDITION	BASE CYCLES	MEMORY FETCH CYCLES P W G	MEMORY STORE CYCLES W G	SA	SO	DO	OTHER
XOP		41	1 - 2	4 -	-	X		
XOR		11	1 1 -	1 -	-	X	-	

*NOTE 1 : Because the floating point instructions may fetch multiple words for each source and destination operand, the delays required for these fetches have been included the calculation of the base cycles for these instructions.

*NOTE 2 : For LDCR add 24 x C cycles, where C is the bit count.

*NOTE 3 : For the MOV instruction do only destination address calculation. MOV does not fetch the destination operand (MOVB does). Use the special DESTINATION MOV time in the calculation for the MOV.

*NOTE 4 : For SLA, SRA, SRC, and SRL add 3 x C cycles, where C is the shift count.

*NOTE 5 : For STCR if 0 $\frac{1}{4}$ C $\frac{1}{4}$ 7 then add 4 x C cycles, and if 8 $\frac{1}{4}$ C $\frac{1}{4}$ 15 then add 14*C cycles, where C is the number of bits to be transferred.

TABLE 4-8. ADDRESS MODIFICATION FACTORS FOR INSTRUCTION EXECUTION TIMES

OPERATION TYPE	SYMBOLIC	NAME	CODE	BASE CYCLES	MEMORY FETCH CYCLES			MEMORY STORE CYCLES
					P	W	G	W
SOURCE ADDRESS	Rn	(register)	TS = 0	0	-	-	-	-
SOURCE ADDRESS	*Rn	(indirect)	TS = 1	4	-	1	-	-
SOURCE ADDRESS	@LOC	(direct)	TS = 2, S=0	7	1	-	-	-
SOURCE ADDRESS	@LOC(Rn)	(indexed)	TS = 2, S≠ 0	11	1	1	-	-
SOURCE ADDRESS	*Rn+	(auto inc)	TS = 3	10	-	1	-	1
SOURCE OPERAND	Rn	(register)	TS = 0	4	-	1	-	-
SOURCE OPERAND	*Rn	(indirect)	TS = 1	8	-	1	1	-
SOURCE OPERAND	@LOC	(direct)	TS = 2, S=0	11	1	-	1	-
SOURCE OPERAND	@LOC(Rn)	(indexed)	TS = 2, S≠ 0	15	1	1	1	-
SOURCE OPERAND	*Rn+	(auto inc)	TS = 3	14	-	1	1	1
DESTINATION OPERAND	Rn	(register)	TD = 0	4	-	1	-	-
DESTINATION OPERAND	*Rn	(indirect)	TD = 1	8	-	1	1	-
DESTINATION OPERAND	@LOC	(direct)	TD = 2, D=0	11	1	-	1	-
DESTINATION OPERAND	@LOC(Rn)	(indexed)	TD = 2, D≠ 0	15	1	1	1	-
DESTINATION OPERAND	*Rn+	(auto inc)	TD = 3	14	-	1	1	1
DESTINATION MOV	Rn	(register)	TD = 0	0	-	-	-	-
DESTINATION MOV	*Rn	(indirect)	TD = 1	4	-	1	-	-
DESTINATION MOV	@LOC	(direct)	TD = 2, D=0	7	1	-	-	-
DESTINATION MOV	@LOC(Rn)	(indexed)	TD = 2, D≠ 0	11	1	1	-	-
DESTINATION MOV	*Rn+	(auto inc)	TD = 3	10	-	1	-	1

TABLE 4-9. MEMORY ACCESS TIMES

BOARD	DEVICE	MEMORY TYPE	ACCESS TIME	WAIT STATES	PLUG PROGRAM	DELAYS/ ACCESS	NOTE
TM990/201-44	TMS 2716	EPROM	486	SLOW	SHFT7	6	
	2114-15	RAM	186	SLOW	SHFTX	2	
	2114-20	RAM	236	SLOW	SHFTX	3	
	2114-30	RAM	336	SLOW	SHFTX	4	
	2114-45	RAM	486	SLOW	SHFT7	6	
TM990/203-13	TMS 4116	RAM	***	0	SHFTX	3	2

NOTES:

1. The plug programming (wiring) options are described in section 2.4.3.
2. The dynamic memory has refresh operations which can occur during a memory access and this prevents assigning a fixed delay and forces the use of the READY line (i.e., SHIFTX).

4.8 TM 990/1481 FLOATING-POINT ARITHMETIC

The TM 990/1481 CPU provides the user with a more accurate method of doing numerical calculations than many other processors that are available today. As an example, the arithmetic instructions of the TMS 9900 microprocessor allow the user to do calculations with integers only. When working with only integers, the position of the decimal point is always known and the range of the numbers is usually small (-32768 to +32767). However, there is a large class of problems which involve calculations that require a greater range and more accuracy. These problems can be solved by using multiple precision programming techniques and the correct algorithm with enough memory and processor time.

The TM 990/1481 provides a unique set of instructions that use floating-point numbers to address the range and accuracy problems. These instructions can be divided into three groups of operations as follows:

1) Arithmetic/Logic Operations
2) Load/Store Operations
3) Conversion Operations.

Floating-point representation is similar to scientific notation. In scientific notation a number is expressed as a numeric value times a power of ten (eg., 4700 = 4.7×10^3 in scientific notation). Floating-point representation separates a number into two distinct parts: a mantissa and an exponent. The mantissa consists of the digits of the number and the exponent is a quantity that denotes the power to which the base is to be raised. This type of number is also called a real number.

The TM 990/1481 provides two types of floating-point or real numbers: single-precision real (real) and double-precision real (Note: single-precision real numbers are usually referred to as simply "real numbers"). The difference between the two types of floating-point numbers is the number of bits that make up the mantissa; double-precision numbers use 56 bits for the mantissa while single-precision numbers use 24 bits for the mantissa. The accuracy of a value is determined by the number of bits that make up the mantissa.

Prior to discussing the internal machine representation of floating-point instructions, several examples involving floating-point representation and floating-point operations will be presented. These examples will be illustrated using base ten for readability.

4.8.1 Floating-Point Representation

4.8.1.1 Mantissa. The mantissa or digits of a number will always be represented with its decimal point placed to the left of the most significant non zero digit. This process is called normalization. For example the following two numbers will look the same normalized.

UN-NORMALIZED	NORMALIZED
27.342	.27342
.027342	.27342

4.8.1.2 Exponent. The exponent determines the position of the decimal point when going from the normalized to the un-normalized representation. The sign of the exponent indicates which way the decimal should be moved when converting to the un-normalized form. A positive exponent means the decimal should be moved right the specified number of positions. A negative exponent means the decimal should be moved left the specified number of positions by inserting zeros between the decimal and the most significant non-zero digit. For example the same two numbers are again shown with their exponents:

UN-NORMALIZED	FLOATING POINT REPRESENTATION
27.342	$.27342 \times 10^{2}$ --- positive exponent
.027342	$.27342 \times 10^{-1}$ --- negative exponent
	mantissa exponent

It is possible to tell which of the two normalized floating point numbers is larger by comparing the exponent parts first; and then testing the mantissas only if the exponents are equal.

Another consideration involves the signing of numbers. Shown below is a positive and a negative number and their corresponding floating point representation.

UN-NORMALIZED	FLOATING POINT REPRESENTATION
-.006134	$-.6134 \times 10^{-2}$
613.4	$.6134 \times 10^{3}$

Now it can be seen there are two signs (exponent and mantissa) to keep track of. To reduce the number of signs, the exponents are biased by a positive value so as to eliminate its sign. For example, if the exponents are biased by 64, we would say we are using excess-64 notation. The same two numbers used above are shown below using excess-64 notation.

UN-NORMALIZED	FLOATING POINT EXCESS-64 REPRESENTATION
-.006134	$-.6134 \times 10^{62}$
613.4	$.6134 \times 10^{67}$

It is obvious that the correct exponent (movement of the decimal) can be determined by subtracting the bias factor 64. All negative biased exponents will have values less than 64 and all positive exponents will have values greater than 64. In cases where the exponent is equal to 64, the mantissa is also the un-normalized representation of the number.

Example:

NORMALIZED	FLOATING POINT EXCESS-64 REPRESENTATION
.6134	$.6134 \times 10^{64}$

4.8.2 Floating-Point Operations

When using floating point operations it is sometimes helpful to understand what takes place when a prescribed operation is performed. For example when adding or subtracting two floating point numbers the exponents are first compared and the decimal point is moved left in the mantissa with the smaller exponent until the exponents are equal. The mantissas are then added. Consider the addition of 54321.09 and 12.34 in floating point.

	UN-NORMALIZED	FLOATING POINT EXCESS-64 REPRESENTATION		
	54321.09	$.5432109 \times 10^{69}$	=	$.5432109 \times 10^{69}$
+	12.345	$.12345 \times 10^{66}$	=	$.0001234 \times 10^{69}$
	54333.435			$.5433343 \times 10^{69}$

An interesting point here is that the low order digit(5) of 12.345 has been lost in the movement of the decimal point because the mantissa countained only seven digits. In extreme cases if one number is a great deal smaller than the other it could be lost completely and the sum would not reflect any change. The user should be aware of the difference in magnitude of the numbers in use. As a general rule, a number that is several orders of magnitude smaller than another will be lost during floating-point operations.

In some addition and subtraction operations the operation itself will cause the exponent to be different than that of the two operators. In the first example shown below an overflow occurs and the exponent is incremented by one.

UN-NORMALIZED	FLOATING POINT EXCESS-64 REPRESENTATION
71	$.71 \times 10^{66}$
+ 83	+ $.83 \times 10^{66}$
154	1.54×10^{66}
	= $.154 \times 10^{67}$

The example below shows the exponent being decremented two after the subtract operation.

UN-NORMALIZED	FLOATING POINT EXCESS-64 REPRESENTATION
	$.1234 \times 10^{68}$
1234	
- 1200	$- .1200 \times 10^{68}$
34	$.0034 \times 10^{68}$
	$= .34 \times 10^{66}$

This process of normalization after the operation is called post-normalization.

By subtracting two numbers which are nearly equal the number of significant digits will be very small even though the number of digits in the mantissa is several times greater. For this reason blind subtraction of floating point numbers which are almost equal is undesirable if the original operands were very hard to obtain.

Multiplication and division of floating point numbers is accomplished by first adding and subtracting the exponents and then subtracting the bias. Secondly the desired operation is performed on the mantissas. Shown below is an example of floating point multiply.

UN-NORMALIZED	FLOATING POINT EXCESS-64 REPRESENTATION	
	$.565 \times 10^{67}$	
565		
	$\times .15 \times 10^{66}$	
X 15		
	$= .565 \times .15 \times 10^{67+66}$	
8475		
	$= .565 \times .15 \times 10^{133-64}$	(subtract bias)
	$= .565 \times .15 \times 10^{69}$	
	$= .08475 \times 10^{69}$	
	$= .8475 \times 10^{68}$	(post normalization)

Shown below is an example of a floating point divide operation:

UN-NORMALIZED	FLOATING POINT EXCESS-64 REPRESENTATION	
	$.555 \times 10^{67} / .4 \times 10^{65}$	
$555 / 4 = 138.75$	$= (.555 / .4) \times 10^{67-65}$	
	$= (.555 / .4) \times 10^{2+64}$	
	$= (.555 / .4) \times 10^{66}$	(add bias)
	$= 1.3875 \times 10^{67}$	
	$= .13875 \times 10$	(post normalization)

4.8.3 Internal Representation of TM 990/1481 Floating-Point Numbers

Floating point arithmetic executing on the TM 990/1481 is done in a slightly different manner than the previous examples have shown. Instead of using base ten as the basis for arithmetic the TM 990/1481 does all floating point arithmetic in base sixteen (hexadecimal). The reason for this is that digital computers can perform operations much quicker when they use a multiple of base two. Therefore, in order to understand the floating-point formats, the user must be able to convert decimal quantities to hexadecimal.

The following procedure can be used to convert a number from base 10 to base 16. In order to illustrate this procedure, an example showing the conversion of 458.765625_{10} to its hexadecimal equivalent will be included.

1. Divide the number into integer and fractional parts.

2. Convert the integer and fractional parts of the decimal number to their hexadecimal equivalents.

3. Conversion of the integer part.

   (1) Divide the integer part by 16 and note the remainders in a separate column (See Example 1).

   (2) Continue the division process until the quotient is zero.

   (3) Convert any remainders greater than 9 to hex value (eg. 10 = A, 11 = B, etc.).

   (4) Read the hexadecimal equivalent from bottom (MSB) to top (LSB).

Example 1: Convert integer part (458) to its' hexadecimal equivalent.

Solution:

	Remainder Base 10	Remainder Base 16		
28 / 16 ⟌ 458	10	A	(LSB)	↑
1 / 16 ⟌ 28	12	C		|
0 / 16 ⟌ 1	1	1	(MSB)	Read Value

$\therefore 458_{10} = 1CA_{16}$

4. Conversion of the fractional part.

   (1) Multiply the fractional part by 16 and note the integer overflow in a separate column (See Example 2).

   (2) Continue the multiplication process until an overflow with no fraction occurs.

   (3) Convert any overflow values greater than 9 into hex values.

   (4) Read the hexadecimal equivalent from top (MSB) to bottom (LSB).

Example 2: Convert fractional part (.765625) to its' hexadecimal equivalent.

Solution:

	Overflow Base 16	Overflow Base 10	
			.765625
			X16
Read Value (MSB)	C	12	.250000
↓			.250000
			X16
(LSB)	4	4	.000000

$\therefore .765625_{10} = .C4_{16}$

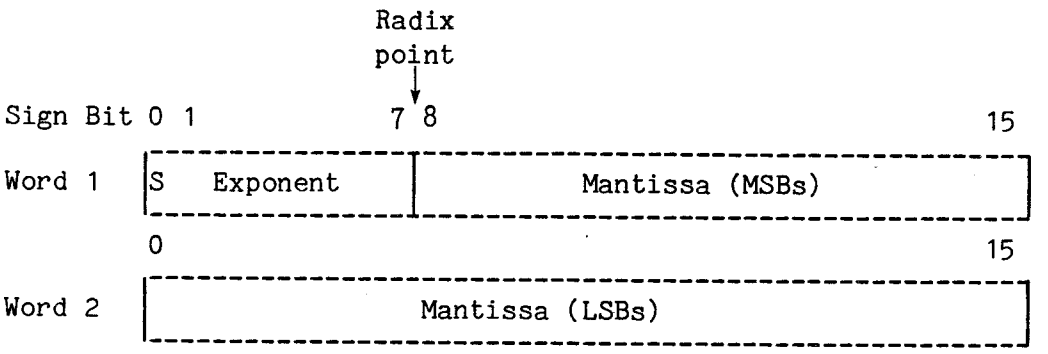
Therefore, the hexadecimal equivalent of $458.765625_{10} = 1CA.C4_{16}$.

NOTE

There are mathematical tables that are available to obtain the desired base conversions - the previous method was given only as an aid in the absense of such tables.

4.8.3.1 TM 990/1481 Single Precision Floating-Point Numbers. Single precision floating point numbers represent any value within the approximate range 10 to 10 , including zero. Single precision floating point numbers are stored in memory in two consecutive numbers 16 bit words as shown below.

SINGLE PRECISION FLOATING POINT NUMBER



Word 1 contains the sign bit, the hexadecimal exponent, and the 8 most significant bits of the hexadecimal mantissa. The second word contains the sixteen least significant bits of the mantissa. The mantissa is normalized by hexadecimal characters with the assumption the radix point is between bits 7 and 8 of Word 1.

The hexadecimal exponent is biased by >40 (excess-64). Exponents of zero are represented by >40, except for the number zero. The number zero is represented with the exponent and mantissa both as zero. The exponent may be any value from >00 to >7F. Using >40 as the bias, these numbers represent exponent values from ->40 to >3F (16^{-64} to 16^{-63}). The seven exponent bits are stored in bits one thru seven of Word 1.

Bit 0 of the first word is used for a sign bit. When this bit is 0, the number is positive; when this bit is a 1, the number is negative.

The following procedure will illustrate the use of the single precision floating-point format. This example will represent 5280_{10} in this format.

Procedure:

1. Convert 5280_{10} to hexadecimal.

   $5280_{10} = 14A0_{16}$

2. Normalize the hexadecimal equivalent.

   $14A0_{16} = .14A0 \times 16^4$ ←——Normalized floating-point representation

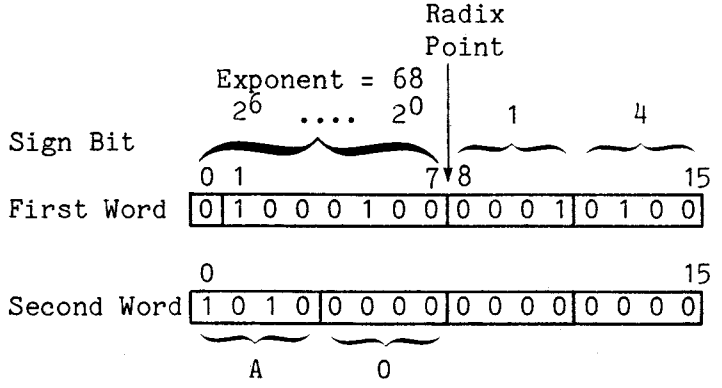
3. Bias the exponent using Excess-64 representation.

   $.14A0 \times 10^4 = .14A0 \times 16^4 \times 16^{40} = .14A0 \times 10^{44}$ ←Excess 40_{16} Rep.

4. Convert exponent to binary.

   $44_{16} = 1000100_2$

5. Assign values to two-word format as indicated below.



Additional examples of single precision numbers are given below.

BASE 10 NUMBER	HEXADECIMAL CONTENTS OF MEMORY WORDS WORD 1	WORD 2
1.0	4110	0000
0.5	4080	0000
5280.0	4414	A000
.0000067353	3C71	0000
-.1210976839	401F	0042

The following procedure can be used to convert the two-word contents into its decimal value.

1. Subtract the bias (64) from the exponent.

2. Move the radix point to the right four places for each power of 16 remaining in the exponent after the bias was removed.

3. Convert the binary value to decimal.

To illustrate this procedure, the two word example given above will be re-converted to its decimal equivalent.

1. $68_{10} - 64_{10} = 4_{10}$ (Exponent with bias removed)

   Radix
   Point
   ↓

2. Word 1 X X X X X X X X 0 0 0 1 0 1 0 0

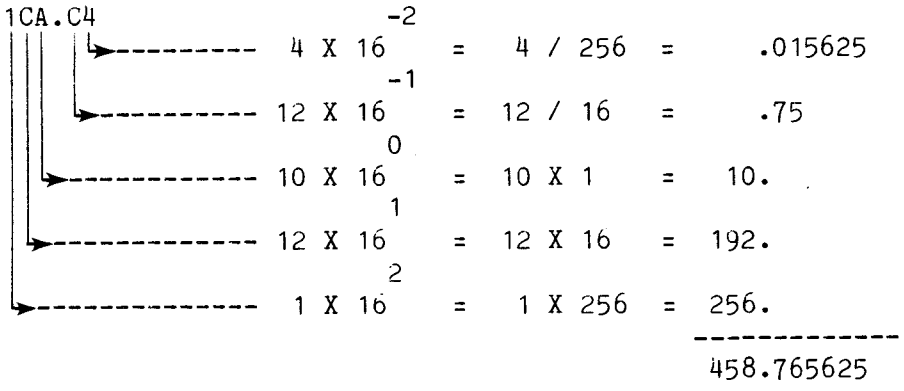
   Word 2 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

   ↑
   Radix Point Moved 16 Places

3. $1010010100000_2 = 5280_{10}$

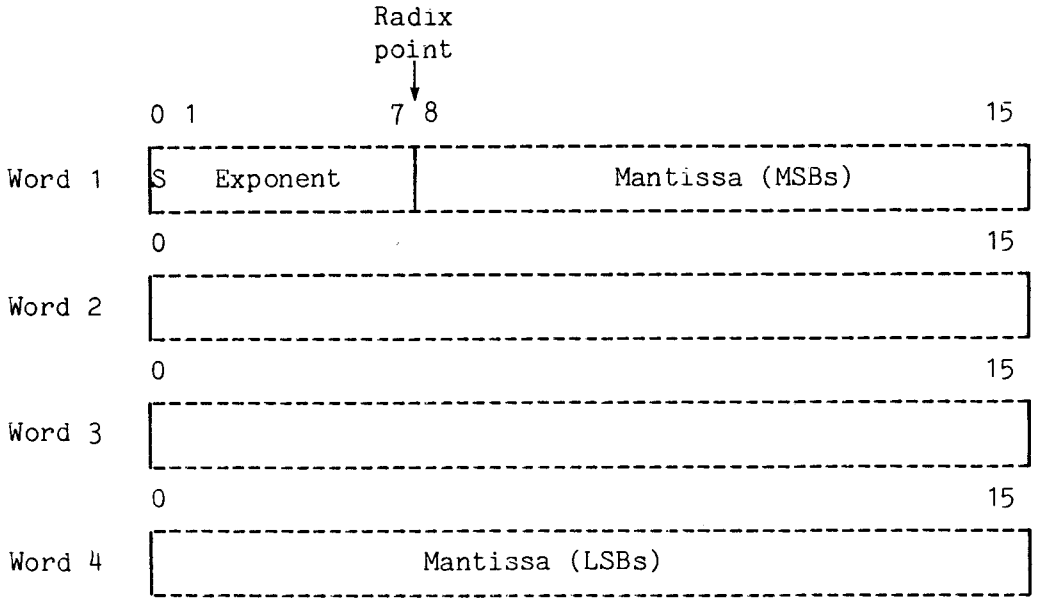
Another method to convert from hexadecimal to decimal is illustrated by the following example.

HEXADECIMAL TO DECIMAL CONVERSION



4.8.3.2 Double Precision Floating-Point Numbers. Double precision floating point numbers are similar to single precision floating point numbers, except that they occupy two more memory words and provide a 56 bit mantissa instead of the 24 bits available with single precision floating point numbers. Double precision floating point numbers have values from 10^{-78} to 10^{75}, including zero. Double precision floating point numbers are stored in four 16 bit words as shown below.

DOUBLE PRECISION FLOATING POINT NUMBER



Word 1 of the double precision floating point number is exactly like Word 1 of the single precision floating point number. Words 2, 3, and 4 contain the remainder of the mantissa with Word 4 containing the 16 least significant bits. The mantissa is normalized by hexadecimal characters with the assumption the radix point is between bit 7 and 8 of Word 1.

4.8.3.3 TM 990/1481 Floating Point Accumulator (FPA). Many microprocessors contain an hardware accumulator; this accumulator may be used for intermediate storage, to form sums, or other intermediate operations. TMS 9900 series microprocessors do no contain an accumulator inherent to the processor; storage for sums and the like is provided by workspace registers that reside in RAM memory external to the processor.

Arithmetic floating-point operations involve two operands; one of these exists in an implicit "accumulator" register created by the results of a load instruction or a previous calculation. The implicit accumulator acts as a single register that participates in all floating point operations as either an operand or result, or both. The outcome of all floating point operations (except the store operations), is placed in the implicit accumulator. Single precision floating point instructions use R0 and R1 of the current workspace as the FPA, leaving R2 and R3 unaltered. Double precision floating point instructions use R0, R1, R2, ans R3 of the current workspace as the FPA.

4.8.4 TM 990/1481 Floating-Point Instruction Overview

As mentioned previously, the TM 990/1481 floating-point instructions can be divided into three functional groups; the instructions that comprise these groups are given below.

Arithmetic/Logic

1. Add (AD, AR)
2. Divide (DD, DR)
3. Multiply (MD, MR)
4. Subtract (SD, SR)
5. Negate (NEGD, NEGR)

Conversion

1. Convert Floating-Point To Integer (CDI, CRI)
2. Convert Floating-Point To Extended Integer (CDE, CRE)
3. Convert Integer To Floating-Point (CID, CIR)
4. Convert Extended Integer To Floating-Point (CED, CER)

Load/Store

1. Load (LD, LR)
2. Store (STD, STR)

It should be noted that there are two types of instructions for each function (eg., there are two ADD instructions: AD and AR). The AD instruction is used for adding double precision floating-point numbers while the AR instruction is used with single precision floating-point numbers. All eleven pairs of these instructions exhibit this same type of duality. It should be noted that the execution time for double precision floating-point numbers exceeds that for single precision floating-point numbers.

Detailed descriptions of these instructions that illustrate their format and application are given in Appendix H.

4.8.5 Sample Programs

Two sample programs will be included to demonstrate some of the uses of floating-point instructions. The first program adds two numbers, then converts the floating-point sum to an integer. In other words, after two numbers are summed, the integer part of the sum is placed in the FPA. The second example also illustrates the addition process; however, double precision floating-point numbers are used.

Problem: Add the numbers 458.765625_{10} and -256.765625_{10} using single precision floating-point arithmetic. Place the integer part of the sum in the FPA (R0,R1).

Solution:

I. One operand (458.765625_{10}) will be placed in the FPA. The first step is to determine the contents for registers R0 and R1 which constitute the FPA for single precision floating-point numbers.

Procedure:

A. Convert the decimal value of the operand into its hexadecimal equivalent, normalize, and express using Excess-64 representation.

(1) $458.765625_{10} = 1CA.C4_{16}$

(2) $1CA.C4_{16} = .1CAC4 \times 10^{3}$

(3) $.1CAC4 \times 10^{3} = .1CAC4 \times 10^{67}$

B. Place the sign bit for the number and the values from the Excess-64 represented value in the two word single precision floating-point format as shown below.

	Sign Bit	0	Radix Point ↓ 7 8	15		
Word 1	0	1 0 0 0 0 1 1	0 0 0 1	1 1 0 0	>431C	Contents for R0
Word 2	1 0 1 0	1 1 0 0	0 1 0 0	0 0 0 0	>AC40	Contents for R1

It should be apparent that Word 1 (>431C) constitutes the contents for R0 and Word 2 (>AC40) constitutes the contents for R1.

II. The second operand (-256.765625_{10}) will be placed in registers R2 and R3. The procedure for determining the contents for these registers is identical to that previously used for the FPA (R0,R1) and will not be repeated here. The two word content representing the second operand is given below.

	Sign Bit	0	Radix Point ↓ 7 8	15		
Word 1	1	1 0 0 0 0 1 1	0 0 0 1	0 0 0 0	>C310	Contents for R2
Word 2	0 0 0 0	1 1 0 0	0 1 0 0	0 0 0 0	>0C40	Contents for R3

Word 1 (>C310) will be placed in R2 and Word 2 (>0C40) will be placed in R3.

The following code will implement the addition process called for and place the integer part of the sum in the FPA (R0, R1).

```
LWPI >FE00          Load Workspace Pointer
LI   R0,>431C       Load FPA (R0,R1) with >431CAC40
LI   R1,>AC40
LI   R2,>C310       Load R2,R3 with >C3100C40
LI   R3,>0C40
AR   R2             Add Single Precision Real, Place Sum  in FPA
CRI                 Convert Single Precision Real Sum to Integer
END
```

The second sample program will illustrate the use of double precision floating-point numbers. As the procedure is similar to that used with single precision floating-point numbers, only an abbreviated explanation will be given.

The FPA consists of registers R0 through R3 when working with double precision floating-point numbers. Therefore, four registers will be used for each operand.

In the sample of code given below, two double precision floating-point numbers will be added. One operand is loaded into the FPA (R0-R3) and the other operand is loaded into registers R4-R7. The sum will be placed in the FPA (R0-R3).

```
LWPI >FE00                                  Load Workspace Pointer
LI   R0,>Value of Word 1 of Operand 1       Load the 4-Word Value
LI   R1,>Value of Word 2 of Operand 1       Representing Operand 1
LI   R2,>Value of Word 3 of Operand 1       Into the FPA (R0-R3)
LI   R3,>Value of Word 4 of Operand 1
LI   R4,>Value of Word 1 of Operand 2       Load the 4-Word Value
LI   R5,>Value of Word 2 of Operand 2       Representing Operand 2
LI   R6,>Value of Word 3 of Operand 2       Into Registers R4-R7
LI   R7,>Value of Word 4 of Operand 2
AD   R4                                     Add DP Real, Place Sum in FPA
END
```

4.9 PROGRAMMING AIDS

The TM990/1481 provides a number of features to aid the programmer in software development.

1) The compatibility of TM990/1481 software with the 990 family of computers means that a large amount of software can be used with little modification.

2) The TM 990/403 version of the TIBUG monitor can be installed on a TM 990/201-44 card. This EPROM resident software monitor allows the programmer to load, edit, debug, and run programs right from power-up.

3) A single step clock allows stepping thru microcode and examination of the result at every step using the optional user supplied LED displays. Recommended LED's are Dialight #547-2007. See paragraphs 2.4.4.3 and 2.4.4.4.

4.10 INTERRUPTS

4.10.1 General. The interrupt logic provides 18 levels (i.e. 18 different trap locations) of interrupt, 16 of which are maskable and two of which are not maskable. The standard set of interrupts consists of 15 interrupt lines on the TM 990 BUS with levels 1 thru 15. The levels are prioritized with level 0 being the highest priority interrupt. Interrupts 1 to 15 and XOPS.

All of the 15 interrupts are individually enabled by setting a bit in a mask register (in the TM 9901) via the CRU. They are also group maskable via the four-bit interrupt mask in the Status Register (ST12-ST15). If the value of the Interrupt Vector (IV) is less than or equal to the value of the mask then the interrupt is allowed (i.e. the IV is higher in priority). The IV represents the level of the highest priority interrupt that is pending. When an interrupt occurs the TM 990/1481 completes the current instruction, fetches a pair of words (WP and PC) called a Transfer Vector from locations 4*IV and (4*IV)+2, then executes the equivalent of a BLWP instruction using the Transfer Vector.

In addition to the general set of 15 interrupts there are three special interrupts; RESET, LOAD, and ARITHMETIC OVERFLOW. The RESET is a level 0 interrupt and it is normally used to initialize the TM 990/1481 following a power-up. The RESET interrupt is initiated via the RESET SWITCH. When the RESET SWITCH is activated the RESET LOGIC forces the TM 990/1481 to start the microprogram execution at location 000 in CONTROL MEMORY when the switch is released. This causes the TM 990/1481 to fetch the interrupt Transfer Vector at memory locations 0000 and 0002 and then execute the equivalent of a BLWP using the Transfer Vector. Since the RESET is a level 0 interrupt it is not maskable.

The RESET interrupt does not wait until the current instruction completes execution and therefore should not be used as a normal program interrupt.

The LOAD interrupt is non-maskable and is initiated by a pulse on the RESTART line on the TM 990 BUS or by the execution of an LREX instruction. The LOAD interrupt on the TM 990/1481 is designed to allow exactly two instructions to be executed before the interrupt trap occurs. The LOAD interrupt fetches the Transfer Vector from the upper two words of logical memory at locations FFFC and FFFE and executes the equivalent of a BLWP. The TIBUG software monitor uses the LOAD interrupt to implement the single instruction step mode of operation. The monitor executes an LREX and then executes an RTWP to the user's program. The user's program can then execute one instruction before the LOAD interrupt occurs and traps back to the monitor.

The other use of the LOAD interrupt is as an alternate to the RESET operation. The RESET switch causes the equivalent of a level 0 interrupt which fetches the Transfer Vector from 0000 and 0002. This is suitable if lower memory is ROM and has valid data when power is turned on, but if this is the case, it also implies that all of the other Transfer Vectors are fixed by the ROM. In some cases it is desirable to have the Transfer Vectors in RAM so that they may be altered by the software. In order to accomplish this, the ROM is placed in upper memory and the RESTART signal is used rather than the RESET to initialize the TM 990/1481. Notice that there must be valid Transfer Vector data either in lower memory at the 0 level interrupt trap location or in upper memory at the LOAD interrupt trap location in order to correctly initialize the TM 990/1481. This can be satisfied by placing ROM in either upper or lower memory or by providing some special hardware to load the trap locations via the DMA interface.

The third special interrupt is the ARITHMETIC OVERFLOW (AO) interrupt or simply the overflow interrupt (OI) which can be enabled by setting bit 10 of the STATUS REGISTER. When enabled the AO interrupt will cause an interrupt trap whenever the OVERFLOW (OV=ST4) bit of the STATUS REGISTER is set. The AO interrupt is assigned to the level 2 interrupt. The AO interrupt trap will occur prior to the execution of the instruction following the one which caused the overflow condition to be generated.

* NOTE : XOP 12 (Write Character) and XOP 7 (Delay Timer) in TIBUG uses a level three interrupt, generated by the TMS 9901 programmable timer, when writing to a 733 ASR. It is therefore recommended that caution be taken by the user when using interrupt level three in software implementation.

4.10.2 Interrupt and XOP Linking Areas Using TM 990/403 TIBUG

This writeup applies to the interrupt and XOP scheme used by the TM 990/403 TIBUG. This scheme allows the user to define service routines for the following:

- Interrupts 1 to 15
- XOPs 2 to 6 (XOPs 1, and 7 to 15 are used by the TM 990/403 monitor)

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 TM 990/403 TIBUG 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 program control to the vectors which point to a linking area. The linking area directs execution to the actual ISR or XSR. The linking areas are shown in Table 4-10. The linking area is designed to leave as much RAM space free as possible when not using all the interrupts. That is, the most frequently used areas are butted up against the TIBUG RAM 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 as well as at the linking area workspace.

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

TABLE 4-10. PREPROGRAMMED INTERRUPT AND USER XOP TRAP VECTORS

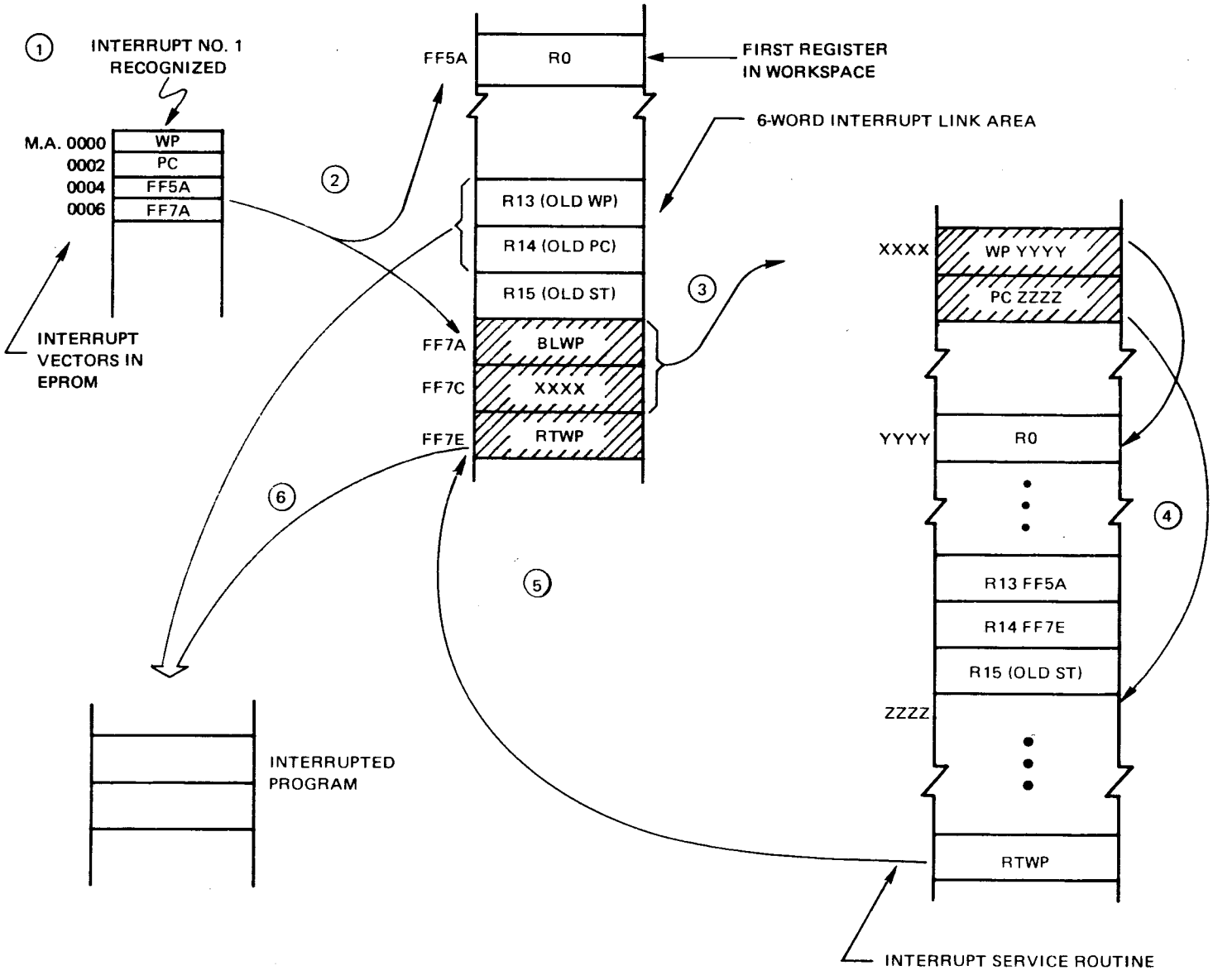
M.A.	INT.	VECTORS (HEX) WP	VECTORS (HEX) PC	M.A.	XOP	VECTORS (HEX) WP	VECTORS (HEX) PC
0000	INT0	TIBUG	TIBUG	0048	XOP2	FF48	FF5A
0004	INT1	FF5A	FF7A	004C	XOP3	FF3A	FF4C
0008	INT2	FF4E	FF6E	0050	XOP4	FF2C	FF3E
000C	INT3	FF8A	FFAA	0054	XOP5	FF1E	FF30
0010	INT4	FF7E	FF9E	0058	XOP6	FF10	FF22
0014	INT5	FF72	FF92				
0018	INT6	FF66	FF86				
001C	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 4-11. 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
				USER RAM AREA				
FE90								
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
FE10	INT7	INT7						
FE20		XOP6	XOP6	XOP6	XOP6	XOP6	XOP6	XOP6
FE30	XOP5	XOP5	XOP5	XOP5	XOP5	XOP5	XOP5	XOP4
FE40	XOP4	XOP4	XOP4	XOP4	XOP4	XOP4	XOP3	XOP3
FE50	XOP3	XOP3	XOP3	XOP3	XOP3	XOP2	XOP2	XOP2
FE60	XOP2	XOP2	XOP2	XOP2	INT2	INT2	INT2	INT2
FE70	INT2	INT2	INT1	INT1	INT1	INT1	INT1	INT1
FE80	INT6	INT6	INT6	INT6	INT6	INT6	INT5	INT5
FE90	INT5	INT5	INT5	INT5	INT4	INT4	INT4	INT4
FEA0	INT4	INT4	INT3	INT3	INT3	INT3	INT3	INT3
FFB0 to FFFE = TIBUG workspace and LOAD (NMI) vectors								

4.10.2.1 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 4-16 and 4-17. 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 a RTWP instruction code (all three words to be entered by the user). When the ISR is completed, control returns to the RTWP instruction in this this linking area after the return values (to the interrupted program) are loaded into the linking area's three registers (R13 to R15). When the interrupt occurs, control is given to the program at the interrupt vectors. The PC vector points to the BLWP instruction (at the PC vector address) which is executed using the destination address 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.)

In returning from the interrupt service routine, the RTWP instruction (routine's last instruction) 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 area is shown graphically in Figure 4-17.



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 4-16. 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 4-10.
- 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 4-17.
- At the vector address for the BLWP operand, place the WP and PC values respectively of the interrupt service routine.

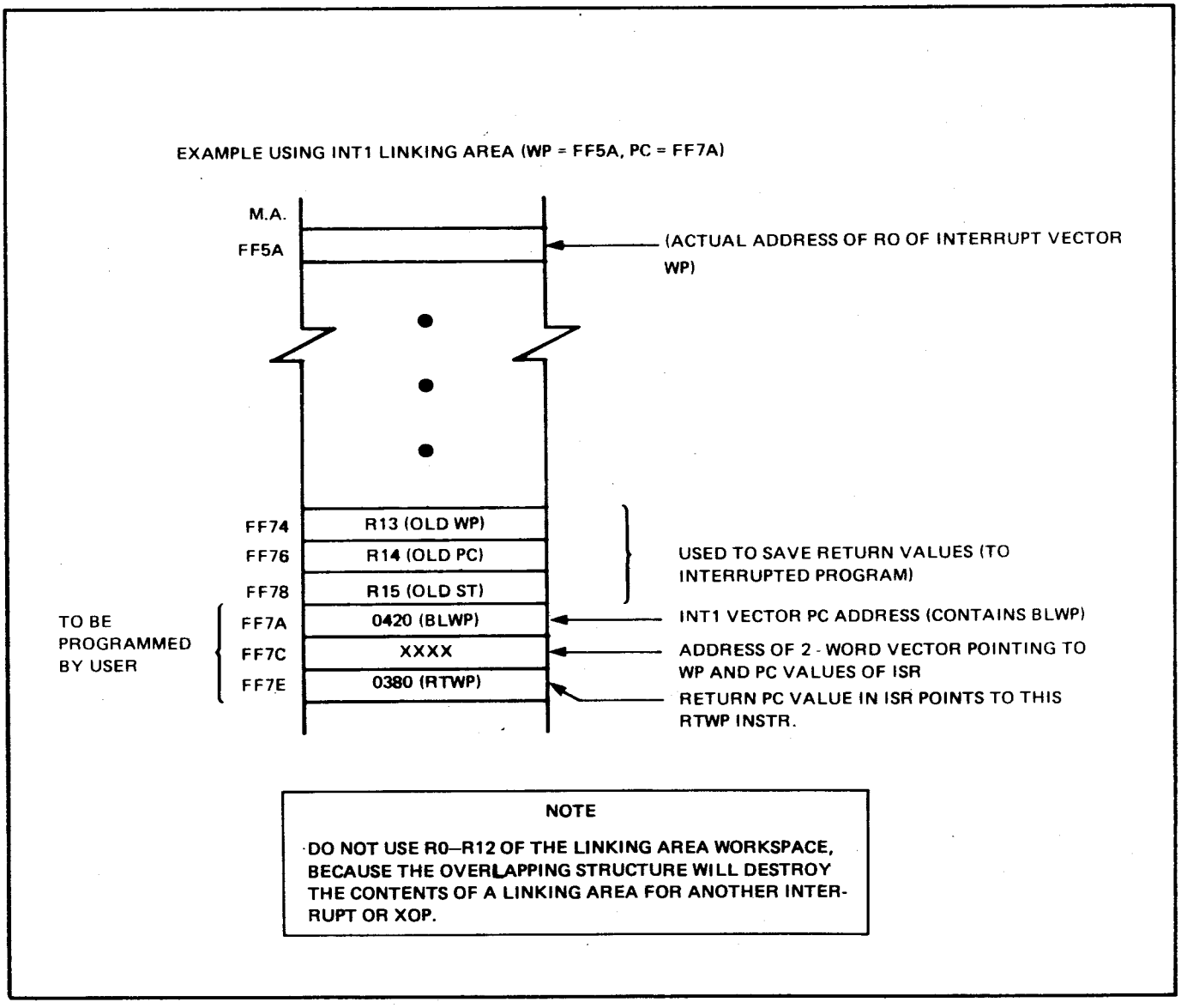


FIGURE 4-17. SIX-WORD INTERRUPT LINKING AREA

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.

4.10.2.2 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 4-18.

For example, when XOP 2 is executed, the PC vector points to the BLWP instruction shown at M.A. $FF5A_{16}$ in Figure 4-18. This executes, transferring control to the preprogrammed WP and PC values at the address in the next word (YYYY as shown in Figure 4-18). To obtain the parameter passed to R11 of the vector WP (M.A. $FF5E_{16}$ in Figure 4-18), 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.

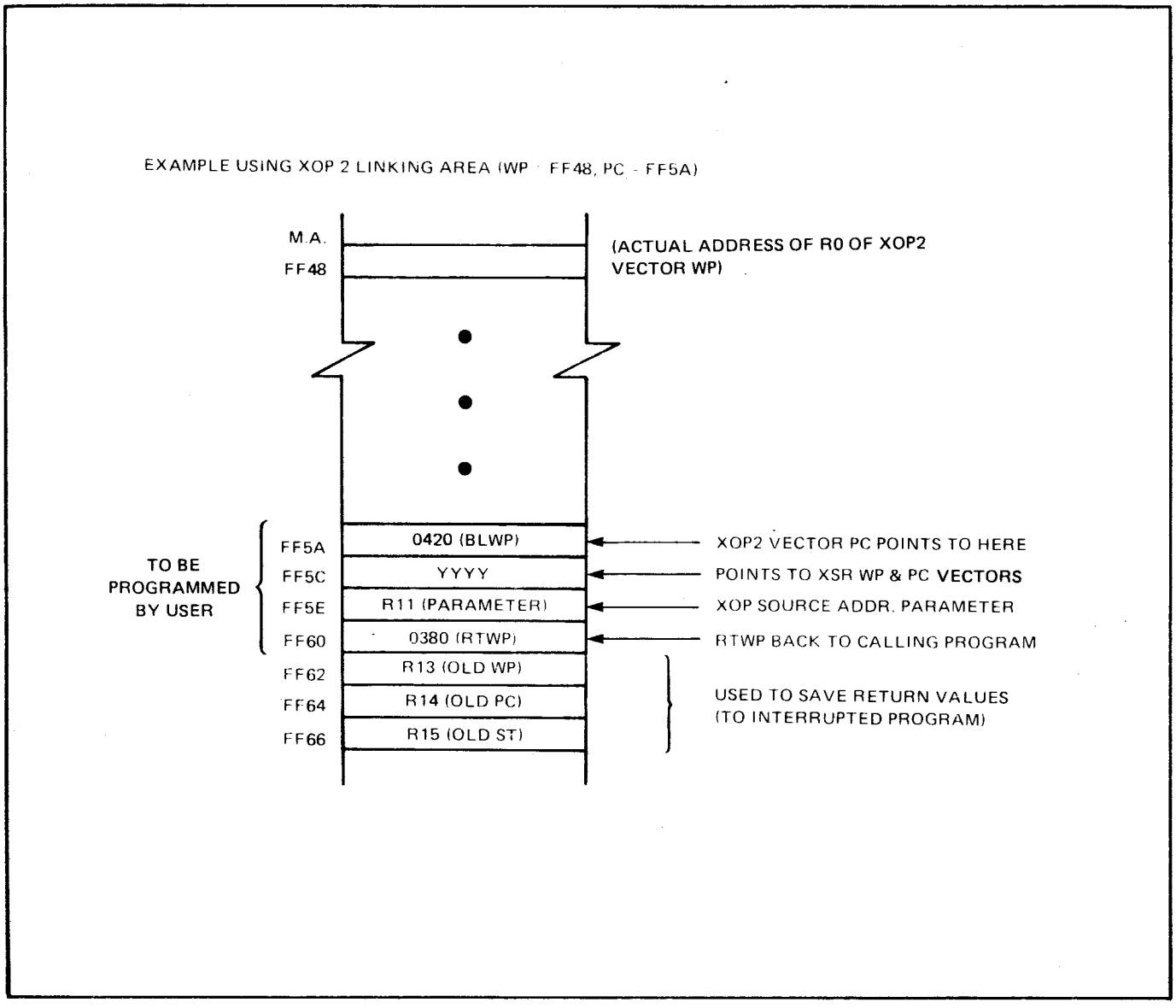


FIGURE 4-18. SEVEN-WORD XOP INTERRUPT LINKING AREA

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 4-10.
- 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 insert 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 4-18 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 has been incremented to point 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.

SECTION 5

SOFTWARE DEVELOPMENT AND APPLICATIONS

5.1 GENERAL

This section covers the various means for developing user software on different software development systems (section 5.2) and the means to install user software on to a TM 990/1481 system using EPROM, cassette, tape, and floppy disk (section 5.3). Also included are debugging hints (section 5.4) and various aspects of floating point operations including the TM 990/433 Demonstration Software (sections 5.5 to 5.7).

5.2 DEVELOPMENT OF SOFTWARE FOR THE TM 990/1481

Floating point instructions can be assembled on Texas Instruments minicomputers which operate under the DX10 disk based operating system. These instructions can be assembled on the following assembler:

- SDSMAC, release 3.3

All but four instructions (listed next) used on the TM 990/1481 can be assembled on the following assemblers:

- SDSMAC, release 3.2
- TXMIRA, release 2.4
- AMPLUS, release 1.0

The four instructions not assembled on the above three systems are:

- LST, load status register
- LWP, load workspace
- SDIV, signed divide
- SMUL, signed multiply

In addition, the user can employ DATA statements in order to create floating point object and source statements. The method of doing this is explained in paragraph 5.2.2.

NOTES

1. The AMPL system does not provide in-circuit emulation of the SN74S481 chips.

2. All memory spaces can be defined by the user. If the TM 990/403 TIBUG is resident, it occupies memory spaces (in EPROM) from 0000_{16} to 1000_{16} and requires RAM from $FE50_{16}$ to $FFFF_{16}$.

5.2.1 Using Floating Point Instruction Assemblers

Figure 5-1 is an example of code generated on one of the systems that assemble floating point instructions.

```
0002                        IDT   'SAMPLE'
0003        FABC    LOC     EQU   >FABC
0004                *
0005                *=======> SHOW OPCODES FOR FLOATING POINT INSTRUCTIONS
0006                *
0007  0000  0C44           AR    R4            ADD REAL R4,R5 TO R0,R1
0008  0002  0CE0           SR    @LOC          SUBTRACT CONTENTS OF LOC
      0004  FABC
0009                *                         AND LOC+2 FROM R0,R1
0010  0006  0DCA           STR   R10           STORE REAL NUMBER IN R0,R1
0011                *                         IN R10,R11
0012  0008  0D96           LR    *R6           LOAD 2 WORDS POINTED TO BY
0013                *                         R6 INTO R0,R1
0014  000A  0E46           AD    R6            ADD DOUBLE PRECISION REAL
0015                *                         R6-R9 TO R0-R3
0016                        END
```

FIGURE 5-1. SOURCE LISTING OF ASSEMBLER USING FLOATING POINT SOURCE

```
0002                        IDT   'SAMPLE1'
0003        0004    R4      EQU   4
0004        0006    R6      EQU   6
0005        000A    R10     EQU   10
0006        0C40    ARCODE  EQU   >0C40              BASIC AR OPCODE
0007        0CC0    SRCODE  EQU   >0CC0              BASIC SR OPCODE
0008        0DC0    STRCOD  EQU   >0DC0              BASIC STR OPCODE
0009        0D80    LRCODE  EQU   >0D80              BASIC LR OPCODE
0010        0E40    ADCODE  EQU   >0E40              BASIC AD OPCODE
0011        0010    INDECT  EQU   >0010              INDIRECT CODE
0012        0020    INDEX   EQU   >0020              INDEX CODE
0013        FABC    LOC     EQU   >FABC              DUMMY LOCATION
0014                *
0015                *=======> SHOW FLOATING POINT INSTRUCTIONS
0016                *
0017  0000  0C44           DATA  ARCODE+R4           AR    R4
0018  0002  0CE0           DATA  SRCODE+INDEX        SR    @LOC
0019  0004  FABC           DATA  LOC
0020  0006  0DCA           DATA  STRCOD+10           STR   R10
0021  0008  0D96           DATA  LRCODE+INDECT+R6    LR    *R6
0022  000A  0E46           DATA  ADCODE+R6           AD    R6
0023                        END
```

FIGURE 5-2. SOURCE LISTING OF ASSEMBLER USING DATA STATEMENTS

5.2.2 Floating Point Support with Other Assemblers

Assemblers that do not assemble floating point instructions can still be used to generate floating point code using the DATA assembler directive. Prior to entering the DATA statement, the user must first hand assemble the desired floating point instruction to determine the correct opcode. Figure 5-2 shows the same program assembled in Figure 5-1 only it is assembled without using floating point source statements. As can be seen, the opcodes and pertinent data are first identified in EQU directives, then the assembler is used to combine the opcode with values such as index bit, indirect bit, and register value. After assembling, the user should scrutinize code for correctness.

Assembling using DATA statements must be used if assembling with the TM 990/302 software development module or the Cross Support Package.

5.3 INSTALLING SOFTWARE INTO THE TM 990/1481 SYSTEM

The TM 990/1481 module does not have user memory onboard; thus, the system must utilize a memory expansion board to contain user software, either in EPROM or RAM. Memory boards available include the TM 990/203 dynamic RAM board and the TM 990/201-44 EPROM/static RAM board.

User-written software may be installed and executed on a TM 990/1481 system in one of the following ways:

1) Burn software into EPROM firmware and install the EPROMs on a memory board. A RESET or LOAD can be used to start the system. The user will be responsible for placing the correct RESET vectors at 0000_{16} and 0002_{16} or LOAD vectors at $FFFC_{16}$ and $FFFE_{16}$ for the WP and PC. A RESET will be issued by actuating the RESET switch. A LOAD can be initiated by enabling the RESTART- signal at connector pin J1-93.

2) Install a loader EPROM. This is similar to the EPROM in 1) above except that it does a load function upon initialization. Loading could be from media such as digital cassette, paper tape, audio cassette, floppy disk, or communication link. An applicable device service routine must be included in the firmware.

3) Install a TIBUG monitor (e.g., TM 990/403) on a memory board and load user software from cassette or paper tape using the TIBUG "L" command. Then, use other interactive TIBUG commands to execute the user programs.

4) The system may be bootloaded from floppy disk using the TM 990/303A disc controller board. This bootload feature will occur upon powerup. The disk controller will read a command from a preformatted diskette, execute the command (such as read user software from diskette to user memory), then give control to the microcomputer board which goes through a level zero (powerup) interrupt. It is important that the diskette be formatted with the desired command at the proper diskette location and also contain the necessary data required by the user. Also, the level zero interrupt handler must be cognizant of the bootstrap command features such as location of data loaded from diskette, etc. Note that only a TM 990/303A system can be used to format the disk as required for the TM 990/303A bootload feature (other systems will use the bootload area in diskette formatting). Format requirements for the diskette are explained in the TM 990/303A Floppy Disk Controller User's Guide.

5.4 DEBUGGING SOFTWARE ON THE TM 990/1481

When debugging software to run on the TM 990/1481, several approaches should be considered. The approach used depends greatly upon the application.

When considering the application, the user should know if the floating point instructions are utilized or whether the operating environment is real time. The floating point instructions may only be debugged two ways on the TM 990/1481 or a TI 990/12 minicomputer. These are the only processors which currently execute the floating point instructions.

The floating point instructions can be debugged on the TM 990/1481 by using a debug package such as the TIBUG monitor. These instructions can be debugged on the TI 990/12 by using its interactive debugger.

Real-time applications can only be debugged by having it loaded and run in the target environment, the TM 990/1481. This is because the application software is reacting to external events and controlling them. Real-time debugging cannot be done on the TI 990/12 because the debugger is one of the many tasks under the operating system.

In applications where real-time control is not used and floating point instructions are not utilized, many debug tools are available. These include:

- Debugger software resident in the TM 990/1481 system
- Interactive debugger software on TI 990 minicomputer systems such as the /4, /5, /10, or /12.

A great deal of debugging can be performed on other machines which execute the same instruction set; however, the user must be aware that certain items such as instruction timing and interrupts vary from processor to processor.

5.5 CHARACTERISTICS OF FLOATING POINT ARITHMETIC

5.5.1 Accuracy Considerations

Floating point algorithms must be written with the goal of minimizing errors rather than eliminating them. Some errors will remain in all floating point calculations due to the finite bit length of the floating point representation. An acceptable solution is to ensure that the small error which can occur on any given floating point operation does not "snowball" so that the final result is not obscured by noise.

An inherent consequence of digital floating point arithmetic is that the associative law does not hold:

(A + B) + C is not equal to A + (B + C) for all values of A, B, and C.

As a result, summing operations concerning elements diverse in magnitude should be arranged so that the smallest elements are added first. This is so the information contained in the least significant bits of the smallest elements are properly accumulated and have the proper impact on the the result.

The associative property does hold for floating point multiplication. The order in which divisors are used on a dividend is not important. However, seemingly equivalent combinations of multiplications and divisions will not yield equal results in all cases. For example:

A * (B / C) is not equal to (A * B) / C for all values of A ,B, and C.

Hence the order of multiplication and division operations should be judiciously chosen to simultaneously minimize error and prevent overflow or underflow.

In general, a small loss of accuracy is to be expected from multiplication or division operations while a substantial loss of accuracy may occur from addition or subtraction operations.

5.5.2 Significant Decimal Digits

The number of significant decimal digits which can be represented by a floating point format is governed by the length of the mantissa. The single precision mantissa is 24 bits long. The largest decimal number which can be represented in 24 bits is $2^{24}-1$ or 16,777,215 which would infer 8 digits of accuracy. The eighth digit will carry very little information. Therefore, it is more realistic to assume 7 digits of accuracy. The largest number which can be represented in the 56 bit double precision mantissa is $2^{56}-1$ or 72,057,594,037,927,935. Again, because of partial information in the least significant digit, 16 digits of accuracy should be assumed.

5.5.3 Range of Value

The range of possible values for both single and double precision is approximately 5.398×10^{-79} to $7.237 \times 10^{+75}$ respectively.

5.5.4 Interrupt Considerations

Hardware floating point operations may interfere with real time sampling functions in certain cases because of their length compared to other instructions. The divide double precision instruction (DD) requires approximately 140 microseconds for execution. In contrast, the unconditional jump instruction (JMP) is executed in under 1 microsecond. Since an interrupt is not acknowledged until completion of the current instruction, an external interrupt may not be serviced for up to 140 microseconds during the execution of the DD instruction.

5.6 RADIX CONVERSION

The conversion between floating point machine representation and ASCII decimal notation is not a straight forward process since the exponent of the machine representation is of a hexadecimal base rather than a decimal base. The machine representation is of the form:

$$A * 16^B$$

while the desired ASCII representation is of the form:

$$C * 10^D$$

There are essentially two ways to perform the conversion:

1) Convert 16^B to $F * 10^D$. A * F is then C.

2) Multiply B by log 16 and round to the nearest integer D. Then divide $A * 16^B$ by 10^D to get C.

The hexadecimal quantities C and D can be converted to decimal by the methods discussed in section 4.8.3. The machine-representation-to-ASCII conversion routines which are described in section 5.6 use method 1, above.

5.7 TM 990/433 FLOATING POINT DEMONSTRATION SOFTWARE

The TM 990/433 floating point demo package is available to illustrate the use of the hardware floating point instructions. It must be used in conjunction with the TM 990/403 TIBUG. It resides in two 2716 EPROMs which must be installed at hex address 1000. The demo software is not intended to be a user utility library; however, user written programs may access the various routines by linking to the appropriate entry point.

The floating point demo monitor may be initialized by a special TIBUG command, the G command, which requires no arguments. Alternatively, the monitor may be initialized by executing a simple branch to address hex 1000. Once initialized, the TM 990/433 requests a command at the system terminal connected to the controller module. The monitor allows the individual demo routines to be run interactively. The various demo routines and their interactive command mnemonics are described in Table 5-1. Included are the two commands HELP and QUIT. The HELP command lists the demo commands and the QUIT command which causes a branch back to the TIBUG monitor.

The demo software package requires 136 bytes of RAM beginning at memory address $F000_{16}$.

TABLE 5-1. DEMO ROUTINES

FPTMON COMMAND	ROUTINE NAME	DESCRIPTION
AD	ADPFP	ASCII decimal to double precision machine representation
AI	AINT	ASCII decimal to integer machine representation
AR	AREAL	ASCII decimal to single precision machine representation
COS	COS	Single precision cosine function
DA	DASCII	Double precision machine representation to ASCII decimal
DCOS	DCOS	Double precision cosine function
DEX	DEX	Double precision exponential function
DSIN	DSIN	Double precision sine function
EX	EX	Single precision exponential function
HELP		Prints a list of demo software commands
IA	IASCII	Integer machine representation to ASCII decimal notation
MINV	MINV	Single precision simultaneous equations
QUIT		Branch back to TIBUG
RA	RASCII	Single precision machine representation to ASCII decimal
SIN	SIN	Single precision sine function

5.7.1 Accessing Demo Routines From Applications Programs

The assembly language listings for all of the demo routines are available in "TM 990/433 Demonstration Software for the TM 990/1481 Board". The calling sequences required by each routine are described in the listing. In general, it is of the following form:

```
BLWP   @entry point
DATA   <one argument>
<normal return>
```

All routines except MINV receive and/or return a floating point argument. This is taken or replaced in the caller's floating point accumulator. The SINE and COSINE routines require radian arguments.

Table 5-2 describes the arguments required by the respective routines and lists the entry points.

TABLE 5-2 ROUTINE ENTRY POINTS AND ARGUMENTS

ROUTINE	ENTRY POINT (Hex)	ARGUMENT
ADPFP	1CBE	Address of user buffer for ASCII input
AINT	1DB4	Address of user buffer for ASCII input
AREAL	1B76	Address of user buffer for ASCII input
COS	1972	None
DASCII	1BF8	Address of user buffer for ASCII result
DCOS	19A6	None
DEX	1A90	None
DSIN	192C	None
EX	1A6A	None
IASCII	1D50	Address of user buffer for ASCII result
MINV	1E12	None
RASCII	1ABA	Address of user buffer for ASCII result
SIN	18FA	None

5.7.2 Machine to ASCII Conversion Routines

Six demo routines provide conversion to and from the three important machine representations:

1) Integer

2) Real (single precision floating point)

3) Double precision floating point

The output of the machine-representation-to-ASCII routines can be read by the ASCII-to-machine-representation routines.

Example 5-1:

```
ENTER COMMAND (OR "HELP"):IA
ENTER THE NUMBER IN INTEGER MACHINE REPRESENTATION
XXXX
FFFF
-1
ENTER COMMAND (OR "HELP"):IA
ENTER THE NUMBER IN INTEGER MACHINE REPRESENTATION
XXXX
000A
+10
ENTER COMMAND (OR "HELP"):IA
ENTER THE NUMBER IN INTEGER MACHINE REPRESENTATION
XXXX
7FFF
+32767
ENTER COMMAND (OR "HELP"):IA
ENTER THE NUMBER IN INTEGER MACHINE REPRESENTATION
XXXX
8000
-32768
ENTER COMMAND (OR "HELP"):
```

Example 5-2:

```
ENTER COMMAND (OR "HELP"):AI
ENTER THE INTEGER ASCII NUMBER (FREE FORMAT)
-1
FFFF
ENTER COMMAND (OR "HELP"):AI
ENTER THE INTEGER ASCII NUMBER (FREE FORMAT)
100
0064
ENTER COMMAND (OR "HELP"):AI
ENTER THE INTEGER ASCII NUMBER (FREE FORMAT)
-256
FF00
ENTER COMMAND (OR "HELP"):AI
ENTER THE INTEGER ASCII NUMBER (FREE FORMAT)
-32767
8001
ENTER COMMAND (OR "HELP"):
```

Example 5-3:

```
ENTER COMMAND (OR "HELP"):AR
ENTER THE REAL NUMBER IN ASCII
S.XXXXXXXX ESYY
+.1          +01
40FF FFFA
ENTER COMMAND (OR "HELP"):AR
ENTER THE REAL NUMBER IN ASCII
S.XXXXXXXX ESYY
-.1          +01
C0FF FFFA
ENTER COMMAND (OR "HELP"):AR
ENTER THE REAL NUMBER IN ASCII
S.XXXXXXXX ESYY
+.625        -01
4010 0000
ENTER COMMAND (OR "HELP"):AR
ENTER THE REAL NUMBER IN ASCII
S.XXXXXXXX ESYY
+.128        +03
427F FFF8
ENTER COMMAND (OR "HELP"):
```

Example 5-4:

```
ENTER COMMAND (OR "HELP"):RA
ENTER THE NUMBER IN REAL MACHINE REPRESENTATION
XXXX XXXX
4110 0000
+.09999996 E+01
ENTER COMMAND (OR "HELP"):RA
ENTER THE NUMBER IN REAL MACHINE REPRESENTATION
XXXX XXXX
41A0 0000
+.09999996 E+02
ENTER COMMAND (OR "HELP"):RA
ENTER THE NUMBER IN REAL MACHINE REPRESENTATION
XXXX XXXX
FFFF FFFF
-.72369260 E+76
ENTER COMMAND (OR "HELP"):RA
ENTER THE NUMBER IN REAL MACHINE REPRESENTATION
XXXX XXXX
0010 0000
+.53975610 E-78
ENTER COMMAND (OR "HELP"):
```

Example 5-5:

```
ENTER COMMAND (OR "HELP"):AD
ENTER THE DOUBLE PRECISION NUMBER IN ASCII
S.XXXXXXXXXXXXXXXX ESYY
-.1                    +01
C0FF FFFF FFFF FFFA
ENTER COMMAND (OR "HELP"):AD
ENTER THE DOUBLE PRECISION NUMBER IN ASCII
S.XXXXXXXXXXXXXXXX ESYY
+.00                   +00
0000 0000 0000 0000
ENTER COMMAND (OR "HELP"):AD
ENTER THE DOUBLE PRECISION NUMBER IN ASCII
S.XXXXXXXXXXXXXXXX ESYY
+.1                    +03
4263 FFFF FFFF FFFD
ENTER COMMAND (OR "HELP"):AD
ENTER THE DOUBLE PRECISION NUMBER IN ASCII
S.XXXXXXXXXXXXXXXX ESYY
-.5                    +01
C150 0000 0000 0000
ENTER COMMAND (OR "HELP"):
```

Example 5-6:

```
ENTER COMMAND (OR "HELP"):DA
ENTER THE NUMBER IN DOUBLE PRECISION MACHINE REPRESENTATION
XXXX XXXX XXXX XXXX
C1A0 0000 0000 0000
-.0999999999999999 E+02
ENTER COMMAND (OR "HELP"):DA
ENTER THE NUMBER IN DOUBLE PRECISION MACHINE REPRESENTATION
XXXX XXXX XXXX XXXX
0000 0000 0000 0000
+.0000000000000000 E+00
ENTER COMMAND (OR "HELP"):DA
ENTER THE NUMBER IN DOUBLE PRECISION MACHINE REPRESENTATION
XXXX XXXX XXXX XXXX
FFFF FFFF FFFF FFFF
-.72370055773322430 E+76
ENTER COMMAND (OR "HELP"):DA
ENTER THE NUMBER IN DOUBLE PRECISION MACHINE REPRESENTATION
XXXX XXXX XXXX XXXX
0010 0000 0000 0000
+.53976053469340183 E-78
ENTER COMMAND (OR "HELP"):
```

Several enhancements could be made to improve the accuracy and utility of these routines:

1) AINT, AREAL, and ADPFP could include error detection logic to flag invalid ASCII input strings.

2) RASCII and DASCII could include an algorithm to round to 6 or 14 significant digits respectively.

3) RASCII and DASCII implement method 1 of section 5-3 by multiplying or dividing the input quantity by 10 until the exponent is zero. The routines could be made to execute faster on the average by converting from 16^B to $F*10^D$ via a table. This would require 128 entries or 512 bytes of memory for single precision or 1 Kilobyte for double precision. Accuracy could also be improved if the proper table entries were used.

4) AREAL and ADPFP could be made to parse the input string so that a free format input could be used.

5.7.3 Transcendental Functions

The transcendental functions are included to illustrate the accuracy to be expected from common implementation of floating point operations. The functions are calculated from Taylor series expansions.

Example 5-7:

```
ENTER COMMAND (OR "HELP"):SIN
ENTER SINGLE PRECISION SINE ARGUMENT IN DEGREES
S.XXXXXXXX ESYY
+.0          +00
+.00000000 E+00
ENTER COMMAND (OR "HELP"):SIN
ENTER SINGLE PRECISION SINE ARGUMENT IN DEGREES
S.XXXXXXXX ESYY
-.30         +02
-.49999971 E+00
ENTER COMMAND (OR "HELP"):SIN
ENTER SINGLE PRECISION SINE ARGUMENT IN DEGREES
S.XXXXXXXX ESYY
+.45         +02
+.70710668 E+00
ENTER COMMAND (OR "HELP"):SIN
ENTER SINGLE PRECISION SINE ARGUMENT IN DEGREES
S.XXXXXXXX ESYY
+.60         +02
+.86602544 E+00
ENTER COMMAND (OR "HELP"):
```

Example 5-8

```
ENTER COMMAND (OR "HELP"):COS
ENTER THE SINGLE PRECISION COSINE ARGUMENT IN DEGREES
S.XXXXXXXX ESYY
+.           +00
+.09999996 E+01
ENTER COMMAND (OR "HELP"):COS
ENTER THE SINGLE PRECISION COSINE ARGUMENT IN DEGREES
S.XXXXXXXX ESYY
+.30         +02
+.86602544 E+00
ENTER COMMAND (OR "HELP"):COS
ENTER THE SINGLE PRECISION COSINE ARGUMENT IN DEGREES
S.XXXXXXXX ESYY
+.45         +02
+.70710668 E+00
ENTER COMMAND (OR "HELP"):COS
ENTER THE SINGLE PRECISION COSINE ARGUMENT IN DEGREES
S.XXXXXXXX ESYY
-.60         +02
+.50000028 E+00
ENTER COMMAND (OR "HELP"):
```

Example 5-9:

```
ENTER COMMAND (OR "HELP"):DSIN
ENTER THE DOUBLE PRECISION SINE ARGUMENT IN DEGREES
S.XXXXXXXXXXXXXXXX ESYY
+.0                  +00
+.0000000000000000 E+00
ENTER COMMAND (OR "HELP"):DSIN
ENTER THE DOUBLE PRECISION SINE ARGUMENT IN DEGREES
S.XXXXXXXXXXXXXXXX ESYY
+.30                 +02
+.4999999999999988 E+00
ENTER COMMAND (OR "HELP"):DSIN
ENTER THE DOUBLE PRECISION SINE ARGUMENT IN DEGREES
S.XXXXXXXXXXXXXXXX ESYY
+.45                 +02
+.7071067811865474 1 E+00
ENTER COMMAND (OR "HELP"):DSIN
ENTER THE DOUBLE PRECISION SINE ARGUMENT IN DEGREES
S.XXXXXXXXXXXXXXXX ESYY
+.60                 +02
+.8660254037844385 5 E+00
ENTER COMMAND (OR "HELP"):DSIN
```

Example 5-10:

```
ENTER COMMAND (OR "HELP"):DCOS
ENTER THE DOUBLE PRECISION COSINE ARGUMENT IN DEGREES
S.XXXXXXXXXXXXXXXX ESYY
+.0                  +00
+.0999999999999999 E+01
ENTER COMMAND (OR "HELP"):DCOS
ENTER THE DOUBLE PRECISION COSINE ARGUMENT IN DEGREES
S.XXXXXXXXXXXXXXXX ESYY
+.30                 +02
+.8660254037844386 4 E+00
ENTER COMMAND (OR "HELP"):DCOS
ENTER THE DOUBLE PRECISION COSINE ARGUMENT IN DEGREES
S.XXXXXXXXXXXXXXXX ESYY
+.45                 +02
+.7071067811865475 5 E+00
ENTER COMMAND (OR "HELP"):DCOS
ENTER THE DOUBLE PRECISION COSINE ARGUMENT IN DEGREES
S.XXXXXXXXXXXXXXXX ESYY
+.60                 +02
+.5000000000000001 3 E+00
ENTER COMMAND (OR "HELP"):
```

Example 5-11:

```
ENTER COMMAND (OR "HELP"):EX
ENTER THE SINGLE PRECISION  EXPONENTIAL ARGUMENT
S.XXXXXXXX ESYY
+.1          +01
+.27182769 E+01
ENTER COMMAND (OR "HELP"):EX
ENTER THE SINGLE PRECISION  EXPONENTIAL ARGUMENT
S.XXXXXXXX ESYY
+.1          +02
+.22026262 E+05
ENTER COMMAND (OR "HELP"):EX
ENTER THE SINGLE PRECISION  EXPONENTIAL ARGUMENT
S.XXXXXXXX ESYY
+.1          +00
+.11051673 E+01
ENTER COMMAND (OR "HELP"):EX
ENTER THE SINGLE PRECISION  EXPONENTIAL ARGUMENT
S.XXXXXXXX ESYY
-.1          +01
+.36787939 E+00
ENTER COMMAND (OR "HELP"):
```

Example 5-12:

```
ENTER COMMAND (OR "HELP"):DEX
ENTER THE DOUBLE PRECISION EXPONENTIAL ARGUMENT
S.XXXXXXXXXXXXXXXX ESYY
+.1                 +01
+.2718281828459026 E+01
ENTER COMMAND (OR "HELP"):DEX
ENTER THE DOUBLE PRECISION EXPONENTIAL ARGUMENT
S.XXXXXXXXXXXXXXXX ESYY
-.1                 +01
+.3678794411714424 E+00
ENTER COMMAND (OR "HELP"):DEX
ENTER THE DOUBLE PRECISION EXPONENTIAL ARGUMENT
S.XXXXXXXXXXXXXXXX ESYY
+.1                 +02
+.2202646579480655 E+05
ENTER COMMAND (OR "HELP"):DEX
ENTER THE DOUBLE PRECISION EXPONENTIAL ARGUMENT
S.XXXXXXXXXXXXXXXX ESYY
+.1                 +00
+.1105170918075463 E+01
ENTER COMMAND (OR "HELP"):
```

Several enhancements could be made to improve the versatility and accuracy of these routines:

1) The arithmetic overflow interrrupt could be set up to detect an overflow condition. At present, out of bounds arguments for all the transcendental functions will simply return an erroneous result.

2) Logic could be added to map the sine and cosine into the first quadrant for any input value. The Taylor series evaluation for sine and cosine for a fixed number of terms becomes more inaccurate as the input value becomes greater in magnitude.

3) The Taylor series terms are added as they are calculated; hence, succeeding smaller terms are added to a sum. For slightly improved accuracy, calculate all the terms first and then add them in the reverse order in which they were calculated.

4) Separate sine and cosine routines are not necessary. The cosine could be calculated using the sine routine and an appropriate mapping algorithm. All the trigonometric functions can be derived from the tangent series.

5) EX and ORX become erroneous for large negative numbers because Taylor series expansion for e^X becomes an alternating series for x less than zero. For large negative numbers, large magnitudes are added and subtracted to the sum to eventually yield a small result. Accuracy can be restored by the method described in 3). As an alternative, calculate e^X for x less than 0 by taking the reciprocal of e^{-X}.

5.7.4 Solution of Simultaneous Equations

The matrix inversion algorithm is designed to operate on a system of simultaneous equations expressed in the form of a matrix with N rows and N+1 columns where N is the number of unknowns. The row elements are in contiguous memory locations. The matrix is inverted "in place" by the Gauss-Jordan matrix inversion method. The original matrix is therefore destroyed and the results appear in the N+1 column of the matrix. For example, given the matrix A with N unknowns, x1 will appear in A(N,1), x2 in A(N,2),...,xN in A(N,N+1).

The FPDMON command MINV assumes that the matrix has been previously placed in memory.

Example 5-13:

```
?M E000,E02E
E000=4110  0000  4120  0000    4130  0000  4160  0000
E010=4110  0000  4110  0000    4110  0000  4130  0000
E020=4170  0000  4120  0000    4110  0000  41A0  0000
?G
FLOATING POINT DEMO MONITOR  REV *
COPYRIGHT 1980 BY TEXAS INSTRUMENTS

ENTER COMMAND (OR "HELP"):MINV
ENTER THE NUMBER OF UNKNOWNS =>3
ENTER THE ADDRESS OF THE MATRIX =>E000
X(+1)=+.09999996 E+01
X(+2)=+.09999996 E+01
X(+3)=+.09999996 E+01
ENTER COMMAND (OR "HELP"):QUIT
?
```

SECTION 6

I/O PROGRAMMING USING THE CRU

6.1 GENERAL

This section describes the fundamental aspects of I/O programming for a system using a TM 990/1481. In this section, the I/O functions on the TM 990/305 module will be used (the 305's memory section is not covered). Additional information on I/O programing can be found in the following manuals:

- Model 990 Computer, TMS 9900 Microprocessor Assembly Language Programmer's Guide (P/N 943441-9701)
- Model 990/12 Computer Assembly Language Programmer's Guide (P/N 2250077-9701 *A)
- TMS 9901 Programmable Systems Interface Data Manual
- TM 990/305 Memory and I/O Expansion Module User's Guide.

6.2 SYSTEM DESCRIPTION

A typical opto-coupled I/O system would consist of a TM 990/1481 and a TM 990/305. A brief description of the TM 990/305 I/O interface follows.

The TM 990/305 has 16 parallel input lines (port 1) and 16 parallel input/output lines (port 0) that can be individually configured as either inputs or outputs. All I/O lines are optically isolated and interface through the communications register unit (CRU). Each input line of port 0 and port 1 has its own socketed series resistor to allow the user to easily reconfigure the module for voltages up to 30 volts. Lines 0 through 3 and line 15 of port 1 can be configured as either inputs or interrupts by selecting the proper jumper option. Line 15 could be used to wire-OR interrupts 0 through 3 if the user desires a board interrupt. Interrupts are edge-triggered and latched.

Port 0 functions in much the same manner as port 1 with the exception that the lines can be used as inputs or outputs. When a line is to be configured as a latched output, its individual line resistor must be removed from its socket. By using high current, open-collector devices in port 0, output currents of 30 mA are permitted. If a line is to be used as an input, the optical isolator in its output section should be removed from its socket. A schematic of a channel of I/O port 0 is shown in Figure 6-1.

As the CRU provides the interface between the TM 990/1481 and the TM 990/305 via the TM 990 system bus (see belwo).

6.3 COMMUNICATIONS REGISTER UNIT (CRU)

The CRU provides a dedicated serial interface for I/O operations. CRU instructions permit transfer of from one to sixteen bits. CRU I/O provides powerful bit manipulation capability, flexible field lengths, and a simple bus structure (a description of CRU single-bit and multibit instructions can be found in Section 4.5.8, entitled CRU Bit Addressing). Both the CRU and address buses are used for this communication which involves 32 CRU bits. It should be noted that the CRU does not use the data bus. A CRU map of the TM 990/1481 is in Appendix B.

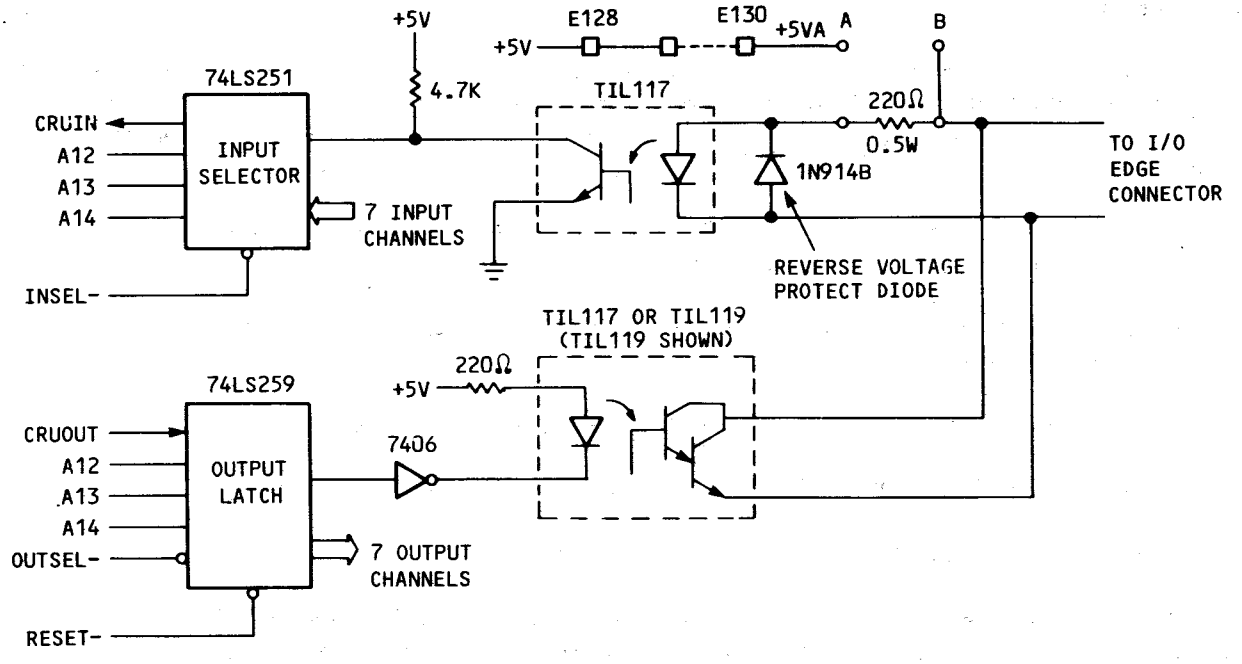


FIGURE 6-1. TM 990/305 PORT 0 I/O CHANNEL

As an example of offboard CRU, the TM 990/305 can be assigned a unique CRU hardware base addresss as explained in the TM 990/305 User's Guide. A sample program using the TM 990/305 is provided in paragraph 6.6 herein. This address is the "base address" for that module and is the reference point from which displacements are taken when using CRU single-bit instructions. The CRU map for the TM 990/305 is given in Table 6-1; this map gives the needed information to address any of the I/O module's channels or interrupts. As an example, after the CRU hardware base address for the I/O module is loaded into R12, then the instruction SBO 5 would activate a load or relay connected to output OUT5 on that module (Note: this high level signal is then inverted thus producing the low signal that is required to activate the TIL 117/TIL 119 opto-coupler. Similarly, this bit could be tested with a test bit 5 (TB5) instruction to verify that the low level that is required to activate the load was actually present.

6.4 LOADING THE CRU HARDWARE BASE ADDRESS

There are two techniques that can be used to load the CRU hardware base address into its designated workspace register (R12). In order to best illustrate these techniques, a review of CRU bit address development will be presented first.

The CRU bit selected by single-bit instructions is determined by the value in bits 3-14 of workspace register R12 plus the value of the signed displacement from the single-bit instruction (See Figure 6-2). The contents of R12 are referred to as the "software base address" while bits 3-14 of R12 constitute the "hardware base address." To properly load R12, a value equal to 2X the

TABLE 6-1. TM 990/305 CRU MAP

Definitions: A = CRU H/W Base Address (R12, Bits 3-14)
S/W Base Address = 2 X H/W Bit Address

CRU Bit	H/W Bit Address	Input		Output	
0	A + 0000	I/O Port 0	IN 0	I/O Port 0	OUT 0
1	A + 0001	"	IN 1	"	OUT 1
2	A + 0002	"	IN 2	"	OUT 2
3	A + 0003	"	IN 3	"	OUT 3
4	A + 0004	"	IN 4	"	OUT 4
5	A + 0005	"	IN 5	"	OUT 5
6	A + 0006	"	IN 6	"	OUT 6
7	A + 0007	"	IN 7	"	OUT 7
8	A + 0008	"	IN 8	"	OUT 8
9	A + 0009	"	IN 9	"	OUT 9
10	A + 000A	"	IN 10	"	OUT 10
11	A + 000B	"	IN 11	"	OUT 11
12	A + 000C	"	IN 12	"	OUT 12
13	A + 000D	"	IN 13	"	OUT 13
14	A + 000E	"	IN 14	"	OUT 14
15	A + 000F	"	IN 15	"	OUT 15
16	A + 0010	INPUT PORT1	IN0/INTERRUPT 0	INTERRUPT RESET 0	
17	A + 0011	"	IN1/INTERRUPT 1	INTERRUPT RESET 1	
18	A + 0012	"	IN2/INTERRUPT 2	INTERRUPT RESET 2	
19	A + 0013	"	IN3/INTERRUPT 3	INTERRUPT RESET 3	
20	A + 0014	"	IN4		
21	A + 0015	"	IN5		
22	A + 0016	"	IN6		
23	A + 0017	"	IN7		
24	A + 0018	"	IN8	INTERRUPT MASK 0	
25	A + 0019	"	IN9	INTERRUPT MASK 1	
26	A + 001A	"	IN10	INTERRUPT MASK 2	
27	A + 001B	"	IN11	INTERRUPT MASK 3	
28	A + 001C	"	IN12	STATUS LED #1	
29	A + 001D	"	IN13	STATUS LED #2	
30	A + 001E	"	IN14	BOARD I/O RESET	
31	A + 001F	"	IN15/BOARD INTERRUPT	BOARD INTER. RESET	

desired CRU hardware base address can be loaded into R12 or a value equal to the desired CRU hardware base address can be loaded into R12 and then shifted one bit to the left.

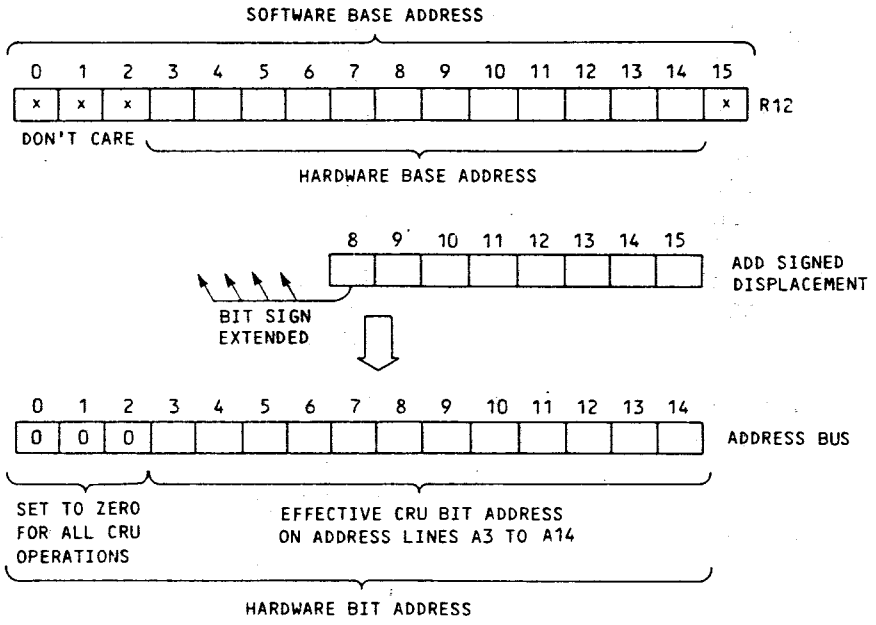


FIGURE 6-2. CRU BIT ADDRESS DEVELOPMENT

As an example of loading the CRU hardware base address, assume that a CRU hardware base address of 180_{16} (384_{10}) is to be loaded into R12. As mentioned previously, there are two ways to accomplish this loading: load a value into R12 that is equal to twice the value of the CRU hardware base address or load the value equal to the desired CRU hardware base address and then shift the value one bit to the left. The following examples place CRU hardware base address 384_{10} (180_{16}) into R12.

```
EXAMPLE 1:  LI   R12,>300     Loads 180(16) (384(10)) into bits 3-14 of R12

                              Notes: 1. 300(16) = 2X 180(16).
                                     2. The greater than sign (>) is used to
                                        indicate a hexadecimal value.

EXAMPLE 2:  LI   R12,>180     Loads 180(16) (384(10)) into R12
            SLA  R12,1        Shifts R12 contents one bit to the left.
```

Now that the two techniques for loading the CRU hardware base address have been described, an example illustrating the loading of a specific bit will be given. Assume that the I/O module is assigned a CRU hardware base address of 180_{16} and it is desired to activate output 5 (OUT5). The displacement from the base address is 5_{10} as given in Table 6-1. The required code that is needed to select this output follows:

```
EXAMPLE 1:  LI   R12,>300     This technique loads R12 with 2X the CRU
            SBO  5            hardware base address that is desired.
                              The SBO instruction activates the bit that
                              is displaced by 5 from the base address.

EXAMPLE 2:  LI   R12,>180     This technique loads R12 with the CRU hard
            SLA  R12,1        ware base address and then shifts this data
            SBO  5            one bit to the left.  The SBO instruction
                              carries out the same function as in the code
                              given above.
```

6.5 USER WORKSPACE

The high usage data registers for the TM 990/1481 are defined as blocks of memory called workspaces; these workspaces are located on the memory module. The starting location of a workspace is defined by a single internal register called the workspace pointer. The workspace pointer contains the memory word address of the first of sixteen consecutive memory words in the workspace, thus the processor has access to sixteen 16-bit registers. When a differnet set of registers is required, the program simply reloads the workspace pointer with the new address.

The load workspace pointer immediate (LWPI) instruction is used to define the starting address for the user workspace. As this workspace resides in RAM (read/write memory), then the type of memory module used and how it is configured will determine the available user workspace.

6.6 SAMPLE PROGRAM

A sample program that will monitor one input line and control one output line on the TM 990/305 is given in Figure 6-3. This control system responds to inputs from a transducer and activates a relay that controls some form of a load. Output OUT1 of I/O port 0 is controlled as a result of constantly reading the state of input IN0 of the same I/O port. Both the input and output signals are active highs. When a one is read at IN0, a one is output to OUT1.

```
START   LWPI >FF90        Define workspace just below TIBUG
OFF     LI   R12,>300     Load TM 990/305 CRU hardware base address
        SBZ  1            Turn output (OUT1) off
LOOP    TB   0            Look at input level
        JNE  OFF          Read "0" = off
        SBO  1            Turn output on
        JMP  LOOP         Look at input level again
        END
```

FIGURE 6-3. MONITOR CONTROL PROGRAM

SECTION 7

THEORY OF OPERATION

7.1 GENERAL

This section covers the theory of operation of the TI 990/1481. Information in the following manuals can be used to supplement material in this section:

- SN74S481, SN54LS/74LS481 4-Bit-Slice Schottky Processor Data Manual
- TMS 9902 Asynchronous Communications Controller Data Manual
- TMS 9901 Programmable Systems Interface Data Manual.

7.2 SYSTEM BLOCK DIAGRAM

Figure 7-1 shows a system using the TM 990/1481(processor and controller boards) and a memory board. All boards interface with the TM990 BUS on the motherboard via the common bottom edge connector J1. The PROCESSOR and the CONTROLLER are also interconnected via the common top edge connectors J3 and J4. Connector J2 on the CONTROLLER allows connection to any RS232 device such as the TI Silent 700 terminal or the TM 990/301 Microterminal.

7.3 THE PROCESSOR BOARD

7.3.1 The 481 Bit-Slice Processor

The PROCESSOR board is designed around the TI SN74S481J bit-slice processor integerated circuit. A functional block diagram of this processor is given in Figure 7-2. The PROCESSOR's major operating registers are contained within the 481 chips. These registers are the PROGRAM COUNTER (PC), the MEMORY COUNTER(MC), the WORKING REGISTER (WR), and the EXTENDED WORKING REGISTER (XWR). Detailed information on the 481 can be found in the data manual entitled the "SN74S481, SN54LS/74LS481 4-Bit-Slice Schottky Processor Elements Data Manual." The TM 990/1481 processor is shown on sheet 2 of the processor board schematics in Appendix A. This processor uses four 481 chips to form a 16-bit processor.

The ALU OP CODE, OP0 to OP10, is routed from the CONTROLLER board via the top edge connector P3 shown on sheet 6 of the schematics. OP8 and OP9 go thru open collector gates to allow the processor chips to drive these lines during micro/macro-operations such as multiply and divide. OP0 thru OP4 are OR'ed with ALUSPLIT to the least significant two chips of the ALU to create a NOP function in the lower byte during byte operations. OP10, which is typically the carry-in function in the ALU OP CODE, goes thru a mux which allows a previously saved carry-out to be substituted as the carry-in.

Two microinstruction commands of the STATUS CONTROL (STC) field, SAVCO and USENSAV, allow the present carry-out of the ALU (CO-) to be saved in the COSAV flip-flop until another SAVCO or USENSAV is executed. The saved carry-out can then be substituted for OP10 as the carry-in to the ALU on any subsequent microinstruction. Again two decodes of the STC field, USECOSAV and USENSAV, allow the substitution of COSAV for OP10.

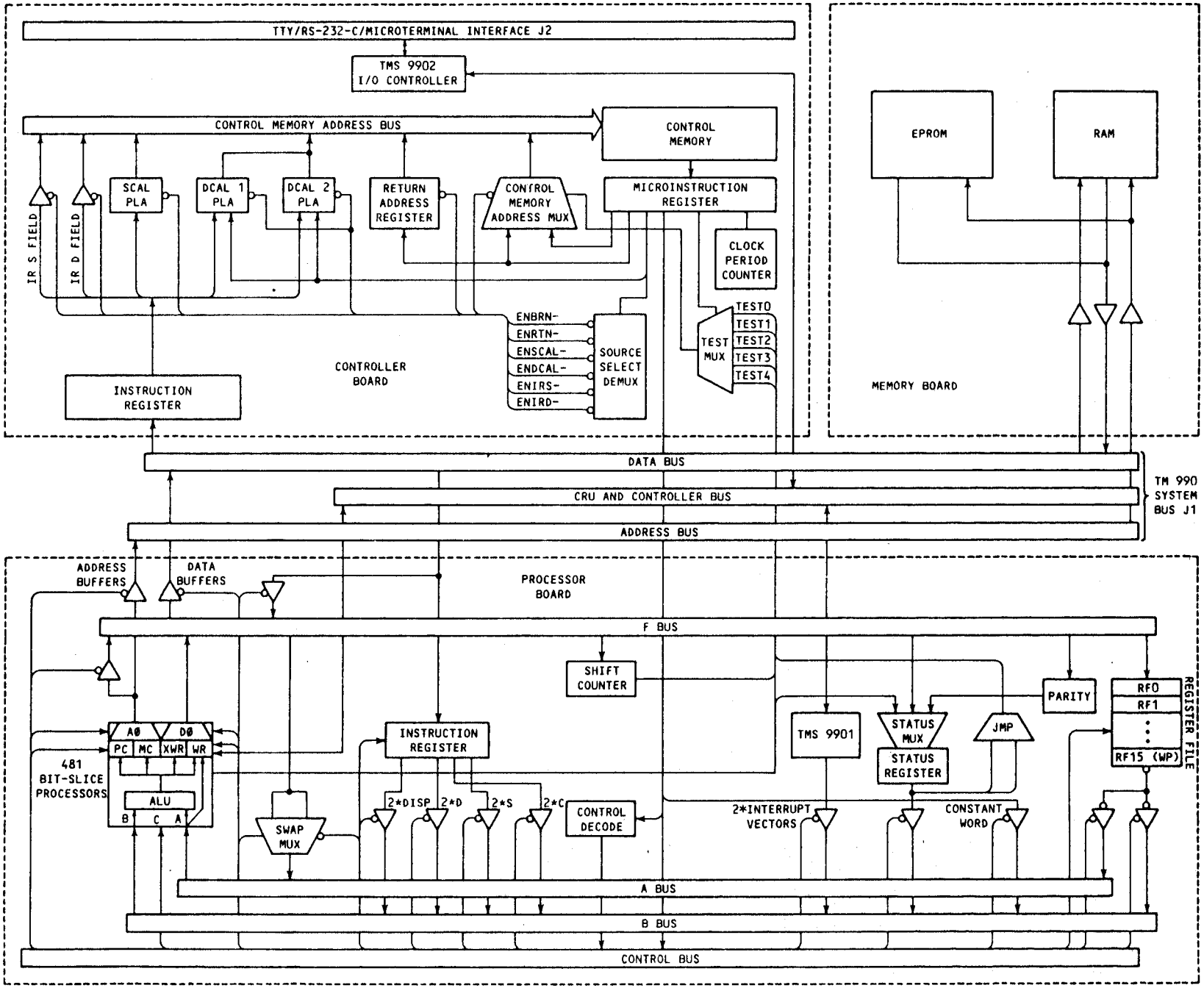


FIGURE 7-1. TM 990/1481 SYSTEM BLOCK DIAGRAM

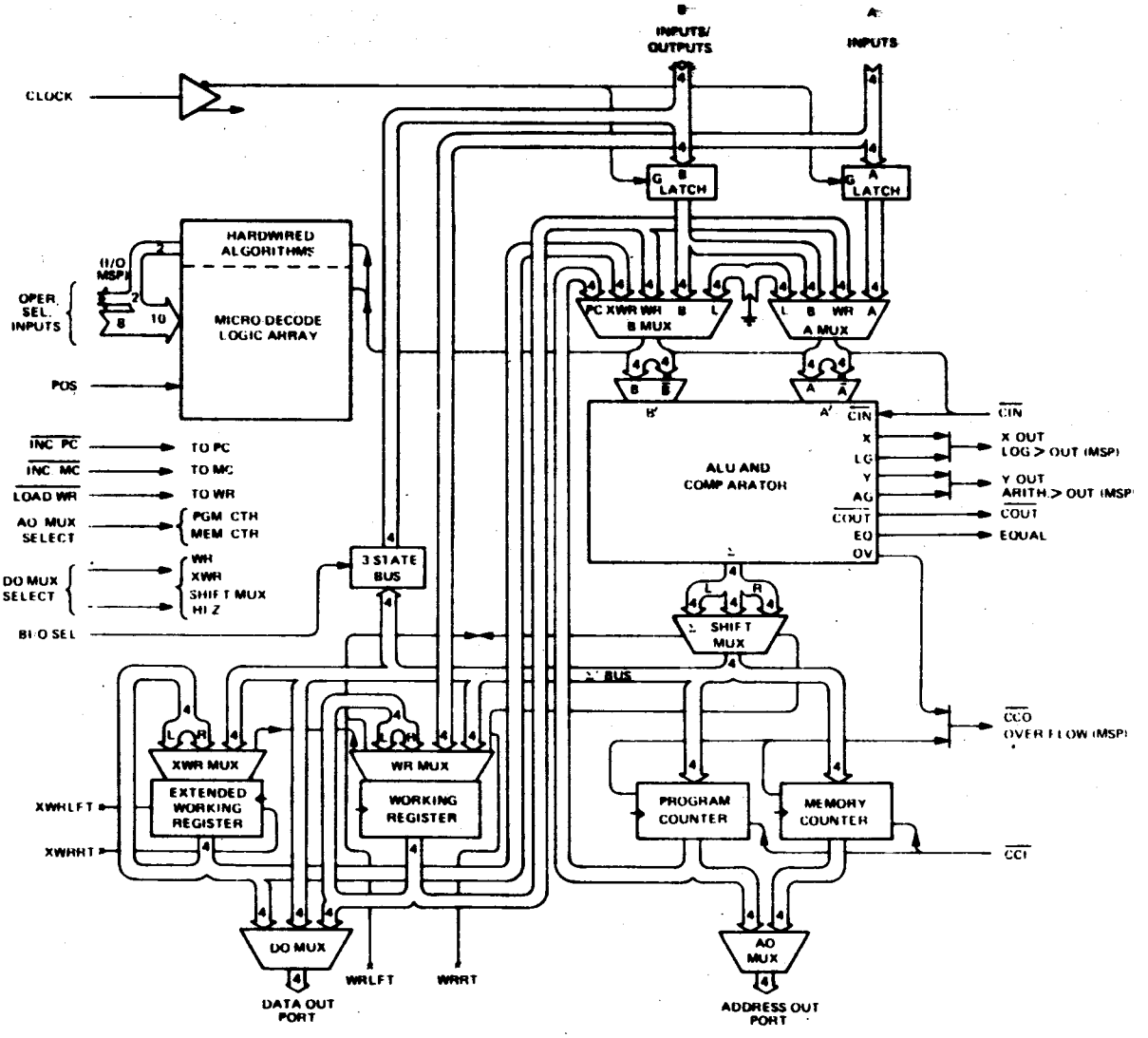


FIGURE 7-2. SN74S481 FUNCTIONAL BLOCK DIAGRAM

The flip-flop which saves the overflow bit functions in a different manner from the COSAV flip-flop. The overflow bit out of the ALU (OV) is saved in the OVSAV flip-flop only when the SAVOV microinstruction is executed, and is reset after the next microinstruction. The OVSAV command is a decode of the STC field. If a string of consecutive microinstructions are executed with SAVCO, and an overflow occurs anywhere in the string, then the OVFL signal will be held from that point to the end of the string and for one additional microinstruction period.

This feature is used for example in the left arithmetic shift. The ALU can do only a single bit shift so a multiple bit shift is performed by looping on a shift and decrementing the counter to zero. The ALU produces an overflow (OV) if the sign bit changes on any shift, but the OV goes away if the sign remains the same on subsequent shifts. The OVSAV allows the overflow condition to be saved and status recorded properly.

7.3.2 Shift Counter

The SHIFT COUNTER counts the microcycles for operations such as shifts,

divides, multiplies, and other iterative operations. The SHIFT COUNTER may be decremented by one, loaded with a count value, or set to 15. The counter is 4 bits wide and is loaded from the F-BUS bits F11 thru F14 when the LDCNT bit of the microinstruction is set. The count loaded is therefore actually F/2. The COUNT=0 signal from the counter goes to the BRANCH TEST MUX so the microprogram may conditionally branch on the COUNT=0 condition.

The SHIFT COUNTER, shown on sheet 4 of the PROCESSOR schematics, is implemented using an SN74LS163N FOUR BIT COUNTER chip. The counter is actually implemented using complement logic, that is, it is loaded with the complement of the F-BUS (F(11-14)-), counts down rather than up, is cleared to zero rather than set to 15, and detects COUNT=15 rather than COUNT=0. All of this is transparent to the microprogrammer however.

The counter is controlled by two bits of the microinstruction, LDCNT and DECCNT. LDCNT causes the counter to be loaded with the complement of the F-BUS value (F(11-14)-) on the next clock (CLKC3-). DECCNT causes the counter to increment by one on the next clock (CLKC3-). If LDCNT and DECCNT are both ONE then the counter is reset to ZERO via the SETCNT15 signal.

7.3.3 The Swap Multiplexer

The SWAP MUX allows the microprogrammer to swap the two bytes of the data on the F-BUS, and the result is available on the A-BUS. The swap operation is normally used to move the byte to be operated on to the most significant half of the word since in byte microoperations the ALU only operates on the upper byte. The normal state of the SWAP MUX allows data to pass to the A-BUS unchanged, and therefore provides a path from F-BUS to A-BUS. There are three swap commands available: FSWAP = swap unconditionally, CSWAPA = swap if bit 15 of the address is one, and CSWAPB = swap if bit 15 of the address is one and the instruction is a byte related instruction. The swap commands are decodes of the DECODE field of the microinstruction.

The SWAP MUX, shown on sheet 3 of the PROCESSOR schematics, is implemented with four SN74S257N QUAD 2 TO 1 MULTIPLEXERS. The chip enables of the multiplexer chips are controlled by the ENSM- signal which is the complement of the A-BUS SELECT bit of the microinstruction bit ENRF- (CMDO(41)). The A/B SELECT inputs of the multiplexer chips are controlled by the SWAP signal, which is a function of FSWAP-, CSWAPA-, CSWAPB-, BYTEX, and AO15. The function is:

$$SWAP = FSWAP + AO15 * (CSWAPA + CSWAPB * BYTEX)$$

where FSWAP, CSWAPA, and CSWAPB are decodes of the DECODE field of the microinstruction word (DEC(0-4) = CMDO(76-80)), BYTEX is a decode of the instruction register, and AO15 is the least significant bit of the ADDRESS BUS. The decoded signals are defined as follows:

DEC(0-4)=00010 ⟶ FSWAP, (FORCE SWAP)

DEC(0-4)=00011 ⟶ CSWAPA, (CONDITIONAL SWAP on ADDRESS)

DEC(0-4)=00100 ⟶ CSWAPB, (CONDITIONAL SWAP on ADDRESS if BYTE)

The BYTEX signal indicates that the instruction register contains a BYTE related instruction, and the least significant bit of the address, AO15, indicates which of the bytes was addressed, the upper byte or the lower byte.

7.3.4 Instruction Register (IR)

The principal INSTRUCTION REGISTER is on the CONTROLLER board where it is decoded to get the microprogram entry points, but a second INSTRUCTION REGISTER is located on the PROCESSOR board where it is used to save the variable parts of the instruction such as the addressing parameters. The fields of the IR which the microprogrammer can access are the C, S, D, and DISP fields. All of these fields are presented to the ALU, via the B-BUS, right justified to the second from the least significant bit. They are therefore actually 2*C, 2*S, 2*D, and 2*DISP. The DISP field is sign extended, and the C, S and D fields are zero filled.

The INSTRUCTION REGISTER, shown on sheet 4 of the PROCESSOR schematics, is implemented using an SN74S374N EIGHT BIT REGISTER and an SN74S175N FOUR BIT REGISTER. The IR is divided in such a way that the least significant four bits may be loaded independently of the rest of the word. Only 12 bits of the IR are implemented on the PROCESSOR since the upper bits do not contain variable data fields. The least significant four bit register is clocked by the GCKIR- signal and the upper register is clocked by CKIR-. GCKIR is the logical OR of CKIR and CKIRX so CKIR clocks the entire register and CKIRX clocks only the least four bits.

The data fields of the IR are driven to the B-BUS using four SN74LS244N QUAD BUS DRIVERS. One of the LS244's drives either the S-FIELD of the IR (IR(12-15)) or the D-FIELD of the IR (IR(6-9)) to the B-BUS (B(11-14)). A second LS244 drives either the upper 4 bits of the 8 bit displacement field of the IR (IR(8-11)) or zeros to the B-BUS (B(7-10)). A third LS244, shown on sheet 8 of the PROCESSOR schematics, drives the C-FIELD of the IR (IR(8-11)) to the B-BUS (B(11-14)). When the C,S,or D fields are driven to the B-BUS (B(11-14)) the remaining 12 bits (B(0-10) and B(15)) are driven to zero to fill out the word by the other LS244's. When the DISPLACEMENT FIELD is driven to the B-BUS the upper bits are sign extended (B(0-7) = B(8)) and the least siginificant bit is set to zero (B(15) = 0).

The control signals that determine which IR field drives the B-BUS are decoded from the B-BUS SELECT field of the microinstruction (BSEL(0-3)⟵ CMDO(43-45)). The decoding is done using an SN74S138N THREE TO EIGHT DEMULTIPLEXER, as shown on sheet 6 of the PROCESSOR schematics.

The decodes are as follows:

BSEL(0-3)=001	→	ENS-	S FIELD
BSEL(0-3)=010	→	END-	D FIELD
BSEL(0-3)=011	→	ENDISP-	DISPLACEMENT FIELD
BSEL(0-3)=111	→	ENC-	C FIELD

7.3.5 Status Register and Status Logic

The STATUS LOGIC allows the microcode to transfer the existing status conditions at any time to the STATUS REGISTER. The microinstruction control word allows either individual status bits to be enabled or certain logical groups of bits to be enabled simultaneously. The STATUS REGISTER can be loaded in three ways, bits 0-7 from current conditions selectively under microinstruction command, bits 12-15 directly from the F-BUS (MASK load), or all 16 bits at once loaded directly from the F-BUS.

The STATUS REGISTER is shown on sheet 7 of the PROCESSOR schematics. The status logic can be logically divided into six parts; the STATUS REGISTER

itself, the STATUS MULTIPLEXER, the STATUS CONTROL section, the F-BUS buffer, the B-BUS buffer, and the CARRY/SHIFT logic.

The STATUS REGISTER is implemented using four SN74LS379N FOUR BIT REGISTERS. The register is divided in this manner to allow the upper 8 bits to be controlled via the STATUS MUX and to allow the lower 4 bits to be loaded independently for MASK LOAD. The register chips are clocked continuously by the system clock CLKP3- . In the normal state the STATUS MUX circulates present status back to the upper 8 bits in the upper two LS379's, and the lower two LS379s have the LOAD line off (HIGH) so that STATUS remains unchanged.

The STATUS MULTIPLEXER, which consists of four SN74LS253N DUAL 4 TO 1 MULTIPLEXERS, can individually select either NO CHANGE or UPDATE for each of the upper 7 bits of the register. The control lines ENST(0-6) determine which of the bits are to be updated and which are to remain the same(the 8th bit, ST7, located in the STATUS MUX, cannot be updated via the STATUS MUX but can be changed as described below).

The source of the new status conditions is the current output of the S481 processor. The LGT,AGT,EQ,and OVFL conditions come directly from the S481s. The PARITY signal is generated from the upper bits of the word currently on the F-BUS using an SN74LS280N PARITY GENERATOR. The COSH signal is either the CARRY-OUT of the ALU or the SHIFT-OUT of the ALU depending on the state of the SAVSH- signal.

The STATUS CONTROL logic decodes the STATUS CONTROL field of the microinstruction to produce the necessary control signals for the status logic and some additional control signals for the ALU section. The STATUS CONTROL produces the individual status bit enable signals, ENST(0-6), the status load signal, LDSR (and LDSR-), and the mask load signal, LDMASK-.

The F-BUS BUFFER allows the F-BUS to drive the upper status bits instead of the STATUS MUX during the status load operation. The F-BUS BUFFER is implemented using half of two SN74LS244N OCTAL BUS DRIVERS. The output enable of the buffer is controlled by the LDSR- signal. When LDSR- is LOW the F-BUS feeds the STATUS REGISTER, and when LDSR- is HIGH the STATUS MUX feeds the STATUS REGISTER.

The B-BUS BUFFER, consisting of four SN74LS244 OCTAL BUS DRIVERS, allows the STATUS REGISTER to be read to the B-BUS when selected via the B-BUS SELECT field of the microinstruction. The signal ENSR which causes the buffer to drive the B-BUS is decoded from the B-BUS SELECT (BSEL(0-2) = 011).

The CARRY/SHIFT logic, shown on sheet 11 of the PROCESSOR schematics, is used to control the condition which sets ST3, either CARRY-OUT(CC) or SHIFT-OUT(SH), and if SH then which bit should be used F(0) or F(15).

TABLE 7-1. STATUS CONTROL ROMS

PROCESSOR PROM P1 (U17/D6)	PROCESSOR PROM P2 (U23/E6)
1=ENCRUCLK	1=SAVSH-
2=ENST6	2=ENSAVCO
3=ENST5	3=SAVOV
4=ENST4	4=ALUSPLIT
5=ENST3	5=COSAVUSE-
6=ENST2	6=LDSR-
7=ENST1	7=LDSR
8=ENST0	8=LDMASK-

DECODE	MNEMONIC	P1 87654321	P2 87654321
00000	NOP	00000000	10110001
00001	COMP	11100000	10110001
00010	COMPB	11100100	10111001
00011	ARITH	11111000	10110001
00100	ARITHB	11111100	10111001
00101	SHIFT	11110000	10110000
00110	SHIFTL	11111000	10110100
00111	CRUCLK	00000001	10110001
01000	SHTEST	00000000	10110000
01001	USENSAV	00000000	10100011
01010	COMPOV	11101000	10110001
01011	NOP	00000000	00000000
01100	NOP	00000000	00000000
01101	NOP	00000000	00000000
01110	NOP	00000000	00000000
01111	NOP	00000000	00000000

DECODE	MNEMONIC	P1 87654321	P2 87654321
10000	ENLGT	10000000	10110001
10001	ENAGT	01000000	10110001
10010	ENEQU	00100000	10110001
10011	ENCO	00010000	10110001
10100	ENOV	00001000	10110001
10101	ENOP	00000100	10110001
10110	ENXOP	00000010	10110001
10111	USECOSAV	00000000	10100001
11000	ALUSPLIT	00000000	10111001
11001	SAVOV	00001000	10110101
11010	SAVCO	00000000	10110011
11011	SAVSH	00010000	10110000
11100	SAVCOST	00010000	10110011
11101	LDSTATUS	00000000	11010001
11110	LDMASK	00000000	00110001
11111	LDSR	00000000	01010001

7.3.6 Register File

The REGISTER FILE contains 16 registers, 15 of which may be used by the microprogrammer for general data storage. Register number 15 is dedicated to holding the WORKSPACE POINTER. The primary use of the REGISTER FILE in the 1481 instruction set is in storing the intermediate results in the floating point routines.

The REGISTER FILE, shown on sheet 5 of the PROCESSOR schematics, is implemented using four SN74S189N 16X4 RAMS, and four SN74S240N OCTAL BUFFERS. The input to the REGISTER FILE is TRUE data from the F-BUS (F(0-15)). The output of the REGISTER FILE is COMPLEMENT data (RF(0-15)-) to the buffers. The buffers are inverting so the data to the A-BUS and B-BUS is again TRUE data. The address to the REGISTER FILE comes from the REGISTER FILE ADDRESS field of the microinstruction (RFAD(0-3) ← CMDO(38-41)). The WRITE ENABLE to the REGISTER FILE (CKRF-) is derived from the system clock (CLKP3-) and the LOAD REGISTER FILE bit of the microinstruction (LDRF- ← CMDO(66)). The output of the REGISTER FILE can go to the A-BUS or to the B-BUS or to both. The signals which drive the buses are ENRFA- and ENRFB- which are derived from the A-BUS and the B-BUS SELECT fields of the microinstruction.

When the REGISTER FILE is being loaded (CKRF- = 0) the outputs of the S189s go to HIGH-Z which looks like ALL ZERO on the A-BUS or the B-BUS if RF is selected. This is normally not a problem since the input latchs on the S481s are simultaneously locking out the BUS data. This is NOT true for the DIRECT A-BUS TO WR LOAD (LDWR←CMDO(65))! The implication of this is that if data must be transferred from the RF to the WR at the same time the RF is loaded from the F-BUS, the direct A-Bus to WR load should not be used; instead, the data should be loaded thru the ALU section via the input latches using ALU opcode A>WR (00000001011).

7.3.7 Constant Word

The CONSTANT WORD comes from the 16 bit BRANCH ADDRESS 1 (BA1) field of the microinstruction. Normally this field contains one of the two branch addresses for a conditional branch, but if an unconditional branch to the BRANCH ADDRESS 0 (BA0) has been selected, then the contents of the BA1 field is a "don't care" as far as the branch logic is concerned. The BA1 field may then be used to introduce a CONSTANT WORD onto the B-BUS of the PROCESSOR. This word might be a number to be added to the data being processed or a mask word used to eliminate unwanted fields in the data.

The 16 bit BA1 field (BX(0-5),BA(10-19)) is routed from the CONTROLLER to the PROCESSOR via the J4 top edge connector on pins J4-25 to J4-40 as shown on sheet 6 of the PROCESSOR schematic. Four SN74LS244 OCTAL BUS DRIVERS allow the CONSTANT WORD to drive the B-BUS when enabled by the ENCONST- signal as shown on sheet 7 of the schematics.

7.3.8 A-Bus, B-Bus, and F-Bus

There are three tri-state buses in the PROCESSOR, the A-BUS, the B-BUS, and the F-BUS. The A-BUS and B-BUS are connected to the AI and BI input ports of the 481 processor, and the F-BUS is driven by the Data Output Port of the 481 processor. The F-BUS can also be driven by the Address Output Port of the 481 and by the Memory Data Input from the TM990 BUS.

The A-BUS can be thought of as the operand bus. Data fetched from memory is

normally brought into the Working Register (WR) via the SWAP MUX and the A-BUS. Intermediate results stored in the REGISTER FILE are available to the 481 via the A-BUS. The AI input to the 481 differs from the BI input in that a direct path exists from AI to WR which bypasses the ALU logic and therefore requires less setup time. This is why the A-BUS was chosen as the data input path.

The B-BUS can be thought of as the modifier bus. The words brought to the B-BUS are typically used to modify addresses or data words. The BI input to the 481 has a more extensive set of options available than the AI input and therefore is used for modifier type data.

There are two sources which can drive the A-BUS, the SWAP MUX and the REGISTER FILE. The ENRFA- bit of the microinstruction (CMDO(42)) determines which of the two drives the A-BUS on each microinstruction (ENRFA- = 0 → REGISTER FILE, ENRFA- = 1 → SWAP MUX). The SWAP MUX, which was described in Section 7.3.3, is implemented with multiplexer chips that have tri-state outputs, and it is therefore connected directly to the A-BUS. The ENSM- signal controls the OUTPUT ENABLEs on the multiplexers, and when ENSM- is LOW the SWAP MUX drives the A-BUS.

The REGISTER FILE, which is described in Section 7.3.6, is implemented with RAM chips which have tri-state outputs, but an intermediate set of tri-state buffers are used for two reasons. First the RAM chips complement the data and an inverting buffer is needed to return true data, and secondly it was necessary to have two sets of buffers to be able to drive both the A-BUS and the B-BUS from the REGISTER FILE. The ENRFA- signal is connected to the OUTPUT ENABLES on one pair of SN74S240N OCTAL BUS DRIVERS, and when ENRFA- is LOW it enables the REGISTER FILE to drive the A-BUS with true data.

There are eight sources which can drive the B-BUS; the CONSTANT WORD, the S field of the INSTRUCTION REGISTER (IR), the D field of the IR, the C field of the IR, the DISP field of the IR, the INTERRUPT VECTOR (IV), the STATUS REGISTER (SR), and the REGISTER FILE (RF).

The BSEL field of the microinstruction word (BSEL(0-2) ← CMDO(43-45)) controls which source drives the B-BUS on each microinstruction. The BSEL field is decoded by an SN74S138N 3-TO-8 DEMULTIPLEXER to produce eight control signals, ENCONST-, ENS-, END-, ENDISP-, ENIV-, ENRFB-, ENSR-, and ENC-, which are used to enable the tri-state buffers to drive the B-BUS.

The F-BUS can be thought of as the result or the function bus. The data which results from ALU operations can be brought out to the F-BUS and then out to memory or saved in the local REGISTER FILE. Data from memory is routed first to the F-BUS from which it can be stored in the REGISTER FILE or routed thru the SWAP MUX to the WR. The ADDRESS OUTPUT can also drive the F-BUS to allow operations or modifications to be performed on the address. Also the COUNTER and the STATUS REGISTER can be loaded with data from the F-BUS.

The F-BUS can be driven from five sources three of which are from the 481 Processor, the ALU SUM BUS, the WORKING REGISTER, and the EXTENDED WORKING REGISTER, and the two other sources are MEMORY DATA IN and ADDRESS OUT.

The FSEL field of the microinstruction word (FSEL(0-2) ← CMDO(47-49)) determines which of these sources drives the F-BUS. Two of the lines, FSEL1 and FSEL2 go directly to the 481 Processor D0 and D1 control inputs to select one of the three 481 internal sources to drive the F-BUS.

FSEL1 (D1)	FSEL2 (D0)	SOURCE
0	0	ALU SUM BUS
0	1	XWR
1	0	WR
1	1	(HI-Z)

When FSEL1 and FSEL2 are both HIGH the 481 DATA OUTPUT lines go to HI-Z and this allows the other sources to drive the F-BUS. The remaining control bit FSEL0 determines which of the two other sources will drive the bus, MEMORY DATA IN (MDI) or ADDRESS OUT. The FSEL(0-2) signals are used to generate the enable signals, ENMDI- and ENATOF- as shown on sheet 9 of the PROCESSOR schematics. The MEMORY DATA IN buffer is implemented using four SN74S241N OCTAL TRI-STATE BUFFERS, shown on sheet 3 of the schematics, which are enabled by ENMDI-. The buffer which routes ADDRESS OUT to the F-BUS is implemented using four SN74S241N OCTAL TRI-STATE BUFFERS, shown on sheet 3 of the schematics, which are enabled by ENATOF-.

7.3.9 Address and Data Out

The address and data lines are driven to the TM990 BUS using four SN74S241N OCTAL TRI-STATE BUFFERS, as shown on sheet 3 of the PROCESSOR schematics. The address lines come from the ADDRESS OUTPUT PORT of the 481 Processor. The data lines come from the F-BUS which would typically be driven by the DATA OUTPUT PORT of the 481 Processor when data is being output to the TM990 BUS.

The ADDRESS OUT buffer is enabled by HOLD ACKNOWLEDGE (HOLDA) so the address is being driven to the bus at all times except when the TM990/1481 has released bus access to some other device.

The DATA OUT buffer is enabled by the signal ENABLE DATA OUT (ENDO) which is derived from the ENABLE MEMORY DATA OUT (ENMDO) bit of the microinstruction.

7.3.10 Interrupt Logic and Jump Control

The Interrupt Logic recognizes interrupt conditions, compares individual bit masks, resolves interrupt priority, generates the Interrupt Vector, checks the interrupt level mask, and controls the Pending Interrupt signal which initiates the firmware interrupt trap routine.

The 15 Maskable Interrupts are controlled by the TMS9901 Programmable Systems Interface integrated circuit shown on sheet 10 of the PROCESSOR schematics. The TMS9901 is controlled via the CRU. It can selectively mask out any of the 15 interrupts, but whether masked or not any of the interrupt lines can be tested via the CRU.

The TMS9901 resolves interrupt priority level and generates a 4-bit Interrupt Vector (IV0-IV3) which represents the level of the highest priority unmasked active interrupt. If any unmasked interrupt is active it generates an Interrupt Request signal (INTREQX-).

The 15 interrupt lines on the TM990 BUS which are inputs to the TMS9901 are pulled-up to +5V on the PROCESSOR using 4.7 Kohm resistors. INT4- can be jumper selected as the normal bus interrupt or as an interrupt from the TM 9902.

The TMS9901 is controlled via the CRU interface and has a CRU address of >080 (i.e. (R12) = >0100). The upper most CRU address bits are decoded by a PROM on the CONTROLLER board to produce the TMS9901 enable signal SEL9901-. The lower five bits of the CRU address, A10-A14, are used by the TMS9901 to address the individual control or data bits. The TMS9901 is clocked by the 3.0 MHz clock REFCLK from the TM990 BUS.

Based on the condition of the 15 interrupt lines on its inputs the TMS9901 produces an interrupt request signal INTREQ- when any unmasked interrupts are pending and generates the 4-bit Interrupt Vector. Since the INTREQX- signal is not synchronous to the TM990/1481 system clock it must be syschronized by reclocking it on CLKP3- to produce the signal INTREQ-.

7.3.11 Special Control Decode Logic

The DECODE field of the microinstruction (DEC(0-4) ← CMDO(76-80)) goes from the CONTROLLER to the PROCESSOR via the top edge connector J4 on pins J4-6 to J4-10, as shown on sheet 6 of the PROCESSOR schematics.

The DECODE field is used as an address to a decode ROM on the PROCESSOR and another on the CONTROLLER to produce some special control signals. The DECODE CONTROL ROMs are implemented using three SN74S288N 32 x 8 PROMs, two on the PROCESSOR and one on the CONTROLLER. Table 7-2 shows the contents of these PROMs.

TABLE 7-2. DECODE CONTROL ROMS (PROCESSOR and CONTROLLER)

CONTROLLER PROM C11 (U52/N5)	PROCESSOR PROM P3 (U14/D2)	PROCESSOR PROM P4 (U20/E2)
1=CKRTNEN	1=CRUEQU-	1=INCBY1-
2=CRUOP	2=IR7CRU-	2=LDIRX-
3=RST-	3=F7CRU-	3=LREX-
4=FLAGSEN	4=F15CRU-	4=IDLE-
5=FLAGC	5=CRUWRO-	5=blank
6=FLAGB	6=SHXWR-	6=CSWAPB-
7=FLAGA	7=SHWR-	7=CSWAPA-
8=FLAGD	8=SHDATA	8=FSWAP-

DECODE	MNEMONIC	C11 87654321	P3 87654321	P4 87654321
00000	NOP	11110100	11111111	11101111
00001	NOP	11110100	11111111	11101111
00010	FSWAP	11110100	11111111	01101111
00011	CSWAPA	11110100	11111111	10101111
00100	CSWAPB	11110100	11111111	11001111
00101	LDIRLSB	11110100	11111111	11101101
00110	CRUWRO	11110110	11101111	11101111
00111	CRUEQU	11110110	11111110	11101111
01000	WR15CRU	11110110	11110111	11101111
01001	WR7CRU	11110110	11111011	11101111

TABLE 7-2. DECODE CONTROL ROMS (PROCESSOR and CONTROLLER)

DECODE	MNEMONIC	C11 87654321	P3 87654321	P4 87654321
01010	IR7CRU	11110110	11111101	11101111
01011	SHWRO	11110100	10111111	11101111
01100	SHWRZ	11110100	00111111	11101111
01101	SHXWRO	11110100	11011111	11101111
01110	SHXWRZ	11110100	01011111	11101111
01111	LREX	11110100	11111111	11101011
10000	NOP	11110100	11111111	11101111
10001	NOP	11110100	11111111	11101111
10010	LDRTN	11110101	11111111	11101111
10011	RESET	11110000	11111111	11101111
10100	SFLG1	10001100	11111111	11101111
10101	RFLG1	00001100	11111111	11101111
10110	SFLG2	11001100	11111111	11101111
10111	RFLG2	01001100	11111111	11101111
11000	SINTFLG	10101100	11111111	11101111
11001	RINTFLG	00101100	11111111	11101111
11010	SXOPFLG	11101100	11111111	11101111
11011	RXOPFLG	01101100	11111111	11101111
11100	SINTLOC	10011110	11111111	11101111
11101	RINTLOC	00011100	11111111	11101111
11110	IDLE	11110100	11111111	11100111
11111	INCBY1	11110100	11111111	11101110

7.4 CONTROLLER BOARD

7.4.1 Control Memory

The microinstructions are stored in CONTROL MEMORY which is implemented using 10 SN74S478N 1K x 8 Schottky PROMs as shown on sheet 2 of the CONTROLLER schematics.

The CONTROL MEMORY ADDRESS lines (CMAX,CMAP,CMA(0-9)) access the next microinstruction word, CONTROL MEMORY DATA OUT (CMDO(1-80)), which will be loaded into the MICROINSTRUCTION REGISTER (MIR) on the rising edge of the next system clock (CLKC3-). The MIR contains the microinstruction which is currently being executed.

7.4.2 Microinstruction Register

The MICROINSTRUCTION REGISTER (MIR), shown on sheets 2 and 5 of the CONTROLLER schematics, is implemented using five SN74S174N HEX D-TYPE FLIP-FLOPS WITH CLEAR and six SN74S374N OCTAL D-TYPE FLIP-FLOPS. The CLEAR on the S174s is connected to RESET- to cause the microprogram to start execution with address 0 when the RESET switch is activated. The input to the MIR is the CONTROL MEMORY DATA OUT (CMDO(4-80)) word. The output of the MIR is the set of command subfields which execute the microinstruction. The CLOCK CONTROL field of the microinstruction (CMDO(1-3)) is not loaded into the MIR but is loaded directly into the CLOCK PERIOD COUNTER. All of the microprogram sequence control fields are loaded into the S174s so the RESET- signal will force the controls to the conditional branch operation and both branch addresses will be zero causing the microprogram to start at location 0 when the RESET- signal goes away.

7.4.3 Clock Control Logic

The CLOCK CONTROL field of the microinstruction word specifies the period of each microinstruction step from 200 ns to 666.6 ns in increments of 66.6 ns. The CLOCK CONTROL bits of the microinstruction are not loaded into the MIR but directly into the CLOCK PERIOD COUNTER shown on sheet 6 of the CONTROLLER schematics. The counter is an SN74S163N SYNCHRONOUS 4-BIT BINARY COUNTER which is clocked by the 15 MHZ MASTER CLOCK (MC). The counter is loaded with a value from 0 to 7 from the microinstruction word (CMDO(1-3)) and counts up to 8. When the count reaches 8 the signal ENCLK goes to ONE and enables a SYSTEM CLOCK (CLKC0) to be produced on the next rising edge of MC. CLKC0 stays high for one MC period of 66.6 ns and is reset. CLKC0 in turn enables the counter to be loaded with the next count value. If the signal FIXMODE is ZERO then CLKC0 will enable the LOAD on the counter, and if FIXMODE is ONE then CLKC0 will enable the CLEAR on the counter. If the counter is loaded with a 7 from the microinstruction word then the ENCLK signal will occur on the next MC clock, and CLKC0 will occur on the following MC clock. The system clock period will then be 3 MC periods or 200 ns. If the counter is cleared or loaded with 0 from the microinstruction word then the system clock period will be 10 MC periods or 666 ns. If FIXMODE=1 then the system clock period will be 666 ns regardless of the value of the CLOCK CONTROL field of the microinstruction. When a LOAD INTERRUPT is about to occur the CLKSLW signal is used to extend the clock to 666 ns for one time to allow the PENDING INTERRUPT signal time to become stable before the next system clock.

7.4.4 Clock Distribution

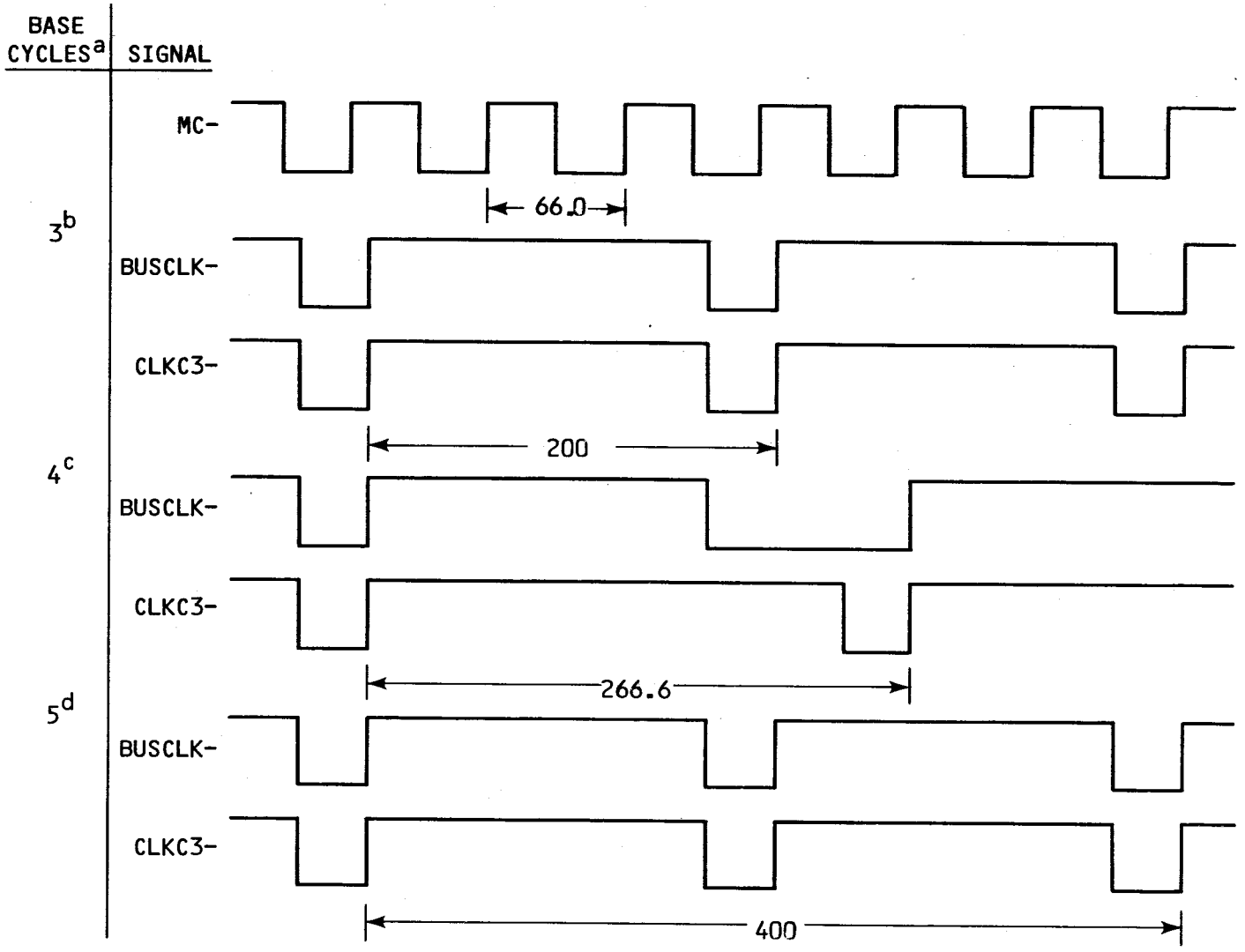
The SYSTEM CLOCK must be distributed to many places on both the PROCESSOR and the CONTROLLER. It must be buffered off the board and buffered for fan-out. In order to minimize the clock skew problem the clock is routed through the same number of levels of propagation on both the CONTROLLER and the PROCESSOR. The clock is named to indicate which board it is on and what propagation level it is. The clock starts on the CONTROLLER board with CLKC0 for example, and the final system clocks which are used on the CONTROLLER and the PROCESSOR are CLKC3- and CLKP3-. The worst case clock skew between CLKC3- and CLKP3- is estimated to be 9 nS. The level 2 clocks are used to produce gated clock pulses for selectively clocking registers as shown on sheet 5 of the CONTROLLER schematics and sheet 9 of the PROCESSOR schematics.

7.4.5 Bus Clock

The signal BUSCLK- on pin J1-22 of the TM990 BUS coincides with the internal system clock of the TM990/1481 (CLKC3-) except when the system clock is stopped for a memory access or a DMA HOLD. When the system clock is stopped the BUSCLK- signal becomes a fixed period 5 MHz clock (200 ns period) with a 33.3% duty cycle. When the system clock resumes the BUSCLK- signal is adjusted or the system clock is delayed so that they occur in synchronism and in a manner that guarantees a BUSCLK period of no less then 200 ns. (See Figure 7-3)

The BUSCLK circuitry is shown on sheet 9 of the CONTROLLER schematics. A signal ENBCK indicates when a system clock is going to be produced and enables the BUSCLK flip flop to track the system clock. A second flip flop BC2, also clocked on MASTER CLOCK (MC), remembers the last state of the BUSCLK flip flop. When the STOP- signal goes LOW indicating that the the system clock has stopped, the circuitry looks at the state of the BUSCLK and BC2 flip flops, and if they indicate that a BUSCLK has not been produced in the last two MC periods then the BUSCLK flip flop is set.

When the STOP- signal goes HIGH the circuitry again tries to track the system clock via the ENCBK signal. The INH signal which indicates that a BUSCLK occurred two MC periods back inhibits ENBCK and system clock in order to prevent a short 133 ns BUSCLK period.



[a]BASE CYCLES = MICROINSTRUCTION MASTER CLOCK CYCLES + MEMORY DELAYS
[b](3N CASE)
[c](3N+1 CASE)
[d](3N+2 CASE)

FIGURE 7-3. BUS CLOCK AND SYSTEM CLOCK TIMING (in nS)

7.4.6 Memory Speed Delay Logic

The Memory Speed Delay logic is designed to allow the TM 990/1481 to operate with several different types of memory boards. The memory expansion boards which were designed to operate with TM 990 microcomputer modules are based on the TMS 9900 microprocessor READY/WAIT handshake protocal. These memory boards do not take the READY signal away fast enough to work with the high speed TM 990/1481 in this asynchronous interface manner. To compensate for this, the TM 990/1481 , when using the READY line, inhibits looking at the READY line for the first 66.67 ns period. This allows the TM 990/1481 to use the READY with the existing and future memory expansion boards but it imposes a speed penalty when operating with the high speed memory boards. In order to resolve this difficulty, the TM 990/1481 implements a synchronous high-speed memory interface which uses the MEMORY SPEED wiring plugs at U94 and U99 to specify the speed of each 2K-word block of the 32K-word logical address space. The TM 990/1481 is thus told how fast a given block of memory is (i.e., what kind of memory board and memory device is being used), and therefore knows how long to wait before assuming the data is stable. (Dynamic RAM memory boards must always used the external READY line because the access time depends upon the refresh operation.)

At the beginning of each memory fetch, the TM 990/1481 stops its system clock and waits for a period of time specified by the wiring plug (shown in Figure 2-2). After that time has expired, the TM 990/1481 assumes that the data is stable on the TM 990 bus and proceeds to load it into the processor. Likewise, on a memory store, the TM 990/1481 waits the specified amount of time and assumes that the address and data have been decoded and enabled; thus, it ends the write operation.

The Memory Speed Delay logic, shown on sheet 6 of the Controller schematics, uses a SN74164N 8-bit shift register to create a set of signals delayed from the system clock in 66.67 ns increments. The signal DCLK- which is delayed from system clock by one 66.67 ns period is used to start the sequence by clearing the shift register. All of the delay signals SHFT1 to SHFT7 go low on the rising edge of SYSTEM CLOCK and each one in succession goes high in 66.67 ns increments (note, SHIFTN represents a delay of N-1 master clock cycles). The SHFTX signal is the latched external READY signal which must be used with dynamic memory boards because of the uncertainty of when the refresh cycles may occur.

One side of each of the two 16-pin memory speed wiring plugs is the set of delay signals SHFT1-SHFT7 and SHFTX, and the opposite side of the two plugs in the set of 16 lines which represent the 16 2K-word blocks of memory whose speed is to be preselected (MDSEL1-MDSEL16). The plugs are programmed by connecting each of the MDSEL lines to one of the memory speed options on the other side of the plug (see Figure 2-2).

The MDSEL lines are routed to a 16-to-1 multiplexer which is implemented using two SN74S251N 8-to-1 mulitplexer chips. The most significant 4 bits of the 16-bit address are used by the multiplexer to decode which 2K-word block of memory is being accessed and to select the appropriate MDSEL signal programmed for that block of memory to drive the internal READY- line. This internal READY- line is used to stop the system clock on a memory access and to release the clock after the programmed delay.

The STOP- signal, which holds the clock off and inhibits the CLOCK PERIOD

COUNTER from advancing, is generated by the SN74S64N AND-NOR logic. The ENABLE MEMORY ADDRESS OUT (ENMAO) signal indicates that a memory access is in progress and is used to enable both the FETCH and STORE to stop the clock via the STOP- signal. Some of the worst case propagation delays are so long, however, that by the time all of the signals which may stop the clock are valid, they may miss the first rising edge of MC after the system clock. Therefore ENMAO is gated with DCLK- to produce the signal GENMAO which is then used as the enable to stop the clock. This guarantees that the STOPsignal will be stable (i.e. HIGH) at the first MC pulse and cannot go low until after the first MC pulse at which time all of the other conditions should be stable.

The memory delays and the microinstruction period control will either overlap or be added together depending on whether the memory access is a STORE or a FETCH. On a FETCH operation the microcode has added extra cycles to the microinstruction period because it intends to do something with the data it has fetched from memory. This extra time must therefore be added after the memory delay circuit has indicated that the FETCH has been completed. On a FETCH the clock is stopped and the CLOCK PERIOD COUNTER inhibited as soon as possible and counting the period is resumed when the memory delay is over. On a STORE operation it is assumed that all of the information for the store is available from the start of the cycle and if the microinstruction period happens to be longer than the memory delay time there is no need to add any more time. On a STORE the CLOCK PERIOD COUNTER is incremented until it reaches the maximum count of 8 and then the clock is stopped only if the memory delay logic indicates that the memory access is not complete.

7.4.7 HOLD and HOLD ACKNOWLEDGE

The HOLD and HOLD ACKNOWLEDGE signals allow external devices to gain access to the TM990 BUS. The HOLD signal on pin J1-92 is received on the CONTROLLER and synchronized to the system clock. The synchronized HOLD signal is used to give the HOLD ACKNOWLEDGE (HOLDA) signal immediately to the TM990 BUS on pin J1-86. The PROCESSOR uses the HOLDA signal to release the data and address busses. The TM 990/1481 then continues to execute microinstructions until the first microinstruction which requires the TM990 BUS. The TM 990/1481 then stops the system clocks (CLKPn and CLKCn) but not the BUSCLK signal. The BUSCLK signal will continue at a fixed period of 200 ns until the system clocks resume, at which time BUSCLK will again track the variable period system clocks.

The INTERLOCK (INTLOC) signal inhibits the TM 990/1481 from responding to the HOLD signal. Currently, the TM 990/1481 generates INTLOC only in the ABS instruction and in all CRU operations. The INTLOC signal allows the ABS instruction to be used to set and reset memory semaphores (flags) without conflicts with other processors communicating with the TM 990/1481 through common memory locations.

The HOLD logic is show on sheet 6 of the CONTROLLER schematics. The HOLD signal is received by an SN74LS132N SCHMIDTT TRIGGER BUFFER to eliminate noise and then clocked thru two flip flops on the 16 MHz MASTER CLOCK (MC) before being synchronized to the system clock (CLKC2) in an SN74S112N JK FLIP FLOP. The synchronized signal is used to generate the HOLDA signal. The signals CRUOP and ENMAO are ORed together to detect that the current microinstruction requires the TM990 BUS and this enables the HOLDEN flip flop to be set to inhibit the system clocks. When the HOLD signal goes away the system clocks will resume with the execution of the pending microinstruction.

7.4.8 Source Select Logic

The SOURCE SELECT LOGIC decodes the SOURCE SELECT field of the microinstruction (SS(0-2) ← CMDO(4-6)) to enable one of six possible sources of addresses to drive the CONTROL MEMORY ADDRESS BUS; - the BRANCH MULTIPLEXER, the RETURN ADDRESS REGISTER, the SCAL PLA, the DCAL PLA, the S field of the IR, and the D field of the IR.

The SOURCE SELECT DEMULTIPLEXER, shown on sheet 3 of the CONTROLLER schematics, is implemented using half of an SN74S139N DUAL 2-TO-4 DEMULTIPLEXER and two SN74LS00N 2-NAND gates. The SOURCE SELECT DEMULTIPLEXER produces the six enable signals, ENBRN-, ENRTN-, ENSCAL-, ENDCAL-, ENIRS-, and ENIRD-.

7.4.9 Branch Multiplexer

The BRANCH MULTIPLEXER provides the capability of a two way branch in CONTROL MEMORY based on the status of the BRANCH- signal. The two branch addresses are provided by the BRANCH ADDRESS 0 (BA0) field of the microinstruction and the BRANCH ADDRESS 1 (BA1) field of the microinstruction. If the BRANCH- signal is ONE (i.e. BRANCH = 0) then the BA0 address is used, and if the BRANCH- signal is ZERO (i.e. BRANCH = 1) then the BA1 address is used.

The BRANCH MULTIPLEXER, shown on sheet 3 of the CONTROLLER schematics, is implemented using three SN74LS257N QUAD 2-TO-1 MULTIPLEXERS. The LS257s have tri-state outputs and are connected directly to the CONTROL MEMORY ADDRESS BUS (CMA(0-9)). The signal ENBRN- from the SOURCE SELECT LOGIC is used to enable the BRANCH MULTIPLEXER to drive the CMA BUS when a conditional branch is selected. The SELECT line on the multiplexers is connected to the BRANCH- signal from the TEST MULTIPLEXER.

7.4.10 Test Multiplexer

The TEST MULTIPLEXER, shown on sheet 3 of the CONTROLLER schematics, selects the test signal that is used to control the two way conditional branch operation in the microcode. The TEST MULTIPLEXER is implemented using three SN74S251N 8-TO-1 MULTIPLEXERS and half of an SN74S139N DUAL 2-TO-4 DEMULTIPLEXER. The most significant two bits of the TEST field, TEST(0-1), are used by the demultiplexer to produce three chip enable signals, TE(0-2)-, which enable one of the three multiplexers. The enabled multiplexer selects one of the eight test signals on its inputs to drive the BRANCH- output line. The BRANCH- line is the signal that controls the SELECT on the BRANCH MULTIPLEXER. If the selected test signal is ZERO then the BRANCH MUX will select the BA0 address, and if the selected signal is ONE then the BRANCH MUX will select the BA1 address.

7.4.11 Test Flags

Four of the test signals into the TEST MULTIPLEXER are called TEST FLAGS. The signals are FLAG1, FLAG2, INTFLG, and XOPFLG, and they come from the TEST FLAG REGISTER shown on sheet 5 of the CONTROLLER schematics. The microprogrammer can set and reset the TEST FLAGS and perform conditional branches based on their condition. The setting and resetting of the TEST FLAGS is controlled via the DECODE field (DEC(0-4)) of the microinstruction word.

The DECODE field is decoded by an SN74S288N 32 X 8 PROM to generate the control signals for the TEST FLAG REGISTER which is implemented using an

SN74LS259N 8 BIT ADDRESSABLE LATCH. The signal FLGSEN enables the FLAG REGISTER to be changed, the signals FLAGA, FLAGB, and FLAGC address the FLAG BIT in the 8 bit register that is to be changed, and the signal FLAGD indicates what value the FLAG is to be set to. The other three decodes out of the ROM are not associated with the FLAG REGISTER; they are the RST- signal which is LOW when the RST instruction is executed, the CRUOP signal which indicates that the address on the TM990 BUS is not a memory address but a CRU address, and the CKRTNEN signal which causes the RETURN ADDRESS REGISTER to be loaded from the BA1 field.

The TEST FLAG REGISTER is cleared by IAQ so the flags are valid only during the execution of a single instruction, and care should be exercised by the microprogrammer not to fetch the next instruction (IAQ) until use of the flags is finished.

The TEST FLAG REGISTER also contains a bit called the INTERLOC bit (INTLOC) which is set to lock out other devices from being given access to the TM990 BUS via the HOLD/HOLD ACKNOWLEDGE protocol. In the 990/9900 instruction set this bit is set during the execution of the ABS instruction to insure that the TM990/1481 will peform a read-modify-write without another DMA device being able to access the unmodified value between the read and write operations.

7.4.12 Return Address Register

The RETURN ADDRESS REGISTER (RTN) is designed to provide one level of microprogram subroutine linkage. The microprogrammer can load the RTN with a 12 bit address value from the BA1 field of the microinstruction at any time when the BA1 field is not being used in a conditional branch or as a literal constant. Also, since the DECODE field is used to enable the RTN load, no other DECODE field function can be perfomed in the same microinstruction. On any subsequent microinstruction the microprogrammer may branch to the location specified by the RTN.

The RETURN ADDRESS REGISTER is implemented using two SN74LS374N OCTAL D-TYPE FLIP FLOP devices, as shown on sheet 3 of the CONTROLLER schematics. The RTN is enabled to drive the CMA BUS by the ENRTN- signal from the SOURCE SELECT LOGIC, and the RTN is loaded (clocked) by the CKRTN- signal generated from CLKC2 and ENCKRTN. ENCKRTN is decoded from the DECODE field by the ROM as shown on sheet 5 of the schematics.

7.4.13 Instruction Register and Entry Point Logic

The INSTRUCTION REGISTER (IR) is duplicated on the PROCESSOR and the CONTROLLER in order to avoid routing the 16 IR bits between the boards. The IR on the CONTROLLER board, shown on sheet 4 of the schematics, is used in conjunction with three PLAs and two buffers in order to generate instruction related or address modification related entry points into CONTROL MEMORY. Since the instruction remains in the IR until the next command to load the IR, it is possible to use the IR multiple times to derive entry points. If several instructions require the same processing for example it is possible for all of these instructions to branch to the same initial entry point. At the end of the common entry point processing it is necessary to again examine the IR to determine a second level of IR derived entry point. The PLAs are arranged to allow 4 different entry points called SCAL, DCAL', DCAL, and OPCAL.

The IR is implemented using two SN74S373N OCTAL TRANSPARENT LATCHES which are clocked on the CKIR signal derived from the LOAD IR (LDIR) bit of the microinstruction word. The entry points are derived using three SN74S330N PLAs and two SN74LS244N OCTAL TRI-STATE BUFFERS. The PLAs have 11 inputs plus an enable compared to the 16 bit length of the IR; the 990/9900 instruction set formats allow different parts of the IR to be used for the different entry points. The source calculation (SCAL) entry point has the most extensive requirements since it is the first entry point and hence all instructions have an SCAL entry point. Some instructions have only the SCAL or first entry point and this reduces the requirements on the remaining entry points.

The TM 990/1481 OP CODE information needed for SCAL does not extend beyond the most significant 12 bits except for some of the floating point instructions. These instructions are all mapped to the same initial entry point and then are individually executed by performing a 16-way branch based on the S field of the IR. The 16 way branch uses the two SN74LS244N OCTAL TRI-STATE BUFFERS to drive least significant bits of the CMA-BUS with the 4-bit S field and two bits of zero fill. The most significant bits are determined by the BA0 MSBs. All of the instructions whose most significant two bits are not both equal to zero are mapped into the same SCAL entry point. ORing these two bits together permits an 11-bit input into the SCAL PLA.

The other two PLAs are used for the remaining three entry points. The least significant two bits of the TEST field determine whether the entry is DCAL', DCAL, or OPCAL and causes the PLAs to generate different entry points for each. The instruction set is split between the two PLAs, one being enabled when the most significant four bits of the IR are not all zero, and the other enabled when they are all zero.

The PLAs only provide the least significant six bits of the entry point. Without modification this would only give 64 unique entry points which is much less than needed. Fortunately each entry point type does not require more than 64 locations, so the four blocks of 64 locations are mapped to different areas of memory by specifying different values for the four most significant bits. This is done by using the most significant four bits of the BA0 field of the microinstruction to drive the most significant bits of the CMA-BUS via one of the two SN74LS244N OCTAL TRI-STATE BUFFERS.

TABLE 7-3. SCAL PLA ENTRY POINTS

INSTRUCTION REGISTER * 23456789AB	ENTRY POINT	OPERATION	INSTRUCTIONS
1 XXXXXXXX00	008	SO,TS=0	SOC,SOCB,MOVB,MOV,AB,A,CB,C,SB,S,SZCB,SZC
1 XXXXXXXX01	009	SA,SO,TS=1	SOC,SOCB,MOVB,MOV,AB,A,CB,C,SB,S,SZCB,SZC
1 XXXXXXXX10	00A	SA,SO,TS=2	SOC,SOCB,MOVB,MOV,AB,A,CB,C,SB,S,SZCB,SZC
1 XXXXXXXX11	00B	SA,SO,TS=3	SOC,SOCB,MOVB,MOV,AB,A,CB,C,SB,S,SZCB,SZC
0 1XXXXXXX00	008	SO,TS=0	DIV,MPY,LDCR,XOR,CZC,COC
0 1XXXXXXX01	009	SA,SO,TS=1	DIV,MPY,LDCR,XOR,CZC,COC
0 1XXXXXXX10	00A	SA,SO,TS=2	DIV,MPY,LDCR,XOR,CZC,COC
0 1XXXXXXX11	00B	SA,SO,TS=3	DIV,MPY,LDCR,XOR,CZC,COC
0 1X10XXXX00	008	SO,TS=0	DIV,MPY,LDCR,XOR,CZC,COC
0 1X01XXXX00	008	SO,TS=0	DIV,MPY,LDCR,XOR,CZC,COC
0 1X00XXXX00	008	SO,TS=0	DIV,MPY,LDCR,XOR,CZC,COC
0 1101XXXX00	028	SO,TS=0	STCR

TABLE 7-3. SCAL ENTRY POINTS (CONTINUED)

INSTRUCTION REGISTER * 23456789AB	ENTRY POINT	OPERATION	INSTRUCTIONS
0 1011XXXX00	02E	SA,TS=0	XOP
0 011111XXXX	033	CRU	TB
0 011110XXXX	003	CRU	SBZ
0 011101XXXX	003	CRU	SBO
0 011100XXXX	002	JUMP	JOP
0 0110XXXXXX	002	JUMP	JH,JL,JNO,JOC
0 010XXXXXXX	002	JUMP	JNC,JNE,JGT,JHE,JEQ,JLE,JLT,JMP
0 0011XXXX00	00C	SO,TS=0	STR,LR,DR,MR,SR,CIR,AR, STD,LD,DD,MD,SD,CID,AD
0 0011XXXX01	009	SA,SO,TS=1	STR,LR,DR,MR,SR,CIR,AR STD,LD,DD,MD,SD,CID,AD
0 0011XXXX10	00A	SA,SO,TS=2	STR,LR,DR,MR,SR,CIR,AR STD,LD,DD,MD,SD,CID,AD
0 0011XXXX11	00B	SA,SO,TS=3	STR,LR,DR,MR,SR,CIR,AR STD,LD,DD,DD,SD,CID,AD
0 0011100000	038		ILLEGAL OP CODES
0 0011X00001	039		ILLEGAL OP CODES
0 0011X00010	03A		ILLEGAL OP CODES
0 0011X00011	03B		ILLEGAL OP CODES
0 0011000000	03C	Ex.Fl.Pt	CED,CER,CDE,CRE,NEGD,NEGR,CDI,CRI
0 0010XX0000	006	SHOP,C=0	SRC,SLA,SRL,SRA
0 0010XX1XXX	004	SHOP,C<>0	SRC,SLA,SRL,SRA
0 0010XXX1XX	004	SHOP,C<>0	SRC,SLA,SRL,SRA
0 0010XXXX1X	004	SHOP,C<>0	SRC,SLA,SRL,SRA
0 0010XXXXX1	004	SHOP,C<>0	SRC,SLA,SRL,SRA
0 0001111X00	038		ILLEGAL OP CODES
0 0001111X01	039		ILLEGAL OP CODES
0 0001111X10	03A		ILLEGAL OP CODES
0 0001111X11	03B		ILLEGAL OP CODES
0 0001XXXX01	009	SA,SO,TS=1	GROUP 7 INSTRUCTIONS
0 0001XXXX10	00A	SA,SO,TS=2	GROUP 7 INSTRUCTIONS
0 0001XXXX11	00B	SA,SO,TS=3	GROUP 7 INSTRUCTIONS
0 0001100X00	028	SO,TS=0	DECT,DEC
0 000101XX00	028	SO,TS=0	INCT,INC,INV,NEG
0 0001110100	02D	SO,TS=0	ABS
0 0001110000	02C	SO,TS=0	SETO
0 0001101100	02B	SO,TS=0	SWPB
0 0001101000	02A	SO,TS=0	BL
0 0001001100	023	SO,TS=0	CLR
0 0001001000	022	SO,TS=0	X
0 0001000100	021	SO,TS=0	B
0 0001000000	020	SO,TS=0	BLWP
0 000011111X	01F		LREX
0 000011110X	01E		CKOF
0 000011101X	01D		CKON
0 000011100X	01C		RTWP

* NOTE: first column is the OR of IR0 and IR1.

TABLE 7-3. SCAL PLA ENTRY POINTS (CONTINUED)

INSTRUCTION REGISTER * 23456789AB	ENTRY POINT	OPERATION	INSTRUCTIONS
0 000011011X	01B		RSET
0 000011010X	01A		IDLE
0 000011001X	030		ILLEGAL OP CODES
0 000011000X	018		LIMI
0 000010111X	017		LWPI
0 0000101101	032		ILLEGAL OP CODE
0 0000101100	015		STST
0 0000101011	008		ILLEGAL OP CODE
0 0000101010	013		STWP
0 000010100X	005	IMMOP**	CI
0 00001001XX	005	IMMOP**	ORI,ANDI
0 0000100X1X	005	IMMOP**	AI
0 000010000X	010	IMMOP**	LI
0 0000011X00	008	SO,TS=0	MPYS,DIVS
0 0000011X01	009	SA,SO,TS=1	MPYS,DIVS
0 0000011X10	00A	SA,SO,TS=2	MPYS,DIVS
0 0000011X11	00B	SA,SO,TS=3	MPYS,DIVS
0 0000010XXX	034		ILLEGAL OP CODES
0 0000001001	036		LWP
0 0000001000	036		LST
0 0000000XXX	036		ILLEGAL OP CODES

* NOTE: first column is the OR of IR0 and IR1.

** NOTE: immeditate operands

TABLE 7-4. PLA 2A ENTRY POINTS: DCAL', DCAL, OPCAL

INSTRUCTION REGISTER 012345678	TEST 34	ENTRY POINT	TYPE	OPERATION	INSTRUCTION
1XXXXXXXX	10	008	DCAL'	SO	SOCB,SOC,MOVB,MOV,AB,A,CB,C
01XXXXXXX	10	008	DCAL'	SO	SB,S,SZCB,SZC
00111XXXX	10	008	DCAL'	SO	DIV,MPY
001101XXX	10	028	DCAL'	SO	STCR
001100XXX	10	008	DCAL'	SO	LDCR
001011XXX	10	02E	DCAL'	SO	XOP
001010XXX	10	008	DCAL'	SO	XOR
00100XXXX	10	008	DCAL'	SO	CZC,COC
000001100	10	028	DCAL'	SO	DECT,DEC
00000101X	10	028	DCAL'	SO	INCT,INC,INV,NEG
000000011	10	008	DCAL'	SO	MPYS,DIVS
111X00XXX	01	090	DCAL	DO,TD=0	SOCB,SOC
110100XXX	01	090	DCAL	DO,TD=0	MOVB
110000XXX	01	095	DCAL	DO,TD=0	MOV
10XX00XXX	01	090	DCAL	DO,TD=0	AB,A,CB,C
1XXX01XXX	01	091	DCAL	DO,TD=1	SOCB,SOC,MOVB,MOV,AB,A,CB,C
1XXX10XXX	01	092	DCAL	DO,TD=2	SOCB,SOC,MOVB,MOV,AB,A,CB,C

TABLE 7-4. PLA 2A ENTRY POINTS: DCAL', DCAL, OPCAL (CONTINUED)

INSTRUCTION REGISTER 012345678	TEST 34	ENTRY POINT	TYPE	INSTRUCTION	
1XXX11XXX	01	093	DCAL	DO,TD=3	SOCB,SOC,MOVB,MOV,AB,A,CB,C
01XX00XXX	01	090	DCAL	DO,TD=0	SB,S,SZCB,SZC
01XX01XXX	01	091	DCAL	DO,TD=1	SB,S,SZCB,SZC
01XX10XXX	01	092	DCAL	DO,TD=2	SB,S,SZCB,SZC
01XX11XXX	01	093	DCAL	DO,TD=3	SB,S,SZCB,SZC
001111XXX	01	0A3	DCAL	DO	DIV
001110XXX	01	0A2	DCAL	DO	MPY
001101XXX	01	0A1	DCAL	DO	STCR
001100XXX	01	0A0	DCAL	DO	LDCR
0010XXXXX	01	090	DCAL	DO	XOR,CZC,COC
1111XXXXX	11	0F7	OPCAL		SOCB
1110XXXXX	11	0F6	OPCAL		SOC
1101XXXXX	11	0F5	OPCAL		MOVB
1011XXXXX	11	0F3	OPCAL		AB
1010XXXXX	11	0F2	OPCAL		A
1001XXXXX	11	0F1	OPCAL		CB
1000XXXXX	11	0F0	OPCAL		C
0111XXXXX	11	0E7	OPCAL		SB
0110XXXXX	11	0E6	OPCAL		S
0101XXXXX	11	0E5	OPCAL		SZCB
0100XXXXX	11	0E4	OPCAL		SZC
00101XXXX	11	0DA	OPCAL		XOR
001001XXX	11	0D9	OPCAL		CZC
001000XXX	11	0D8	OPCAL		COC
00001011X	11	0CB	OPCAL		SRC
00001010X	11	0CA	OPCAL		SLA
00001001X	11	0C9	OPCAL		SRL
00001000X	11	0C8	OPCAL		SRA

TABLE 7-5. PLA 2B ENTRY POINTS: DCAL', DCAL, OPCAL

INSTRUCTION REGISTER * 456789AB	TEST 34	ENTRY POINT	TYPE	INSTRUCTION
0 111111XX	10	03F	DCAL'	STD
0 111110XX	10	03E	DCAL'	LD
0 111101XX	10	03D	DCAL'	DD
0 111100XX	10	027	DCAL'	MD
0 111011XX	10	026	DCAL'	SD
0 111010XX	10	025	DCAL'	CID
0 111001XX	10	024	DCAL'	AD
0 110111XX	10	037	DCAL'	STR
0 110110XX	10	016	DCAL'	LR
0 110101XX	10	035	DCAL'	DR
0 110100XX	10	014	DCAL'	MR

TABLE 7-5. PLA 2B ENTRY POINTS: DCAL', DCAL, OPCAL (CONTINUED)

INSTRUCTION REGISTER * 456789AB	TEST 34	ENTRY POINT	TYPE	INSTRUCTION
0 110011XX	10	011	DCAL'	SR
0 110010XX	10	012	DCAL'	CIR
0 110001XX	10	031	DCAL'	AR
0 011101XX	10	02D	DCAL'	ABS
0 011100XX	10	02C	DCAL'	SETO
0 011011XX	10	02B	DCAL'	SWPB
0 011010XX	10	02A	DCAL'	BL
0 010011XX	10	023	DCAL'	CLR
0 010010XX	10	022	DCAL'	X
0 010001XX	10	021	DCAL'	B
0 010000XX	10	020	DCAL'	BLWP
0 111111XX	01	0BF	DCAL	STD
0 111110XX	01	0BE	DCAL	LD
0 111010XX	01	0BA	DCAL	CID
0 110111XX	01	0B7	DCAL	STR
0 110110XX	01	0B6	DCAL	LR
0 110010XX	01	0B4	DCAL	CIR
0 001000XX	01	0A4	DCAL	LI
0 011001XX	01	0B2	DCAL	DECT
0 011000XX	01	0B0	DCAL	DEC
0 010111XX	01	0AE	DCAL	INCT
0 010110XX	01	0AC	DCAL	INC
0 010101XX	01	0AA	DCAL	INV
0 010100XX	01	0A8	DCAL	NEG
0 0011XXXX	01	098	DCAL	LIMI
0 001011XX	01	096	DCAL	LWPI
0 001010XX	01	094	DCAL	ILLEGAL OP CODE
0 000111XX	01	08E	DCAL	MPYS
0 000110XX	01	08C	DCAL	DIVS
0 111111XX	11	0FF	OPCAL	STD
0 111110XX	11	0FE	OPCAL	LD
0 111010XX	11	0FA	OPCAL	CID
0 110111XX	11	0C7	OPCAL	STR
0 110110XX	11	0C6	OPCAL	LR
0 110010XX	11	0C4	OPCAL	CIR
0 0010100X	11	0DC	OPCAL	CI
0 0010011X	11	0DB	OPCAL	ORI
0 0010010X	11	0EA	OPCAL	ANDI
0 0010001X	11	0EC	OPCAL	AI

* NOTE: first column is the OR of IR0--IR3.

7.4.14 RS232 Serial Communication Controller

The TMS9902 UNIVERSAL ASYNCHRONOUS COMMUNICATION CONTROLLER integrated circuit is used to control the RS232 interface. The TMS9902 is addressed via the CRU and has a CRU address of >40 (i.e. (R12) = >80).

The TMS9902 is shown on sheet 8 of the CONTROLLER schematics. The 9902 is clocked using a 3.2 MHz clock generated from the 16 MHz master clock. The RS232 inputs and outputs of the 9902 are buffered using SN75189N LINE RECEIVERs and SN75188N LINE DRIVERs

7.4.15 Reset/Preset/Load Controls

Three methods are available for initializing the TM 990/1481 , they are RESET, PRESET, and LOAD. The RESET SWITCH, shown on sheet 9 of the CONTROLLER schematics, is debounced by the SN74LS279N QUAD LATCH to produce the RS- signal which is then synchronized to MASTER CLOCK (MC) as shown on sheet 7 of the schematics to produce RESET-. The RESET- signal clears the Sequence Control fields of the microinstruction to force the computer to start executing microinstructions at CONTROL MEMORY ADDRESS 000 when the RESET signal is removed. The starting sequence of microinstructions performs an interrupt trap using the Transfer Vector at memory locations 0000 and 0002. The RESET also causes an I/O RESET (IORST- on J1-88) which is synchronized to the 3.0 MHz REFCLK- on the TM990 BUS, and is a minimum of two REFCLK periods in duration. The PRESET (PRES-) signal on the TM990 BUS, pin J1-94, has the same effect as the RESET SWITCH.

The RESTART- signal on the TM990 BUS, pin J1-93, causes a LOAD INTERRUPT to occur after two instructions have been allowed to execute. The LOAD INTERRUPT causes an interrupt trap using the Transfer Vector at locations FFFC and FFFE. Executing an LREX instruction has the same effect as the RESTART- signal.

7.4.16 TM990 Bus Memory Control Logic

Three of the Memory Control lines for the TM990 BUS, MEMEN-, DBIN, and IAQ are bits of the microinstruction word. The Memory Enable signal (MEMEN-) on pin J1-80 is the Enable Memory Address Out (ENMAO) bit of the microinstruction (CMDO(50)). The Data Bus In (DBIN) signal on pin J1-82 is the Enable Memory Data Out (ENMDO) bit of the microinstruction (CMDO(51)). The Instruction Acquisition (IAQ) signal on pin J1-19 is the IAQ bit of the microinstruction (CMDO(68)).

MEMEN-	=	J1-80	=	ENMAO	=	CMDO(50)
DBIN	=	J1-82	=	ENMDO	=	CMDO(51)
IAQ	=	J1-19	=	IAQ	=	CMDO(68)

The Write Enable signal (WE-) on pin J1-78 is derived from the ENMDO microinstruction bit and a special Write Enable pulse designed to provide adequate set-up and hold times for the memories. The WE- pulse goes LOW 94 nS after the system clock goes HIGH and it goes HIGH 31 nS before the next system clock goes HIGH. The Memory Cycle signal (MEMCYC-) on pin J1-84 is normally HIGH and goes LOW (synchronously with BUSCLK-) only if the system clock is stopped for a memory access and the time exceeds 266.6 ns. Microinstructions which access memory and do not exceed 266.6 ns do not generate a MEMCYC- signal. *

* In the variable clock mode, memory fetches where SHFT1 is selected and memory store operations where SHFT1 through SHFT4 is selected will not generate a MEMCYC- signal.

7.4.17 Debug Clock Options

Two features have been provided to aid in hardware and firmware development and in trouble-shooting hardware failures; these are the FIXED PERIOD SLOW CLOCK and the SINGLE STEP modes of operation. The FIXED MODE causes the clock to operate at a fixed period of 666.6 ns. This eliminates the complication of the variable clock period, and the slow speed allows the operation with EPROMs without recoding the clock control field of the microinstruction. The SINGLE STEP MODE gives the user the ability to step through the microcode and examine the results statically.

The FIXED MODE SWITCH, shown on sheet 9 of the CONTROLLER schematics, is debounced using the SN74LS279N QUAD LATCH and then synchronized to BUSCLK using an SN74LS175N QUAD D-TYPE FLIP FLOP to produce the signal FIXMODE. The FIXMODE signal is used by the CLOCK PERIOD COUNTER to cause it to clear the counter to zero every time and therefore count the longest period of 666.6 ns.

The SINGLE STEP MODE SWITCH, shown on sheet 9, is debounced and synchronized to produce the signal SSMODE which is then used to inhibit the regular clock circuitry via the INHCLK signal. The SINGLE STEP CLOCK momentary contact switch is debounced and enabled with SINGLE STEP MODE (SSM) to produce a single clock pulse identical to and synchronous with BUSCLK.

7.4.18 Upper Memory Page Bits

Two bits, CMAX and CMAP, have been provided to expand the CONTROL MEMORY address space to 4K words. These two bits are stored in flip flops and define the current 1K page of CONTROL MEMORY being accessed. A branch to BRANCH ADDRESS 0 (BA0) will address a microinstruction within the current page of CONTROL MEMORY. A branch to BRANCH ADDRESS 1 (BA1) can access a microinstruction anywhere within the 4K address space and will cause the two page bits to be loaded with the new page number. The RETURN ADDRESS REGISTER (RTN) can be loaded with a 12 bit return address from the BA1 field, and a branch to RTN can therefore also access a microinstruction anywhere within the 4K address space and will set the page bits. The ENTRY POINT LOGIC always accesses the 0 page, so a branch to PLA1, PLA2, IRS, or IRD will zero the page bits. To summarize:

1. All entry points are to page 0.
2. The RETURN ADDRESS REGISTER can access anywhere in 4K memory.
3. BRANCH ADDRESS 1 can access anywhere in 4K memory.
4. BRANCH ADDRESS 0 always refers to the current page.

SECTION 8

MICROPROGRAMMING

8.1 GENERAL

This section describes the microinstruction word format used in the TM 990/1481. The microinstruction word controls the execution of the TM 990/1481. Other topics include the following:

- Clock and sequence control
- Data routing and selection
- Operation control
- Status control
- Special control.

8.2 MICROINSTRUCTION WORD

The microinstruction word is composed of control and data bits which perform the following general functions: clock and microprogram sequence control, data routing and selection, ALU operation control, status control, and some special control functions. The microinstruction word is 80 bits wide and it can be subdivided into functional groups as follows:

CLOCK AND SEQUENCE CONTROL

```
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

[..CC.]  [.SS..]  [...TEST....]  [...BRANCH ADDRESS 0.......]
```

```
2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7

[.....BRANCH ADDRESS 1 / CONSTANT.....]
```

DATA ROUTING AND SELECTION

```
3 3 4 4 4 4 4 4 4 4 4 4 5 5
8 9 0 1 2 3 4 5 6 7 8 9 0 1

[..RF....]  [A]  [..B..]  [P]  [..F..]  [.MC.]
```

OPERATION CONTROL

```
5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 7
2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0

[.......ALU OP CODE.....]  [ REG CONTR ] [IAQ ][CNTR ]
```

STATUS CONTROL

```
7 7 7 7 7
1 2 3 4 5

[....STC....]
```

SPECIAL DECODED CONTROL

```
7 7 7 7 8
6 7 8 9 0

[ ....DEC.... ]
```

8.3 CLOCK AND SEQUENCE CONTROL

The CLOCK AND SEQUENCE CONTROL fields control how the microprogram is entered, and how it is sequenced thru conditional and unconditional branches and subroutine linkages, and how long the clock period will be for each microinstruction step.

```
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

[ ..CC. ] [ .SS.. ] [ ...TEST.... ] [ ...BRANCH ADDRESS 0..... ]

                2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
                2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7

                [ .....BRANCH ADDRESS 1 / CONSTANT..... ]
```

The CLOCK AND SEQUENCE CONTROL consists of the following subfields:

CC = CLOCK CONTROL --defines the time to the next clock

SS = SOURCE SELECT --selects the source of the next Control Memory address

TEST = TEST SELECT --selects the test condition for the conditional branch and is also used for entry point control

BA0 = BRANCH ADDRESS 0 -- branch address if TEST=0

BA1/CONST = BRANCH ADDRESS 1 or CONSTANT word
-- branch address if TEST=1 or constant word to be used by Processor or link address to be loaded into Return Address Register

8.3.1 Clock Control

The CLOCK CONTROL field defines the time to the next clock. The TM990/1481 does not have a constant period microinstruction clock but rather allows the microprogrammer to adjust the clock period to suit the microinstruction being

executed. The clock period is counted from a 15 MHz master clock so the time is in increments of 66.6 ns with a minimum period of 3 increments or 200 ns.

Since a high speed variable period clock presents some problems during test and repair, two switch options have been provided. The first option provides a constant period clock of the longest period, 666.6 ns. The second option provides a single step microinstruction clock to allow the technician to step thru the microprogram and examine the results.

TABLE 8-1. CLOCK CONTROL....CMDO(1-3)

CODE	MNEMONIC	DESCRIPTION
000	C0	666.6 ns
001	C1	600.0 ns
010	C2	533.3 ns
011	C3	466.6 ns
100	C4	400.0 ns
101	C5	333.3 ns
110	C6	266.6 ns
111	C7	200.0 ns

8.3.2 Source Select

The SOURCE SELECT field selects the source of the next Control Memory address. The CONTROL MEMORY ADDRESS BUS is a tri-state bus to which are connected several blocks of address generation logic. If a conditional or unconditional branch is desired then the BRANCH ADDRESS MULTIPLEXER is specified as the source. If a return from a subroutine is desired then the RETURN ADDRESS REGISTER is specified as the source. If the INSTRUCTION REGISTER has just been loaded with a new instruction, execution may be initiated by entering the source operand derivation routine for that instruction, then selecting the SCAL PLA as the source for CONTROL MEMORY ADDRESS. The further reentry points of DCAL', DCAL, and OPCAL are obtained by selecting the DCAL PLA as the source and identifying the specific reentry type in the TEST field.

The remaining two SOURCE SELECT options allow the microprogram to branch to an address in CONTROL MEMORY specified partially by the microinstruction word and partially by a field (S or D) in the INSTRUCTION REGISTER. This is used for generating XOP entry points and for the two word (extended) floating point instructions.

TABLE 8-2. SOURCE SELECT....CMDO(4-6)

CODE	MNEMONIC	DESCRIPTION
000	BRN	BRANCH ADDRESS MULTIPLEXER
001	RTN	RETURN ADDRESS REGISTER
010	---	---
011	---	---
100	PLA1	SCAL ENTRY POINT
101	PLA2	DCAL', DCAL, OR OPCAL ENTRY POINTS
110	IRS	IR S FIELD (EXT F.P.) ENTRY POINTS
111	IRD	IR D FIELD (XOP) ENTRY POINTS

8.3.3 Test Select

When the SOURCE SELECT field specifies the BRANCH MUX as the source for CONTROL MEMORY ADDRESS then the TEST field specifies which test signal line to use to determine which of the multiplexer's two inputs to use for the next CONTROL MEMORY ADDRESS, BRANCH ADDRESS 0 or BRANCH ADDRESS 1. When the SOURCE SELECT field specifies the DCAL PLA as the source for CONTROL MEMORY ADDRESS then the TEST field specifies which type of reentry point decode is desired, DCAL', DCAL, or OPCAL.

TABLE 8-3. TEST....CMDO(7-11)

CODE	MNEMONIC	DESCRIPTION
00000	UNCBA0	unconditional branch to BA0
00001	SEQZ	S field of IR equals 0
00010	DEQZ	D field of IR equals 0
00011	MOVW	IR contains the MOV instruction
00100	BYTE	IR contains a byte instruction
00101	JUMP	the jump condition is TRUE
00110	CNTEQZ	COUNTER = 0
00111	UNCBA1	unconditional branch to BA1
01000	LGT *	logical greater than
01001	AGT *	arithmetic greater than
01010	EQ *	equal
01011	COSH *	carry out or shift out
01100	OVFL *	overflow
01101	OP	odd parity
01110	PINT	pending interrupt
01111	POS *	operand positive
10000	A015	ADDRESS LSB (even/odd byte)
10001	F0	F-BUS MSB (sign)
10010	COSAV	saved carry-out
10011	FLG1	general purpose flag 1
10100	FLG2	general purpose flag 2
10101	INTFLG	interrupt flag or G.P. flg 3
10110	XOPFLG	XOP flag or G.P. flg 4
10111	LOAD	LOAD interrupt
11000	---	---
-----	---	---
-----	---	---
11111	---	---

* NOTE: If a conditional branch is executed using these test signals, then the clock period specified in the CLOCK CONTROL field must be 266.6 ns or greater.

If the SOURCE SELECT field is PLA2 then the TEST field is specified as follows:

00001	DCAL	DCAL entry point
00010	DCALP	DCAL' entry point
00011	OPCAL	OPCAL entry point

8.4 DATA ROUTING AND SELECTION

There are three tri-state buses in the PROCESSOR, the A-BUS, the B-BUS, and the F-BUS. The A-BUS and B-BUS are connected to the AI and BI input ports of the S481 processor, and the F-BUS can be driven by the Data Output Port of the S481 processor. The F-BUS can also be driven by the Address Output Port of the S481 and by the Memory Data Input from the TM990 BUS. The DATA ROUTING AND SELECTION subfields control the selection of data to the A-BUS, B-BUS, and F-BUS within the PROCESSOR and controls the access to the TM990 BUS.

```
3 3 4 4 4 4 4 4 4 4 4 4 5 5
8 9 0 1 2 3 4 5 6 7 8 9 0 1

[..RF....][A] [..B..][ P ] [..F..] [.MC.]
```

RF = REGISTER FILE -- address of the register being used

A = A-BUS SELECT -- source of A-BUS data

B = B-BUS SELECT -- source of B-BUS data

P = PC/MC SELECT -- select either Program Counter (PC) or Memory Counter (MC) to go to the Address Output Port (AOP) of the 481 Processor

F = F-BUS SELECT -- source of F-BUS data

MC = MEMORY CONTROL-- specifies memory operation

8.4.1 Register File Address

The REGISTER FILE ADDRESS field selects one of 16 internal registers for data storage or retrieval. The microcode can use any of these registers for temporary storage, but the contents of register R15 must not be destroyed since the processor uses R15 as the WORKSPACE POINTER storage register. The microprogrammer may use R15 within his program as long as he saves and restores the value. The microcode must also be aware that the contents of these registers may be altered by some instructions so that while it can use these registers within an instruction, it should not try to transfer data from one instruction time to another.

Any value on the F-BUS may be loaded into a register in the file by selecting the REGISTER ADDRESS and by setting the LOAD REGISTER FILE (LDRF) bit of the microinstruction. Any register in the file may be brought to the A-BUS or the B-BUS (or both) by selecting the REGISTER ADDRESS and selecting the RF option in either case.

It is possible to read from a register and to write into that register in the same microinstruction cycle, but there is one particular case in which this is not permitted. If the register being used for the simultaneous read and write is selected to drive the A-BUS and the LOAD WORKING REGISTER (LDWR) option is selected, then selecting the LDRF will result in 0 being loaded into WR rather than the register contents. In the case the code can still

load the WR but he must do so via the ALU OP CODE that routes AI to WR rather than via the LDWR command. See section 7.3.6 for a hardware explanation of how the REGISTER FILE operates.

TABLE 8-4. REGISTER FILE ADDRESS....CMDO(38-41)

CODE	MNEMONIC	DESCRIPTION
0000	RF0	register file address 0
0001	RF1	register file address 1
0010	RF2	register file address 2
0011	RF3	register file address 3
0100	RF4	register file address 4
0101	RF5	register file address 5
0110	RF6	register file address 6
0111	RF7	register file address 7
1000	RF8	register file address 8
1001	RF9	register file address 9
1010	RF10	register file address 10
1011	RF11	register file address 11
1100	RF12	register file address 12
1101	RF13	register file address 13
1110	RF14	register file address 14
1111	RF15	WORKSPACE POINTER REGISTER

8.4.2 A-BUS SELECT A

The A-BUS SELECT field selects the source to drive the A-BUS which goes to the AI port of the 481 ALU. The two possible sources are the REGISTER FILE and the SWAP MUX.

The A-BUS can be thought of as the operand bus. Data fetched from memory is normally brought into the Working Register (WR) via the SWAP MUX and the A-BUS. Intermediate results stored in the REGISTER FILE are brought back to the 481 via the A-BUS. The AI input to the 481 differs from the BI input in that a direct path exists from AI to WR which bypasses the ALU logic and therefore requires less setup time. This is why the A-BUS was chosen as the data input path.

The SWAP MUX allows the microcode to swap the two bytes of the data on the F-BUS, and the result is available on the A-BUS. The swap operation is normally used to move the byte to be operated on to the most significant half of the word since in byte microoperations the ALU only operates on the upper byte. The normal state of the SWAP MUX allows data to pass to the A-BUS unchanged, and therefore provides a path from F-BUS to A-BUS. There are three swap commands available: FSWAP = swap unconditionally, CSWAPA = swap if bit 15 of the address is one, and CSWAPB = swap if bit 15 of the address is one and the instruction is a byte related instruction. The swap commands are decodes of the DECODE field of the microinstruction.

TABLE 8-5. A-BUS SELECT....CMDO(42)

CODE	MNEMONIC	DESCRIPTION
0	A-RF	REGISTER FILE to A-BUS
1	A-SM	SWAP MUX to A-BUS

8.4.3 B-BUS SELECT B

The B-BUS SELECT field selects the source to drive the B-BUS which goes to the BI port of the 481 ALU. There are 8 possible sources for the B-BUS; the REGISTER FILE, the STATUS REGISTER, the CONSTANT word, two times the S field of the IR, two times the D field of the IR, two times the DISPLACEMENT field of the IR, two times the COUNT field of the IR, and two times the INTERRUPT VECTOR.

The B-BUS can be thought of as the modifier bus. The words brought to the B-BUS are typically used to modify addresses or data words. The BI input to the 481 has a more extensive set of options available than the AI input and that is why the B-BUS is used for modifier type data.

TABLE 8-6. B-BUS SELECT....CMDO(43-45)

CODE	MNEMONIC	DESCRIPTION
000	B-CON	CONSTANT word to B-BUS
001	B-2S	two times the S field of the IR to B-BUS
010	B-2D	two times the D field of the IR to B-BUS
011	B-2H	two times the DISPLACEMENT field of the IR to B-BUS
100	B-2IV	two times the INTERRUPT VECTOR to B-BUS
101	B-RF	REGISTER FILE to B-BUS
110	B-SR	STATUS REGISTER to B-BUS
111	B-2C	two times the C field of the IR to B-BUS

8.4.4 Address Output Select P

The ADDRESS OUTPUT SELECT field determines whether the MEMORY COUNTER or the PROGRAM COUNTER is output by the 481 ALU.

TABLE 8-7. ADDRESS OUTPUT SELECT....CMDO(46)

CODE	MNEMONIC	DESCRIPTION
0	AOP-MC	MEMORY COUNTER to ADDRESS OUTPUT
1	AOP-MC	PROGRAM COUNTER to ADDRESS OUTPUT

8.4.5 F-Bus Select F

The F-BUS SELECT field specifies the source of the F-Bus data. Three of the sources, SUM, XWR, and WR, are sources internal to the 481 Processor which come out to the Data Output Port (DOP) of the 481 to drive the F-BUS. The Memory Data In (MDI) source connects the data lines on the TM 990 bus to the F-Bus, and the A to F (ATF) source connects the address output port (AOP) of the 74S481 which could contain the output of the MC or PC registers to the F-Bus.

TABLE 8-8. F-BUS SELECT....CMDO(47-49)

CODE	MNEMONIC	DESCRIPTION
000	F-SUM	ALU SUM BUS to F-BUS
001	F-XWR	EXTENDED WORKING REGISTER to F-BUS
010	F-WR	WORKING REGISTER to F-BUS
011	F-MDI	MEMORY DATA IN to F-BUS
100	---	--
101	---	--
110	---	--
111	F-ATF	ADDRESS OUT to F-BUS

8.4.6 Memory Control MC

The MEMORY CONTROL field determines what type of memory operation will be performed FETCH, STORE or TRANSFER.

TABLE 8-9. MEMORY CONTROL....CMDO(50-51)

CODE	MNEMONIC	DESCRIPTION
00	NOP	
01	MEMFTD	F-BUS to DATA BUS (no memory operation)
10	MEMFET	FETCH
11	MEMSTO	STORE

8.4.7 CONSTANT WORD

The CONSTANT WORD comes from the 16 bit BRANCH ADDRESS 1 (BA1) field of the microinstruction. Normally this field contains one of the two branch addresses for a conditional branch, but if the microprogrammer knows that he has selected an unconditional branch to the BRANCH ADDRESS 0 (BA0) word then the contents of the BA1 field is a "don't care" as far as the branch logic is concerned. The microprogrammer is then free to use the BA1 field to introduce a CONSTANT WORD onto the B-BUS of the PROCESSOR. This word might be a number to be added to the data being processed or a mask word used to eliminate unwanted fields in the data.

8.5 OPERATION CONTROL

OPERATION CONTROL

```
5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 7
2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0

[.......ALU OP CODE.....] [ REG CONTR ] [IAQ] [CNTR]
```

ALU OP CODE = OPERATION CONTROL LINES TO 481 ALU

REG CONTR = REGISTER LOADING AND INCREMENTING CONTROL LINES

IAQ = INSTRUCTION ACQUISITION SIGNAL

CNTR = COUNTER CONTROL LINES

8.5.1 ALU Operation Control

The ALU OP CODE controls the SN74S481N processor chips. The format of the op code is explained in the device data manual for the S481. Of all the possible op codes the following are those used to implement the TM990/1481 instruction set.

TABLE 8-10. ALU OP CODE....CMDO(52-62)

ALU CODE	MNEMONIC	OPERATION	
11111000001	NOP	NOP, 0 → SUM-BUS	1F04
10001001011	FFFF>SUM	-1 → SUM-BUS	112C
00001001111	0>WR	0 → WR	013C
00001001011	-1>WR	-1 → WR	012C
00000001011	A>WR	AI → WR(THRU ALU)	002C
00000001111	/AI>WR	/AI → WR	003C
00001000101	B>WR	BI → WR	0114
00011001111	WR/>WR	/WR → WR	033C
00001101101	XWR>WR	XWR → WR	01B4
00011001010	WR+1>WR	WR + 1 → WR	0328
00011001110	WR/+1>WR	/WR + 1 → WR	0338
00011001001	WR-1>WR	WR - 1 → WR	0324
00000000010	AI-BI>WR	AI - BI → WR	0008
00000011001	AI+WR>WR	WR + AI → WR	0064
00000011011	AI+/W>WR	/WR + AI → WR	006C
00011000001	WR+B>WR	WR + BI → WR	0304

TABLE 8-10. ALU OP CODE....CMDO(52-62) (CONTINUED)

ALU CODE	MNEMONIC	OPERATION	
00011000010	WR-BI>WR	WR - BI → WR	0308
00000011010	AI-WR>WR	AI - WR → WR	0068
00011101001	WR+XWR>W	WR + XWR → WR	03A4
00011101010	WR-XWR>W	WR - XWR → WR	03A8
10110001000	AIOWR>WR	AI"OR"/WR → WR	1620
10111010000	WROXWR>W	WR"OR"XWR → WR	1740
10111000000	BIOWR>WR	BI"OR"WR → WR	1700
10111000010	B/OWR>WR	/BI"OR"WR → WR	1708
10100000110	AIABI>WR	AI"AND"BI → WR	1418
11001000000	WR@BI>WR	WR"XOR"BI → WR	1900
10101000110	BIAWR>WR	WR"AND"BI → WR	1518
10101000100	B/AWR>WR	WR"AND"/BI → WR	1510
10101010110	WRAXWR>W	WR"AND"XWR → WR	1558
10101010100	WRAX/>WR	WR"AND"/XWR → WR	1550
11001010000	WREX>WR	WR"XOR"XWR → WR	1940
11101101001	WR<SRL>	SHIFT RT LOGICAL → WR	1DA4
11101101011	WR<SRA>	SHIFT RT ARITHMETIC→WR	1DAC
11101101101	WR<SRC>	SHIFT RT CIRCULAR→WR	1DB4
11101101010	WR<SLA>	SHIFT LF ARITHMETIC→WR	1DA8
11101101100	WR<SLC>	SHIFT LF CIRCULAR→WR	1DB0
01100001011	A>MC	AI → MC	0C2C
01101000101	B>MC	BI → MC	0D14
01101111101	PC>MC	PC → MC	0DF4
01111001011	WR>MC	WR → MC	0F2C
01110000001	B*2>MC	BI * 2 → MC	0E04
11010011001	A*2>MC	AI * 2 → MC	1A64
11010010101	4*XWR>MC	XWR * 4 → MC	1A54
01100000001	A+B>MC	AI + BI → MC	0C04
01111000001	WR+B>MC	WR + BI → MC	0F04
01110111001	PC+B>MC	PC + BI → MC	0EE4
01110101001	XWR+B>MC	XWR + BI → MC	0EA4
01100000010	A-B>MC	AI - BI → MC	0C08
11011011111	B/2>MC	BI/2 → MC	1B7C
01111101001	WR+XR>MC	WR + XWR → MC	0FA4
00101001111	0>XWR	0 → XWR	053C
00101001110	1>XWR	1 → XWR	0538
00100001011	A>XWR	AI → XWR	042C
00101000101	B>XWR	BI → XWR	0514
00101000111	B/>XWR	/BI → XWR	051C
00111001011	WR>XWR	WR → XWR	072C

TABLE 8-10. ALU OP CODE....CMDO(52-62) (CONTINUED)

ALU CODE	MNEMONIC	OPERATION	
00110000001	B*2>XWR	BI*2 → XWR	0604
11011111111	B/2>XWR	BI/2 → XWR	1BFC
00101000001	B-1>XWR	BI - 1 → XWR	0504
00111001110	WR/+1>XR	/WR + 1 → XWR	0738
00101101100	/XR+1>XR	/XWR + 1 → XWR	05B0
00100000001	A+B>XWR	AI + BI → XWR	0404
01000101011	AI+/X>XR	AI + /XWR → XWR	08AC
00100101001	AI+XR>XR	AI + XWR → XWR	04A4
00111101001	WR+XR>XR	WR + XWR → XWR	07A4
11011110001	X+BI*2>X	BI*2 + XWR → XWR	1BC4
00100101010	AI-XR>XR	AI - XWR → XWR	04A8
00111101010	WR-XWR>X	WR - XWR → XWR	07A8
00111101100	XR-WR>XR	XWR - WR → XWR	07B0
10100000111	AAB>XWR	AI"AND"BI → XWR	141C
10101000111	WRABI>XR	WR"AND"BI → XWR	151C
10110010001	AIOXR>XR	AI"OR"XWR → XWR	1644
10100010111	AIAXR>XR	AI"AND"XWR → XWR	145C
10111010001	WROXWR>X	WR"OR"XWR → XWR	1744
10101010101	WRAX/>X	WR"AND"/XWR → XWR	1554
11011110101	B+XWRS>X	(XWR + BI)<SLA> → XWR	1BD4
11010111101	A*2>XWR	A*2 → XWR	1AF4
00010001110	AI/+1>XR	2'S COMP AI → XWR	0438
00010101100	XWR-AI>X	XWR - AI → XWR	04B0
11000001001	WR@AI>XR	WR"XOR"AI → XWR	1824
00111100001	<BX>+W>X	BI"AND"XWR + WR → XWR	0784
11101110000	XRLS>XR	XWR<SLL)> → XWR	1DC0
11010110001	X+AILS>X	(XWR + AI)<SLL> → XWR	1AC4
11010101001	A+WRLS>X	(AI + WR)<SLL> → XWR	1AA4
11010101000	A+W1LS>X	(AI + WR + 1)<SL> → XWR	1AA0
01000001011	A>PC	AI → PC	082C
01001011101	WR>PC	WR → PC	0974
01101001111	0>MC	0 → MC	0D3C
10001001111	0>S	0 → SUM	113C
10001001010	1-1>SUM	1 - 1 → SUM	1128
10000001011	A>S	AI → SUM	102C
10001000101	B>S	BI → SUM	1114
10011001011	WR>S	WR → SUM	132C
10001101101	XWR>S	XWR → SUM	11B4
10001000001	B-1>S	BI - 1 → SUM	1104
10011001110	WR/+1>S	2'S COMP WR → SUM	1338

TABLE 8-10. ALU OP CODE....CMDO(52-62) (CONTINUED)

ALU CODE	MNEMONIC	OPERATION	
10011101001	WR+XR>S	WR + XWR → SUM	13A4
10001101001	XWR-1>S	XWR - 1 → SUM	11A4
10010101001	B+XWR>S	BI + XWR → SUM	12A4
10010101010	B-XWR>S	BI - XWR → SUM	12A8
10011000001	BI+WR>S	BI + WR → SUM	1304
10000101001	AI+XR>S	AI + XWR → SUM	10A4
10000011001	AI+WR>S	AI + WR → SUM	1064
10000001010	AI+CIN>S	AI + 1 → SUM	1028
10000000010	AI-BI>S	AI - BI → SUM	1008
10000101010	AI-XWR>S	AI - XWR → SUM	10A8
10000101011	AI+/XR>S	AI + /XWR → SUM	10AC
10000011011	AI+/WR>S	AI + /WR → SUM	106C
10000001110	/AI+1>S	2'S COMP AI → SUM	1038
10000011100	/AI+WR>S	/AI + WR → SUM	1070
10000101100	/AI+XR>S	/AI + XWR → SUM	10B0
10001101100	XWR+1>S	XWR + 1 → SUM	11B0
10011101100	XR-WR>S	XWR - WR → SUM	13B0
11000100001	AI@BI>S	AI"XOR"BI → SUM	1884
11000110001	AI@XWR>S	AI"XOR"XWR → SUM	18C4
11000101001	AI@WR>S	AI"XOR"WR → SUM	18A4
10110101001	AIOWR>S	AI"OR"WR → SUM	16A4
10100101111	AIAWR>S	AI"AND"WR → SUM	14BC
10110110001	AIOXWR>S	AI"OR"XWR → SUM	16C4
10100100111	AIABI>S	AI"AND"BI → SUM	149C
10110100001	AORB>S	AI"OR"BI → SUM	1684
10111110001	WROXWR>S	WR"OR"XWR → SUM	17C4
10101100111	BAWR>S	BI"AND"WR → SUM	159C
10111100001	BIOWR>S	BI"OR"WR → SUM	1784
10011100001	<BX>+W>S	BI"AND"XWR + WR → SUM	1384
10001100100	<BX>+1>S	BI"AND"XWR + 1 → SUM	1190
10000010001	A+BAWR>S	AI + B"AND"WR → SUM	1044
11101100001	AIRS>SUM	AI<SRL> → SUM	1D84
11101100000	AILS>S	AI<SLL> → SUM	1D80
10011001011	WR<0>	WR COMPARED TO 0	132C
10001101101	XWR<0>	XWR COMPARED TO 0	11B4
11100101111	XWR<WR>	XWR COMPARED TO WR	1CBC
11100101101	WR<XWR>	WR COMPARED TO XWR	1CB4
11100011001	AI<WR>	AI COMPARED TO WR	1C64
11100011011	<WR>AI	WR COMPARED TO AI	1C6C
11100101001	AI<XWR>	AI COMPARED TO XWR	1CA4

TABLE 8-10. ALU OP CODE....CMDO(52-62) (CONCLUDED)

ALU CODE	MNEMONIC	OPERATION	
11100101011	XWR<AI>	XWR COMPARED TO AI	1CAC
10101110011	XAWR/<0>	XWR"AND"/WR COMPARED TO 0	15CC
11010110111	XAWR<0>	XWR"AND"WR COMPARED TO 0	15DC
10001001010	GENOV	FORCE OVERFLOW	1128
11101111000	LDBLSHFT	LEFT DOUBLE SHIFT LOGICAL	1DE0
11101111001	RDBLSHFT	RIGHT DOUBLE SHIFT LOGICAL	1DE4
11101111100	LDBLSHCR	LEFT DOUBLE SHIFT CIRCULAR	1DF0
11101111101	RDBLSHCR	RIGHT DOUBLE SHIFT CIRCULAR	1DF4
11101011000	SUBLSDBL	SUBTRACT AND SHIFT LEFT DOUBLE	1D60
11101001001	ADDLSDBL	ADD AND SHIFT LEFT DOUBLE	1D24
11110010011	MPY	MULTIPLY	1E4C
11110111111	SMPY	SIGNED MULTIPLY	1EFC
11110000110	DIVA	DIVIDE	1E18
11110001110	DIVB	DIVIDE	1E38
11110001010	DIVC	DIVIDE	1E28
11110010111	SDIVA	SIGNED DIVIDE	1E5C
11110101111	SDIVB	SIGNED DIVIDE	1EBC
11110110111	SDIVC	SIGNED DIVIDE	1EDC
11110100111	SDIVD	SIGNED DIVIDE	1E9C
11110011111	SDIVE	SIGNED DIVIDE	1E7C

8.5.2 Processor Register Control

The REGISTER CONTROL bits allow the microcode to perform some frequently occurring register operations in parallel with the operation specified by the remainder of the microinstruction. The Program Counter (PC) for example can be incremented at any time independent of the operations specified by the other control fields. Note that INCMC and INCPC cannot both be executed in the same microinstruction.

INCMC....CMDO(63)....Increment the Memory Counter

INCPC....CMDO(64)....Increment the Program Counter

LDWR.....CMDO(65)....Load the Working Register directly from the AI input port

LDRF.....CMDO(66)....Load the Register File register specified by the RF subfield with the F-BUS data

LDIR.....CMDO(67)....Load the Instruction Register from the TM 990 bus

8.5.3 Instruction Acquisition

The Instruction Acquisition (IAQ) bit is used to indicate, via the IAQ signal on pin J1-19 of the TM990 BUS, that the current memory fetch operation is an instruction fetch. The actual instruction fetch is programmed via the commands AOP-PC, MEMFET, and LDIR.

8.5.4 Counter Control

The SHIFT COUNTER counts the clock times for operations such as shifts, divides, multiplies, and other iterative operations. The SHIFT COUNTER may be decremented by one, loaded with a count value, or set to 15. The counter is 4 bits wide and is loaded from the F-BUS bits F11 thru F14 when the LDCNT bit of the microinstruction is set. The count loaded is therefore actually F/2. The COUNT=0 (CNTEQZ) signal from the counter goes to the BRANCH TEST MUX so the microprogrammer may do a conditional branch on the CNTEQZ condition.

In the normal loop structure the counter is loaded with a value N, an operation is performed, the counter is decremented by one, and a conditional branch is executed on CNTEQZ. If the loop is a small loop and the decrement command and the conditional branch are in the same microinstruction, then the loop will be executed N+1 times because the branch uses the current counter status at the beginning of the microinstruction period and the decrement does not occur until the end of the period. The SETCNT15 command therefore allows a loop to be set up to execute 16 times.

TABLE 8-11. COUNTER CONTROL....CMDO(69-70)

CODE	MNEMONIC	DESCRIPTION
00	NOP	
01	DECCNT	decrement the counter by one
10	LDCNT	load the counter from the F-BUS
11	SETCNT15	set the counter to 15

8.6 STATUS CONTROL

The STATUS LOGIC allows the microcode to transfer the existing status conditions at any time to the STATUS REGISTER. The microinstruction control word allows either individual status bits to be enabled or certain logical groups of bits to be enabled simultaneously. The STATUS REGISTER can be loaded in three ways, bits 0-7 from current conditions selectively under microinstruction command, bits 12-15 directly from the F-BUS (MASK load), or all 16 bits at once loaded directly from the F-BUS.

TABLE 8-12. STATUS CONTROL....CMDO(71-75)

CODE	MNEMONIC	DESCRIPTION	
00000	NOP		
00001	COMP	COMPARE	ENABLE=012XXXX
00010	COMPB	COMPARE BYTES	ENABLE=012XX5XX
00011	ARITH	ARITHMETIC	ENABLE=01234XXX
00100	ARITHB	ARITHMETIC BYTES	ENABLE=012345XX
00101	SHIFT	SHIFT	ENABLE=0123XXXX
00110	SHIFTL	SHIFT LEFT ARITHMETIC	ENABLE=01234XXX
00111	CRUCLK	ENABLE CRU CLOCK	
01000	SHTEST	SAVE SHIFT IN ST3	ENABLE=XXX3XXXX
01001	USENSAV	USE SAVED CARRY AND SAVE NEW CARRY	
01010	COMPOV	COMPARE & OVERFLOW	ENABLE=012X4XXX
01011	----		
01100	----		
01101	----		
01110	----		
01111	----		
10000	ENLGT		ENABLE=0XXXXXX
10001	ENAGT		ENABLE=X1XXXXX
10010	ENEQU		ENABLE=XX2XXXX
10011	ENCO		ENABLE=XXX3XXXX
10100	ENOV		ENABLE=XXXX4XXX
10101	ENOP		ENABLE=XXXXX5XX
10110	ENXOP		ENABLE=XXXXXX6X
10111	USECOSAV	USE SAVED CARRY	
11000	ALUSPLIT	OPERATE ON UPPER BYTE OF ALU ONLY	
11001	SAVOV	SAVE OVERFLOW	
11010	SAVCO	SAVE CARRY OUT IN COSAV FLIP FLOP	
11011	SAVSH	SAVE SHIFT OUT (IN STATUS BIT 3)	
11100	SAVCOST	SAVE CARRY OUT IN STATUS REGISTER BIT 3	
11101	LDSTATUS	LOAD STATUS FROM F-BUS BUT NOT MASK	
11110	LDMASK	LOAD MASK FROM F-BUS	
11111	LDSR	LOAD STATUS REGISTER FROM F-BUS	

8.7 SPECIAL CONTROL FIELD

The DECODE field is used to generate a number of different control signals within the PROCESSOR and the CONTROLLER. These functions can be summarized as follows:

1. control of the SWAP MUX
2. control of CRU input and output data
3. control of ALU serial shift lines during logical shifts
8. IDLE and LREX signals to the TM990 BUS
5. loading the Return Address Register (RTN) from the BA1 field
6. setting and resetting FLAGS
7. I/O RESET function
8. allow the MC or PC to be incremented by 1 rather than 2

TABLE 8-13. SPECIAL (DECODE) CONTROL....CMDO(76-80)

CODE	MNEMONIC	DESCRIPTION
00000	NOP	
00001	NOP	
00010	FSWAP	FORCE SWAP BYTES
00011	CSWAPA	CONDITIONAL SWAP BYTES (AO15=1)
00100	CSWAPB	CONDITIONAL SWAP BYTES (AO15=1*BYTE=1)
00101	LDIRLSB	LOAD LEAST SIGNIFICANT 4 BITS OF IR
00110	CRUWRO	CRUIN TO WRO
00111	CRUEQU	CRUIN TO ST3
01000	WR15CRU	WR15 TO CRUOUT
01001	WR7CRU	WR7 TO CRUOUT
01010	IR7CRU	IR7 TO CRUOUT
01011	SHWRO	SHIFT 1 → WR
01100	SHWRZ	SHIFT 0 → WR
01101	SHXWRO	SHIFT 1 → XWR
01110	SHXWRZ	SHIFT 0 → XWR
01111	LREX	LREX INSTRUCTION EXCUTING
10000	-----	
10001	NOP	
10010	LDRTN	LOAD RETURN ADDRESS REGISTER
10011	RESET	EXECUTE I/O RESET
10100	SFLG1	SET FLAG 1
10101	RFLG1	RESET FLAG 1
10110	SFLG2	SET FLAG 2
10111	RFLG2	RESET FLAG 2
11000	SINTFLG	SET INTERRUPT FLAG (SET FLAG 3)
11001	RINTFLG	RESET INTERRUPT FLAG (RESET FLAG 3)
11010	SXOPFLG	SET XOP FLAG (SET FLAG 4)
11011	RXOPFLG	RESET XOP FLAG (RESET FLAG 4)
11100	SINTLOC	SET PROCESSOR INTERLOCK (SET HOLD INHIBIT)
11101	RINTLOC	RESET PROCESSOR INTERLOCK (RESET HOLD INHIBIT)
11110	IDLE	IDLE INSTRUCTION EXECUTING
11111	INCBY1	INCREMENT MC OR PC BY 1 RATHER THAN 2

SECTION 9

INTERFACE DESCRIPTIONS

9.1 GENERAL

This section provides a description of the TM 990/1481 interfaces. Topics include the following:

- TM 990 Bus Interface
- Processor/Controller Interface
- Terminal Interface.

9.2 TM 990 BUS INTERFACE

The TM 990 bus is specified in the TM 990 System Specification.

9.3 PROCESSOR/CONTROLLER INTERFACE

The Processor and Controller are connected via their common top edge connectors J3 and J4. The cables are simple one to one connections and they can be of the flat cable variety. Holes have been provided for pin-and-socket connectors if a more secure connection is desired.

TABLE 9-1. PROCESSOR AND CONTROLLER TM990 BUS CONNECTOR (P1)

1 GND	26 (CLK.B-)	51 (VBATT)	76 +12V
2 GND	27 GND	52 (VBATT)	77 GND
3 VCC	28 (EXTCLK.B-)	53 (XA0.B)	78 WE.B-
4 VCC	29 CRUIN.B	54 (XA1.B)	79 GND
5 INT8.B-	30 CRUOUT.B	55 (XA2.B)	80 MEMEN.B-
6 INT7.B-	31 GND	56 (XA3.B)	81 GND
7 INT10.B-	32 BUSY.B-	57 A0.B	82 DBIN.B
8 INT9.B-	33 D0.B	58 A1.B	83 GND
9 INT12.B-	34 D1.B	59 A2.B	84 MEMCYC.B-
10 INT11.B-	35 D2.B	60 A3.B	85 GND
11 INT14.B-	36 D3.B	61 A4.B	86 HOLDA.B
12 INT13.B-	37 D4.B	62 A5.B	87 CRUCLK.B-
13 INT2.B-	38 D5.B	63 A6.B	88 IORST.B-
14 INT15.B-	39 D6.B	64 A7.B	89 GND
15 INT3.B-	40 D7.B	65 A8.B	90 READY.B
16 INT1.B-	41 D8.B	66 A9.B	91 GND
17 INT5.B-	42 D9.B	67 A10.B	92 HOLD.B-
18 INT4.B-	43 D10.B	68 A11.B	93 RESTART.B-
19 IAQ.B	44 D11.B	69 A12.B	94 PRES.B-
20 INT6.B-	45 D12.B	70 A13.B	95 (GRANTOUT.B-)
21 GND	46 D13.B	71 A14.B	96 (GRANTIN.B-)
22 BUSCLK.B-	47 D14.B	72 A15.B	97 VCC
23 GND	48 D15.B	73 -12V	98 VCC
24 REFCLK.B-	49 (VAUX)	74 -12V	99 GND
25 GND	50 (VAUX)	75 +12V	100 GND

NOTE : Signals in parentheses are neither used by nor generated by the TM990/1481

TABLE 9-2. PROCESSOR/CONTROLLER INTERFACE CONNECTOR (P3)

1 TC	21 /LOAD
2 TD	22 RFAD(3)
3 /SEL9901	23 RFAD(2)
4 /TINT	24 RFAD(1)
5 /MAP	25 RFAD(0)
6 /CLKP1	26 SELPC
7 BYTEX	27 DECCNT
8 COSAV	28 LDCCNT
9 CNTEQZ	29
10 /LREX	30 OP(10)
11 /IDLE	31 OP(9)
12 FO	32 OP(8)
13 PARITY	33 OP(7)
14 OVFL	34 OP(6)
15 COSH	35 OP(5)
16 EQ	36 OP(4)
17 LGT	37 OP(3)
18 AGT	38 OP(2)
19 JUMP	39 OP(1)
20 PINT	40 OP(0)

TABLE 9-3. PROCESSOR/CONTROLLER INTERFACE CONNECTOR (P4)

1	LDIR	21	STC(3)
2	LDRF	22	STC(2)
3	LDWR	23	STC(1)
4	INCPC	24	STC(0)
5	INCMC	25	BA(19)
6	DEC(4)	26	BA(18)
7	DEC(3)	27	BA(17)
8	DEC(2)	28	BA(16)
9	DEC(1)	29	BA(15)
10	DEC(0)	30	BA(14)
11	BSEL(2)	31	BA(13)
12	BSEL(1)	32	BA(12)
13	BSEL(0)	33	BA(11)
14	FSEL(2)	34	BA(10)
15	FSEL(1)	35	BX(5)
16	FSEL(0)	36	BX(4)
17	/ENRFA	37	BX(3)
18	ENMDOD	38	BX(2)
19	ENMDO	39	BX(1)
20	STC(4)	40	BX(0)

9.4 TERMINAL INTERFACE

The J2 connector on the CONTROLLER allows connection to any RS-232 device such as the TI Silent 700 terminal or to the TM 990/301 Microterminal.

TABLE 9-4. CONTROLLER RS-232 CONNECTOR (P2)

1	GND	14	+5V
2	RS232 RCV	15	XMT CLK
3	RS232 XMT	16	RESTART
4		17	RCV CLK
5	CTS	18	TTY RCV
6	DSR	19	
7	GND	20	DTR
8	DCD	21	
9		22	
10		23	TTY RCV RTN
11		24	TTY XMT RTN
12	+12V	25	TTY XMT
13	-12V		