SPNU007

TMS7000 Evaluation Module User's Guide

8-Bit Microcomputer Family

TMS7000
Evaluation Module
User's Guide

TEXAS
INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes in the devices or the device specifications identified in this publication without notice. TI advises its customers to obtain the latest version of device specifications to verify, before placing orders, that the information being relied upon by the customer is current.

In the absence of written agreement to the contrary, TI assumes no liability for TI applications assistance, customer's product design, or infringement of patents or copyrights of third parties by or arising from use of semiconductor devices described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor devices might be or are used.

Contents

1. Introduction to the RTC/EVM7000 Evaluation Module

The RTC/EVM7000 is a TMS7000 Evaluation Module, referred to throughout this manual as the "EVM". It is designed to emulate the Single-Chip mode of the TMS7000 NMOS and CMOS families. It provides all the signals that would be available from masked ROM parts including the UART functions. The EVM provides the ability to develop, debug, and test programs prior to factory masking.

> **Note:**
>
> The EVM does not support the expansion modes of the TMS7000 family of processors.

Two different EVMs exist, one each to support the NMOS and CMOS technologies. The EVM part numbers are as follows:

NMOS Family:	RTC/EVM7000N-1
CMOS Family:	RTC/EVM7000C-1

Organization of this section:

- Section 1.1 is a manual overview, showing manual organization.
- Section 1.2, Section 1.3, and Section 1.4 cover a general description of the EVM and its different modes.
- Section 1.5 is a list of additional documentation.

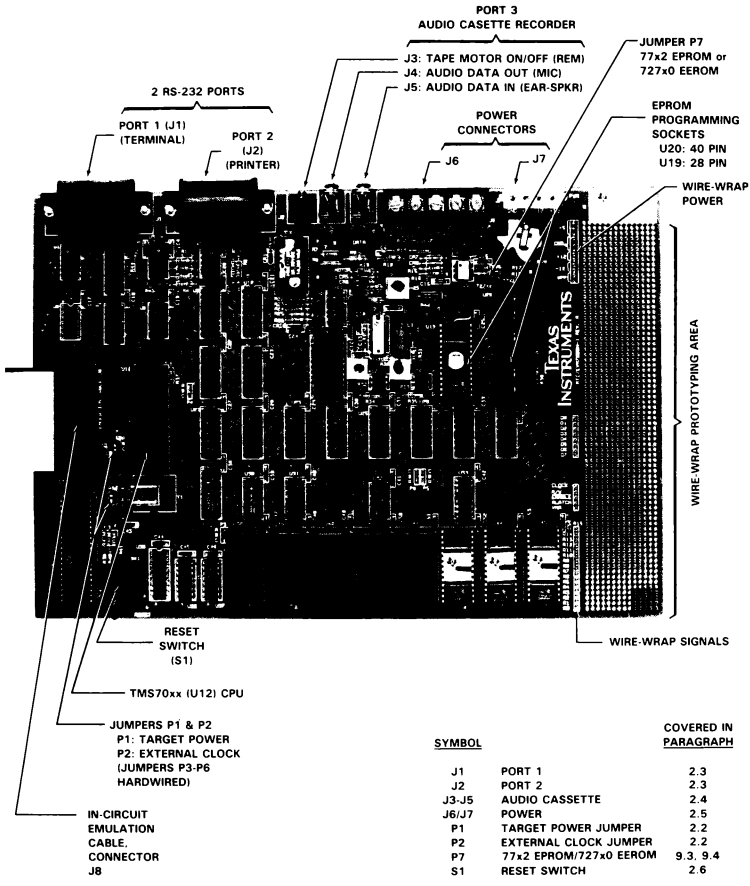
1.1 Manual Organization

This manual is organized as follows:

Section 2	Installation. How to set up the EVM, upgrade it (if necessary) to support the TMS70x2, set jumpers, attach power, attach cables (such as for terminals, printers, and audio recorder), RESET switch, crystal options, UART setup, 7000 family device selection, etc.
Section 3	Development Tools Example. Example of using some of the EVM's software tools to load and execute a small program that blinks the DATA LED when executed. Tools used include the Monitor to hand insert the program by changing memory, "writing" the program using the Text Editor, assembling the program with the Assembler, and executing the program. A more extensive example is in the walkthrough in Appendix C.
Section 4	Text Editor. Using the editor to generate source programs.
Section 5	Assembler. How to input programs into the assembler, assembler fields, assembler directives.
Section 6	Debug Monitor. Explanations of more than 50 commands used to control program execution and debug. Also included are commands to access the EVM system.
Section 7	In-Circuit Emulation. Emulate a program in a target system. Signals are transmitted via a 40-pin emulation cable between the EVM and the target-system's processor socket.
Section 8	Audio Recorder Storage. How to set up to record digital programs using an audio recorder. Includes cabling, tapes, filing.
Section 9	EPROM Programmer. Programming EPROMs using the onboard EPROM sockets and programming software, check for erased conditions, and verify for programming.
Section 10	I/O Ports. I/O ports and corresponding Peripheral File registers.
Section 11	Diagnostics. How to send messages or flags to the terminal during program execution.
Section 12	Creating Monitor Commands. How to develop your own commands.
Appendix A	Schematics of the EVM
Appendix B	Command Format Summary
Appendix C	EVM Walkthrough
Appendix D	Crystal Frequency Dependent Constants

1.2 Functional Overview

The EVM is a single-board development system capable of emulating the Single-Chip mode of the TMS7000 family of microcomputers. Figure 1-1 shows the major functional areas of the EVM.



SYMBOL		COVERED IN PARAGRAPH
J1	PORT 1	2.3
J2	PORT 2	2.3
J3-J5	AUDIO CASSETTE	2.4
J6/J7	POWER	2.5
P1	TARGET POWER JUMPER	2.2
P2	EXTERNAL CLOCK JUMPER	2.2
P7	77x2 EPROM/727x0 EEROM	9.3, 9.4
S1	RESET SWITCH	2.6

Figure 1-1. Major Functional Areas of RTC/EVM7000

The EVM can stand alone as a development system using its Text Editor for creation of TMS7000 assembly language text files with storage on cassette tape. The tape recorder has limited directory- and file-search capabilities. The EVM can also accept text files from a host CPU through either of the two EIA ports. In both situations, the resident assembler will convert the incoming text into executable code in the second pass after resolving labels from the first assembly pass.

The EVM firmware supports three ports in the operations of loading and dumping data (text and object code) for storage and/or display. Ports 1 and 2 conform to the EIA RS-232-C standard, and Port 3 is the audio tape connection. Details about connecting a device to these ports are contained in Section 2.

Port 1	User Terminal/Terminal Emulator
Port 2	Uplink/Downlink to/from Host CPU, or Line Printer
Port 3	Audio Tape

No UART is visible onboard; the EVM implements the UART function for both EIA ports in software, supporting the following baud rates:

110, 150, 300, 600, 1200, 2400, 4800, 9600

The baud rate of Port 1 (terminal) is determined automatically at powerup by striking the carriage return on the terminal (cabled to Port 1) after hitting the RESET switch on the EVM. This feature is called "autobaud" and eliminates the need to select baud rates. The baud rate of Port 2 defaults to 9600 baud at reset, and the baud rates of both ports can be changed with the Monitor command BR.

The EVM firmware is contained in 24K bytes of EPROM. The unused portion of the U45 EPROM is accessed with the Monitor commands U0 through U9. The EVM requires 2K bytes of system RAM, which is separate from the 32K bytes of user RAM. A wire-wrap development area, with all required signals provided and labeled, is available for additional logic.

Since the EVM is intended to be a development tool by using the emulation cable, the crystal frequency of the EVM can be altered to fit the needs of the target system. This procedure is detailed in Section 7 and Appendix D.

1.3 Operating System

The EVM operating system firmware resides in 24K bytes of EPROM and can be divided into three major parts:

- Debug Monitor and EPROM Programmer (Section 6 and Section 9)
- Text Editor (Section 4)
- Assembler (Section 5)

All the software is designed to interact with the user to provide a complete, powerful, and easy to use development tool.

During assembly and debug operations, the EVM RAM can be configured to emulate all TMS7000 family members. For emulation of TMS7020 and TMS704x devices, the EVM allows assembly of text files from RAM, leaving the text intact for immediate editing after execution. After assembly of the Text Editor output, breakpoints can be set based on either addresses or line numbers.

During execution, several modes of fixed displays are available, providing a hex display of the entire register and peripheral files or a binary display of the peripheral ports. During a fixed display, subsequent execution to a breakpoint or execution of a single instruction step will overwrite the old data on the screen with new data. A programmable line of up to six register or peripheral locations is provided for display with breakpoints and instruction steps.

The Text Editor is cursor and line number oriented with autoincrement line numbers, resequence line numbers, change line number, duplicate line, and find string commands. The cursor-oriented edit capability simulates a screen editor by allowing editing of the previous or next line by moving the cursor up or down.

1.4 EVM Configurations

The EVM can function as a peripheral to a host computer, or in a standalone configuration using cassette tape as a mass storage device. Since the EVM's mode depends only on how it is connected, combinations of modes are also possible. All modes support all TMS7000 family members.

1.4.1 Peripheral Mode

Peripheral mode highlights the ability of the EVM Assembler to accept an edited source file from a host system. During the assembly on the EVM, the source listing is sent to Port 1 (terminal connection). After assembly, the address, trap, and interrupt breakpoints can be set. Figure 1-2 shows the peripheral devices in this mode.

Host computers running TMS7000 cross-support software will produce either a 7000 object file that is loaded with the LM command or a Tektronix object file that is loaded with the LT command.

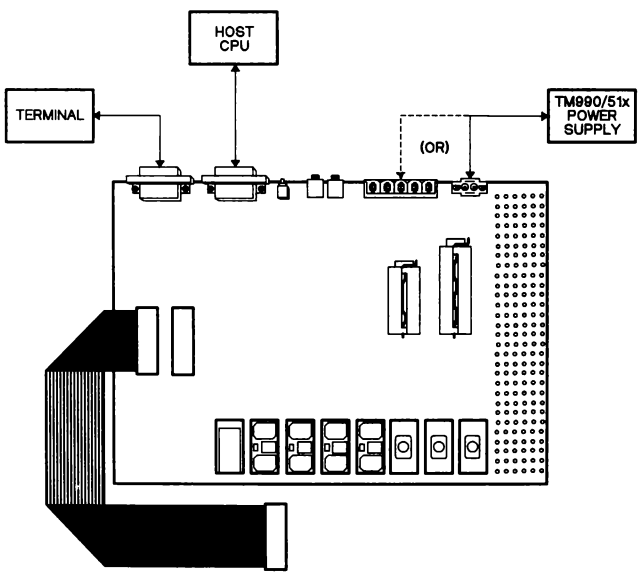


Figure 1-2. EVM Configuration, Periperhal Mode

1.4.2 Standalone Mode

Standalone mode requires no external computer support for the EVM to function as a development system. The file- structured cassette tape interface (with file search) is the mass storage medium. The Text Editor provides for creation and modification of large text files and, for emulation of TMS7020 and TMS704x devices, allows assembly directly from the Text Editor as well as from cassette tape. Figure 1-3 shows the peripheral devices in this mode.

A line printer is not necessary for the EVM to function effectively in Standalone mode with assembly from the Editor. If used, it would be connected at the EVM's Port 2. A unique set of Monitor commands allow breakpoints to be set based on Text Editor line numbers (such numbers are easier to remember and are quickly displayed from the Text Editor as well as the Monitor).

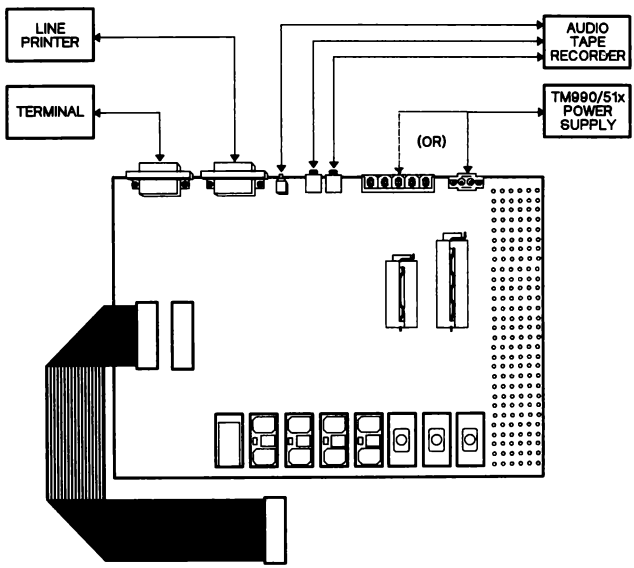


Figure 1-3. EVM Configuration, Standalone Mode

1.4.3 Combining Modes

To optimize the debug aids of the Standalone mode with the mass storage of the Peripheral mode, the EVM can be operated in a way that combines the best features of both. A hardware configuration similar to Peripheral mode places emphasis on the disk storage of the host while still using the EVM Text Editor. A hardware configuration similar to Standalone mode utilizes a personal computer or smart terminal at Port 1 running terminal emulation software, thus allowing the EVM to store data on disks tied to this device. In either instance, there are two options:

1) Use a text editor on the host system. Download the file to the EVM Text Editor for assembly from RAM. This gives access to a full screen editor on the host (if available) and allows storage of the file to disk without doing an upload from the EVM Text Editor. In this configuration, the EVM Text Editor is used only to hold source for the Assembler. Since the host editor does not have line numbers, the No Line Numbers flag (N) is required in the download command string.

2) Use the Text Editor on the EVM. Download the file from the host to the EVM Text Editor at the start of a debug session, and upload the file to the host for storage at the end of the debug session. Since the EVM Text Editor is used for editing in this configuration, the No Line Numbers flag (N) is not needed in the download command string. Line numbers are stored with the file on the host with the upload command.

1.5 Other Applicable Documents

Other documents that may be helpful when using the EVM include:

- TMS7000 Family Data Manual
- TMS7000 Assembly Language Programmer's Guide

2. Installation And Operation

The EVM can be installed as received from the factory or it can be configured according to custom requirements. Section 2.1 to Section 2.6 should be read before attempting walkthroughs in either Section 3 or Appendix C. Covered in this section are:

Upgrade to different TMS7xxx devices	Section 2.1
Jumper settings	Section 2.2
Cabling	Section 2.3, Section 2.4
Power and initialization	Section 2.5, Section 2.6
EVM, RAM, and EPROM memories	Section 2.7
System access commands	Section 6.7
Crystal frequencies	Section 2.9
UART and data transmission	Section 2.10
Configure device-type memory	Section 2.11
Change default EPROM address	Section 2.12
Change Port 2 default baud rate	Section 2.13
Change defaults - bell, tab, buffer timeout	Section 2.14
Configure cursor control	Section 2.15

Section 3 and Appendix C contain walkthrough exercises to help in understanding the installation and operation of the EVM and to become familiar with program development. Covered in these "hands-on" walkthroughs are:

Board installation including cabling and initialization

Use of major commands

Program developing using the Text Editor and Assembler

EPROM programming (Appendix C only).

2.1 Upgrade to Support TMS70x2

The RTC/EVM7000N-1 and RTC/EVM7000C-1 EVM boards are designed to support the Single-Chip mode of the TMS70x2 microcomputers without hardware or software modification.

It is important that the EVM processor in U12 match the emulated processor in Peripheral-File size and number of clocks. The easiest means to do this is to change out the EVM processor with the processor-type being emulated.

In Single-Chip mode emulation, the difference between the TMS70x2 and other TMS7000 family members is the addition of 128 bytes of RAM, designated R128 through R255 in the Register File. The EVM Debug Monitor commands that access the Register File automatically sense the presence of the extended Register File and set the upper limit of the commands to either R127 or R255.

Monitor commands affected are:

DS Display Machine State (Section 6.6.17 on page 6-15)
MR Display/Modify Register File (Section 6.6.39 on page 6-32)
FR Fill Register File With Value (Section 6.6.23 on page 6-22)

Prior to installing the TMS70x2, entry of a Register File value greater than 127 would cause an address error. After installing the TMS70x2 the legal range of values is from 0 to 255. Entries greater than 255 will cause an address error.

The machine state display output by the DS Command effectively doubles in size for the TMS70x2 with the addition of 128 registers. This display can be saved to a host computer (Port 2) or audio tape (Port 3) and reloaded with the LS (Load Machine State) command.

User programs are emulated from EVM RAM; however, the Peripheral File registers and general-purpose RAM registers in emulated programs use these same register areas on the EVM's CPU chip. If a program for a 127-register TMS70x1 device is reloaded onto an EVM with a 256-register CPU, the upper 128 registers are ignored and unchanged. If a 256-register TMS70x2-device program is reloaded onto an EVM with a 128-register TMS70x0 CPU, the 256-register contents are accepted with the top 128 registers not stored; however, no error message will be issued.

The EVM firmware maintains a flag value to tell the Debug Monitor of the presence of the additional 128 bytes of RAM. This flag is set at cold reset when the memory is sized, and is toggled by the command "/R".

With direct control of the Register File size flag, this command allows a TMS70x2 EVM to ignore the additional 128 bytes of RAM and to emulate all other NMOS TMS7000 family members.

2.2 Jumper Settings

Caution:

Avoid possible board damage by removing power from the board before connecting or disconnecting any jumpers or cables.

Hardware jumpers P1 and P2 control clock and V_{CC} sources for in-circuit emulation. These are set prior to shipping. Their functions and settings are covered in Section 7.

Jumpers P3 to P6 are hardwired as follows:

P3 B-C: to pin 23 of RAM socket U38:
- Connect A-B for 8K byte RAM, or
- Connect B-C for 2K byte RAM

P4 A-B: +5 Vdc to MC pin of TMS7000, sets Microprocesor mode

P5 A-B: A13 to A input of memory controller to sockets U38 and U40

P6 No connections

2.3 Terminal, Printer, Host Connections (Ports 1 and 2)

Using a standard EIA RS-232-C, type DB25P, Ports 1 and 2 can be connected as follows. (The ports' data configuration contains 10 data bits, no parity.)

- Port 1
  - Connector J1
  - Terminal, RS-232 connection
- Port 2
  - Connector J2
  - Printer, host system (uplink/downlink)

The EVM does not support a 20 mA current loop interface, but adapters can be purchased from outside vendors to perform the conversion. Pins 9 and 10 on the J1 and J2 connectors are provided with ± 12V to power a converter.

The EVM supports terminals operating in the full-duplex mode. For operation with a modem or mainframe computer, some RS-232-C lines need to be swapped, such as pins 2 and 3. Typically, other handshaking lines may need to be changed in these situations. Section 2.10 details the requirements of the EVM EIA structure.

2.4 Connecting the Audio Tape (Port 3)

When used in a standalone configuration, the EVM supports one audio tape recorder as a mass storage device. Connections to the audio tape are detailed in Table 2-1. The recommended tape recorder is a Radio Shack CTR-XX series or equivalent. The audio tape is referred to throughout the manual as "Port 3".

Table 2-1. Audio Tape Connections

FUNCTION	EVM CONNECTION	CASSETTE	CABLE†
Audio Tape Motor	J3	Remote	Submini - Submini
Audio Data Out	J4 (OUT)	Mic/Record	Mini - Mini
Audio Data In	J5 (IN)	Ear/Monitor	Mini - Mini

†Cable connectors are male on each end. Cables are available at local electronics vendors.

The motor control provided by the EVM insures proper starting of the tape during multiple tape block operations. This connection is required. The proper setting of the volume control when reading data from the tape is 5-7 on a scale of 10. Since the optimum volume setting varies from recorder to recorder, you may need to experiment. Recorders with a tone control should be set to 6 on a scale of 10 or to LO for HI/LO settings.

2.5 Applying Power

The power supply must be UL approved with current limiting on all outputs and should have the voltage/minimum current capabilities of +5V/1 A, +12V/0.1 A, and -12V/0.1 A. An onboard connector, J7, is provided for power connection to a TM990/519 power supply. Connector J7 is AMP 1-480702-0 (pins are 350550-1). An assembled EVM/519 power cable is available from the Texas Instruments Atlanta (Ga.) Regional Technology Center [1]. The J7 connections are:

Pin:	1	2	3	4
	GND	+5 V	-12 V	+12 V

Additionally, a screw terminal power connector (J6) is provided for use with a TM990/518 or equivalent power supply. With the EVM positioned such that J6 is at the upper right (component side up), the terminals are from left to right:

+5 V +12 V -12 V GND V_{PP} output

V_{PP} is produced on the EVM from +5 V and changes under program control to match the device being programmed. V_{PP} will be +5 V when not enabled. When power is properly applied to the EVM, the "PWR" LED is lit.

2.6 RESET and ESCAPE

2.6.1 Using the Reset Switch to Start EVM Operation

Use the RESET switch (SW1) to initialize the EVM board:

1) Press RESET.

2) Press <CR> at the terminal connected to Port P1. The EVM automatically determines the terminal baud rate for communication.

If the EVM starts properly, the reset banner message will appear as shown in Figure 2-1.

---

[1] The address and phone of the TI Atlanta RTC is:

Atlanta Regional Technology Center
5515 Spalding Drive
Norcross, Ga. 30092
(404) 662-7945

```
TMS7000 DEBUG MONITOR REV 2.X
DEVICE TYPE = 2 (704X)
SYSTEM RAM = 32256 BYTES
HELP : HE /H $HE
MODIFY : MM/IM MR/IR MP/IP A/MA B/MB MS-P/PC,SP
DISPLAY: DM/DH DS  IO   DV
STATE : C/CP D/DP xC,xN,xZ,xI (x=C/S) EI DI SR
SAVE : SM ST DS
LOAD : LM LT LS
MOVE : MV
FIND : FB
FILL : FM NP  FR
RESET : RT TO
BRKPT : B1 B2 C1 C2 CB DB BT CT DT
TRACE : TO TF IT PT IS TC TR
DEBUG : SS TS FS CS CY   GO EX ET EF   RU
EDITOR : XE L1 L2 LL LA LN
ASSM : XA XL XP  AT   XR
EPROM : PE VE CE RE BC   12 21 25   43 44 45
MATH : HC DC AR
PORTS : BR DR MO
?
```

Figure 2-1. Command Menu at Reset

At this point (banner and Monitor cursor displayed), you can continue in this section or go to one of the walkthroughs in either Section 3 or Appendix C.

If the EVM does not start properly, confirm proper connection of the terminal (see Section 2.10). The RAM is mapped from >0200 to >7FFF, and the RAM size is displayed after a memory check of user RAM. The results of the RAM sizing routine are used throughout the EVM in control of the memory map for emulation of different TMS7000 family members (see Table 2-5). Removal of one of the middle RAM chips will cause addresses below it in the memory map to be ignored.

The EVM distinguishes between warm and cold resets and also knows whether power has been cycled when a reset occurs. If power has been cycled, the following things are done:

- Determine RAM size
- Load default device type (listed in Table 2-5)
- Initialize Text Editor
- Reset assembler label table
- Clear breakpoints on address
- Clear breakpoints on TRAP
- Autobaud Port 1 baud rate
- Default Port 2 baud rate
- Disable software handshake

A power cycle reset can be done with the Monitor command "$RT".

```
?$RT
ARE YOU SURE? (N)
```

Press Y and <CR> for autobaud, and the reset banner will appear along with the monitor prompt (?). Press only <CR> and only the monitor prompt will come up.

2.6.2 Escape Key

The Escape key (<ESC>) is used throughout the EVM firmware to abort the current line being entered. In the Monitor and Text Editor, it aborts the command if it is entered before all characters required to satisfy the command. In the Line-By-Line Assembler (LBLA), it is used to abort the line of code currently being entered. In this case, a label on the line is not placed in the label table.

2.7 Memory

The EVM can address 64K bytes of memory. The entire memory space of the EVM is decoded and sockets provided for on the board in 8K-byte sections. Figure 2-2 shows EVM memory and its two general areas of EPROM and RAM. The explanations of RAM and EPROM that follow describe how the EVM uses RAM in lower memory for software development but has it logically reside in upper memory, the same as on the device. Section 2.8 further explains this by describing the commands used for system access, comparing one command (inspect/change memory) when used for system access and when used for device memory access.

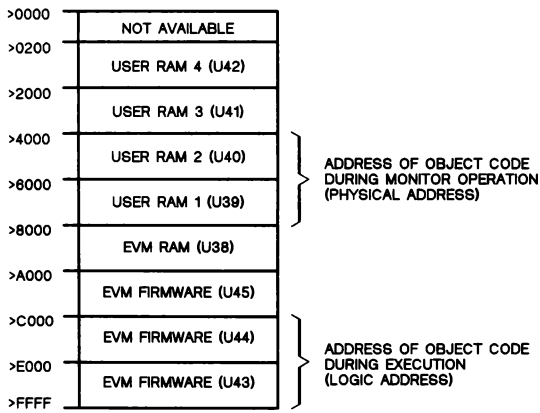


Figure 2-2. EVM Memory Map

2.7.1 RAM

The EVM is equipped with 32K bytes of RAM physically located in the lower addresses of the 64K address map. Programs are developed and debugged in the higher 16K bytes of this RAM. At the time of execution, the higher 16K bytes of this RAM is moved logically to the higher-address end of the address map (bank select), providing execution at the true microcomputer address range.

During Monitor operation, the higher-address 16K bytes of this RAM actually resides from >4000 to >7FFF, but is interfaced to the user as though it was residing from >C000 to >FFFF. This RAM provides storage for up to 16K bytes of object code, allowing full emulation of the entire TMS7000 microcomputer family.

Figure 2-2 details the EVM memory map. Addresses below >200 (512_{10}) are automatically locked out by the address decoding since these locations are used in the emulated register and peripheral files. Thus, USER RAM 4 actually adds only 7.5K bytes.

One 8K-byte socket (U38) is occupied by a 2K-byte static RAM. This RAM is used as system RAM for storing user data during emulation and executing the program to bank-select the user RAM during emulation. It also stores all operating system constants and variables and contains the 512-byte I/O buffer.

Socket U38 is identical to all other 8K-byte sockets on the board with the exception of hardware jumper P3, which allows use of either a 2K-byte or an 8K-byte RAM in the socket. When pins A and B on P3 are connected the EVM expects an 8K-bytes RAM, and when pins B and C are connected the EVM expects a 2K-byte RAM.

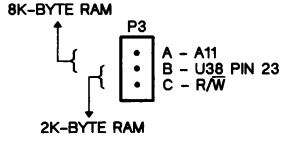


Figure 2-3. 2K/8K RAM Jumper (P3)

2.7.2 EPROM

The EVM firmware occupies 24K bytes (>A000->FFFF). A third socket is provided for EPROM expansion (see Figure 2-2). The EVM firmware resides in three TMS2764 EPROMs (U43, U44, and U45).

2.8 System Access Commands

Special commands are used to access system memory (vs. device memory) and registers on the EVM board. These are covered in the Debug Monitor section (Section 6) as to function since they are similar to the Monitor commands but have a dollar-sign ($) prefix. These commands are in the top of Table 2-2. (do not have a parenthetical reference).

The EVM's memory map is shown in Figure 2-2 on 2-6. The EVM firmware uses memory addresses >4000 to >7FFF range for internal use while displaying these addresses in the >C000 to >FFFF range, which is the range that the RAM will occupy after the memory bank swap that accompanies program execution. In this way, you work in the true microcomputer address range (e.g., >F000 to >FFFF for a TMS7040).

For example, the Monitor MM command accesses addresses corresponding to the memory on the microcomputer. The $MM command accesses addresses actually used by the EVM. As shown in Figure 2-2, these areas are separated by >8000. Thus, a value found by the MM command at >F010 will also be found by the $MM command at >7010.

Table 2-2. System Access Command Summary

COMMAND	DESCRIPTION
$DM	Display Memory
$DS	Display Machine State
$FB	Find Byte in Memory
$FM	Fill Memory
$MM	Display/Modify Memory
$MP	Display/Modify Peripheral File from P0-P255
$MR	Display/Modify Register File
$MV	Move Memory
$PE	Program EPROM (see Section 9.1.1)
$RE	Read EPROM (see Section 9.1.3)
$CE	Compare EPROM (see Section 9.1.2)
$LM	Load 7000 Object Code (see Section 12.2)
$LT	Load Tektronix Object Code (see Section 12.2)
$XA	For Creation of User Commands (see Section 12.2)
$XL	For Creation of User Commands (see Section 12.2)
$XP	For Creation of User Commands (see Section 12.2)
$RU	Execute EVM Firmware in RAM (see Section 12.2)
$RT	EVM Power Cycle Reset (see Section 2.6.1)

2.9 Crystal Frequency Dependent Constants

Crystal frequency dependent constants used in the EVM are listed in the tables in Appendix D. These values reside in EPROM starting at location >FFB2 and are accessible with the Monitor command $MM described in Section 6.7. If the crystal frequency is changed, the appropriate values for that frequency must be placed in the table by creating a new EPROM. This is further explained in Appendix D.

2.10 Software UART

The software UART transmits and receives data (via Ports 1 and 2) with a character format of one start bit, eight data bits, one stop bit, and no parity.

2.10.1 EIA Port Description

The EVM EIA I/O structure communicates through P250 in the Peripheral File. This port is referred to as "PIO". The assignment of the bits is as follows:

PIO7	'6	'5	'4	'3	'2	'1	PIO0
AUDIO USE	DSR IN	EIA DATA IN	AUDIO USE	EIA DATA OUT	AUDIO USE	DTR OUT	EIA PORT SEL.
Msb							Lsb

EIA PORT P250

PIO0 (Lsb)	EIA Port Select (0=Port 2, 1=Port 1)
PIO1	DTR-OUT to EIA
PIO2	FSK OUT to Port 3
PIO3	EIA Data Out
PIO4	Tape Motor (0=ON, 1=OFF)
PIO5	EIA Data In
PIO6	DSR-IN from EIA
PIO7 (Msb)	FSK Data In from Port 3

2.10.2 EIA Communications Protocol (HS Command)

The EVM uses handshaking in transmitting and receiving data through Port 1 and Port 2. Before the EVM transmits data through the selected port it will place +12 V on pin 6 (DTR) of the same port. The EVM will then wait until pin 20 (DSR) of the selected port rises above +4 V (+4 V to +12 V). After the handshake signal is received the data is transmitted. The EVM will check the handshake line before each byte of data is transmitted. If at any time the DSR line is not satisfied, the EVM will wait in a loop until it is satisfied. If the terminal has no handshaking, then pin 20 should be left unconnected since the EVM pulls the handshake line to +12 V through a 2.2 kΩ resistor.

The EVM downloads from Ports 1 and 2 through a 512-byte input buffer. While data is loaded into the buffer, the DTR line from the EVM remains at +12 V. When 500 characters have been received, the DTR line is driven to -12 V after reception of the stop bit of the 480th character. A character timer will continue to receive characters transmitted within two character times of each other. This character timer is in effect whenever buffer input is enabled and will terminate buffer input at any point if transmission stops.

After buffer reception stops, the buffer is unloaded internally by the command currently executing. Control characters (outside the range >20 to >7E) are skipped as the buffer is read internally.

The EVM supports several software handshake protocols via the HS command. Software handshake protocols are disabled at reset. Hardware handshake as described above is enabled at reset and remains enabled during the software protocols selected by the HS command:

HS *current value* {0=Disabled, 1=XON, 2=ACK, 3=Both}<CR,SP>

If no entry is made, the current value of the software handshake flag is retained. If 0 is entered, software handshake is disabled. If both types are enabled (3 is entered), the XON is transmitted first. If an entry out of the legal range is made, an error is issued.

XON Handshake (1 or 3). The EVM indirectly supports XON/XOFF by transmitting the XON character (>11) when input to the download buffer is expected. It then expects 512 (the length of the buffer) or fewer characters. The software UART prohibits transmission of a XOFF character while reading from Port 2. Figure 2-4 shows a FORTRAN routine to transmit 512 characters to a device and then wait for a XON character.

ACK Handshake (2 or 3). The EVM indirectly supports ACK/NACK by transmitting the ACK sequence (0<CR> -- Tektronix handshake) when input to the download buffer is expected. After the record has been received, the buffer times out and is processed internally. Re-entry in the buffer load routine will cause the ACK to be transmitted for the next record. The EVM never transmits the NACK code (7<CR> -- Tektronix handshake), but sends load error messages out Port 1 if input is through Port 2 and out Port 2 if input is through Port 1.

```
C
C    SOURCE FILE NAME = UNIT 3
C
C    ERROR MESSAGE OUTPUT = UNIT 2
C
C    EVM = UNIT 5
C
          DIMENSION ILINE(512)
          INTEGER XON
          DATA XON/>11/
4         READ(3,EOF=3)ILINE
          READ(5,7)IXON
          IF(IXON.EQ.XON)GOTO5
          WRITE(2,2)
2         FORMAT("CHARACTER RECEIVED NOT XON")
          GOTO4
5         WRITE(5)ILINE
3         WRITE(2,7)
7         FORMAT("DOWNLOAD COMPLETE")
          STOP
          END
```

Figure 2-4. Example FORTRAN Download Program

2.10.3 Terminal Emulation Support

Terminal Emulation mode is used when a host computer or an intelligent terminal is connected to the EVM at Port 1, the terminal connection. Input from Port 1 for the commands listed below uses the 512-byte input buffer and hardware/software handshake as described in the previous section.

COMMAND	
LM	Load Memory - 7000 Format
LT	Load Memory - Tektronix Format
LS	Load Machine State
XA	Execute Assembler (only if output port is not Port 1)
XE	Execute Text Editor

Any non-fatal download errors occurring during execution of these commands with input from Port 1 will cause the error message to be transmitted out Port 2. These error messages are transmitted out Port 1 when input is through Port 2.

2.10.4 Upload/Download Procedures

Once any handshake requirements for a given EVM host connection have been resolved, upload/download operations will require a terminal on the EVM at Port 1 and a terminal on the host with the host connected to the EVM at Port 2. If the host is a smart terminal or a personal computer running terminal emulation software, the Port 2 connection is not required as all upload/download is done through Port 1.

The host EIA port must be handled by a device service routine (DSR) that allows both reception (upload) and transmission (download). A common EIA port DSR on a multi-user host is a keyboard service routine (KSR). Note that a KSR usually permits download at baud rates up to 9600 baud, but will limit upload to 300 baud or less since it is designed to handle relatively slow keyboard input. Before starting an upload or download, ensure that all baud rates and character formats match.

Table 2-3 shows the signals associated with the pins on the EVM EIA port. Following the table are cabling examples.

Table 2-3. EVM EIA Port Pin Description

PIN	DESCRIPTION	I/O
1	PROTECTIVE GND	
2	DATA RX	I
3	DATA TX	O
6	DTR(handshake output)	O
7	SIGNAL GND	
8	PDCD (+12V when power on)	O
9	+12V (current limited)	
10	-12V (current limited)	
20	DSR (handshake input)	I

Example 2-1. EVM to 820 KSR Connections

7000 EVM			820 KSR†	
PIN	SIGNAL		SIGNAL	PIN
2	RX	←	TX	2
3	TX	→	RX	3
6	DTR	→	DSR	6
7	GND	↔	GND	7
8	PDCD	→	DCD	8
20	DSR	←	SCA(BUSY)	11

† CONNECT RTS (PIN 4) TO CTS (PIN 5)

Example 2-2. EVM to 743 KSR Connections

7000 EVM			743 KSR	
PIN	SIGNAL		SIGNAL	PIN
1	GND	↔	GND	9
2	RX	←	TX	13
3	TX	→	RX	12
7	GND	↔	GND	1
8	PDCD	→	DCD	11
20	DSR	←	DTR	15

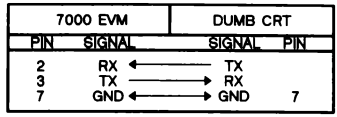
Example 2-3. EVM to 810 Printer Connections

7000 EVM			810 LINE PRINTER	
PIN	SIGNAL		SIGNAL	PIN
1	GND	↔	GND	1
3	TX	→	RX	3
6	DTR	→	DTR	6
7	GND	↔	GND	7
8	PDCD	→	DCD	8
20	DSR	←	DTR(BUSY)	11

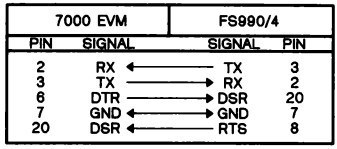
Example 2-4. EVM to 990 EIA Card Connections

7000 EVM			990 EIA CARD	
PIN	SIGNAL		SIGNAL	PIN
2	RX	←	TX	3
3	TX	→	RX	2
6	DTR	→	DSR	20
7	GND	↔	GND	7
8	PDCD	→	DCD	18
20	DSR	←	RTS	8

Example 2-5. EVM to Dumb CRT Connections



Example 2-6. EVM to FS990/4 Connections



Notes:

1. PDCD is +12V when EVM power is on.
2. Dumb CRT has all handshaking disabled.

Table 2-4 lists upload and download EVM commands and data file formats.

Table 2-4. Upload/Download Commands and File Markers

FORMAT	COMMANDS UPLOAD	COMMANDS DOWNLOAD	BOF	EOF
7000 Object	SM	LM†	K	<CR>:
TEK Object	ST	LT	<CR>/	/00000000<CR>
Machine State	DS	LS	none	none
Execute Assembler		XA	*>	*<
Quit Text Editor	Q	XE	*>	*<

† The LM command also accepts "O" as a BOF marker (9900 format).

Upload. At the host terminal, configure the host to accept whatever data will be uploaded (open an edit file for write or execute a copy from the EIA port to a file). Then execute the desired EVM command (Table 2-4) at the EVM terminal, specifying either Port 1 or 2 for the configuration.

Download. Execute the desired EVM command at the EVM terminal (Table 2-4), specifying either input Port 1 or 2 for the configuration above. Then execute a utility at the host terminal to copy or print the file to the EIA port to which the EVM is connected.

2.11 TMS7000 Family Device Type Memory (DV Command)

The EVM can be configured to support the various onboard ROM requirements of the TMS7000 family members, from 2K bytes to 16K bytes. This is referred to as "device type". Note that a device type is not restrictive in reality to that family member, but is used within the EVM to establish default PC values and AORG values and to optimize RAM to best support the family member.

The Monitor command DV is used to change device type:

?DV
current device-type index <device type index><CR,SP>
PC, ST, SR, RA, & RB contents for device

After "DV" is entered, the current device type index is displayed. Legal values for the device-type index are 1 to 5 (listed in Table 2-5). Entering a value out of this range will cause an error and return control to the Monitor without causing any change. If any of the RAM chips have been removed, entering an index for a device requiring more RAM than is on the EVM will cause the device type to default to 1 (see Table 2-5 for device types and required RAM).

After entry, the current device type is displayed. Entering a <CR,SP> will return to the Monitor and save the last digit entered as the new device type if no error occurs. If the device type is changed (or if the present device type is re-entered), the Text Editor will be initialized to adjust for the new memory map and the Monitor CP command will be automatically executed to adjust the program counter. The device type defaults at reset to a value stored in EPROM at >FFB1.

Table 2-5. TMS7000 Family Device Types

INDEX	DEVICE	DEFAULT PC/AORG	MINIMUM RAM REQUIRED
1	TMS702x, TMS70C2x	>F806	8K
2	TMS704x, TMS70C4x	>F006	16K
3	Reserved	>E006	16K
4	Reserved	>D006	16K
5	Reserved	>C006	24K

Notes: 1. The default PC/AORG value is offset by six bytes to allow for mask identification.
2. EPROM address >FFB1 contains device-type default.

2.11.1 RAM Usage by Device Type

Device types 1 and 2 (TMS702x and TMS704x) allow assembling directly from the EVM Text Editor without affecting its contents. This allows for rapid debug of text files limited in size only by the amount of RAM left after the Assembler sets aside space for object code and label table storage. For device types 1 and 2, the RAM memory map is automatically configured as follows:

TYPE:DEVICE	MEMORY MAP CONFIGURATION
1: TMS7020x, TMS70C2x	Upper 2K bytes: Object code storage Next lower 3K bytes: Label table storage Balance: Text Editor text storage

2:	TMS7040x, TMS70C4x	Upper 4K bytes: Object code storage Next lower 6K bytes: Label table storage Balance: Text Editor text storage

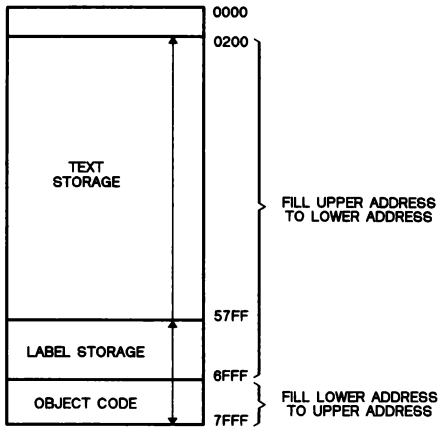


Figure 2-5. Assembly Map for TMS704x/TMS70C4x

For device types 3, 4, and 5, the size of the EVM RAM reserved for object code storage is so large that assembly from RAM is not feasible. These modes are intended to assemble text files sourced external to the EVM, either from a host computer connected at Port 2 or an audio tape connected at Port 3. Device types 1 and 2 can also have files sourced externally.

TYPE:DEVICE	MEMORY MAP CONFIGURATION
3:Reserved	Upper 8K bytes: Object code storage at >6000->7FFF, and labels stored from low end of RAM to >5FFF
4:Reserved	Upper 12K bytes: Object code storage at >5000->7FFF, and labels stored from low end of RAM to >4FFFF
5:Reserved	Upper 16K bytes: Object code storage at >4000->7FFF, and labels stored from low end of RAM to >3FFF

Note that no mechanism exists to stop the label table from expanding into the first part of the text file. The label table uses eight bytes per label, or 1K bytes per 128 labels. Unresolved labels are removed from the label table when resolved and the entire table moved up, providing efficient use of memory. Overflows will be rare, and loss of text can be minimized by starting a file with comments.

2.11.2 Changing the Default Device Type

EPROM address >FFB1 contains the default device type. This location is initially >01 (TMS7020) but can be changed to one of the types listed above by burning another Monitor EPROM with this location modified. The procedure to create a new EPROM is detailed in Section 9.6. If the value is changed to one outside the legal range of 1 to 5, it is treated as a value of 1.

2.12 Changing the Default EPROM Programmer Destination

EPROM address >FFB0 contains the default EPROM type used in place of the last parameter in the Monitor EPROM Programmer commands. This location is initially >04 (TMS2764) but can be changed to one of the values listed in Table 2-6 by burning another Monitor EPROM with this location modified. The procedure to create a new EPROM is detailed in Section 9.6. If the value is changed to one out of the legal range, it is treated as a value of >04.

Table 2-6. Default EPROM Programmer Destinations

DESTINATION	EPROM
>04	TMS2764
>08	TMS27128

2.13 Changing Port 2 Default Baud Rate

The location in EPROM at address >FFAF contains the default Port 2 baud rate. This location is initially >08 (9600 baud) but can be changed to one of the values listed in Table 2-7 by burning another Monitor EPROM with this location modified. The procedure to create a new EPROM is detailed is Section 9.6. If the value is changed to one outside the legal range, it is treated as a value of >08.

Table 2-7. Default Port 2 Baud Rate Values

VALUE	BAUD RATE
1	110
2	150
3	300
4	600
5	1200
6	2400
7	4800
8	9600

2.14 Default Changes for Bell, Tab, and Buffer Timeout

Defaults can be modified for these three functions by changing values in EPROM U43 as shown in the following table. Addresses are EPROM addresses. A comparable system memory address is EPROM address plus >E000. Thus, for example, the unmodified bell value (ASCII >07) can be inspected with a $MM FECA.

DESCRIPTION	U43 ADDRESS
Disable terminal bell	>1ECA
Expanded TAB character CNTL(I)	>1ECB
Default buffer timeout	>1ED5

2.15 Configuring Cursor Control

The EVM supports cursor operations for the Text Editor and the Monitor fixed display commands EF (Fixed Display) and FS (Single Step with Fixed Display). The tasks of recognizing cursor characters and sending the appropriate response to the terminal are handled separately.

Cursor transmission to the terminal can be of sequences from one to three characters in length and is limited to cursor up (Monitor fixed display) and cursor left (Text Editor). Cursor right is done by overprinting the display and cursor down is done with line feeds. Therefore, only the first two values (or sequences) must absolutely match the terminal used. The Monitor CU command allows modification of the cursor-up character(s) from the default value. The Monitor CL command allows modification of the cursor-left character(s) from the default value. Section 2.15.3 discusses the method of changing the default value.

Cursor character recognition is done by table lookup, and it recognizes the set of Lear/Siegler, Televideo, Adds, and Hazeltine single control characters and the VT-52 type two character <ESC> sequences. Section 2.15.4 discusses the lookup tables and how to add additional characters for recognition.

2.15.1 Display/Modify Cursor-Up Character(s) (CU)

FORMAT: CU *current cursor up* <cursor-up><CR>

PARAMETERS: Cursor up keystroke

Purpose: to set the cursor-up character or character sequence to be recognized by the terminal.

The current cursor up is displayed first (four bytes). If the terminal recognizes a cursor-up character or sequence other than the default value, then execution of this command is necessary before fixed display execution commands (EF and FS) can be executed. The cursor sequence can be up to three characters long. If more than three characters are loaded before reception of a <CR>, then the default value is reloaded. If a <CR> is the first character entered, the cursor value is not changed.

2.15.2 Display/Modify Cursor-Left Character(s) (CL)

FORMAT: CL *current cursor left* <cursor-left><CR>

PARAMETERS: Cursor-left keystroke

Purpose: to set the cursor-left character or character sequence to be recognized by the terminal.

The current cursor-left is displayed first (four bytes). If the terminal recognizes a cursor-left character or sequence other than the default value, then execution of this command is necessary before several Text Editor functions can be executed. The cursor sequence can be up to three characters long. If more than three characters are loaded before reception of a <CR>, then the default value is reloaded. If a <CR> is the first character entered, the cursor value is not changed.

2.15.3 Changing the Default Cursor-Up and Cursor-Left Values

The cursor-up and cursor-left default values are each stored in four successive bytes in EPROM. The cursor sequence can be one, two, or three characters long and must be followed immediately by at least one zero byte. Method for changing values in the Monitor EPROM is discussed in Section 9.6.

Table 2-8. Default Cursor-Up and Cursor-Left Locations

CURSOR-UP	CURSOR-LEFT
>FF98 = 0B CNTL-K	>FF9C = 08 CNTL-H
>FF99 = 00	>FF9D = 00
>FF9A = 00	>FF9E = 00
>FF9B = 00	>FF9F = 00

2.15.4 Adding Recognized Cursor Characters

Two lookup tables reside in EPROM with spare locations for recognition of:

- single control characters, and
- single characters following an <ESC> if received within two character times after the <ESC> (see Table 2-9 and Table 2-10).

These tables are used by the Text Editor. The format of both tables in memory is:

```
BYTE >XX        Cursor Character XX
BYTE >YY        Cursor Character YY


BYTE >ZZ        Last Spare Cursor Character

DATA >xxxx      Address of Handler for Character XX
DATA >yyyy      Address of Handler for Character YY
```

DATA >zzzz Address of Handler for Last Spare Character

In Table 2-9 and Table 2-10, BYTE statements containing the control character are conveniently placed on the same line with their corresponding DATA statement containing the function vector.

Entries are added to the table by placing the new cursor character in the first spare location in the appropriate table (non-<ESC> and <ESC>), and placing the address of the proper handler (from those already in the tables) in the corresponding address section of the table. Section 7 describes the procedure for creating a new Monitor EPROM. After creation of a new EPROM, the added cursor characters will be recognized automatically.

Table 2-9. Cursor Single Control Characters

ADDRESS IN CHARACTER TABLE	CHARACTER OR HEX VALUE	ASCII VALUE MSB ADDR	FUNCTION VECTOR	FUNCTION
>FF20	CNTL-F	>06	>FF38	Cursor-right
>FF21	CNTL-L	>0C	>FF3A	Cursor-right
>FF22	CNTL-P	>10	>FF3C	Cursor-right
>FF23	CNTL-H	>08	>FF3E	Cursor-left
>FF24	CNTL-N	>0E	>FF40	Insert char(s)
>FF25	CNTL-D	>04	>FF42	Delete char(s)
>FF26	<CR>	>0D	>FF44	Terminator
>FF27	<LF>	>0A	>FF46	Cursor down
>FF28	CNTL-V	>16	>FF48	Cursor down
>FF29	CNTL-K	>0B	>FF4A	Cursor up
>FF2A	CNTL-Z	>1A	>FF4C	Cursor up
>FF2B	CNTL-I	>09	>FF4E	Tab
>FF2C	CNTL-A	>01	>FF50	Home
>FF2D	CNTL-¬	>1E	>FF52	Home
>FF2E	CNTL-E	>05	>FF54	Undo line
>FF2F	<RUB>	>7F	>FF56	Delete char
>FF30	>FF	>FF	>FF58	Spare
>FF31	>FF	>FF	>FF5A	Spare
>FF32	>FF	>FF	>FF5C	Spare
>FF33	>FF	>FF	>FF5E	Spare
>FF34	>FF	>FF	>FF60	Spare
>FF35	>FF	>FF	>FF62	Spare
>FF36	>FF	>FF	>FF64	Spare
>FF37	>FF	>FF	>FF66	Spare

For example, pressing the <CNTL> (CONTROL) key and F key creates an ASCII >06 (from address >FF20) which causes a cursor-right move via a subroutine at the vector in EPROM address >FF38. EPROM addresses can be checked with the $MM command.

Table 2-10. Cursor <ESC> Sequence Characters

ADDRESS IN CHARACTER TABLE	CHARACTER OR HEX VALUE	ASCII VALUE MSB ADDR	FUNCTION VECTOR	FUNCTION
>FF68	A	>41	>FF78	Cursor-up
>FF69	B	>42	>FF7A	Cursor-down
>FF6A	C	>43	>FF7C	Cursor-right
>FF6B	D	>44	>FF7E	Cursor-left
>FF6C	H	>48	>FF80	
>FF6D	CNTL-R	>12	>FF82	
>FF6E	I	>49	>FF84	
>FF6F	CNTL-L	>0C	>FF86	Cursor-up
>FF70	>FF	>FF	>FF88	
>FF71	>FF	>FF	>FF8A	
>FF72	>FF	>FF	>FF8C	
>FF73	>FF	>FF	>FF8E	
>FF74	>FF	>FF	>FF90	
>FF75	>FF	>FF	>FF92	
>FF76	>FF	>FF	>FF94	
>FF77	>FF	>FF	>FF96	

NOTE: Cursor controls result from <ESC> pressed before the designated character is pressed.

Table 2-11. Recognized Cursor Character Summary

Cursor-up:	CNTL-K CNTL-Z <ESC>A <ESC>CNTL-L
Cursor-down:	CNTL-J CNTL-V <ESC>B
Cursor-right:	CNTL-F CNTL-L CNTL-P <ESC>C
Cursor-left:	CNTL-H <ESC>D
Home:	CNTL-A CNTL-¬ <ESC>H <ESC>CNTL-R
Tab:	CNTL-I
Back tab:	<ESC>I

Note:

Cursor reaction also depends upon terminal type and any intermediate software (e.g., cross-communication).

3. Development Tools Example

The walkthrough in this section:

- Initializes the system
- Enters program object into memory
- Runs the program which causes the DATA LED on the EVM to blink
- Uses the Text Editor to generate the same program
- Uses the Assembler to generate object

This is an abbreviated walkthrough which goes through board powerup and use of several commands such as memory change, set PC, and execute program. In addition, the same program is entered using the Text Editor and assembled with the Assembler.

A more extensive walkthrough is provided in Appendix C that demonstrates additional debugging commands, and shows in more detail the workings of the Text Editor, Assembler, and EPROM Programmer.

For this walkthrough, the following are assumed:

- Special cursor control characters do not have to be defined (such as with the CU or CL commands).
- All default values at the time of EVM board powerup are in effect (e.g., device type is 2; thus the DV command is not used).
- The board confuguration is as installed at the factory.
- Nomenclature used:

  <CR> = RETURN key

  <SP> = SPACE key

  Underlined characters are those input at the keyboard.

3.1 Configuring the System

As specified in Section 2, the following procedures must be completed to set up a minimum system configuration:

1) Connect EVM Port 1 to a terminal using a standard RS-232 cable.

2) Connect a power supply to port J6 or J7.

3.2 System Initialization

1) Apply power to terminal and EVM. The EVM's POWER LED (upper right) will light.

2) Toggle the RESET switch and enter <CR>. The command menu in Figure 3-1 will come up on the terminal followed by a "?" prompt.

```
TMS7000 DEBUG MONITOR REV 2.X
DEVICE TYPE = 2 (704X)
SYSTEM RAM = 32256 BYTES
HELP : HE /H $HE
MODIFY : MM/IM MR/IR MP/IP A/MA B/MB MS-P/PC,SP
DISPLAY : DM/DH DS IO DV
STATE : C/CP D/DP xC,xN,xZ,xI (x=C/S) EI DI SR
SAVE : SM ST DS
LOAD : LM LT LS
MOVE : MV
FIND : FB
FILL : FM NP FR
RESET : RT TO
BRKPT : B1 B2 C1 C2 CB DB BT CT DT
TRACE : TO TF IT PT IS TC TR
DEBUG : SS TS FS CS CY GO EX ET EF RU
EDITOR : XE L1 L2 LL LA LN
ASSM : XA XL XP AT XR
EPROM : PE VE CE RE BC 12 21 25 43 44 45
MATH : HC DC AR
PORTS : BR DR MO

?
```

Figure 3-1. Command Menu at Reset

3.3 Entering Object by Memory Change

Figure 3-2 is a listing of the LED blink program. The third column of the listing contains the object values (one byte, hexadecimal) for the memory addresses shown in the second column. Do the following to set up the object values in memory:

1) Enter:

   ?MM F006<CR>

   to display and change values beginning at memory address >F006.

2) The display should show the memory address and both the hex value and binary value at that address. For example:

```
?MM F006<CR>
  F006=99 (10011001) ¬
```

3) Enter the value "A2". This is the hex value of the first line of the LED blink program as shown in Figure 3-2.

```
?MM F006<CR>
  F006=99 (10011001) A2<SP>
```

4) By using the SPACE bar, you can display successive memory addresses for updating. Thus, the entire program can be loaded between the addresses >F006 and >F015 as shown in Figure 3-3.

5) Finish the last memory change with a <CR> as shown in Figure 3-3.

6) Enter the RU command to execute the program. The EVM's DATA LED should blink.

7) If the DATA LED does not blink, check program values using the MM command (you may have to call the Monitor first by toggling the RESET switch). If necessary, verify that the program counter is set to program start (F006) using the PC command. Then reissue the RU command.

```
0010 F006           AORG   >F006
0020 F006 A2 LOOP   MOVP   %>01,P254     LEAVE RAM BIT SET
     F007 01
     F008 FE
0030 F009 DB D1     DECD   R20           TIME OUT
     F00A 14
0040 F00B E3        JC     D1
     F00C FC
0050 F00D A2        MOVP   %>03,P254     SET LED, RAM BIT
     F00E 03
     F00F FE
0060 F010 DB D2     DECD   R20           TIME OUT
     F011 14
0070 F012 E3        JC     D2
     F013 FC
0080 F014 E0        JUMP   LOOP          REPEAT SINGLE BLINK
     F015 F0
```

Figure 3-2. Blink Program Assembler Listing

```
?MM F006
 F006=99 (10011001) A2<SP>
 F007=99 (10011001) 01<SP>
 F008=99 (10011001) FE<SP>
 F009=99 (10011001) DB<SP>
 F00A=99 (10011001) 14<SP>
 F00B=99 (10011001) E3<SP>
 F00C=99 (10011001) FC<SP>
 F00D=99 (10011001) A2<SP>
 F00E=99 (10011001) 03<SP>
 F00F=99 (10011001) FE<SP>
 F010=99 (10011001) DB<SP>
 F011=99 (10011001) 14<SP>
 F012=99 (10011001) E3<SP>
 F013=99 (10011001) FC<SP>
 F014=99 (10011001) E0<SP>
 F015=99 (10011001) F0<CR>
?RU
```

Figure 3-3. Entering Blink Program Into Memory

3.4 Source Entry Using the Text Editor

This example uses the Text Editor to enter text and the Assembler to get object code.

1) Call the Text Editor with the XE command:

```
?XE <CR>
```

2) Set automatic line number increment (default increment is 10) using the editor command "A":

```
?XE <CR>
EVM TEXT EDITOR
"H" HELP
EDITOR RAM = 22016 BYTES
*A
0010
```

Enter the first source line (shown in Figure 3-3). End with a <CR>, and line number 0020 is printed automatically:

```
?XE <CR>
EVM TEXT EDITOR
"H" HELP
EDITOR RAM = 22016 BYTES
*A
0010       AORG  >F006<CR>
0020 LOOP  MOVP  %>01,P254       LEAVE RAM BIT SET <CR>
0030
```

The label field starts next to the line number. The mnemonic field starts (at least) one space to the right of the line number. Operand and comment fields continue to the right, separated by at least one space. Continue to input the eight source lines (those numbered 0010 to 0080 in Figure 3-2). When complete, press <CR> twice and then "Q" (quit) to exit the editor and call the Monitor.

3.5 Assembling a Source File

Call the Assembler with the XA command (without parameters - the most recently edited file will be the source file).

```
.
.
.
0070        JC     D2<CR>
0080        JMP    LOOP<CR>
0090 <CR>
*O <CR>
TMS7000 DEBUG MONITOR REV 2.X
?XA <CR>
INITIALIZE? (Y) <CR>
TMS7000 ASSEMBLER
>
0010 F006           AORG   >F006
0020 F006 A2 LOOP MOVP   %>01,P254   LEAVE RAM BIT SET
.
.
.
```

Because of the AORG directive, object is loaded by the assembler beginning at address >F006 (if you had preset values at these addresses, you could check the entry of the new object code with the "MM >F006" command and compare the new code to the third column of Figure 3-2).

If you re-execute the editor (XE command), the same source program will be in memory for editing. Other features of the Text Editor are described in Section 4 and the walkthrough in Appendix C.

4. Text Editor

The EVM Text Editor is line-number oriented and is entered from the Monitor with the XE command. The Text Editor may be used to build assembly language source files as well as general text files. These source files can be uploaded to a host computer for storage or saved on cassette tape. To identify beginning-of-file and end-of-file for the Assembler, the Text Editor brackets the file with "*>" and "*<" signs respectively (each is preceded by an asterisk -- like a comment). The format for calling the Text Editor and its banner message is:

FORMAT: XE {port **0**,1,2,3}

PARAMETERS: Initial source input port
(default = 0 = keyboard/editor RAM)

For example:

```
?XE<CR>
EVM TEXT EDITOR
"H" HELP
EDITOR RAM = 22016 BYTES
*                                    (editor command prompt)
```

The number in the banner is the number of bytes available for text storage and depends on the device type being emulated by the EVM. As text is entered, this number will decrease. The editor command-input prompt is "*". As in the Monitor, <ESC> is used to abort the current activity and return control to the Text Editor top level.

Besides dedicated keys, the cursor can be controlled during text entry with the following keys and key combinations (the Edit command has further controls covered in Section 4.1.5):

Cursor left	CNTL-H or BACKSPACE
Cursor right	CNTL-F
Tab right	TAB
Insert character	CNTL-N
Delete character	CNTL-D
To left margin	CNTL-A

This sample Text Editor session is referred to in examples in this section.

```
*0010 LABEL     EQU R25   PROGRAM TO OUTPUT
*0020 BPORT     EQU P6    A SQUARE WAVE ON
*0030 START     EQU $     ALL BITS OF PORT B
*0040           CLR       LABEL
*0050           CALL      @OUT
*0060           INV       LABEL
*A
 0070           CALL      @OUT
 0080           JMP       START
 0090 OUT       MOV       LABEL,A
 0100           MOVP      A,BPORT
 0110           RETS
 0120           END
 0130
*
```

4.1 Text Editor Commands

The Text Editor commands are listed below:

COMMAND	DESCRIPTION	SECTION
A	Autoincrement Line Mode	Section 4.1.1
C	Change Line Number	Section 4.1.2
<CR>	Delete Line	Section 4.1.3
D	Duplicate Line	Section 4.1.4
E	Edit Line	Section 4.1.5
F	Find Character String	Section 4.1.6
H	Help Menu	Section 4.1.7
I	Input File to Editor	Section 4.1.8
L	List (Print) Lines to Terminal	Section 4.1.9
M	Display Free RAM	Section 4.1.10
Q	Quit Editor, Save File	Section 4.1.11
R	Resequence Line Numbers	Section 4.1.12
T	Display/Modify Tab Value	Section 4.1.13
Z	Initialize Editor	Section 4.1.14
+	Line Number Pointer to EOF	Section 4.1.15
-	Line Number Pointer to BOF	Section 4.1.16
=	Display Current Line Number Pointer	Section 4.1.17

Note:

The following applies to entering line numbers:

If more than four digits are entered, the last four digits will be used, and leading zeros are assumed if less than four digits are entered. If no line number is entered, the current line number is assumed.

4.1.1 Autoincrement Line Number Mode (A)

FORMAT: [start line number]A

PARAMETER: Line number (optional; default=last line number plus increment)

Purpose: After a source line is entered, automatically print the next line number to the terminal, incremented by a set value, ready for input of the next source line.

The Autoincrement command initially uses the default line-number increment (10) set at the Text Editor initialization. A Resequence Line Numbers command (R) can change the increment. When a line number precedes the "A", the Autoincrement mode starts with the given line number. If no line number is entered, the Text Editor positions the Autoincrement pointer at the end of the text file and starts with the last used line number plus the current increment.

To leave Autoincrement mode, enter a carriage return immediately after the line number. If a line of text with that line number already exists, the text will not be deleted. Entering and immediately exiting Autoincrement mode will cause the pointer to point to the last line in the file.

EXAMPLES:

1) *A

```
0010
```

Autoincrement mode is entered after memory has been initialized (first line number is 10).

2) *A
```
0860
```

Autoincrement mode is entered after a file has been downloaded. The pointer is automatically positioned to the line following the last line of text (line number 860).

3) *500A
```
0500
```

Autoincrement mode is entered starting at line 500.

4.1.2 Change Line Number (C)

FORMAT: <line number to change>C <new line number><CR,SP>

PARAMETERS: 1) Line number to be changed
2) New number of line

Purpose: to change the line number of a line of text in memory.

This command will execute properly only if two conditions are satisfied: 1) the line number to be changed must already exist in the text file and 2) the new line number must not already exist in the text file.

EXAMPLES:

1) *50C 70<CR>
```
LINE NUMBER ERROR
*
```

A line number cannot be changed to one that already exists.

2) *55C
```
LINE NUMBER ERROR
*
```

A non-existent line number was specified.

3) *60C 45<CR>
*40C 60<CR>
*R<CR>
*

The "CLR LABEL" and "INV LABEL" statements in the sample program have been swapped by changing the line numbers, and resequenced back to increments of 10 by the "R" command. The sample session is shown on 4-1.

4.1.3 Delete Line (<CR>)

FORMAT: <number of line to delete><CR>

PARAMETERS: 1) Number of the line to be deleted
2) "Y" (yes) to enable deletion

Purpose: to delete a line of text in memory.

If the line does not exist, LINE NUMBER ERROR is issued. If the line exists, it is deleted from the text file.

EXAMPLE:

```
*70<CR>
ARE YOU SURE? (N) Y
*
```

Line 70 is deleted.

4.1.4 Duplicate Line (D)

FORMAT: <line to duplicate>D <new line no.><CR,SP>

PARAMETERS: 1) Line number to be duplicated
2) New line number

Purpose: to duplicate a line of text in memory.

The first line number specified must already exist, and the target line number must not exist.

EXAMPLES:

1)
```
*65D
LINE NUMBER ERROR
*
```

The first line must exist in order to be duplicated.

2)
```
*60D 80<CR>
LINE NUMBER ERROR
*
```

The second line number cannot already exist.

3)
```
*60D 150<CR>
*60<CR>
ARE YOU SURE? (N) Y
*150C 60<CR>
*
```

Line 60 was duplicated at line 150. The original line was then deleted and line 150 was changed back to 60.

4.1.5 Edit Line (E)

FORMAT: <line number>E

PARAMETERS: Line number (default=current line)

Purpose: to insert, delete, replace and add characters to an existing line.

The Edit command dumps the current line to the terminal with the cursor (¬) positioned at the end of the line. If only the "E" is entered, the current line is displayed. If a line number is specified, that line becomes the current line displayed for edit.

EXAMPLE:

```
*10E
0010 LABEL   EQU R25¬
```

In the Edit mode, the Text Editor will accept characters from the keyboard and store them at the cursor position, moving the cursor one position to the right for each character entered. The character input routine does not accept control characters as text and will send an audible beep to the terminal when one is entered that is not a legal sub-command.

The following sub-commands are available within the Edit mode:

KEYBOARD INPUT	RESULT
CNTL-F/-L/-P	Cursor-right
CNTL-H, BACKSPACE	Cursor-left
CNTL-A	Home
Tab	Tab right
Back Tab	Tab left
CNTL-E	Undo line
CNTL-N	Insert character(s)
CNTL-D	Delete character(s)
<RUB> or <DEL>	Delete previous character
<CR>	Save line
Cursor-down/CNTL-J	Save line/edit next line
Cursor-up/CNTL-K/-Z	Save line/edit previous line

The Insert (CNTL-N) and Delete (CNTL-D) commands function in one of two modes depending on the baud rate of the terminal. Manual mode is for terminal baud rates of 1200 baud and below; it expects a count parameter to be entered after an insert or delete command and displays the results on a new line. Interactive mode is for terminal baud rates of 2400 baud and above and automatically inserts or deletes one character for each entry of the command, redisplaying the new line on top of the old line. Manual mode is used for printing terminals, and interactive mode is used for video terminals.

Characters recognized for cursor control are stored in tables in EPROM. Cursor control can be either single control characters or escape sequences. See Section 2.15.3 for a list of default cursor characters and the procedure for adding new characters.

Cursor-Right. The Text Editor will not allow the cursor to be positioned beyond the right end of the text with this command. The cursor-right character is not transmitted to the terminal, but is simulated with overprinting of the current character. Example cursor-right keys are CNTL-F, CNTL-L, and CNTL-P.

Cursor-Left. The Text Editor will not allow the cursor to be positioned beyond the left end of the text with this command. An audible beep is issued to the terminal when the cursor encounters the left end of the line. The cursor-left character must be transmitted to the terminal. If the terminal recognizes a cursor character other than the default value, the Monitor command CL (Modify Cursor-Left Character) must be executed. Section 2.15.3 discusses changing the default cursor-left value. An example of a cursor-left character is CNTL-H.

Home. This moves the cursor to the first (leftmost) character on the line. The home character is not transmitted to the terminal, but is simulated with a computed number of cursor left character. Examples of HOME characters are CNTL-A and CNTL-¬ (the "¬" character is found as the "upper-case" 6 on some keyboards).

Tab (CNTL-I). This sends the cursor to the next tab location. Only four right tabs are allowed, all of a length equal to the tab value. Attempting to tab further will move the cursor to the first character on the line. A tab is not a specific character location on a line; instead it is a single value set by the "T" command which designates how many characters to space for a tab input.

Back Tab (<ESC>-I). This sends the cursor back to the previous tab position. If in the process of performing a back tab the Home character position is encountered, an audible beep sent to the terminal.

Undo Line (CNTL-E). This causes all changes to the current line to be ignored and the same line reloaded from memory onto the screen. The reloaded (without changes) line is displayed on the next line.

Insert Up to 9 Characters (CNTL-N).

FORMAT: CNTL-N {<ESC>, 1-9} (manual mode)
CNTL-N (interactive mode)

PARAMETERS (manual mode): Number of characters to insert. This is used to insert up to nine characters in manual mode or one character in interactive mode starting at the cursor position.

In interactive mode (for terminal baud rates of 2400 baud and above), one space is automatically inserted at the cursor position and the line is redisplayed with the cursor in the original position.

In manual mode (rates of 1200 baud and below), the CNTL-N command does not echo. After it is entered, the legal entries are <ESC> or a digit from 0 to 9. Any other entry will cause an audible beep and wait for legal input. Entering 0 aborts the command and returns control to the cursor level, with no visible action occurring.

If a digit from 1 to 9 is entered, the line is dumped with that number of spaces inserted at the original cursor position, and the cursor will be positioned at the first blank character. Text can be entered in the blank spaces. If the blanks created by the Insert command are filled, continued typing will replace the characters already existing. The Text Editor will not accept and store as text any control character.

If a digit is entered that would cause the line to exceed 64 characters, an audible beep is issued and another digit input is expected. If <ESC> is entered, the entire edit is aborted.

EXAMPLE:

```
*10E                          (display line for edit)
0010 LEL    EQU R25           (put cursor after first "L")
    ¬CNTL-N 2                 (insert 2 spaces)
0010 L  EL     EQU R25
   AB<CR>                     (fill spaces, exit)
*L<CR>                        (check changes)
0010 LABEL     EQU R25
```

"LEL" has been changed to "LABEL."

Delete Up to 9 Characters (CNTL-D).

FORMAT: CNTL-D<# of characters to delete> (manual mode)
CNTL-D (interactive mode)

This deletes up to nine characters in manual mode or one character in interactive mode starting at the cursor position.

In interactive mode (terminal baud rates 2400 baud and above), one character is automatically deleted at the cursor position and the line is redisplayed with the cursor in the original position. <ESC> aborts the entire edit (including changes).

In manual mode (terminal baud rates of 1200 baud or lower), the CNTL-D command does not echo. After it is entered, the legal entries are <ESC> or a digit from 0 to 9. Any other entry will cause an audible beep and a wait for legal input. Entering 0 aborts the command and returns control to the cursor level, with no visible action occurring. If a digit from 1 to 9 is entered, the new line is redisplayed on the next line with the cursor in the original position.

If the cursor is at the start of the line and a digit is entered that would delete all characters in the line, an audible beep is issued and another digit input is expected. <ESC> aborts the entire edit (including changes).

EXAMPLE:

```
*10E                          (display line for edit)
0010 LABELXYZ   EQU R25
          ¬CNTL-D 3           (cursor at "X"; delete 3 chars)
0010 LABEL    EQU R25         (3 characters deleted; close up)
          ¬<CR>               (exit)
*
```

The "XYZ" in "LABELXYZ" is deleted.

Delete Previous Character (<RUB> or <DEL>). This deletes the character before the cursor. Entry of either the RUBout or DELete key will replace the character before the cursor with a space. This will not shorten the line because no characters are removed.

Save Edited Line (<CR>). This exits the Edit mode and saves the edited line. If the newly edited line is the same length or shorter than the old version of the line, the Text Editor stores it in the same place in the text buffer, so that the byte count of a long file can be shortened before collecting the text together with the Quit command. A <CR> can be entered with the cursor at any position in the line.

Save Line/Edit the Next Line (Cursor-Down). This saves the line being currently edited and displays the next line for edit. This command is a quick way of scrolling through a file. If the last line is encountered, scrolling will stop. When this command is used while editing a line from the Find command, the next line is displayed until a <CR> is entered, returning control to the Find command starting at the last line displayed. The relative position of the cursor stays the same as successive lines are displayed. Example cursor-down keys are CNTL-J and CNTL-V.

Save Line/Edit the Previous Line (Cursor-Up). This saves the line being currently edited and displays the previous line for edit. This command is the opposite of the cursor-down command described above. If the first line in the file is encountered, scrolling will stop. Example cursor-up keys are CNTL-K and CNTL-Z.

4.1.6 Find Character String (F)

FORMAT: [start line number]F <string>

PARAMETERS: 1) Line number to begin search (optional)
2) String (up to 8 characters)

Purpose: to quickly locate for edit a string of up to eight characters in the text file.

The F command matches the specified string with its occurrence in the text file and prints the line to the terminal for edit. A <CR> continues the search for further occurrences of the string. The characters in the string can only be within the range of >20 to >7E of ASCII values (no control characters).

If the F command is not preceded with a line number, the entire text file is searched. If a line number is entered, searching begins at that line or the first line that exists after that line number.

If the string is found, the command enters the Edit mode with the cursor over the start of the string in the line. If the string occurs more than once in the line, entering a <CR> to save the line will cause the line to be redisplayed, with the cursor over the next occurrence. Once in the edit mode, the entire contents of the line can be altered. Lines before or after the one displayed can also be changed using the cursor-up and cursor-down commands (see Section 4.1.5). When finished with the edit, entering <CR> will return to the Find mode at the last line displayed and search for the next occurrence of the string.

If the Edit mode is terminated by <ESC>, control returns to the Text Editor and the "*" prompt is printed. Note that any changes made to a line from which <ESC> is entered will not be saved. If searching continues to the end of the file either while searching with a <CR> or failure to find the string at all, the message "END OF TEXT" is printed.

EXAMPLES:

1)
```
*F BPORT
0020 BPORT   EQU  P6<CR>
0100         MOVP A,BPORT<CR>
END OF TEXT
*
```

Two occurrences of "BPORT" were found.

2) *35F LABEL
0040 CLR LABEL<ESC>
*

The first occurrence of "LABEL" after line 35 was found.

3) *F APORT
END OF TEXT
*

END-OF-TEXT was encountered before finding the string "APORT."

4.1.7 Help (H)

FORMAT: H {port **1**,2}

PARAMETERS: Output port (default=1)

Legal values for the output port are 1 (terminal) and 2 (printer). This command displays a list of the Text Editor commands to the specified port.

4.1.8 Input File to the Text Editor (I)

FORMAT: I {port 1,2,**3**}
LINE NUMBERS? (N)

PARAMETERS: Input port to load file from (default=3)

Purpose: to load the Text Editor with a file.

The current contents of the Text Editor are discarded only after input data is detected. Values for the input port are 1 (terminal emulator), 2 (host download), or 3 (audio tape).

When downloading a file from a host system, the file must contain a beginning-of-file character (^) and an end-of-file character (^<). These are used by the EVM Text Editor and Assembler to mark the beginning and end of the incoming ASCII string during download.

Also, the Text Editor inserts a space between the line number and the text when dumping a file, and strips off the first character after the last line number (expecting it to be a space) during download.

4.1.9 List Line(s) to Terminal (L)

FORMAT: <start line number>L <no. of lines><CR,SP>

PARAMETERS: 1) First line to be listed (default=current line)
2) Number of lines to list (default=1)

Purpose: to list lines of text in order of ascending line numbers.

The number of lines that can be listed is in the range of 1 to 9999. Entering 0 is the same as entering 1. If no line number is entered, the first line listed is the current line. The current line is pointed to by the line number most recently displayed. If a line

number is entered, listing starts with that line. In either case, listing continues until either the given number of lines are dumped or the end of the text file is encountered.

While the command is dumping to the terminal, the display can be stopped and started by keyboard input. Any key will stop the display at the end of the current line of output. After the display is stopped, <ESC> will abort the command, <SP> will cause display of successive lines one at a time for each <SP> entry, and any other key will restart the display.

EXAMPLES:

1)
```
*1L 2<CR>
0010 LABEL    EQU R25
0020 BPORT    EQU P6
*
```

Two lines are requested beginning with line 1. Since line number 1 does not exist, the list begins with the first line found after line number 1.

2)
```
*10L 2<CR>
0010 LABEL    EQU R25
0020 BPORT    EQU P6
*
```

Two lines are listed beginning with line 10.

3)
```
*L<CR>
0020 BPORT    EQU P6
*
```

The current line is displayed.

4)
```
*1L 9999<CR>
0010 LABEL    EQU R25        PROGRAM TO OUTPUT
0020 BPORT    EQU P6         A SQUARE WAVE ON
0030 START    EQU $          ALL BITS OF PORT B
0040          CLR LABEL
0050          CALL @OUT
0060          INV LABEL
0070          CALL @OUT
0080          JMP START
0090 OUT      MOV LABEL,A
0100          MOVP A,BPORT
0110          RETS
0120          END
*
```

An attempt to list 9999 lines starting with line number 1 was made. The end of the text file was encountered first.

4.1.10 Display Free RAM Remaining (M)

FORMAT: M

PARAMETERS: None

Purpose: to display the number of bytes of RAM available for text storage.

The maximum RAM space available for text storage occurs just after execution of the Z command. This space is equal to the total amount of RAM detected by the memory sizing routine at power-up reset (less the RAM set aside by the Assembler for object

code and label table storage for device types 1 and 2). When the Text Editor is entered with a download of text from either Port 1 or 2, the remaining free RAM value is printed out on the line above the "*" prompt.

EXAMPLES:

1) *M
```
EDITOR RAM = 21637 BYTES
*
```

The Text Editor has 21637 bytes of available RAM.

2) *M EDITOR RAM = 21592 BYTES
*

A line of text 38 characters long has been entered. Each line stored requires 2 bytes for Text Editor use, 2 bytes for the line number, the text, the <CR> ending the line, and 2 bytes for the Assembler address tag. In all, 45 bytes were used.

4.1.11 Quit Edit and Save File (Q)

FORMAT: Q {port **0**,1,2,3}

PARAMETERS: Output port (default=0)

Purpose: to exit the Text Editor and enter the Monitor. If an output port is specified, the text file is dumped to that port prior to entering the Monitor.

Because the Text Editor does not destroy its internal pointers, it can be exited and entered at will as long as nothing occurs to alter the contents of Text Editor RAM. For example, the text file can be saved to audio tape (output port = 3); then the Text Editor can be entered specifying Port 2, so that when it is exited, the program listing will be dumped to the printer.

If either Port 1 or 2 is specified, a "LINE NUMBERS? (N)" prompt asks if the text lines are to be dumped with or without line numbers. An "N" (or <CR> for default) response dumps without numbers. Any other response causes line numbers to be dumped with the file.

If no port is specified, Port 0 is assumed. If 0 is entered for the port number, no dump takes place and the control returns directly to the Monitor. When the Q command is entered, the Text Editor will immediately return to the Monitor if RAM is empty.

Quitting the Text Editor with output to the terminal allows the display to be stopped and started by keyboard input. Any key will stop the display at the end of the current line of output. After the display is stopped, <ESC> will abort the command with control remaining in the Text Editor, <SP> will cause display of successive lines one at a time for each <SP> entry, and any other key will restart the display.

EXAMPLES:

1) *Q <CR>
```
TMS7000 DEBUG MONITOR  REV 2.X
?
```

The Text Editor is exited without dumping the text, and control returns to the Monitor. The text remains intact in RAM. You can continue editing the text file with the XE command.

2) *Q 3
FILENAME: A<CR>
READY TO RECORD? <CR>

```
TMS7000 DEBUG MONITOR  REV 2.X
?
```

The Text Editor is exited and the text is dumped to Port 3 (cassette tape) to file "A". The text also remains intact in RAM.

3) *Q 1 LINE NUMBERS? (N) Y
EVM TEXT EDITOR

```
*>
0010 LABEL   EQU R25        PROGRAM TO OUTPUT
0020 BPORT   EQU P6         A SQUARE WAVE ON
0030 START   EQU $          ALL BITS OF PORT B
0040         CLR LABEL
0050         CALL @OUT
0060         INV LABEL
0070         CALL @OUT
0080         JMP START
0090 OUT     MOV LABEL,A
0100         MOVP A,BPORT
0110         RETS
0120         END
*<

TMS7000 DEBUG MONITOR  REV 2.X
?
```

The Text Editor is exited and the text is dumped to Port 1 (terminal) with line numbers. The text remains intact in RAM.

4.1.12 Resequence Line Numbers (R)

FORMAT: [No. of 1st line after resequence]R <increment>

PARAMETERS: 1) No. of first line (optional)
2) Increment between lines (0-9; default=10)

Purpose: to resequence line numbers of text in memory.

Legal increments are from 1 to 9. If a line number is entered, it will be the first number of the resequenced lines (e.g., 400R 3 results in line numbers of 0400, 0403, 0406, etc). If no line number is entered, the first resequenced line will be the same as the increment (e.g., R 4 causes line numbers of 0004, 0008, etc.). If no increment or a zero (0) is entered, 10 is assumed. If only an "R" is entered (no start line or increment given), lines will be incremented by 10 starting at line 10.

If execution of the command causes a line number to exceed the maximum of 9999, a LINE NUMBER ERROR is issued and the command is automatically re-executed with an increment equal to 1 starting with line number 0001.

EXAMPLES:

1) *R<CR>
*

All lines of text in memory are resequenced by the default value of 10, and the first line number is 0010.

2) *500R<CR>
*

All line numbers are resequenced with the first line starting at 500 and a default increment of 10.

3) *R 5
*

All lines are resequenced by a default increment of 5, starting at line 0005.

4) *9000R<CR>
LINE NUMBER ERROR
*

The line number register has exceeded 9999. When this occurs, file is resequenced by 1 starting at 1.

4.1.13 Display/Modify Tab (T)

FORMAT: T *current tab value* <tab value><CR,SP>

PARAMETERS: Tab value (4 to 12; default=8)

Purpose: to display and/or modify the tab value.

Only one tab value is defined. "Tab" is not a column location on the screen; it is the distance in characters the cursor moves when the TAB key is pressed. The value is displayed immediately after entry of the command. The tab defaults at reset to 8. Legal range of values is from 4 to 12. Entering <CR> or <SP> in place of data will return to the Text Editor prompt without changing the tab. The last two digits entered before the <CR> or <SP> are saved as the tab value.

4.1.14 Initialize Text Editor (Z)

FORMAT: Z

PARAMETERS : "Y" (yes) to enable initialization

Purpose: to clear all text from memory and to initialize the Text Editor workspace pointers.

> **Caution:**
>
> **This command destroys all contents of the text buffer.**

This command executes automatically prior to loading text from Ports 2 or 3 when the Text Editor is executed. This command is also automatically executed when entering the Text Editor from Port 1 for the first time after power-up (but not reset). This command may be executed at any time to cause the Text Editor to "forget" the contents of the text file in memory.

EXAMPLE:

*Z

```
ARE YOU SURE? Y
27136
*
```

The Text Editor responds by printing out a warning message, giving you a chance to not initialize. The only response that will echo and initialize is "Y". After initialization the number of bytes of useable RAM is displayed.

4.1.15 Line Number Pointer to EOF (+)

FORMAT: +

PARAMETERS: None

Purpose: to set the value of the current line number to the last line in the file and display the line number.

By pointing to the last line in the file, the Edit command can be entered from the end of the file without knowing the last line number in the file.

4.1.16 Line Number Pointer to BOF (-)

FORMAT: (minus sign)

PARAMETERS: None

Purpose: to set the value of the current line number to the first line in the file and display the line number.

By pointing to the first line in the file, the Edit command can be entered from the beginning of the file without knowing the first line number in the file.

4.1.17 Display Current Line Number (=)

FORMAT: =

PARAMETERS: None

Purpose: to display the current line number.

4.2 Text Entry

The format for text entry is:

XXXX<SP><text><CR>

where:

XXXX All lines begin with a 4-digit line number from 0001 to 9999. Line number 0000 is illegal. The line number can be entered manually, followed by a space, or is provided automatically in Autoincrement mode (A). Entering more than four digits results in the last four entered used as the line number. If less than four digits are entered, leading zeros are assumed.

<SP> Entering a space after the line number tells the Text Editor that text will follow, as opposed to a line number preceding a command. This space is provided automatically in the Autoincrement mode. It is not stored with the text.

<CR> A carriage return signifies the end of the line of text and tells the Text Editor to store the line from the I/O buffer to RAM in order of increasing line number.

The Text Editor fills RAM from the highest address down. When text storage uses RAM to within 64 characters of the bottom of RAM, the RAM FULL error is issued after any operation that involves storing text to RAM. The M command can be used to make best use of the remaining space. The Text Editor will quit storing lines when they are too long to fit in the remaining RAM, but lines can still be entered as usual, and the RAM FULL error will continue to be issued. Figure 4-1 shows the Text Editor memory map.

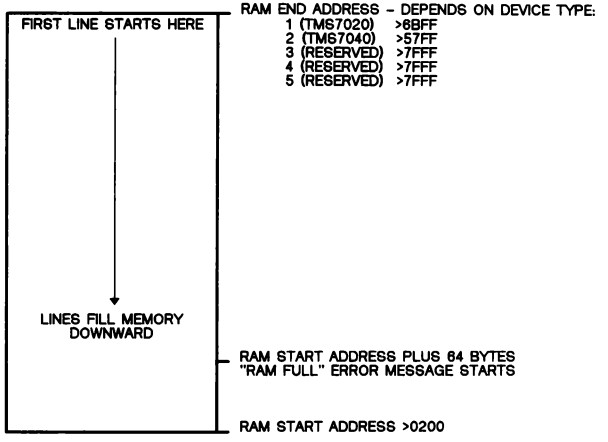


Figure 4-1. Text Editor Memory Map

Three limitations are placed on the content of the text.

1) The Text Editor will not allow control characters to be entered as text. If a control character that is not a command is entered, an audible beep is issued to the terminal.
2) If the text of a line consists only of spaces, the line will not be stored.
3) The maximum number of characters allowed in a line is 64. If 64 characters are entered, further entry is inhibited to allow editing of the line. A <CR> will save the line.

Cursor control and character insert and delete capability are available during text entry. The functions are identical to those in the Edit Line command described in Section 4.1.5. The Undo Line command (CNTL-E) deletes all characters on the line, allowing the line to be started over. If done in Autoincrement mode, the same line

number is dumped. This command differs from <ESC> in that pressing <ESC> will abort the Autoincrement mode also.

4.3 Monitor and Text Editor Debug Aids

In the Standalone mode of EVM operation (Section 1.4.2), the most rapid way to assemble a file from the Text Editor is with suppressed listing ("XA 0 0"). A disadvantage to this approach is that breakpoint addresses are unknown without an assembled source listing. To minimize this disadvantage, when a file is assembled from the Text Editor with any listing port, the location in program memory at which the first byte (for a multiple-byte instruction) is stored is appended to the line in the Text Editor. This "tag" address is used by a group of Monitor commands that allows the file in the Text Editor to be used as the assembled source listing, linking line numbers to memory addresses for setting breakpoints and tracking program flow. With the ability to set a breakpoint only by entering a line number, the need for an assembled listing is minimized. The appropriate commands are shown in Table 4-1.

Table 4-1. Monitor and Text Editor Debug Aids

MONITOR COMMAND	DESCRIPTION
L1	Set Breakpoint 1 with a Line Number
L2	Set Breakpoint 2 with a Line Number
LA	List Tag Address of a Line Number
LN	Show a Line with a Given Tag Address
LL	List Line(s) to the Terminal

During fixed-display debug operations (EF and FS), the line in the Text Editor with a tag address equal to the Program Counter is displayed above the fixed display. For execution to breakpoint (EF), the line at which the breakpoint has been set is displayed. This line will be the first line executed after the breakpoint is displayed. For single-step execution (FS), the line to be executed next is displayed. Therefore the data output in the fixed display leads the Text Editor line by a single step, allowing the user to anticipate the results of the next step.

When a file is loaded into the Text Editor, all address tags are 0000. After assembly, only comment lines will retain the address tag of 0000. Subsequent execution of the Text Editor will clear the address tag of a line only if the line is altered. Lines added from the terminal will also be tagged as 0000. Further assembly will correct all address tags.

4.4 Text Editor Errors

ERROR — This is the general error message issued by the Text Editor when an illegal character is entered as a parameter in a command string.

INPUT ERROR — This error is issued after a character has been input that is not within the legal range of 0-9 and A-F when hexadecimal input is expected.

RAM FULL — This error is issued whenever the amount of available RAM for text storage (as displayed with the M

command) drops below 64 bytes. The error will continue to be issued until RAM is exhausted, although lines can still by edited or added to the text file. If a file is loaded into the Text Editor, this error is issued only when the RAM is full, at which point the load is aborted and control returned to the Text Editor command handler.

LINE NUMBER ERROR This error is issued whenever an operation involving line numbers causes the line number holding register to underflow below 0001 or overflow above 9999. This error is also issued for line number violations involving already-used line numbers and in the following instances:

1) During download with the Text Editor creating line numbers, this error will terminate the load operation.

2) Deleting, changing, and duplicating lines in violation of the line number rules for each command will cause this error and terminate of the command.

3) In Autoincrement mode, this error will terminate the mode.

4) During execution of a resequence line numbers command, issuance of this error will automatically resequence the file by 1 starting with line number 0001.

5. Assembling and Executing Programs

The EVM has two types of assemblers:

1) EVM Assembler (XA):

FORMAT: XA [**0**,1,2,3] [**1**,2,3]

PARAMETERS: 1) Port no. of source input (default = 0 = text editor RAM)
2) Port no. of listing output (default = 1)

This assembler can assemble an edited source file either from the EVM Text Editor, from audio cassette, or downloaded from a host. It can provide a listing file. Prompts ask to initialize the system and if line numbers are included in the source file (responses are covered in Section 5.1). Absolute (untagged) object is placed in memory starting at the AORG directive value or default value for the processor specified in Table 2-5. Assembly starts following the first greater-than sign (>) and is complete upon reaching an END directive or an end-of-file mark (^<). Then control goes back to the Monitor.

2) Two Line-by-Line Assemblers (XL and XP)

FORMAT: XL

PARAMETERS: None

This line-by-line assembler (LBLA) creates a new label table for the program under assembly, and immediately assembles each line as input from the keyboard. Absolute (untagged) object is placed at memory starting at the AORG directive value or default value for the processor specified in Table 2-5 END directive in code completes assembly, gives control back to the Monitor.

FORMAT: XP

PARAMETERS: None

The XP command assembles programs without destroying the present label table in memory. This allows assembling code to be used with a program in memory while being debugged. The XP command use is described in Section 5.4.

Note: The reverse assembler (XR) is described in Section 6.6.56 on page 6-46.

5.1 EVM Assembler

In the below example, the EVM Assembler is executed with the Monitor XA command, with source coming from Port 2 and listing going to Port 1:

```
?XA 2 1<CR>
LINE NUMBERS? (N) Y<CR>
INITIALIZE? (Y)<CR>
TMS7000 ASSEMBLER
>
0010   F006                 AORG    >F006
0020   F006 A2    LOOP      MOVP    %>01,P254
       F007 01
       F008 FE
        .
        .
        .
0080   F014 E0              JMP     LOOP
       F015 F0
0090   F016                 END

0 ERRORS

TMS7000 DEBUG MONITOR  REV 2.X
?
```

If the source is to come from Port 1 or 2, a "LINE NUMBERS? (N)" prompt asks if incoming source has a line number and space at the beginning of each source line. If a "N" (or <CR> for default), the assembler will automatically generate these for use with error messages. Any other response indicates line numbers present.

The "INITIALIZE? (Y)" prompt makes a new label table with a "Y" (or <CR> for default "Y") for yes, or it keeps and uses the latest-generated label table with an "N". Contents of this current label table are printed by the Monitor AT command. A "Y" response also fills device memory with >FF (TRAP 0 opcode); otherwise, memory contents are retained except for newly assembled object.

The Assembler accepts the registers R0 to R255, even though this range does not exist on all TMS7000 devices.

During EVM Assembler execution with the listing output sent to the terminal, execution (and the listing display) can be stopped and started by keyboard input. The <ESC> key stops the display at a point where a <CR> is to be printed. After the display is stopped, the <ESC> key aborts the Assembler and returns control to the Monitor, the <SP> key causes assembly and display of text one line at a time for each <SP> entry, and any other key restarts Assembler execution.

Files from editors on the EVM or on a different computer will be accepted by the Assembler. If the text is sourced from a separate system, the text file must be bracketed with the ">" and the "<" symbols (the editor precedes these with an asterisk, like a comment). For example:

```
*>
REG   EQU   R10
PER   EQU   P30
      AORG >F006
      MOV   %>30,REG
*     COMMENT LINE
      INC   R40
      END
*<
```

The > and < in the "*>" and "*<" character sets are the beginning-of-file (BOF) and end-of-file (EOF) markers recognized by the EVM Text Editor and EVM Assembler. If an END directive is the last line, the *< is not necessary.

The EVM Assembler accepts both forward- and backward-referenced labels. All equate statements (EQU) must be read before the equated label is used. The assembled source listing will show "00" for relative jumps and "0000" for absolute jumps for all forward referenced labels. These locations are resolved when the label is assembled and the correct values are placed into the appropriate RAM locations. The object code in RAM may be inspected with the DM or MM commands described in Section 6 (program start indicated by AORG operand or by device PC value).

After the Assembler receives the END directive, it lists all forward referenced relative jumps in which the displacement was out of range. All relative jumps must be between -128 and 127 bytes. The output listing will list the label, the address of the label, and the out-of-range indicator ("OR"). Unresolved labels are also listed. The format of an unresolved label listing is the label, the address of the label, and the unresolved label indicator ("UL"). These labels are printed as they occur in the Assembler label table, three on a line. Also, an error count is printed. This number is a decimal number from 0 to 255 and does not include a count of the label errors mentioned above. The following is an example of the listing:

```
LABEL  F820 OR
LABEL2 F83A UL

12 ERRORS
```

A complete listing of the label table after assembly is available with the Monitor AT command. The label table remains intact until another assembly is performed.

When the number of errors in the assembly exceeds 255, the assembly is aborted, and the above display is given for the assembly to that point. Then the Monitor is entered.

Any time a file is assembled, only the absolute object code is loaded into RAM. The original source and listing are not saved. The assembled listing is generated only during the assembly and sent to the output port.

For emulation of device types 1 and 2 (TMS7020, TMS70C20, TMS704x, and TMS70C4x), execution of the Assembler in any form will not destroy the contents of the Text Editor. If the device type is other than 1 or 2, execution of the Assembler automatically initializes the Text Editor (resets internal flags and editor contents are lost as the editor and assembler use the same memory space).

5.1.1 Assembling Files From a Host System

Once a file is edited with the proper beginning-of-file and end-of-file markers (as described earlier in this section), the EVM is ready to assemble the file.

Two methods exist for entering files into the EVM Assembler: download and terminal emulation. Download mode involves source input at Port 2 and listing output at Port 1. This mode allows the EVM to function as a peripheral to a host system. In order to get a hard copy of the listing, a printing terminal (or a printer daisy chained onto the terminal) must be used at Port 1. The second method, terminal emulation, involves input at Port 1 and listing output at either Port 1 or Port 2. This mode allows an intelligent terminal running terminal emulation software to be connected to the EVM at Port 1. The commands for assembling files from a host system are:

XA 2 1<CR>	Download (source input: Port 2, listing output: Port 1)
XA 1 2<CR>	Terminal Emulation (source input: Port 1, listing output: Port 2,1)
XA 1 1<CR>	

Since these command strings specify source input from Port 1 or 2, the "LINE NUMBERS? (N)" prompt will be issued. Any response other than "N" means that line numbers are included in the file and none need be created by the file-input routine.

Example 5-1. File With Line Numbers

```
*>
0001 LABEL EQU R20
0002       INC A
0003 LOOP  CLR R30
0004       JMP LOOP
0005       END
*<
```

Note: If there is to be a label in the label field, then there must be only one space between the line number and the label. There must also be at least one space between the label and the mnemonic, or at least two spaces between the last digit in the line number and the mnemonic if there is no label.

Example 5-2. File Without Line Numbers

```
*>
LABEL EQU R20
      INC A
LOOP  CLR R30
      JMP LOOP
      END
*<
```

Another option is the ability to suppress the assembled listing. If any errors are detected, they are output to the terminal in the standard error format:

```
XXXX ****ERROR YYY
```

where XXXX is the line number and YYY is the error code. To suppress the listing, use a 0 (zero) as the XA second parameter:

XA 2 0<CR>
XA 1 0<CR>

Remember that since the source listing is not stored by the EVM; suppressing it during assembly will require another assembly to generate it.

5.1.2 Assembling Source Files From Audio Tape

The EVM accepts files from tape which were loaded to tape by the EVM Text Editor. When the text file is loaded to tape, the EVM will automatically include the beginning-of-file and end-of-file markers previously mentioned. The format of the tape assembly command is as follows:

```
XA 3 {output port 1,2}<CR>
```

The output port can be Port 1 or 2 and is for the listing. There is not a line number option in the assembly from tape command because all files from tape have line numbers, since they were created by the EVM Text Editor. The EVM Text Editor also automatically provides the space after the line numbers required by the Assembler.

There is also a suppress-listing option with the command to assemble from tape:

```
XA 3 0
```

Concatenation of Audio Tape Files. For software development with the EVM Text Editor and audio tape, file concatenation allows text files to be created, stored, manipulated, and assembled. When the Text Editor issues the RAM FULL error, save the file to tape, initialize the Text Editor and continue entering text. This process can be repeated as necessary, but the last file created must have the END assembler directive. When the Assembler is executed with input from tape, it will assemble until it finds the END directive. If it finds the end-of-file mark first (as automatically provided for each section by the Text Editor) it assumes file concatenation and responds with:

```
FILENAME:
```

at the terminal, accompanied by a beep. A filename may now be specified for the next section of text. If the user does not intend to concatenate files but simply forgot to put an END directive in the text, the <ESC> key should be pressed, causing the Assembler to proceed to the Assembler termination routine as though an END directive had been present. By using this method of file concatenation, long text files can be broken up into smaller, more manageable parts, edited and stored in any order, and assembled in the proper order automatically using the built-in file search capability of the EVM.

5.1.3 Assembling from RAM

Assembling a text file stored in RAM by the EVM Text Editor provides a way to quickly debug programs of moderate size. The error messages are the same as when assembling from an external source. This feature is for device types 1 (TMS702x, '70C2x) and 2 (TMS704x, '70C42) only. The format is:

```
XA 0 {output port 0,1,2}
```

where the output port can be: 0 (no listing), 1 (terminal), or 2 (line printer).

5.2 Object Code Loading and Dumping

The EVM accepts object code in three formats (7000, 9900, and Tektronix) and outputs object code in two formats (7000 and Tektronix). Input files must be at absolute or load module level. For an explanation of each of these formats, see the appropriate assembler manual (a listing of these assembler manuals is given in Section 1.5). Object files other than load level may cause the EVM to generate an error for that file. Note that the EVM assemblers do not output tagged object, merely absolute values loaded at set memory locations.

5.2.1 7000 Dump Format

If a dump is performed from locations >FF34 to >FF46, then the 7000 dump format would appear as follows:

```
K0000PROGRAM 9FF34BDEADBDEADBDEADBDEADBDEADBDEADBDEADBDEADBDEAD7F0BFF
BDEAD*DE7FDC6F
:<CR>
```

If a dump is performed from locations >FF34 to >FF47, then the 7000 dump format would appear as follows:

```
K0000PROGRAM 9FF34BDEADBDEADBDEADBDEADBDEADBDEADBDEADBDEADBDEAD7F0BFF
BDEADBDEAD7FD29F
:<CR>
```

The 7000 format tags are explained below:

K 4-digit count of relocatable bytes followed by 8-character program identifier (this is a "0" for 9900 format).

Load address:

9 4-digit origin address follows the "9".

Object code data:

B 4-digit word follows a "B".
***** 2-digit byte follows an asterisk (*) (7000 format only).

End of line:

7 4-digit checksum follows - compare the checksums.
8 4-digit checksum follows - ignore the checksum.
F ignore all characters until <CR>.

The 7000 checksum is the 2's complement of the 16-bit sum of the ASCII values of all characters on the line between carriage returns, excluding all control characters such as <CR>, <LF>, and <FF>.

5.2.2 Tektronix Dump Format

The two previous dumps (in Section 5.2.1 above) would be represented in Tektronix format as follows:

```
/FF34130EDEADDEADDEADDEADDEADDEADDEADDEADDEADDEDD
/00000000<CR><LF>
```

and:

```
/FF34140FDEADDEADDEADDEADDEADDEADDEADDEADDEADDEADF4
/00000000<CR><LF>
```

respectively.

The Tektronix line starts with a "/" that is never included in checksum calculations. The next four digits are the address to store the first byte of data on that line. The next two bytes are the number bytes of data on that line. The next two bytes is the first checksum, an 8-bit value consisting of the sum of the hex values of each digit in the address and byte count fields. The data bytes follow, equal to the count given at the start of the line, followed by another checksum, an 8-bit value equal to the sum of the hex values of each digit of the data bytes only. The dump is terminated with a line of zeros in the first three fields.

5.3 LBLA Assembler (XL)

The Line-By-Line-Assembler (LBLA) is entered any time the user wants to assemble code a line at a time with input and output at Port 1 (terminal). The LBLA is entered with the XL command (start new label table) or the XP command (use old label table).

Upon entering the LBLA, a banner message will appear followed a line number and memory location (default AORG). The cursor (shown here as "¬") will then be positioned for entry of a line of code (in the start of the label field of the source line). For example:

```
?XL
INITIALIZE? (Y)

TMS7000 ASSEMBLER

0001  F806       ¬
```

In response to the "INITIALIZE? (Y)" prompt:

- An "N" response keeps all memory locations unchanged (i.e., maintains previous programs in memory).
- Any other response (e.g., <CR> or "Y") initializes all processor-addressable memory to the value >FF (TRAP 0 opcode).

Contents of the current label table are inspected with the AT command.

The LBLA will indicate errors as they occur. If an error is detected, an error message is printed on the next line and the entire line containing the error is ignored. After printing the error number, a new line number and memory location are printed on the next line, but the memory location is unchanged from the line with the error. For example:

```
0001   F806      LOOP CLB
0001  ****ERROR 5
0001   F806       ⌐
```

Error 5 indicates that "CLB" is an illegal mnemonic. The label "LOOP" is not stored as a label because the entire line has been ignored.

The ESC key, entered while inputting a source line, will delete all inputs made for the line and reprompt the same line number, ready for new inputs. Any labels defined in the deleted line (i.e, in the label field) will not be placed in the label table.

As each line is entered and properly terminated, it is assembled and the opcode(s) stored in RAM. All forward-referenced labels will assemble as "00" for byte values or "0000" for address values. All forward reference labels are resolved as the label is encountered and the value is put in RAM. If a previously defined label is referenced, the LBLA will calculate the relative offset (if it is a relative jump) and place it in RAM. The offset will appear in the listing. If the instruction is an absolute jump, the value of the label will be placed in RAM and in the listing.

Registers from R0 to R255 are accepted by the LBLA.

```
0001  F806 B5    LOOP CLR A
0002  F807 D3         INC R10
      F808 0A
0003  F809 8C         BR  @LOOP
      F80A F806
0004  F80B E0         JMP LOOP
      F80C F8
0005  F80D            END
```

The END directive terminates the LBLA, and control returns to the Monitor. The END directive is discussed in Section 5.6.

5.4 Altering Programs After Assembly (Using XP)

Once the program is entered and completed with the END assembler directive, object code can be changed two ways. First, code in RAM *can be changed* directly using the Monitor command MM. Second, code *can be added* to a program using the LBLA Patch command XP. The new code can reference any labels used in the initial program. When the XP command is entered, the EVM responds in the same manner as it does in the LBLA except that the label table is not reset. The following restrictions apply:

1) Patching can be done only on the most recently assembled program.

2) The first line of the patch must be an AORG assembler directive. The AORG directive points to the location in the program where the patch is to be inserted. The AORG directive is discussed in Section 5.6.

The patch writes over the existing code. To insert a block of code in the program, a Branch (BR) statement should be patched into the existing program to branch to a location outside of the existing program limits. Code the patch at that location and branch back into the existing program. Any instructions overwritten by the first branch must be included at the start of the patch.

For example, assume that the following is the existing program:

```
          .
          .
          .
0035 FA34 B3      INC A
0036 FA35 D5      CLR R10
     FA36 0A
0037 FA37 C2      DEC B
0038 FA38 CC      RR  B
          .
          .
          .
0097 FB03         END
          .
          .
```

To insert a patch between the "INC A" and "CLR R10", do the following:

1) Enter the LBLA Patch mode with the XP command.

2) Use the AORG >FA35 assembler directive to place the branch (BR) statement at Program Counter location >FA35.

3) Code in the branch statement to branch around the end of the existing program (>FB03 or greater).

4) Use the AORG assembler directive again to place the Program Counter location at the point you choose to branch to.

5) Code in the patch, remembering to include the code eliminated by the branch statement, and include another branch statement to jump back into the original program. Use the END assembler directive to terminate the patch.

```
0001 F806         AORG >FA35    (set up assemble address)
0002 FA35 8C      BR   @>FB03
     FA36 FB03
0003 FA38         AORG >FB03    (set up patch start address)
0004 FB03            .          (patch code start)
                     .
                     .
                     .          (end of new instructions)
0043 FB91 D5      CLR  R10      (repeat 2 instructions
     FB92 0A                      overwritten by the
0044 FB93 C2      DEC  B          branch to the patch)
0045 FB94 8C      BR   @>FA38   (to rest of main program)
     FB95 FA38
0046 FB97         END           (end patch)
```

This process can be repeated as necessary.

5.5 Instruction Format

The TMS7000 Assembler instructions are described in the TMS7000 Assembly Language Programmer's Guide and TMS7000 Family Data Manual.

5.5.1 Constants

Hexadecimal constants are preceded by ">". For example:

```
0001   F806  8C     BR @>10FF
       F807  10FF
0002   F809
```

Binary constants are preceded by "?" or "<". For example:

```
0001   F806  22     MOV   %?0101,A
       F807  05
0002   F808
```

Octal constants are preceded by an exclamation mark (!). For example:

```
0001   F806  22     MOV   %!10,A
       F807  08
0002   F808
```

Decimal constants are not preceded by a special character. For example:

```
0001   F806  A8     MOVD   %100(B),R3
       F807  0064
       F809  03
0002   F80A
```

String constants must be enclosed in single quotes and must be printable characters with ASCII character code representation. The null string is also a legal entry. For example:

```
0001   F806  29     ADC   %'A',A
       F807  41
0002   F808  29     ADC   %'',A
       F809  00
0003   F80A
```

5.5.2 Label Format

In the operand field, labels are preceded by the "@" symbol:

```
HERE    LDA   @LABEL
```

Labels can be equated to labels:

```
HERE    EQU   THERE            (provided THERE is predefined)
LABEL1  EQU   LABEL2+X         (provided LABEL2 is predefined;
                                 offset of X is 0 to 65535)
LABEL3  EQU   LABEL1+LABEL2    (LABEL1 & LABEL2 predefined)
```

5.5.3 "$" Indicates PC Value

Examples are:

```
JEQ    $+6     IF EQUAL, SKIP NEXT 6 BYTES

JMP    $       CONTINUOUS LOOP, WAIT FOR RESET
```

5.5.4 Register/Peripheral File Requirements

The number following the "R" or "P" designations must be an unsigned decimal number in the range 0 to 255. For example:

```
0001  F806  48  ADD   R5,R6
      F807  05
      F808  06
0002  F809  94  ORP   B,P7
      F80A  07
0003  F80B
```

5.6 Assembler Directives

Both the EVM Assembler and LBLA support the following directives.

5.6.1 AORG

Format: AORG >XXXX

The AORG command allows a program or piece of a program to be placed at a specific location in memory. During assembly, all references to memory locations (BR, JMP, Jcnd, LDA, STA, etc.) are based on the AORG statement. For each device type, the AORG defaults to a specified value but can be set to any value greater than the default.

5.6.2 EQU

Format: <label> EQU {value, register, or peripheral register}

The EQU directive assigns a label to a decimal or hex value, a register, or a peripheral register. An equate to a register or peripheral register must be defined before that label is used. For example:

```
REG       EQU       R25
OUTPUT    EQU       P4
          CLR       REG             CLEAR REG 25
          MOVP      %>F0,OUTPUT     SET P4
ONES      EQU       >FFFF
          MOVD      %ONES,REG+1     SET R25, R26 ALL ONES
```

See label usage in Section 5.5.2 on page 5-10.

5.6.3 BYTE

The operand of the BYTE directive is a one-byte constant (any format) to be loaded into the next byte of memory. Constants must be positive or negative values in the range of >00 to >FF, and can be entered in hex, decimal, or binary form. The operand can also be the sum of labels previously defined by EQU statements, with an unlimited number of terms. The last term can be a hex or decimal value. Values greater than 8 bits are truncated. For example:

```
0001  F806  FF        BYTE   >FF              (hexadecimal)
0002  F807  0A        BYTE   ?0001010         (binary)
0003  F808  FF        BYTE   -1               (decimal)
0004  F809  41        BYTE   'A'              (string)
0005              *
0005  F80A  20 TST1    EQU    >20
0006  F80A  10 TST2    EQU    >10
0007  F80A  0E TST3    EQU    14
0008  F80A  0D TST4    EQU    13
0009  F80A  00 TST5    EQU    0
0010              *
0010  F80A  30        BYTE   TST1+TST2
0011  F80B  4D        BYTE   TST1+TST2+TST5+TST3+TST4+2
```

5.6.4 DATA

The operand of the DATA directive is a two-byte constant (any format) that is to be loaded into the next two bytes of memory. Only the last four valid hexidecimal digits entered are stored. Data may be entered as positive or negative values. DATA statements can be the the sum of previously defined EQU statements, with an unlimited number of terms, and the last term can be a hex or decimal value. DATA statements can be used with labels and when assembled will be equal to the address of the label. Values greater than 16 bits are truncated. For example:

```
0001  F806  DD        DATA   >DDAD            (hexadecimal)
      F807  AD
0002  F808  04        DATA   ?10011000100     (binary)
      F809  C4
0003  F80A  00        DATA   32               (decimal)
      F80B  20
0004  F80C  41        DATA   'AB'             (string)
      F80D  42
0005              *
0005  F80D      TABLE  EQU    >FC00
0006  F80D      TAB1   EQU    >0100
0007              *
0007  F80D  FC        DATA   TABLE+2
      F80E  02
0008  F80F  FD        DATA   TABLE+TAB1
      F810  00
```

5.6.5 TEXT

The operand of the TEXT directive is a string of arbitrary length. The string must be enclosed in single quotes and follow the format for string entries. Null strings are not allowed.

```
0001  F806  53  TEXT   'STRING'
0002  F80C
```

Note that the ASCII values of the TEXT string are not printed. This is a space-saving measure for listings. The next address printed indicates the next available byte after the TEXT string. The Monitor commands MM and DM can be used to display and modify assembled contents.

5.6.6 BSS

Format: [label] BSS <value>

The BSS directive reserves a block of memory the size of the value. The value can be hex or decimal. If no label is present, then the location counter is advanced an amount equal to the value. If a label is present, it is assigned the value of the location of the first byte in the block, and the location counter is advanced an amount equal to the value. For example:

```
LABEL   BSS  >20      SAVE A BUFFER OF 32 BYTES
```

5.6.7 BES

Format: [label] BES <value>

The BES directive reserves a block of memory the size of the value. The value can be hex or decimal. If no label is present, then the location counter is advanced an amount equal to the value. If a label is present, it is assigned the value of the location of the next byte following the block, and the location counter is advanced an amount equal to the value. For example:

```
LABEL   BES  >20      SAVE A BUFFER OF 32 BYTES
```

5.6.8 Comment

If "*" is the first character on a line, the entire line is a comment, and the LBLA ignores assembling any characters following the "*". Comments can also be included on any line if separated from the last field by at least one space. For example:

```
0001  F806     ****************************************
0001  F806     *   (these three lines are comments)
0001  F806     *
0001  F806
```

5.6.9 END

The END directive prints unresolved labels, out-of-range labels, and the total error count. Control is then returned to the Monitor.

5.6.10 Additional Assembler Directives

The following assembler directives are not processed by the Assembler, but will be ignored without error: TITL, IDT, and PAGE. All other directives will cause an error.

5.7 Assembler Errors

When using the LBLA or the Assembler, the error listing will occur on a new line and will contain the line number the error occurred on. The LBLA will indicate an error immediately after it occurs, while the Assembler will wait until the entire line is input, including comments. A label defined on the line containing the error *is ignored*, as is the assembly mnemonic. Also, in the Assembler mode (downlink from a host system or input from tape) the EVM Assembler will store in RAM four consecutive NOP instructions each time it encounters an error. This allows error correction after the rest of the file is assembled, using the MM or XP commands.

Errors are indicated by:

XXXX ****ERROR YYY

where "XXXX" is the line number where the error occurred and "YYY" is the error number. Error numbers and explanations are given in Table 5-1.

Table 5-1. Error Codes

ERROR NUMBER	EXPLANATION
1	The object code for user program has exceeded the available RAM for storage of object code for that device type (see Section 2.11).
2	Unable to AORG at location specified due to RAM limits (see Section 2.11).
5	Mnemonic entered not found in TMS7000 instruction set.
15	Expression is not proper for the specified TMS7000 instruction.
17	Not a printable character.
20	Expected a character A-Z, did not receive.
25	Register/peripheral value greater than 255.
30	Comma expected but was not found.
35	Premature <CR> detected.
40	Duplicate label definition.
60	Operand should be an absolute address or a label equating to an absolute address.
70	<CR> detected as first character in mnemonic field.
80	Greater than 6 characters in label.
200	Operand must be Register A, Register B, a register number (R0-R255), or a label equating to a register. In the case of a PUSH or POP command the operand may also be ST (Status Register).
203	Jump displacement too large (acceptable values are -128 to +127).
205	Operand must be Register A, Register B, a register number (R0-R255), or a label equating to a register.
207	Addressing must be direct, indirect, or indexed.
208	TRAP number must be less than 24.
209	The operand source must be Register A, Register B, a register number (R0-R255), a label equating to a register, or an absolute value.
210	If the operand source is Register A then the destination must be Register B, a register number (R0-R255), or a label equating to a register. If the operand source is Register B then the destination must be Register A, a register number (R0-R255), or a label equating to a register.

Table 5-1. Error Codes (Concluded)

ERROR NUMBER	EXPLANATION
212	The operand source must be Register A, Register B, a register number (R0-R255), or a label equating to a register.
219	The operand source must be Register A, Register B, or an absolute value. If the instruction is MOVP then the operand source may also be a peripheral register (P0-P255) or a label equating to a peripheral register.
220	The operand destination must be a peripheral register.
223	The operand destination must be Register A or Register B.
224	Improper format in the operand.
245	Index register must be Register B.
254	Format is improper for an equate statement.

5.8 Executing Programs

This section describes a sample debug session. The user-entered commands and keys are underscored throughout the example. First, the program below is assembled using the XA command.

```
?XA<CR>

TMS7000 ASSEMBLER
INITIALIZE? (Y)<CR>
0001  F006  22  MOV  %>55,A
      F007  55
0002  F008  D0  MOV  A,R2
      F009  02
0003  F00A  D3  INC  R2
      F00B  02
0004  F00C  18  ADD  R2,A
      F00D  02
0005  F00E  01  IDLE
0006  F00F      END
```

Once the program has been loaded into memory, the PC, ST, and SP registers must be set using the MS command.

```
?MS
PC=FF16 F006<SP>
SP=R1 02<CR>
```

The program begins at >F006; therefore, the PC must be set to >F006. For this example, the Status Register (ST) may be set to zero. The Stack Pointer (SP) has been set at R2 to preserve register space. The EX command may now be used to run the program to the breakpoint address >F00E as shown:

```
?B1 F00E<CR>
BP1=F00E  BP2=xxxx

?EX<CR>
NEXT INST---> 0080          IDLE
PC=F00E  C=0 N=1 Z=0 I=0  SP=R2  A=AB  B=00<CR>
```

Checking the registers shows that the program has executed properly:

```
?MR
+ R0=AB
+ R1=00
+ R2=56
+ R3=F8
+ R4=0E
+ R5=40
+ R6=00
```

Note that the current memory address and status of the program were saved within the user register file in registers R3, R4, and R5, respectively. R6 is an undefined user register.

Once the MS command has been reset as shown below, single-stepping the previous program with the SS command results in the following displays. The MR command is used between single-steps to examine the registers.

```
?XA<CR>
INITIALIZE? (Y)<CR>
TMS7000 ASSEMBLER
>
0010   F006 22            MOV     %>55,A
       F007 55
0020   F008 D0            MOV     A,R2
       F009 02
0030   F00A D3            INC     R2
       F00B 02
0040   F00C 18            ADD     R2,A
       F00D 02
0050   F00E 01            IDLE
0060   F00F               END

0 ERRORS

TMS7000 DEBUG MONITOR  REV 2 X   (assembler returns to Monitor)
?
```

Execute MS command.

```
?MS
PC=F006
SP=R1 R2<CR>
```

In the upcoming examples, Register A (R0) and R2 are used for data manipulation. Register B (R1) is not used. R3, R4, and R5 are used to process the single-step (SS command) breakpoint. R3 and R4 contain the address of the last byte of the next instruction, and R5 contains the Status Regsiter contents after the single step.

In these examples, the program is single stepped starting at >F006 (Text Editor source statement 0010). Stack Pointer has been set to R2; thus R3/R4 = last byte of next instruction, and R5 = Status Register contents, updated after each SS instruction.

```
?SS
LAST INST---> 0010     MOV     %>55,A
  CYCLE COUNT = 000171
NEXT INST---> 0020     MOV     A,R2
PC=F008  C=0 N=0 Z=0 I=0  SP=R2 A=55 B=00
```

Check register contents to match just-executed step. A >55 has been moved into RA (R0). R3/R4 contain the last-byte address of the next instruction. R5 contains Status Register contents.

```
?MR
+ R0=55 (01010101) <SP>
+ R1=00 (00000000) <SP>
+ R2=00 (00000000) <SP>
+ R3=F0 (11110000) <SP>
+ R4=09 (00001001) <SP>
  R5=00 (00000000) <CR>
```

Single step next address.

```
?SS
LAST INST---> 0020      MOV     A,R2
 CYCLE COUNT = 000179
NEXT INST---> 0030      INC     R2
PC=F00A  C=0 N=0 Z=0 I=0  SP=R2 A=55 B=00
```

Check register contents. R2 now contains a copy of R0 (RA) contents.

```
?MR
+ R0=55 (01010101) <SP>
+ R1=00 (00000000) <SP>
+ R2=55 (01010101) <SP>
+ R3=F0 (11110000) <SP>
+ R4=0B (00001011) <SP>
  R5=00 (00000000) <CR>
```

Single step next address.

```
?SS
LAST INST---> 0030      INC     R2
 CYCLE COUNT = 000186
NEXT INST---> 0040      ADD     R2,A
PC=F00C  C=0 N=0 Z=0 I=0  SP=R2 A=55 B=00
```

Check registers. R2 has been incremented.

```
?MR
+ R0=55 (01010101) <SP>
+ R1=00 (00000000) <SP>
+ R2=56 (01010110) <SP>
+ R3=F0 (11110000) <SP>
+ R4=0D (00001101) <SP>
  R5=00 (00000000) <CR>
```

Single step next address.

```
?SS
LAST INST---> 0040      ADD     R2,A
 CYCLE COUNT = 000194
NEXT INST---> 0050      IDLE
PC=F00E  C=0 N=1 Z=0 I=0  SP=R2 A=55 B=00
```

Check registers. R0 contains R2+R0.

```
?MR
+ R0=AB (10101011) <SP>
+ R1=00 (00000000) <SP>
+ R2=56 (01010110) <SP>
+ R3=F0 (11110000) <SP>
+ R4=0F (00001111) <SP>
+ R5=40 (01100000) <SP>
  R6=FE (11111110) <CR>
```

6. Debug Monitor

This section contains a description of each Monitor command, presented in alphabetical order, except for EPROM programmer commands covered in Section 9.1. Covered in this section:

Table 6-1. Monitor Commands, Alphabetical Summary

COMMAND	DESCRIPTION	SECTION	PAGE
AR	+/- Hex Arithmetic	Section 6.6.1	6-7
AT	Display Assembler Label Table	Section 6.6.2	6-7
BR	Display/Modify Baud Rate	Section 6.6.3	6-8
BT	Set Breakpoint on Trap	Section 6.6.4	6-8
B1/B2	Set Breakpoints 1 and 2	Section 6.6.5	6-9
CB	Clear Breakpoints	Section 6.6.6	6-10
CC	Clear SR Carry Bit	Section 6.6.46	6-36
CE	Compare EPROM to Memory	Section 9.1.2	9-3
CI	Clear SR Interrupt Bit	Section 6.6.46	6-36
CL	Designate Cursor-Left Key	Section 2.15.3	2-18
CN	Clear SR Negative Bit	Section 6.6.46	6-36
CP†	Clear Processor Status	Section 6.6.7	6-10
CS	Designate Cursor-Up Key	Section 2.15.3	2-18
CT	Clear Breakpoint on TRAP	Section 6.6.9	6-11
CY	Display/Clear Cycle Counter	Section 6.6.10	6-11
CZ	Clear SR Zero Bit	Section 6.6.46	6-36
C1/C2	Clear Breakpoints Individually	Section 6.6.11	6-12
DB	Display Breakpoints	Section 6.6.12	6-12
DC	Decimal-Hex Byte Conversion	Section 6.6.13	6-12
DI	Disable (Clear) SR Bits	Section 6.6.46	6-36
DM	Display Memory	Section 6.6.14	6-13
DP†	Display Processor Status	Section 6.6.15	6-14
DR	Audio Tape Directory	Section 6.6.16	6-14
DS	Display Machine State	Section 6.6.17	6-15
DT	Display Breakpoint on Trap	Section 6.6.18	6-15
DV	Select TMS7000 Family Device	Section 6.6.19	6-16
EF	Execute to Breakpoint with Fixed Display	Section 6.6.20	6-17
EI	Enable (Set) SR Bits	Section 6.6.46	6-36
ET	Execute to Breakpoint with Trace	Section 6.6.20	6-18
EX	Execute to Breakpoints	Section 6.6.20	6-18
FB	Find Byte in Memory	Section 6.6.21	6-20
FM	Fill Memory	Section 6.6.22	6-21
FR	Fill Register File	Section 6.6.23	6-22
FS	Single Step with Fixed Display	Section 6.6.24	6-23
GO	Go Execute at Address	Section 6.6.20	6-20
HC	Hex-Decimal Word Conversion	Section 6.6.25	6-24
HE	Help	Section 6.6.26	6-24

†The following are five commands (and their shortened versions) that can be used: CP (C), DP (D), MA (A), MB (B), and PC (P).

Table 6-1. Monitor Commands, Alphabetical Summary (Concluded)

COMMAND	DESCRIPTION	SECTION	PAGE
IO	Display I/O Status	Section 6.6.27	6-24
IS	Inspect Instruction Trace Sample Count	Section 6.6.49	6-40
IT	Inspect instruction Trace Samples	Section 6.6.49	6-40
LA	Show Address of Line	Section 6.6.28	6-25
LL	List Lines from Text Editor	Section 6.6.29	6-25
LM	Load Memory - 7000 Format	Section 6.6.30	6-26
LN	Show Editor Line at Address	Section 6.6.31	6-26
LS	Load Machine State	Section 6.6.32	6-27
LT	Load Memory - Tektronix Format	Section 6.6.33	6-28
L1/L2	Set Breakpoint 1 or 2 by Line Number	Section 6.6.34	6-28
MA/MB†	Display/Modify Registers A and B	Section 6.6.35	6-29
MM	Display/Modify Memory	Section 6.6.36	6-29
MO	Audio Tape Motor On	Section 6.6.37	6-30
MP	Display/Modify Peripheral File	Section 6.6.38	6-31
MR	Display/Modify Register File	Section 6.6.39	6-32
MS	Display/Modify PC, SP, Regs. A and B	Section 6.6.40	6-33
MV	Move (Copy) Memory	Section 6.6.41	6-33
NP	Fill Memory With NOPs	Section 6.6.42	6-34
PC†	Display/Modify PC, SP, Regs. A and B	Section 6.6.40	6-33
PE	Program EPROM From Memory	Section 9.1.1	9-2
PT	Print Trace Contents to Port 1 or 2	Section 6.6.49	6-41
RE	Read EPROM to RAM	Section 9.1.3	9-4
RT	Reset Target Processor	Section 6.6.43	6-34
RU	Execute Program Without Breakpoints	Section 6.6.44	6-35
SC	Set SR Carry Bit	Section 6.6.46	6-36
SI	Set SR Interrupt Bit	Section 6.6.46	6-36
SM	Save Memory - 7000 Format	Section 6.6.45	6-36
SN	Set SR Negative Bit	Section 6.6.46	6-36
SP	Display/Modify PC, ST, SP; start at SP	Section 6.6.40	6-33
SR	Display Status Register	Section 6.6.46	6-36
SS	Single Step Program	Section 6.6.47	6-37
ST	Save Memory - Tektronix Format	Section 6.6.48	6-38
SZ	Set SR Zero Bit	Section 6.6.46	6-36
TC	Configure Single-Step Trace	Section 6.6.50	6-42
TF	Turn Off Instruction Trace	Section 6.6.49	6-41
TO	Turn On Instruction Trace	Section 6.6.49	6-41
TS	Single-Step Program With Trace	Section 6.6.51	6-42
T0	Load PC With Trap 0 (Zero) Vector	Section 6.6.52	6-43
VE	Verify EPROM Erased	Section 9.1.4	9-4
XA	Execute Assembler	Section 6.6.53	6-44
XE	Execute Text Editor	Section 6.6.54	6-45
XL/XP	Execute Line Assembler	Section 6.6.55	6-46
XR	Execute Reverse Assembler	Section 6.6.56	6-46

†The following are five commands (and their shortened versions) that can be used: CP (C), DP (D), MA (A), MB (B), and PC (P).

6.1 Command Parameters

Except for noted default values, command parameters must be entered in the order shown in this section. Failure to enter the required number of parameters will result in termination of the command. The default values for these commands are defined in more detail in the command descriptions that follow.

After command entry, the Monitor prints a space. The first parameter must be entered immediately following this space. When more than one parameter is required, enter one space before the next parameter(s). The general command format is:

?CMD PARM1 PARM2 PARM3 PARM4<CR,SP>

6.1.1 Numerical Parameters

When the Monitor is accepting numerical input data, any entry other than valid hex or decimal digits (or their symbolic equivalents) will cause the command to abort with no change made to effect any memory location. All address and data inputs are in hexadecimal format only. File and Peripheral File locations and modifications to the Stack Pointer can be either in decimal or in hex if preceded by ">".

For the following two-address commands, a "+" can be used as the second parameter to define a byte offset from the first parameter:

DM FM MV FB SM ST XR

For example, the command

```
?FM   >F800 +>100  >AA<CR>
```

means to fill memory from >F800 to >F900 (>F800+>100) with the value >AA. If only the "+" is used in the above example, it means one byte or fill byte address >F800 with >AA. The sum of the byte offset and the start address cannot exceed >FFFF.

6.1.2 Symbolic Parameters

The $ (dollar sign) and source-statement symbols can be used in Monitor commands. The dollar sign indicates current Program Counter (PC) value. The following are examples:

```
?HC $<CR>                    (display PC in hex and decimal)
>F300=62208                  (current PC = F300 hex)
?B1 $<CR>                    (set breakpoint 1 to PC value)
BP1=F300 BP2=0000
```

Source-statement symbols currently in the Assembler symbol table can be used in Monitor commands. (To view the symbol table, issue AT command.) For example:

```
?MM @START                   (display address labeled START)
  FCA0=FE (11111110)

?MM @START+6                 (show memory at START + 6 bytes)
  FCA6=AA (10101010)
```

Note: Offsets (like the "6" above) must be in the range of 1 to 9.

```
?DM $ @TABLE          (dump from PC to address of TABLE)
```

?DM @HERE @TABLE (dump from addresses of HERE to TABLE)

6.2 Defining Registers

On initial powerup, register contents will be undefined. Afterwards, the register contents will remain intact regardless of reset or execution breakpoint operations. The only exception involves TRAP 0 discussed in Section 6.9.

6.3 Command Termination

If the Escape key (<ESC>) is pressed during entry of a command string, the command will immediately terminate and return control to the Monitor top level. This escape mechanism is invalid during program execution because the Monitor is no longer performing keyboard polling. Use RESET to halt the program.

Where a carriage return (<CR>) or space (<SP>) can be used interchangeably to terminate a command, the notation <CR,SP> will be used.

6.4 Display/Modify Procedures

The EVM display/modify commands:

- MA and MB (Registers A and B, Program Counter, and Stack Pointer)
- MS, PC, and SP (Program Counter, Stack Pointer, and Registers A and B)
- MM (Program Memory)
- MR (Register File)
- MP (Peripheral File)
- SR (Status Register - display only)

When a display/modify command is entered, there are six options for proceeding:

- Change by entering hex digits (0-255 decimal or hex preceded by ">" for the Stack Pointer)
- Carriage return (<CR>) to terminate
- Space (<SP>) to continue
- Plus sign (+) except for MA and MS (same as <SP>)
- Minus sign (-) except for MA and MS
- The redisplay command (<)

After data is displayed, valid digits can be entered to replace those currently displayed. Any number of digits may be entered, but only the last two (four in the case of Program Counter modification) are retained at the current memory location. If only one digit is entered (or less than four in the case of the Program Counter), then leading zeros are assumed.

A <CR>, <SP>, "<", or "-" may be entered, regardless of whether new data has been entered. A <CR> terminates the command and control returns to the Monitor command processor. A <SP> proceeds to the next display/modify location. If "<" is entered, the value currently at the location is redisplayed for further update or continuation of the command. If "-" is entered, the command goes back to the previous display/modify location.

The MA, MB, MS, PC, and SP commands, will continue to display all four locations in a round-robin fashion using <SP>. A <CR> will return to the Monitor command entry level.

6.5 Additional Command Notes

The Program Counter, Status Register and Stack Pointer hardware registers contain the following default values upon powerup:

Program Counter:	default for device type (see Table 2-5 on page 2-14)
Status Register:	>00
Stack Pointer:	R1 (B register)

Two breakpoints on address can be set using the commands B1 and B2. The commands DB, CB, C1, and C2 display or clear the address breakpoints. Also, up to 24 breakpoints on TRAP (or interrupt) locations can be enabled using the BT command, and the commands DT and CT display or clear these breakpoints.

Commands that load RAM (LM, LT, XA, and XL) will initialize the RAM to >FF before loading. The initialized RAM is the RAM designated as object code storage for the device type selected. There are two reasons for initializing RAM:

- >FF is the unprogrammed EPROM state, and the programming routine skips memory locations with >FF (assuming the device being programmed already has >FF at that location). It also insures that only the user's program gets programmed into the EPROM.
- >FF is the TRAP 0 (software breakpoint) opcode. Surrounding the program with >FF will automatically return control to the Monitor if execution jumps outside the program memory range of the emulated device type.

Commands that prompt for "FILENAME" when dealing with Port 3 will allow more than one character but will use the last character entered as the file name. Commands that read from tape will also accept "*", which will load the first file encountered of the proper type.

All memory dump, display, or modify commands restrict address entry to the range from the Program Counter (PC) default value to >FFFF.

6.6 Monitor Command Descriptions

The Monitor commands are presented here in alphabetical order. The EPROM programmer commands are discussed in the next section. The formats of the commands use the following symbols:

<>	Angle brackets enclosing a parameter in lower case means that the parameter must be entered. Upper case letters indicate that the specified key must be pressed. <CR,SP> indicates that either carriage return or space may be used.
{}	Braces enclose a list from which the user must choose one option. The value printed in boldface is the default for the parameter.
[]	Brackets indicate that the parameter is optional.
underscore	An underscored parameter is one displayed on the terminal by the Monitor. (In an *example,* underscore represents user inputs.)

boldface A boldfaced parameter is the default.

italics The value of a parameter in italics is a command response.

In the examples, the user enters the characters or keys that are underscored.

6.6.1 +/- Hex Arithmetic (AR)

FORMAT: AR <hex number> <hex number><CR,SP>

PARAMETERS: 1) hexadecimal number (a), 4-digits maximum
2) hexadecimal number (b), 4-digits maximum

Purpose: to compute the sum (a+b) and difference (a-b) of the numbers entered and display the results.

This command allows quick hex arithmetic for address or offset computation or for determining the length of programs, subroutines, etc., by entering address values. For each data entry, if more than four numbers are entered, the last four entered are used. If less than four numbers are entered, leading zeros are assumed. Legal entries are 0-9 and A-F. If <SP> is entered for the first parameter, that parameter is assumed to be zero and the subtraction part of the command forms a 2's complement of the second number. If <CR> is entered instead of the first parameter, the command aborts. If <CR> or <SP> is entered for the second parameter, it is assumed to be zero.

EXAMPLES:

1)
```
?AR 20 10<CR>
0020+0010=0030 0020-0010=0010
?
```

2)
```
?AR <SP>FFFF<CR>
0000+FFFF=FFFF 0000-FFFF=0001
?
```

6.6.2 Display Assembler Label Table (AT)

FORMAT: AT {port **1**,2}

PARAMETERS: Output port (default=1)

Purpose: to list the label table of the most recent assembly.

If no output port is entered, the terminal is assumed. Legal values for the output port are 1 and 2. If this command is executed without a label table existing, nothing will be displayed. The resolved labels are printed first, followed by unresolved or out-of-range labels. Unresolved labels are denoted by "UL", and out-of-range labels for jumps are denoted by "OR".

While the command is dumping to the terminal, the display can be stopped and started by keyboard input. Any key will stop the display at the end of the current line of output. While the display is stopped, <ESC> will abort the command, <SP> will cause display of successive lines one at a time for each <SP> entry, and any other key will restart the display.

EXAMPLE:

```
?AT 1
LOOP  >F808       LOOP1  >F812

?
```

The label table is listed on the terminal.

6.6.3 Display/Modify Baud Rate (BR)

FORMAT: BR
(BR1-index,BR2-index) <port> <index>

PARAMETERS: 1) EIA port number (1 or 2)
2) Baud rate index (1-8)

Purpose: to examine and change the baud rates of both EIA Ports.

Legal port numbers are 1 and 2, and legal baud rate indices are:

INDEX	BAUD RATE	
1	110	
2	150	
3	300	
4	600	
5	1200	
6	2400	
7	4800	
8	9600	(Port 2 powerup default)

Immediately after BR is entered, the current values of the baud rate indices are displayed in parentheses on the next line, the first number for Port 1, the second for Port 2. Entering <CR,SP> here will terminate the command. Using this command to change the baud rate of Port 1 (terminal) will require the baud rate at the terminal also to be changed before Monitor execution can continue. At reset, the baud rate of Port 1 is determined automatically, and the baud rate of Port 2 defaults to 9600. Table 2-7 and Section 2.13 detail the steps necessary for changing the Port 2 default value to any of the legal baud rates.

EXAMPLES:

1)
```
?BR
(8,3) <CR>
?
```

The baud rates for both ports are displayed.

2)
```
?BR
(8,8) 2 3
?
```

The baud rate of Port 2 is changed from 9600 baud to 300 baud.

6.6.4 Set Breakpoint on Trap (BT)

FORMAT: BT <vector> ... <vector><CR>

PARAMETERS: Trap vector number(s), 0 to 23

Purpose: to set breakpoint(s) to occur when a TRAP location is accessed.

The range of legal vectors is from 0 to 23 and the vectors are entered in decimal. When a breakpoint on trap location is set, the execution of a TRAP in a program will cause a breakpoint at the first location in the TRAP routine. For those trap locations that are also interrupt vector locations, the breakpoint can also be considered a breakpoint on serviced interrupt.

After the space printed by the Monitor, an unlimited number of locations can be entered, separated by spaces. Entering locations already set will cause the terminal to beep. Not entering locations already set will not clear them. Entering <CR> will terminate the command and execute the DT (Display Breakpoints on TRAP) command. Only the CT command can clear breakpoint on trap locations.

EXAMPLE:

```
?BT 1 3<CR>
1 3 5 9
?
```

Breakpoints on TRAP 1 and TRAP 3 are set. Breakpoints on TRAPS 5 and 9 were previously set.

6.6.5 Set Breakpoints 1 and 2 (B1 and B2)

FORMAT: B1 <addr><CR,SP>
B2 <addr><CR,SP>

PARAMETERS: Breakpoint address

Purpose: to set Breakpoint 1 or 2 on address.

Note:

1) Breakpoint addresses must be on an instruction boundary.

2) When a breakpoint is set, the TRAP 0 opcode (>FF) will be inserted into the breakpoint address. For instance, in the example below, >FF will be inserted at location >F882. This may cause problems when the "CHECKSUMS" algorithm (see Section 6.8.1) is executed.

3) When executing to an event counter value (number of breakpoints to encounter before stopping), only BP1 can be used. See descriptions of the breakpoint commands in Section 6.6.20.

Breakpoint 1 is used with the event counter in the EX command. If no address is entered, the breakpoint remains unchanged. The last four hex digits entered are used for the address. If the breakpoint is changed, the DB command (Display Breakpoints) is executed. The address entered must be in the range of the PC default for that device (see Table 2-5) to >FFFF. The CB command clears both breakpoints.

EXAMPLES:

1) ?B1 F882<CR>
```
BP1=F882  BP2=F903
?
```
Breakpoint 1 is set to >F882.

2) ?B2 F977<CR>
```
BP1=F882  BP2=F977
?
```
Breakpoint 2 is set to >F977.

6.6.6 Clear Breakpoints (CB)

FORMAT: CB

PARAMETERS: None

Purpose: to clear both address breakpoints (B1 and B2).

The CB command is a quick way of clearing both address breakpoints. Breakpoints can also be cleared with the C1 and C2 commands below. Before clearing the breakpoints, the old breakpoint values are displayed. After clearing the breakpoints they are displayed as "0000".

EXAMPLE:

```
?CB
BP1=F882  BP2=F997
BP1=0000  BP2=0000
?
```
Both address breakpoints are cleared.

6.6.7 Clear Processor Status (CP)

FORMAT: CP (or C)

PARAMETERS: None

Purpose: to clear and display the processor status.

The Program Counter is reset to the default for the current device type being emulated (see Table 2-5). After clearing the processor status, the status is redisplayed. This command is executed automatically after the device type is altered with the DV command. This command can be abbreviated as "C."

EXAMPLE:

```
?CP
PC=F806  C=0 N=0 Z=0 I=0  SP=R1  A=00  B=00
```

6.6.8 Cycle Count Single Step (CS)

FORMAT: CS <start address> <stop address>
CLEAR CYCLE COUNT? (Y){N,Y)

PARAMETERS: 1) Address to start single step execution
2) Address of next instruction **after single step completed**
3) "N" to use present cycle-counter contents; otherwise, start at zero

Purpose: Single step program from PC start to the instruction **before** the stop address; then display cycle counter contents.

Instruction single stepping will begin at "start address" and continue until "stop address" is reached; the instruction at the stop address will not be executed. The only display will be the cycle counter start value (before single stepping) and final value after execution. Counter can be started at present contents by a "N" response to "CLEAR CYCLE COUNT? (Y)". Any other response will cause the default (counter set to zero) before starting.

Because actual single steps take place with necessary execution overhead, large programs may take minutes to execute. The command can be aborted by the board RESET switch. Then use the CY command to show cycles accumulated to that point.

EXAMPLE:

```
?CS F810 F815<CR>
CLEAR CYCLE COUNT? (Y) N          (don't clear counter)
CYCLE COUNT = 000674              (beginning cycle counter value)
CYCLE COUNT = 000876              (ending cycle counter value)
?

?CS F810 F815<CR>
CLEAR CYCLE COUNT? (Y) <CR>       (take default)
CYCLE COUNT = 000000
CYCLE COUNT = 000202
?
```

6.6.9 Clear Breakpoint on Trap (CT)

FORMAT: CT {vector numbers, A}

PARAMETERS: Trap vector number(s) 0 to 23 or "A" (all)

Purpose: to clear as many of the breakpoint on trap locations as desired.

Vector locations are cleared by entering the vector number as displayed by the DT command - a decimal number from 0 to 23. After the space printed by the Monitor, successive vectors can be entered, separated by spaces. Entering locations already cleared will cause the terminal to beep. Entering a <CR> will terminate the command and execute the DT command. If "A" is entered, all locations are cleared and the DT command is bypassed. The "A" must be the first item entered or it is interpreted as an error, since it is a hex value.

EXAMPLE:

```
?CT A
?
```

All breakpoints on trap locations are cleared.

6.6.10 Display/Clear Cycle Counter (CY)

FORMAT: CY

PARAMETERS: None

Purpose: to clear to zero the Cycle Counter (used in single step modes to show instruction cycles executed).

After executing the command, the cycle count is printed. A "0" (zero) input resets the count to zero. Any other entry (e.g., <CR> or <SP>) returns to the Monitor with the count unchanged. For example:

```
?CY
CYCLE COUNT = 003762 <CR>          (don't change counter)
?CY
CYCLE COUNT = 003762 0
CYCLE COUNT = 000000               (counter set to zero)
?
```

6.6.11 Clear Breakpoints Individually (C1 and C2)

FORMAT: C1 or C2

PARAMETERS: none

Address breakpoint 1 can be cleared by entering C1, and address breakpoint 2 can be cleared by entering C2. In both cases, the breakpoint values are redisplayed.

6.6.12 Display Breakpoints (DB)

FORMAT: DB

PARAMETERS: None

Purpose: to display the current values of the two breakpoints on address (B1 and B2).

Breakpoints can be modified only with the B1 and B2 commands and cleared with the CB, C1, and C2 commands below.

EXAMPLE:

```
?DB
BP1=F882  BP2=F977
?
```

The address breakpoints are displayed.

6.6.13 Decimal-Hex Byte Conversion (DC)

FORMAT: DC <decimal number, 0-255><CR,SP>

PARAMETERS: Up to three decimal digits for conversion (maximum value is 255)

Purpose: to convert a decimal number to hex.

This command is a quick way to convert decimal register numbers, peripheral numbers, or data to hex equivalents. If more than three numbers are entered, the last three numbers are used. If less than three numbers are entered, leading zeros are assumed.

EXAMPLE:

```
?DC 100<CR,SP>
100=>64
?
```

6.6.14 Display Memory (DM)

FORMAT: DM <start addr> <stop addr> {port 1,2,3}

PARAMETERS:
1) Memory start address (at reset, default=PC default; otherwise, default=previous value)
2) Memory stop address (default=start address + 127)
3) Output port (default=1)

> **Note:**
>
> Addresses must be within the device default range.

Purpose: to display memory in hexadecimal format.

Memory is displayed 16 bytes on a line. The 16 bytes are preceded by the address of the first byte on the line. The first address dumped is at the zero nibble boundary, so that all lines start at >XXX0.

The command remembers all address values entered for use as defaults when the command is executed again. At reset, the start address defaults to the PC default value for the device type being emulated. The end address defaults to the start address plus >7F (decimal 127) bytes. When entering new address values, the end address will always default to the start address plus >7F bytes unless a stop address is entered. This provides a default display of 8 lines (i.e., subsequent entry of a start address only will default the command to the 8-line display). If both values are entered, both values are used for defaults for further execution of the command.

While the command is dumping to the terminal, the display can be stopped and started by keyboard input. Any key will stop the display at the end of the current line of output. After the display is stopped, <ESC> will abort the command, <SP> will cause display of successive lines one at a time for each <SP> entry, and any other key will restart the display.

EXAMPLES:

1) ?DM F806 F810<CR>
```
F800  31 44 45 6F  D9 E0 33 4A  55 44 55 8F  F9 D9 C1 AB
F810  44
```
2) ?DM F806 +2<CR>
```
F800  31 44 45 6F  D9 E0 33 4A  55
?
```

Memory is displayed up to >F808 (>F806 +2).

3) ?DM F806 F810 2
```
?
```

Dump memory to Port 2.

4) ?DM F810 F806<SP>
```
ADDRESS ERROR
?
```

No operation has been performed because the dump end address is less that the dump start address.

6.6.15 Display Processor Status (DP)

FORMAT: DP (or D)

PARAMETERS: None

Purpose: to display the processor status.

"D" is the abbreviated version of this command. The processor status is defined as:

Program Counter
Status Register (C, N, Z, and I bits)
Stack Pointer
Register A
Register B

The display is the same as the breakpoint line (see examples in Section 6.6.20).

EXAMPLE:

```
?DP
PC=F773  C=0 N=0 Z=0 I=0  SP=R20  A=00  B=04
```

6.6.16 Audio Tape Directory (DR)

FORMAT: DR {port **1**,2}

PARAMETERS: Output port (default=1)

Purpose: to display the files stored on a cassette tape to either Port 1 or Port 2.

Audio tape use is discussed in detail in Section 8. The display consists of filename, file type, and length in 512-byte blocks. The tape must be at the scan-start position, and the tape recorder set to PLAY before the command.

EXAMPLE:

```
?DR 1
```

```
TAPE DIRECTORY

A  SOURCE  05
A  7000    02
A  STATE   02
R  SOURCE  04
R  TEK     01<RESET>
?
```

The audio tape directory is listed to the terminal. Since cassette tape cannot support logical "END OF TAPE" (last file), a reset must be performed to exit the DR command.

6.6.17 Display Machine State (DS)

FORMAT: DS {port 1,2,3}

PARAMETERS: Output port (default=1)

Purpose: to dump the machine state for display to terminal or storage. The command dumps the following items:

Processor Status line (same as DP command)
Register File contents
Peripheral File contents

Register and Peripheral File displays are grouped 16 on a line and start with the decimal register or peripheral number and the hex address of the start of the line. This display is also the fixed display type 2 used by the EF and FS commands. The size of the Register File display is determined automatically by the size of the Register File in the TMS7000 on the EVM (either 128 or 256 registers).

While the command is dumping to the terminal, the display can be stopped and started by keyboard input. Any key will stop the display at the end of the current line of output. After the display is stopped, <ESC> will abort the command, <SP> will cause display of successive lines one at a time for each <SP> entry, and any other key will restart the display. The machine state can also be dumped to audio tape (Port 3) or uplinked to a file (Port 2) and restored (see LS command) for execution of a program without having to reinitialize registers.

EXAMPLE:

```
?DS 1
PC=0000  C=1 N=1 Z=1 I=1  SP=R1  A=00  B=00

R0  =0000  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
R16 =0010  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
R32 =0020  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
                    .                           .
                    .                           .
                    .                           .
R224=0060  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
R240=0070  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00

P0  =0100  A2 A2 00 00  21 21 02 F3  FF 00 FF 00  FF FF FF FF
P16 =0110  FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF
P32 =0110  FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF
```

6.6.18 Display Breakpoint on Trap (DT)

FORMAT: DT

PARAMETERS: None

Purpose: to display the current values of the breakpoint on trap locations.

Breakpoints on trap can be modified with the BT command only, and can be cleared with the CT command.

EXAMPLE:

```
?DT
1 3 5 9
?
```

The breakpoints are set on trap locations 1, 3, 5, and 9.

6.6.19 Select TMS7000 Family Device (DV)

FORMAT: DV

PARAMETERS: None

Purpose: to display/designate the device being emulated so that ROM size can be correctly configured for the device. Index numbers are 1 to 5:

1 702x, 70C2x
2 704x, 70C4x
3 Reserved
4 Reserved
5 Reserved

This command is also discussed in Section 2.11.

EXAMPLE:

```
?DV
DEVICE TYPE = 2 (704X) 1<CR,SP>
DEVICE TYPE = 1 (702X)
PC=F006  C=0 N=0 Z=0 I=0  SP=R1  A=00  B=00
?
```

6.6.20 Execute Program with Breakpoints (EF,ET,EX,GO)

Four breakpoint execution commands can be used:

- EF (Execute Breakpoint with Terminal Fixed Display)
- ET (Execute to Breakpoint with Trace)
- EX (Execute to Breakpoint)
- GO (Execute to Breakpoint; Optional Start Address, BP1 Value)

When using these commands, remember that the program will stop execution and display the breakpoint line **before** execution of the instruction at the breakpoint address. Subsequent EX commands using the existing breakpoints will **begin** execution at the instruction displayed.

Breakpoint commands also use a slow-speed event counter (EC) that functions with Breakpoint 1 (BP1) **only** (not BP2). If an EC value is not given, the default is one. At a breakpoint display, <SP> continues execution with the last EC value; entry of <CR> clears the EC value and returns control to the Monitor. Or, execution will begin at a value entered (1-9; any other key will cause an EC value of key ASCII value minus >30).

When the program executes with an EC value, the instruction at BP1 will be executed the number of times in the event counter before issuing the breakpoint. Note that to actually count the events, the program is suspended for approximately 20 milliseconds every time the breakpoint occurs, but execution then resumes and the cycle repeats until the event count expires.

Other breakpoints on address or trap can be encountered before the event count expires. As execution continues from breakpoint to breakpoint, the event count is displayed in decimal in parentheses at the point where input is expected. When the event count expires, the format of the breakpoint display is:

```
PC=xxxx  C=x N=x Z=x I=x  SP=Ryyy  A=xx  B=xx    EVC1   (count)
```

Note that "EVC1" replaces "BP1" and that the event count value is reloaded. The event count is a decimal number from 1 to 255. An entry of 0 means no event count and an entry of 1 will break on the first occurrence of BP1. If more than three numbers are entered, the last three before <CR> will be used as the event count. Entering a number greater than 255 will cause "ERROR".

Note:

For the EF, ET, and EX commands, the following are defined:

1) Breakpoint display:

```
NEXT INST---> 0060 D2       DECD    R20
PC=F010   C=1  N=1  Z=0  I=0   SP=R4   A=AA   B=EE       BP1
(17)
```

2) Breakpoint/trace display (example):

```
NEXT INST---> 0060 D2       DECD    R20
PC=F010  C=1 N=1 Z=0 I=0  SP=R4  A=AA  B=EE    BP1
P6=FF (11111111)     >0777=00 (00000000)   R4=C0 (11000000)
>EE00=F0 (11110000)   P33=FF (11111111)    P5=00 (00000000)
```

This is a combination of the breakpoint display and the trace display values set by the TC command.

Execute to Breakpoint with Terminal Fixed Display (EF).

FORMAT: EF <display type> [event count]<CR>

PARAMETERS: 1) Display type 1, 2, or 3 (default=1; see list below)
2) Optional event count (default=1)

Purpose: to execute a program to a breakpoint and show one of three updated fixed displays (1, 2, or 3; see list below) on the terminal screen:

Display

1 Breakpoint/Trace-Line -- trace as set by TC command (default)

2 Machine State (Breakpoint display, register display, Peripheral File display)

3 I/O Status (Breakpoint/trace-line display and Peripheral File display.

The Trace Line is that set by the TC command. When a breakpoint is encountered, the breakpoint display is dumped to the terminal. Entry of a <CR> returns control to the Monitor. Entry of a <SP> continues execution. Occurrence of the next breakpoint causes the new data to overwrite the old data, giving a stable, easy to read display. If more than one display-type number is entered, the last number entered is used as the display type.

> **Note:**
>
> The fixed display is produced by printing to the terminal a number of cursor-up characters equal to the number of line feeds in the display. This requires that the cursor-up character output by the EVM is recognizable by the terminal. The default value is listed in Section 2.15.3 and can be changed with the CU command.

Execute to Breakpoint with Trace (ET).

FORMAT: ET [event count]<CR,SP>

PARAMETERS: Optional event count (default=1) Purpose: to execute a program to a breakpoint and show an updated breakpoint/trace line.

This command is identical to the EF 1 command. If no trace line is set by the TC command, a blank line is printed after the breakpoint line.

Execute to Breakpoints (EX).

FORMAT: EX [event count]<CR,SP>

PARAMETERS: Optional event count (default=1) Purpose: to execute a program to a breakpoint and show an updated breakpoint display. The instruction at the breakpoint address can be executed the number of times in the optional event count before the breakpoint occurs.

> **Note:**
>
> The EVM software breakpoints are accomplished by placing 'TRAP 0' opcodes (>FF) at desired addresses, saving the user reset (TRAP 0) vector, and placing the Monitor breakpoint processing address in >FFFE and >FFFF. During normal in-circuit emulation operation, this is transparent to the user since the proper data is restored when the Monitor is running. But if the program is accessing itself as data, such as in a "CHECKSUM" operation, and a breakpoint is set, an >FF is read. This can be overcome by using the RU command, which executes without breakpoints.

If a breakpoint is encountered (either on address or call through trap vector), the program breaks **before** the code at the breakpoint address and returns control to the Monitor.

> **Note:**
>
> Breakpoint addresses must be:
>
> 1) greater than >00FF, and
> 2) set on an instruction boundary.

Two breakpoints on address and up to 24 breakpoints on trap are allowed, and execution will continue until either is encountered. When a breakpoint is encountered, the cursor will stay at the end of the display. Entering a <CR> will return control to the Monitor. Entering anything else will automatically execute to the next breakpoint. The format of the address breakpoint display is:

```
PC=xxxx  C=x N=x Z=x I=x   SP=Ryyy  A=xx  B=xx    BP1
```

if Breakpoint 1 is encountered (BP2 for Breakpoint 2).

The format of the breakpoint on trap display is:

```
PC=xxxx  C=x N=x Z=x I=x   SP=Ryyy  A=xx  B=xx    TRAP nn
```

where "nn" is the trap location encountered. The current PC, ST, SP, A, and B registers are printed.

EXAMPLES:

1) Before execution, PC = >F64E.

   ?EX <CR>

   Begins execution at location >F64E. No breakpoint is defined so execution continues. Program execution can be terminated and the Monitor re-entered by pushing the RESET switch (Section 6.11).

2) Before execution, PC = >F64E and BP1 = >F681.

   ?EX <CR>

   ```
   PC=F681  C=0 N=0 Z=0 I=1   SP=R20  A=00  B=05    BP1  <SP>
   PC=F681  C=0 N=1 Z=0 I=1   SP=R20  A=14  B=12    BP1  <CR>
   ?
   ```

   Execution begins at location >F64E. When location >F681 is encountered, program execution is halted, and the PC, ST, and SP are displayed. A space is entered, causing execution to begin again. The same breakpoint is encountered again, and the Monitor is entered.

3) ?EX 11<CR>

   ```
   PC=F681  C=1 N=0 Z=0 I=0   SP=R23  A=22  B=01    EC1 (11)<SP>
   PC=F681  C=1 N=0 Z=0 I=0   SP=R23  A=44  B=31    EC1 (11)<CR>
   ?
   ```

   The program is executed to BP1 with an event count of 11. The process was repeated with a <SP>. After the second occurrence of BP1, <CR> terminated execute mode.

Go Execute at Address (GO).

FORMAT: GO [start execution address] [BP1 event count]

PARAMETERS: 1) Memory address to start execution (default=PC value)
2) Optional event count (default=1) Purpose: Execute to breakpoint with trace with starting address specified.

This command is identical to ET with the added options of specifying an execution start address (instead of the default PC address). As with EF, ET, and EX, an optional even count can be specified for BP1.

EXAMPLES:

1) ?GO @LOOP<CR>

Begin execution at the address labeled LOOP. Event count before breakpoint is the default 1.

2) ?GO F336 10<CR>

Begin execution at address >F336, execute a breakpoint display after 10 occurrances of BP1.

6.6.21 Find Byte in Memory (FB)

FORMAT: FB <start addr> <stop addr> <value> [mask]<CR,SP>

PARAMETERS:
1) Memory start address
2) Memory stop address
3) Value in hex to be searched for (default=>FF)
4) Optional mask value to specify which bits to check (default=>FF)

Purpose: to find the occurrences of the byte specified in the memory limits specified.

Following are examples of the FB command:

```
FB ADDR1 ADDR2<CR>          finds >FF
FB ADDR1 ADDR2<SP><CR>      finds >00
FB ADDR1 ADDR2 0A<CR,SP>    finds >0A
FB ADDR1 ADDR2 'A'<CR,SP>   finds the ASCII value of "A" (>41)
FB ADDR1 ADDR2 F5 3         finds binary 01 (1s in value AND mask)
```

When entering a hexadecimal value, only the last two characters entered before the <CR,SP> will be used. An ASCII character can be entered for <value>, by enclosing the character in single quotes.

When the value is found, the address and value are printed in a format similar to the MM command. At this point, the value can be changed. The following can be entered:

<SP> or +	displays successive locations for access.
-	displays previous locations for access.
<	redisplays the same location.
<CR>	restarts the search.

<new data>Q enters new data, then displays it

<ESC> exits search, returns to Monitor.

Any occurrence of one of the boundary addresses will cause the command to terminate and return control to the Monitor. Data changed on the same line can be redisplayed by finishing the entry with a "Q." <ESC> aborts the FB command and data entered on the same line is not retained.

EXAMPLES:

```
1)  ?FB F006 FFFF 83<CR>
      F0E1=83 84<CR>
      F682=83<CR>
      FAB3=83 84<SP>
      FAB4=7F 80Q
      FAB4=80<CR>
    ?
```

The "ANDP" opcode (>83) was found, and the first and third occurrences were changed to "ORP" (>84). Location >FAB4 was changed from >7F to >80. No >83 values were found after the last <CR>.

```
2)  ?FB $  +>300  83<CR>
```

This instruction searches for the value >83 from the PC value to PC + >300.

```
3)  ?FB F800 F900 B6 83<CR>
    F810 = C2 (11000010)
    F812 = D3 (11010011)<ESC>
    ?
```

The value (B7) is first ANDed with the mask (82). The result is then the mask to match with bytes being searched. For example:

```
B6 =  1011 0110   (byte value in command)
83 =  1000 0011   (mask in command)
                  (AND above)
      1000 0010   (resulting mask to compare with bytes in memory)
```

The last line shows the mask to identify bytes with desired value. The command will identify bytes that have ones corresponding to the ones in the final mask. For example, the following bytes would be selected using the above mask:

```
1000 1110   (ones correspond to ones in mask above)
1011 1111
1100 1011
1110 0111
1111 1110
```

6.6.22 Fill Memory (FM)

FORMAT: FM <start addr> <stop addr> <value><CR,SP>

PARAMETERS:
1) Memory start address
2) Memory stop address
3) Fill value (default=>FF)

Purpose: to fill RAM or a part of RAM with a specified value.

Following are examples of the FM command:

```
FM ADDR1 ADDR2<CR>          fills RAM with >FF
FM ADDR1 ADDR2<SP><CR>      fills RAM with 00
FM ADDR1 ADDR2 0A<CR,SP>    fills RAM with >0A
FM ADDR1 ADDR2 'A'<CR,SP>   fills RAM with the ASCII value of "A" (>41)
```

When entering a hexadecimal value, only the last two characters entered before the <CR,SP> will be used. Attempting to fill RAM below the PC default address will cause an error. For device types 3, 4, and 5, this command resets flags to indicate no text in the Text Editor.

EXAMPLES:

1)
```
?FM F0FF F006
ADDRESS ERROR
?
```

Error occurred because the start address is greater than the end address.

2)
```
?FM F006 F0FF 'B'
?
```

Locations >F006 to >F0FF are filled with the ASCII value of "B" (>42).

3)
```
?FM F006 F0FF FFDD<CR>
?
```

Locations >F006 to >F0FF are filled with >DD.

6.6.23 Fill Register File (FR)

FORMAT: FR <start reg> <stop reg> <value><CR,SP>

PARAMETERS:
1) Start register
2) Stop register
3) Fill character (default=>FF)

Purpose: to fill the portion of the Register File with the value specified.

This command is similar to the FM command. Following are examples of the FR command:

```
FR REG1 REG2<CR>            fills with >FF
FR REG1 REG2<SP><CR>        fills with 00
FR REG1 REG2 0A<CR,SP>      fills with value >0A
FR REG1 REG2 'A'<CR,SP>     fills with ASCII value of "A"
```

When entering a hexadecimal value, only the last two characters entered before the <CR,SP> will be used. After execution, control returns to the Monitor. Do not enter an "R" in front of the register value. Entering a register value in excess of 127 or 255 (depending on the member of the TMS7000 family on the EVM) will cause an error. Entering a start register value greater than the stop register value will cause ADDRESS ERROR. During entry of the register value, the last three digits entered will be used. If less than three digits are entered, leading zeros are assumed.

EXAMPLE:

```
?FR 0 127 FF
?
```

The Register File is filled with >FF.

6.6.24 Single-Step Program with Fixed Display (FS)

FORMAT: FS <display type> <step count><CR,SP>

PARAMETERS: 1) Display type 1, 2, or 3 (default=1)
2) Number of fixed steps from 1 to 255 (default=1)

Purpose: to single-step the program and display one of three updated fixed displays.

The three types of fixed displays:

Type

1) Single-step breakpoint display.

```
?FS <CR>
LAST INST---> F010          DECD   R20
CYCLE COUNT = 000148
NEXT INST---> F012          JC     >F010
PC=F012  C=1 N=0 Z=0 I=0  SP=R1  A=F0  B=13
```

2) Single-step breakpoint display, Register File and Peripheral File contents.

3) Single-step breakpoint display and Peripheral File contents.

If more than one display type number is entered, the last number entered is used. The number of fixed steps entered is the "step count". The step count parameter is the decimal number from 1 to 255 of instruction steps to be executed before stopping. After each step, the display selected is updated and dumped to the terminal.

After the initial step count has expired, the following can be entered:

<SP>	executes one step
<CR>	returns control to the Monitor
1-9	executes number of steps entered
A-Z	executes steps equal to letter ASCII value minus >30 (e.g., "A" executes >42->30 or >12 steps) -- ASCII values less than >30 execute one step
0 (zero)	executes one step after setting cycle counter to 000000
*	asterisk means execute continuously until a key is pressed
*ADDR	executes until PC = ADDR or until a key is pressed

The Monitor command CY clears the cycle count. If the cycle count exceeds 999999, it will "roll over" (i.e., increment to 000000 and continue from there).

The fixed display is produced by printing to the terminal a number of cursor-up characters equal to the number of line feeds in the display. This requires that the cursor-up character output by the EVM is recognizable by the terminal. The default value is listed in Section 2.15.3 and can be changed with the CU command.

6.6.25 Hex-Decimal Word Conversion (HC)

FORMAT: HC <1-4 hex digits><CR,SP>

PARAMETERS: Up to four hex digits for conversion

Purpose: to convert a hex number to decimal.

If more than four numbers are entered, the last four are used. If less than four numbers are entered, leading zeros are assumed. Legal entries are 0-9 and A-F.

EXAMPLE:

```
?HC 345FFFF<CR>
>FFFF=65535
?
```

>FFFF is displayed in decimal.

6.6.26 Help (HE)

FORMAT: HE {**M**,E} {port **1**,2}

PARAMETERS: 1) Monitor (M) or Text Editor (E) commands (default=M)
2) Output port (default=1)

Purpose: to list the Monitor commands or the Text Editor commands.

If no parameter or the "M" parameter is entered, the Monitor commands are displayed on the terminal. If the first parameter is "E", the Text Editor commands are listed. The default output port is 1 (terminal); Port 2 (printer) can also be specified.

EXAMPLE:

```
?HE M 2
?
```

The Monitor commands are listed to Port 2 (printer).

6.6.27 Display I/O Status (IO)

FORMAT: IO {port **1**,2}

PARAMETERS Output port (default=1)

Purpose: to display the Peripheral File locations from P0 to P31 in hex and binary along with the processor status line of the SP command and trace parameters of the TC command.

The output port can be either 1 (terminal) or 2 (printer). The Peripheral File is displayed in two columns, the contents displayed in hex followed by binary in

parentheses. This display is also the fixed display type 3 used by the EF and FS commands.

While the command is dumping to the terminal, the display can be stopped and started by keyboard input. Any key will stop the display at the end of the current line of output. After the display is stopped, <ESC> will abort the command, <SP> will cause display of the next lines, and any other key will restart the display.

EXAMPLE:

```
?IO <CR>
PC=F7F1  C=1 N=1 Z=0 I=0  SP=R44  A=21  B=63
R4=17   R5=81   R6=72   R7=73

P0 =2A (00101010)     P16=FF (11111111)     P32=FF (11111111)
   .                     .                     .
   .                     .                     .
   .                     .                     .
P15=FF (11111111)     P31=FF (11111111)     P47=FF (11111111)
```

6.6.28 Show Address of Line (LA)

FORMAT: LA <line number><CR,SP>

PARAMETERS: Text Editor line number

Purpose: to display the address containing the opcode(s) assembled from a line in the Text Editor.

When a file is assembled from the Text Editor, the address at which the opcodes are stored is remembered. After assembly, this command displays the address for possible use with the MM command to change a value in program memory associated with a line in the Text Editor without having to reassemble. A comment line always has address "0000". New lines in the Text Editor have address "0000" prior to assembly. If the line number entered does not exist, no address is output.

EXAMPLE:

```
?LA 450<CR> F882
?
```

The first byte assembled for line 450 is stored at >F882.

6.6.29 List Lines from Text Editor (LL)

FORMAT: LL <line number> <line count><CR,SP>

PARAMETERS: 1) Starting line number (default=first line)
2) Number of lines to list (default=1)

Purpose: to display selected source lines from the Text Editor.

All entries are decimal, from 1 to 9999 with leading zeroes assumed. If more than four digits are entered, the last four entered are used. If the line number entered does not exist, the next existing line starts the display. The count parameter is the number of lines to be displayed and defaults to one. Entering zero causes one line to be displayed.

If this command is entered and the Text Editor is empty, no display occurs.

While the command is dumping to the terminal, the display can be stopped and started by keyboard input. Any key will stop the display at the end of the current line of output. After the display is stopped, <ESC> will abort the command, <SP> will cause display of successive lines one at a time for each <SP> entry, and any other key will restart the display.

EXAMPLE:

```
?LL 40 3<CR>
0040            CLR SUM2
0050            CALL @OUT
0060            INV SUM2
?
```

Three lines from the program in the Text Editor are listed, starting with line 40.

6.6.30 Load Memory - 7000 Format (LM)

FORMAT: LM {port 1,2,**3**}

PARAMETERS: Input port (default=3)

Purpose: to load memory from Port 1, 2, or 3. Data to be loaded is in 7000 (or 9900) object format. Object code RAM will be initialized to >FF.

The object file must be at load module (absolute) level (see Section 6.8). An object file at any other level may produce an error during loading operations. If an error occurs, the LM command sends a message to the EIA port that is not the source of the input; the message includes the approximate location of the error. This command can be aborted with a reset.

Since the LM command, when executed with input from Port 1 or 2, loads data through the I/O buffer, the DATA LED will go on when the buffer is being filled and go off when the buffer is being emptied into RAM. This command waits until input data is detected before initializing object code RAM to >FF.

EXAMPLES:

1)
```
?LM<CR>
FILENAME: A<CR>
?
```

After execution, memory has been loaded with a 7000 or 9900 object file from file "A" on cassette tape (Port 3).

2)
```
?LM 2
CHECKSUM ERROR  F901
?
```

A checksum error was found in the object file. The operation does not abort at the point of the error.

6.6.31 Show Editor Line at Address (LN)

FORMAT: LN <addr><CR,SP>

PARAMETERS: Memory address

Purpose: to display the complete source line that produced the opcode found at the specified address.

Functionally the opposite of the LA command, the LN command displays the Text Editor line associated with a known memory address. If the address is not at an instruction boundary, no equivalent Text Editor line will be found.

This command is useful when the contents of the stack are examined after a breakpoint on trap or interrupt (BT) because the break occurs after the service routine is entered. The old PC LSB is at the SP location and the old PC MSB is at the SP-1 location. Since the old PC value is placed on the stack, executing the LN command in the Standalone mode displays the line from the Text Editor that was executed prior to the breakpoint. Breakpoint displays that include stack locations are enabled with the TC command and fixed display commands (EF and FS).

EXAMPLE:

```
?LN F806<CR>
0040          CLR SUMREG
?
```

The equivalent Text Editor line is displayed for the opcode at address >F806.

6.6.32 Load Machine State (LS)

FORMAT: LS {port 1,2,**3**}

PARAMETERS: Input port (default=3)

Purpose: to restore a machine state produced by the execution of a program and stored with the DS command.

This command restores the items listed in the display format in the DS command description (PC, ST, SP, Register File, and Peripheral File values).

Since the LS command, when executed with input from Ports 1 and 2, loads data through the I/O buffer, the DATA LED will go on when the buffer is being filled and go off when the buffer is being emptied into RAM.

The LS command does not care whether the machine state file being loaded has 128 or 256 register locations. Regardless of which member of the TMS7000 family is on the EVM, any size machine state file will load without error.

EXAMPLES:

1)
```
?LS<CR>
FILENAME: A<CR>
?
```

After execution, the machine state is restored to the contents found in file "A" loaded from Port 3 (tape). Execution of "DS 1" will display results.

2) ?LS 2
INPUT ERROR
?

During the load operation, an unrecognizable character was loaded when a hex value was expected. The command is aborted.

6.6.33 Load Memory - Tektronix Format (LT)

FORMAT: LT {port 1,2,**3**}

PARAMETERS: Input port (default=3)

Purpose: to load memory from Port 1, 2, or 3. Data to be loaded is in Tektronix object format. Object code RAM will be initialized to >FF.

The object file must be at load module (absolute) level. An object file at any other level may produce an error during loading operations. If an error occurs, the LT command sends a message to the EIA port that is not the source of the input; the message includes the approximate location of the error. Since the LT command, when executed with input from Port 1 or 2, loads data through the I/O buffer, the DATA LED will go on when the buffer is being filled and go off when the buffer is being emptied into RAM. This command waits until input data is detected before initializing object code RAM to >FF.

EXAMPLES:

1) ?LT<CR>
FILENAME: A<CR>
?

After execution, memory has been loaded with Tektronix object file from file "A" on cassette tape (Port 3).

2) ?LT 2
CHECKSUM ERROR F901
?

A checksum error was found in the object file. The operation does not abort at the point of the error.

6.6.34 Set Breakpoint 1 or 2 by Editor Line Number (L1/L2)

FORMATS: L1 <line number><CR,SP>
L2 <line number><CR,SP>

PARAMETERS: Text Editor line number

Purpose: to set Breakpoint 1 or 2 on address using a Text Editor line number.

If no line number is entered, the breakpoint value remains unchanged. The last four decimal digits entered are used for the line number. Breakpoint 1 is used with the event counter in the EX command. If the breakpoint is changed, the DB (Display Breakpoints) command is executed. The line number must exist in the Text Editor and the file must have been assembled for an address to be associated with the line.

EXAMPLES:

1) ?L1 450<CR>
BP1=F882 BP2=F903
?

Breakpoint 1 is set on line 450 (>F882).

2) ?L2 620<CR>
BP1=F882 BP2=F977
?

Breakpoint 2 is set on line 620 (>F977).

6.6.35 Display/Modify Registers A and B (MA and MB)

FORMAT: MA and MB(or A and B)

PARAMETERS: None

Purpose: to display the contents of Register A and/or Register B and allow modification of their contents.

Command inputs MA and MB can be abbreviated as A or B.

When the MA command is entered, the contents of Register A are displayed. A new value can be entered, or a <SP> displays the contents of Register B. Entering MB will display the contents of Register B. The contents of A can be modified by keying in the desired hexadecimal value. Pressing <SP> will continue to B and pressing <CR> will terminate the command and return control to the Monitor. To leave the register contents intact, press <SP> or <CR> after the register contents have been displayed. Registers A and B can also be modified as R0 and R1 with the MR command.

For either register, entry of "<" will redisplay the value just entered. New hex digits entered on the line that a command termination takes place will be retained if a <CR> or <SP> is pressed but will not be retained if <ESC> is entered. INPUT ERROR caused by entry of an illegal character is non-fatal, redisplaying the line the error occurred on for update. After display/modify of Register B, the command continues to the MS command.

EXAMPLES:

1) Before execution, A = >34 and B = >F6.

```
?MA
A=34 (00110100) 123456<SP>
B=F6 (11110110)<CR>
?
```

After execution, A = >56 and B = >F6.

2) Before execution, A = >34 and B = >F6.

```
?B
B=F6 (11110110) 9<CR>
?
```

After execution, A = >34 and B = >09.

6.6.36 Display/Modify Memory (MM)

FORMAT: MM <start addr><CR,SP>

PARAMETERS: Memory address (at reset, default=PC default; otherwise, default=previous entry)

Purpose: to display and allow modification of program memory. An optional memory start location may be specified for initial displacement.

The memory address defaults after reset to the PC default value for the current device type. Addresses must be in the legal memory display/modify range for the device type. The new address becomes the default for the next command.

Once displayed, memory address contents may be modified by entering valid hexadecimal digits. Only the last two hex digits entered will be retained.

The command line terminator controls the continuation of the command, whether or not data is entered. The command continues to the next location if <SP> or a "+" is pressed or goes to the previous location if "-" is entered. In either case, the "+" or "-" is displayed to denote direction. Entry of "<" will cause the value just entered to be redisplayed. The command is terminated if <CR> is entered. The last hex digits entered will be retained if <CR> is pressed, but will not be retained if <ESC> is pressed. Following address >FFFF, the MM command will terminate. INPUT ERROR caused by an illegal character is non-fatal, redisplaying the line for update.

EXAMPLES:

1) Before execution, location >FF06 contains value >56, >FF07 contains >45, and >FF08 contains >A5. Last MM command was to address >FF06.

```
?MM<CR>
+ FF06=56 (01010110) <SP>
+ FF07=45 (01000101) 5676<SP>
  FF08=A5 (10100101) 6<CR>
?
```

After execution, location >FF06 contains >56, >FF07 contains >76, and >FF08 contains >06. Default start address after reset remains >FF06.

2) Before execution, >FFFE contains >F8 and >FFFF contains >06.

```
?MM FFFE<CR>
+ FFFE=F8 (11111000) F9F0<SP>
+ FFFF=06 (00000110)<SP>
?
```

After execution, >FFFE contains >F0 and >FFFF contains >06. This is the reset vector. >FFFE is the new default start address.

3) Before execution, >0200 contains >AA.

```
?MM 200<SP>
ADDRESS ERROR
?
```

No operation is performed since the requested memory location is outside the legal memory display/modify range.

6.6.37 Audio Tape Motor On (MO)

FORMAT: MO

PARAMETERS: None

Purpose: to allow using cassette tape motor control keys while the cable is inserted between J3 and REM input.

With the cassette motor under program control, this command turns on the motor to allow use of the recorder keys (rewind, fast forward, etc.). Execution continues until any subsequent terminal keyboard input, which stops the motor and resumes Monitor execution. This command is discussed in Section 8.

EXAMPLE:

```
?MO<SP>
?
```

The tape motor was enabled with the MO command, then turned off by hitting <SP>.

6.6.38 Display/Modify Peripheral File (MP)

FORMAT: MP <peripheral register no.><CR,SP>

PARAMETERS: Peripheral register number (at reset, default=0; otherwise, default=previous entry)

Purpose: to display and allow modification of peripheral registers P0 to P47 inclusive. The peripheral number is the start location.

The start location defaults to P0 at reset. If a location is specified, it must be in the legal file display/modify range. The last location input is the default for the next command. Entry of "P" is optional. The entry can be in hexadecimal if preceded by ">".

The entry terminator controls the command continuation. whether or not data is entered. The command continues to the next location if <SP> or "+" is pressed or goes to the previous location if "-" is entered. In either case, the "+" or "-" is displayed to denote direction. Entry of "<" will cause the value just entered to be redisplayed. <CR> terminates the command.

The command terminates following display/modify of the last location. New hex digits entered will be retained if <CR> is pressed but will not be retained if <ESC> is entered. INPUT ERROR caused by entry of an illegal character is non-fatal, redisplaying the line for update.

EXAMPLES:

1) Before execution, location P8 contains value >09, and P10 contains >BC.

```
?MP 8<CR>
+ P8=09 (00001001) 5432<SP>
+ P9=FF (11111111) <SP>
  P10=BC (10111100) <CR>
?
```

After execution, P8 contains >32 and P10 contains >BC. P8 is now the default start location.

2) ?MP 249<CR>
ADDRESS ERROR
?

No operation is performed since the requested memory location is outside the legal Peripheral File display/modify range.

6.6.39 Display/Modify Register File (MR)

FORMAT: MR <register><CR,SP>

PARAMETERS: Register number (at reset, default=0; otherwise, default=previous entry)

Purpose: to display and allow modification of internal registers in the range R0 to R127 (R255 for TMS7042) inclusive. Start location is specified by entering a register number.

The start location defaults after reset to R0 and afterward to the last address entered. Locations must be a decimal number in the legal Register File display/modify range (0 to 127 or 255 inclusive) (note that the contents of Register A (R0) and Register B (R1) can be read and modified using the MA, MB, or MR command. Entry of the "R" is optional. The entry can be in hexadecimal if preceded by ">". At reset, the EVM sets the upper limit of the MR command to either R127 or R255.

The command continues to the next location if <SP> or "+" is pressed or goes to the previous location if "-" is entered. Entry of a "<" will cause the value just entered to be redisplayed. The command is terminated if a <CR> is entered. INPUT ERROR caused by entry of an illegal character is non-fatal, redisplaying the line the error occurred on for update.

EXAMPLES:

1) Before execution, R0 contains >56, R1 contains >45, and R2 contains >A5.

```
?MR<CR>
+ R0=56 (01010110) <SP>
+ R1=45 (01000101) 5676<SP>
  R2=A5 (10100101) 6<CR>
?
```

After execution, location R0 contains value >56, R1 contains >76, and R2 contains >06. Default start location after reset is R0.

2) ?MR 88<CR>
+ R88=09 5432<SP>
R89=BC <CR>
?

After execution, R88 contains >32 and R89 contains >BC. R88 is now the default start location.

3) ?MR 256<SP>
ADDRESS ERROR
?

No operation is performed since the requested memory location is outside the legal Register File display/modify range.

6.6.40 Display/Modify PC, SP, RA and RB (MS, PC, SP, A, B)

FORMAT: MS or PC (or P) or SP or A or B

PARAMETERS: None

Purpose: to display and allow modification of the Program Counter, Stack Pointer, and Registers A and B.

The registers are displayed in the order of PC, SP, RA, and RB (program counter, stack pointer, and Registers A and B). The next register is displayed when <SP> is entered. Displayed values can be changed to a value within the legal range for the register. INPUT ERROR from entry of an illegal value will result in redisplay of the register. A "<" following the change value will redisplay the changed contents. Values will be used for upcoming program execution. Entry of a <CR> or <ESC> will terminate the display and return to the Monitor.

EXAMPLES:

1) Before execution, PC = >F773 and SP = R50.

```
?MS
PC=F773 F123<SP>
SP=R50 20<CR>
?
```

After execution, PC = >F123 and SP = R20.

2) Before execution, PC = >F773 and SP = R50.

```
?MS
PC=F773 <SP>
?
```

After execution, PC = >F773 and SP = R50.

6.6.41 Move (Copy) Memory (MV)

FORMAT: MV <start addr> <stop addr> <dest addr><CR,SP>

PARAMETERS:
1) Move source start address
2) Move source stop address
3) Move destination start address

Purpose: to move (copy) a block of memory. This command will not read from the EPROM programming sockets.

Address parameters are limited to the range from the PC default to >FFFF. The source locations are not changed except where they are overwritten by the copied data.

EXAMPLES:

1)
```
?MV F076 F876 F006<CR>
?
```

After execution, locations >F006 to >F806 inclusive contain a copy of what was contained in locations >F076 to >F876 inclusive.

2) ?MV F006 F806 F076<SP>
?

After execution, locations >F076 to >F876 inclusive contain a copy of what was contained in locations >F006 to >F806 inclusive.

3) ?MV C000 C400<CR>
ADDRESS ERROR
?

No operation was performed because the destination start address is missing.

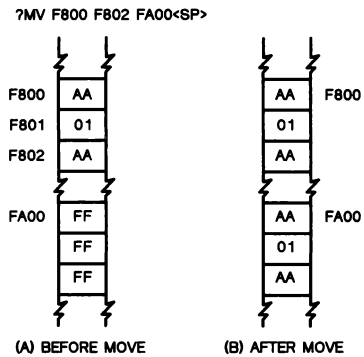


Figure 6-1. Block Memory Move

6.6.42 Fill Memory with NOPs (NP)

FORMAT: NP

PARAMETERS: None

Purpose: to fill program memory with the NOP opcode (>00).

The program memory range from default PC to >FFFF is determined by the current device type. Table 2-5 lists default PCs for each device. The NOP opcode is >00. For device types 3 to 5, this command resets flags in the Text Editor, indicating no text in the editor.

6.6.43 Reset Target Processor (RT)

FORMAT: RT

PARAMETERS: None

Purpose: to reset the emulated processor.

The RT command simulates a target reset. Due to differences in the peripheral file map of TMS7000 family members, the command requests that the EVM RESET switch be toggled to ensure proper initialization of the TMS7000. It then prints out values of the PC, ST, SR, RA, and RB. Results of initialization are:

LOCATION	VALUE
PC	contents of user's TRAP 0 vector location, >FFFE & >FFFF
ST	>00
SP	R1 (Register B)
A	PC MSB
B	PC LSB
Port A	inputs (P5 = 00)
Port B	>FF
Port C	inputs (P9 = 00)
Port D	inputs (P11= 00)

EXAMPLE:

```
?RT
PRESS RESET<RESET>
PC=F006  N=0 R=0 Z=0 I=0  ST=R1  RA=F8  RB=00
?
```

After the RESET switch is toggled, the processor status is displayed, the PC contains the RESET vector taken from >FFFE and >FFFF, and the A and B registers contain the MSB and LSB of the previous PC value. Note that the PC value taken from >FFFE and >FFFF must agree with the processor device type PC value or ADDRESS ERROR will be given. ADDRESS ERROR will be displayed after setting SP to R1, and RA and RB to the MSB and LSB of the current PC value.

6.6.44 Execute Program without Breakpoints (RU)

FORMAT: RU

PARAMETERS: None

Purpose: to execute a program without breakpoints, without having to clear all currently set breakpoints. Execution begins at the current PC location.

The only way to terminate the RU command is by reset. This will yield the message:

```
STOP AT PC=xxxx
```

where "xxxx" is the interrupted Program Counter. If no breakpoints of any kind are set, the EX command will function much like the RU command with one exception: the RU command does not swap the Monitor TRAP 0 for the user TRAP 0, allowing a program to emulate software reset in real time (see Section 6.6.20 for further explanation of the EX command).

EXAMPLE:

```
?RU
<RESET>
STOP AT PC=F448
?
```

The RESET switch terminated the RU command.

6.6.45 Save Memory - 7000 Format (SM)

FORMAT: SM <start addr> <stop addr> {port 1,2,**3**}

PARAMETERS:
1) Memory start address
2) Memory stop address
3) Output port (default=3)

Purpose: to dump memory to Port 1, 2, or 3. Data will be dumped in 7000 object format.

The object file must be at load module (absolute) level as outlined in load/dump formats in Section 6.8.

EXAMPLES:

1)
```
?SM F806 FFFF<CR>
FILENAME: A<CR>
READY TO RECORD?<CR>
?
```

Memory at locations >F806 to >FFFF inclusive was dumped to tape to filename "A" in 7000 object file format.

2)
```
?SM F806 FFFF 2
?
```

These locations are dumped to Port 2.

3)
```
?SM FFFF F806<SP>
ADDRESS ERROR
?
```

No operation has been performed because the dump end address is less than the dump start address.

6.6.46 Status Register Display/Change Commands

Display Status Register (SR).

FORMAT: SR

PARAMETERS: None

This command displays the status register contents.

EXAMPLE:

```
?SR
C=0 N=0 Z=0 I=0
?
```

Set/Reset Status Register Bits. Individual or all Status Register bits can be set (to one) or cleared using the following ten commands:

	Command	Results
EI	Enable (set) SR bits	C=1 N=1 Z=1 I=1
		(If all bits=1)
CC	Clear Carry (C) bit	C=0 N=1 Z=1 I=1
CN	Clear Negative (N) bit	C=1 N=0 Z=1 I=1
CZ	Clear Zero (Z) bit	C=1 N=1 Z=0 I=1
CI	Clear Interrupt (I) bit	C=1 N=1 Z=1 I=0
DI	Disable (clear) SR bits	C=0 N=0 Z=0 I=0
		(If all bits=0)
SC	Set Carry (C) bit	C=1 N=0 Z=0 I=0
SN	Set Negative (N) bit	C=0 N=1 Z=0 I=0
SZ	Set Zero (Z) bit	C=0 N=0 Z=1 I=0
SI	Set Interrupt (I) bit	C=0 N=0 Z=0 I=1

6.6.47 Single-Step Program (SS)

FORMAT: SS <count> {port **1**,2}

PARAMETERS: 1) Number of single-steps (1 to 155; default=1)
2) Output port (default=1)

Purpose: to single step a program from the initial conditions specified by the PC, ST, and SP registers and give the single-step display for each step.

For each count, one program step is executed beginning at the PC value. A count value of 0 (zero), 1, or no entry (default) specifies one step. As each step (up to 255 max.) is executed, the display (example below) shows the contents of PC, SR (status register), and registers A and B as well as instruction executed, cycle count, next instruction, parameters selected by the TC command (single-step trace), and next-step number.

After the initial step count has expired, the following can be entered:

<SP>	executes one step
<CR>	returns control to the Monitor
1-9	executes number of steps entered
A-Z	executes steps equal to letter ASCII value minus >30 (e.g., "A" executes >42->30 or >12 steps) -- ASCII values less than >30 execute one step
0 (zero)	executes one step after setting cycle counter to 000000
*	asterisk means execute until a key is pressed
*ADDR	executes until PC = ADDR or until a key is pressed

The Monitor command CY clears the cycle count. If the cycle count exceeds 999999, it will "roll over" (i.e., increment to 000000 and continue incrementing).

The display can be sent to Port 1 or 2. If to Port 2 (printer), the count must be 3 or higher.

EXAMPLE:

One step requested; then three more asked:

```
?SS<CR>
LAST INST---> F009          DECD   R20
CYCLE COUNT = 000519
NEXT INST---> F00B          JC     >F009
PC=F00B  C=1 N=1 Z=0 I=0  SP=R1  A=F0  B=0C    2<CR>
CYCLE COUNT = 000526
NEXT INST---> F009          DECD   R20
PC=F009  C=1 N=1 Z=0 I=0  SP=R1  A=F0  B=0C    (1)
CYCLE COUNT = 000537
NEXT INST---> F00B          JC     >F009
PC=F00B  C=1 N=1 Z=0 I=0  SP=R1  A=F0  B=0C    <SP>
CYCLE COUNT = 000544
NEXT INST---> F009          DECD   R20
PC=F009  C=1 N=1 Z=0 I=0  SP=R1  A=F0  B=0C    <CR>
?
```

Four single-steps were taken. The first was from the SS command. Then the number "2" was entered, executing two more single-steps. Then <SP> caused the fourth to be taken. Finally <CR> returned control to the Monitor.

6.6.48 Save Memory - Tektronix Format (ST)

FORMAT: ST <start addr> <stop addr> {port 1,2,**3**}

PARAMETERS:
1) Memory start address
2) Memory stop address
3) Output port (default=3)

Purpose: to dump memory to Port 1, 2, or 3. Data will be dumped in Tektronix object format.

The object file must be at load module (absolute) level (see Section 6.8).

EXAMPLES:

1)
```
?ST F806 FFFF<SP>
FILENAME: A<CR>
READY TO RECORD?<SP>
?
```

Memory at locations >F806 to >FFFF inclusive are dumped to file "A" on cassette tape in Tektronix object file format.

2)
```
?ST F806 FFFF 2
?
```

Locations are dumped to Port 2.

3)
```
?ST F8FF F500<CR>
ADDRESS ERROR
?
```

No operation occurs because the dump end address is less than the dump start address.

4) ?ST F806<CR>
ADDRESS ERROR
?

No operation occurs because the dump end address is missing.

6.6.49 Instruction Trace Execution

The EVM provides two means of tracing program execution. One is by entering the TC command (Section 6.6.50) so that one-to-six different values (registers, memory addresses) are printed when one of the following program/single-step commands are executed:

EF	Execute to breakpoint with trace and fixed display
ET	Execute to breakpoint with trace
FS	Single step with fixed display
TS	Single step with trace

In addition to that above, another form of trace can be employed that accumulates a record of the source statements executed. Besides working with the above four commands, this trace also works with:

EX	Execute to breakpoint, and
SS	Single step execution

Using the latter form of trace with one of these six commands, execution continues until the breakpoint is reached or trace memory is filled (with source-statement data). If the breakpoint is reached, the breakpoint display is issued along with the amount of instructions executed in a "SAMPLE COUNT = " message. If memory containing the trace steps is exceeded before reaching the breakpoint, a "TRACE MEMORY FULL" message is issued and execution stops.

In order to use this accummulated trace sample, the following commands are used. These are described in the following paragraphs.

		Page
IS	Print count of trace samples (qty. of instructions)	6-40
IT	Inspect trace (view instructions executed)	6-40
PT	Print trace memory contents to port 1 or 2	6-41
TF	Turn off trace	6-41
TO	Turn on trace	6-41

Capacity to accumulate sample counts is limited by the RAM in socket U38:

In U38,	
2K static RAM	500 trace-sample capacity
8K static RAM	3500 trace-sample capacity

If it is necessary to modify U38 memory, observe the following:

U38 RAM Size	Set Jumper P3
2K RAM	P3 B-C
8K RAM	P3 A-B

Inspect Trace Sample Counts Accummulated (IS).

FORMAT: IS

PARAMETERS: None

Purpose: to display the hexadecimal amount of sample counts (instructions) executed under trace.

The hexadecimal number of sample counts taken during trace is displayed. This command is automatically executed (i.e., message displayed) when trace is on and a breakpoint reached or trace memory is full.

EXAMPLE:

```
?IS
SAMPLE COUNT = >00C                    (12 samples recorded)
?
```

Inspect Trace Samples (IT).

FORMAT: IT <trace number or prgm. memory addr.> <CR,SP>

PARAMETERS: Trace sample number (hex) to start display (default = 000), or
Program memory address to start display specified by:
- hexadecimal memory address, or
- label preceded by @ Purpose: To display up to 22 trace samples of executed source lines.

EXAMPLE:

```
?IT @D1                                (start trace at label D1)
0AA      S    PC=FAE0  D1      DECD   R20
               .
               .
               .
0BF      B    PC=F904          MOV    %3,R1
[]                                     (wait for keyboard input)
```

The [] at the bottom of the display means a keyboard input can be made to do one of the following:

<SP>	display the next 22 samples
<sample number>	display 22 samples beginning at a sample number
<memory addr>	display 22 samples beginning at a memory address - hexadecimal value (e.g., F99A) - source label (e.g., @SUM)
<CR>	exit to Monitor

If less than 22 samples are available beginning with the first location to be displayed, all remaining samples will be displayed. If no samples are at the location, the "SAMPLE COUNT =" message will be issued. A display dump can be stopped by any key. When stopped, any key will restart and <ESC> will return to the Monitor.

Trace-line fields are explained in this trace example:

```
0AA      S     PC=FAE0   D1       DECD    R20
(1)     (2)    --(3)--   --------(4)-------
```

(1) trace sample number
(2) S = single step
    B = first instruction or breakpoint
(3) PC address of statement executed
(4) Source statement (label has to be in label table)

Print Trace Sample (PT).

FORMAT: PT <output port>

PARAMETER: Output port (default=1)

Purpose: To print entire trace memory to an output port (1 = terminal, 2 = printer). An output port parameter other than 1 or 2 will cause an error message. The trace must be on (TO command). If the trace is empty, the "SAMPLE COUNT=000" will be displayed.

EXAMPLE:

```
?PT 1
000      S     PC=F800   STRT     MOVD    %>100,R1
001      S     PC=F802            MOV     %>FF,R2
                 .
                 .
                 .
4AA              PC=FAE0   S4       DECD    R4
4AB              PC=FAE2            JNC     S1
4AC      B       PC=FAE4   D4       DECD    R20
```

The trace shows execution (with trace on or TO command issued) of a program starting at >F800 and interrupted by a breakpoint at >FAE4.

Turn Off Trace Sample (TF).

FORMAT: TF

PARAMETERS: None

Purpose: To turn off trace function.

Turn On Trace Sample (TO).

FORMAT: TO

PARAMETERS: None

Purpose: To turn on trace and clear trace-sample memory.

EXAMPLE:

```
?TO  SAMPLE COUNT = >000
```

The sample-count display of >000 indicates trace-sample memory is cleared. It is not necessary to execute TF before TO in order to clear trace-sample memory.

6.6.50 Configure Single-Step Trace (TC)

FORMAT: TC

PARAMETERS: None

Purpose: to set the values to be displayed by four program-execute and single-step commands.

Up to six "fields" can be defined with this command. Legal range of register locations is R0 to R127 or R255, depending on the size of the Register File. These values will be displayed when using the following commands:

EF	Execute to breakpoint with trace and fixed display
ET	Execute to breakpoint with trace
FS	Single step with fixed display
TS	Single step with trace

When the command is first executed, the cursor is positioned at the start of the next line. You can enter three types of data:

- Register locations (R0 to R127 or R255)
- Peripheral File locations (P0 to P255)
- Memory locations (hex address preceded by > or label preceded by @ if label table exists -- use AT command to check)

An <SP> moves the cursor to the next field after entering data. A <CR> concludes the command. Entering the command when field(s) contain data, the fields are printed and you can:

- Enter a new value below the value shown, or
- Enter "S" to "skip" to the next field without changes, or
- Enter "B" to delete (blank) the field directly above.

Each field is 11 characters long and as each field is entered, the cursor will be positioned at the start of the next field. If six "S" commands are entered, the command will terminate. If a "B" command is used on a blank field, the command will terminate. If <SP> is entered immediately after "R", "P", or ">", then "R0", "P0" or >0000 is assumed.

Any time a <CR> is entered, the command terminates. The trace line is redisplayed if any changes have been made. If an error occurs during the command, no change is made to any field.

EXAMPLE:

Three trace areas are designated, then the EF command is executed with values at each area printed out.

```
?TC<CR>
R20<SP>      P254<SP>     >F9AA<CR>
R20=DA       P254=FF      >F9AA=FF
?EF 1<CR>
NEXT INST---> 0030 D1       DECD    R20
PC=F009  C=1 N=1 Z=0 I=0  SP=R1  A=00  B=00     BP1
R20=D9 (11011001)     P254=FF (11111111)    >F9AA=FF (11111111)

?
```

6.6.51 Single-Step Program with Trace (TS)

FORMAT: TS <count> {port 1,2}

PARAMETERS: 1) Number of trace steps (default=1)
2) Output port (default=1)

Purpose: to single-step a program providing the single-step display used by the Monitor SS command as well as the memory trace display entered with the TC command. Except for the expanded display, this is the same as the SS command.

For each count, one program step is executed beginning at the PC value. A count of 0, 1, or no entry (default) specifies one step. As each step (up to 255 maximum) is executed, a display is output.

After the initial step count has expired, the following can be entered:

<SP>	executes one step
<CR>	returns control to the Monitor
1-9	executes number of steps entered
A-Z	executes steps equal to letter ASCII value minus >30 (e.g., "A" executes >42->30 or >12 steps) -- ASCII values less than >30 execute one step.
0 (zero)	executes one step after setting cycle counter to 000000
*	asterisk means execute until a key is pressed
*ADDR	executes until PC = ADDR or until a key is pressed

The display can be sent to Port 1 or 2. If sent to Port 2 (printer), the count must be 3 or higher.

EXAMPLE:

One step requested; then two more asked:

```
?TS<CR>
LAST INST---> F009          DECD   R20
CYCLE COUNT = 000519
NEXT INST---> F00B          JC     >F009
PC=F00B  C=1 N=1 Z=0 I=0  SP=R1  A=F0  B=0C
>FFFF=F0 (11110000)   R20=25 (00100101)
P254=FF (11111111)
P9=00 (00000000)      P9=00 (00000000)
   2<SP>
CYCLE COUNT = 000526
NEXT INST---> F009          DECD   R20
PC=F009  C=1 N=1 Z=0 I=0  SP=R1  A=F0  B=0C
>FFFF=F0 (11110000)   R20=25 (00100101)
P254=FF (11111111)
P9=00 (00000000)      P9=00 (00000000)
(1)
CYCLE COUNT = 000537
NEXT INST---> F00B          JC     >F009
PC=F00B  C=1 N=1 Z=0 I=0  SP=R1  A=F0  B=0C
>FFFF=F0 (11110000)   R20=24 (00100100)
P254=FF (11111111)
P9=00 (00000000)      P0=00 (00000000)
   <CR>
?
```

The Monitor command CY clears the cycle count. If the cycle count exceeds 999999, it will "roll over" (i.e., increment to 000000 and continue from there).

6.6.52 Load Program Counter with TRAP 0 (Zero) Vector (T0)

FORMAT: T0 (T-zero)

PARAMETERS: None

Purpose: to load the PC with the reset vector.

This command is a subset of the RT command (Reset Target Processor)

EXAMPLE:

```
?T0
PC=FFFF  C=0 N=0 Z=1 I=0   SP=R1   A=FF   B=FF
```

6.6.53 Execute Assembler (XA)

FORMAT: XA {port **0**,1,2,3} {port 0,**1**,2}

PARAMETERS: 1) Source input port (default=0)
2) Listing output port (default=1)

Purpose: to execute the Assembler.

The Assembler is discussed in Section 5. The source input port can be 0 (internal RAM), 1 (terminal), 2 (host download), or 3 (audio tape). If the input port is 2, the download mode is selected. If the input port is 1, terminal emulator mode is selected. For TMS7020 and TMS7040 devices, assembly from RAM is allowed. Input port 0 enables assembly from RAM. If the Text Editor is empty, the command will terminate after entry of the 0. Specifying 0 for the output port suppresses the listing except for error messages. The error messages are sent to the terminal if the input port is 2 or 3, and they are sent to Port 2 if the input port is 1. The listing scroll can be stopped and restarted with <SP>.

If source input is from Ports 1 or 2, a "LINE NUMBERS? (N)" prompt is issued. An "N" (or <CR> for default) response means the source file has no line numbers. Any other response means the Assembler must expect a line number and a space at the beginning of each source statement. If a "N" response is given before receiving a file with line numbers, the numbers will be mistakenly assembled as if part of the statement.

An "INITIALIZE? (Y)" prompt asks if program memory should be filled with >FF values and clear the label table before assembly. This will be done with a "Y" (or <CR> for default). A "N" will leave program memory unchanged except for locations containing the new assembled object code.

The Assembler does not output tagged object code. Instead, absolute values are loaded into respective memory locations. Assembled object code is saved using SM or ST commands (Save Memory, 7000 or Tektronix format), and reloaded with the LM or LT commands (Load Memory, 7000 or Tektronix format). When data is loaded in download or terminal emulator mode and during assembly from RAM, the DATA LED is on when data is being loaded into the buffer and off when it is being assembled into RAM.

EXAMPLES:

1)
```
?XA<CR>
INITIALIZE? (Y)<CR>
TMS7000 ASSEMBLER
```

A file is assembled from the Text Editor.

2) ?XA 2 1
LINE NUMBERS? (N) <CR> INITIALIZE? (Y) <CR>
TMS7000 ASSEMBLER

The Assembler is entered with source incoming on Port 2, listing on Port 1 (terminal), and no line numbers with the text.

6.6.54 Execute Text Editor (XE)

FORMAT: XE {port 0,1,2,3}

PARAMETERS: Initial source input port (default=0)

Purpose: to execute the Text Editor.

The Text Editor commands are described in detail in the Text Editor section (Section 4).

The input port can be 0 (user RAM), 1 (terminal emulator), 2 (host download), or 3 (audio tape). If no input port is specified, Port 0 is assumed and text entry from the terminal keyboard is enabled. After display of the Text Editor banner message "EVM TEXT EDITOR", the Text Editor will do one of two things depending upon the input port specification. If Port 0 (terminal) is specified, the number of bytes of available text storage space is printed out (in decimal) on the next line, and the Text Editor prompt "*" is displayed on the following line. If Ports 1,2 or 3 are given, the banner message is displayed and the Text Editor waits until the download is finished before printing out the remaining free RAM on the line and printing the prompt on the following line.

When data is loaded in download or terminal emulator mode from Ports 1 or 2, the DATA LED is on when data is being loaded into the buffer and off when it is being linked into RAM.

Data downloaded through Ports 1 or 2 that has been created with another editor must be bracketed with ">" and "<" characters that signify the beginning and end of file (BOF and EOF) to the EVM Text Editor. These characters must be the first and only character on the first and last lines of the file with each one being bracketed by a <CR>. Line feed characters (<LF>) can optionally follow a <CR>.

Like the XA command, the Text Editor can accept an input text file without line numbers and create line numbers with an increment of 1 during download. This feature is available only with input from Ports 1 and 2 and is achieved by using the "N" or <CR> (default) answer to the "LINE NUMBERS? (N)" prompt. An "N" (or <CR>) response tells the Text Editor that no line numbers precede lines in the incoming file; thus the editor creates a four-digit line number as text lines are received. Otherwise, any other entry means the Text Editor should expect a four-digit line number followed by a space at the start of each line in the file.

EXAMPLES:

1) ?XE<CR>

```
EVM TEXT EDITOR
"H" HELP
EDITOR RAM = 21777 BYTES
*
```

The Text Editor is entered with initial text input from Port 0 (terminal). This is mode used for initial entry of text.

2)
```
?XE 3<CR>
FILENAME: *<CR>
EVM TEXT EDITOR
A
16432
*
```

The Text Editor is entered with initial text input from the cassette tape. The first SOURCE file encountered ("*" after "FILENAME:") is to be loaded. The file name ("A") is displayed after load is complete.

3)
```
?XE 2
LINE NUMBERS? (N) <CR>
EVM TEXT EDITOR
11014
*
```

A file is loaded from a host computer and line numbers are created.

6.6.55 Execute Line Assemblers (XL and XP)

FORMAT: XL or XP

PARAMETERS: None

Purpose: to enter the Line-By-Line Assembler (LBLA).

The XL command creates a new symbol table, and the XP command uses the old (present) symbol table. These commands are described in detail in Section 5.3 and Section 5.4, respectively.

The XL command fills memory with >FF starting at the device default PC address. It also clears the label table.

6.6.56 Execute Reverse Assembler (XR)

FORMAT: XR <start addr><end addr><port **1**,2,3>

PARAMETERS:
1) Address to start reverse assemble
2) Address to end reverse assemble
3) Port to receive listing (default = 1)

Purpose: to enter the Reverse Assembler.

The Reverse Assembler allows reassembling from object code, using the present assembler label table (shown by the AT command), to generate comparable source code. Memory between the start-address and end-address parameters are decoded into source statements to be sent to the port selected. If no parameters are entered, defaults equal previous values used. If no values have been entered since reset, 128 bytes starting at device PC default are reverse assembled with source sent to port 1.

Object values equated to labels (in the label table) will be represented by labels in the source statements; otherwise, these will be represented by absolute values.

A single line describes each source line with 1) execution cycles followed in parentheses by additional cycles if conditional jumps were executed, 2) one to four bytes of object, 3) address 4) source statement consisting of label, mnemonic, and operand. For example, a one-line reverse assembly:

```
?XR F812 F812<CR>
 5 (+2)      E3FC          F812    LABEL     JC      D2
```

Instr. cycles (7 if a jump) — Opcode — Mem. addr. — Source statement

To reverse assemble one line at the PC value, enter:

```
?XR $ +<CR>
 5 (+2)     E3FC          F812    LABEL     JC      D2
```

A single "+" in the second parameter reverse assembles one line at the PC value ("$" = PC). For multiple-line assembly, use a number with the plus sign to specify bytes (e.g., XR $ +>20 reverse assembles from the PC address to PC+>20).

6.7 System Utilities and Access Commands

System utilities can be used to enhance program debug and execution while system access commands allow access to the EVM system (vs. the emulated system).

The following one-letter (preceded by slash) commands can be used to assist in program development and debug:

Command	*Use*
/A	Reset the assembler label table. This will disable the table used in reverse assembly used in displays for single step and breakpoint commands. The table is set by the assemblers.
/C	Return cursor-left and cursor-up key inputs to default values stored in PROM U43. These values can be changed by the CL and CU commands (described in Section 2.15)
/D	Demo mode where dumps are made to both port 1 and port 2 simultaneously. The enabling/disabling of port 2 is toggled by each entry of the /D command.
/E	Reset the assemble-from-Text-Editor flag. After assembling with XA 0, execution by single step and to breakpoints will display source lines from the Text Editor. This command causes displays to come instead from the reverse assembler (do not have comments, etc.)
/H	Displays help menu of system utility commands.
/I	Toggle Port 1 baud rate. For each "/I" entry, the baud rate is set to 110-1200 baud (printer mode) or to 2400-9600 baud (terminal mode).
/M	Resets response of five commands to default value: DM (128 bytes from default PC), MM (default PC for device), MP (P0), MR (R0), and XR (128 bytes from default PC).
/N	Display/modify number of nulls transmitted after a carriage return. This defualts to 00 at reset and can be any number from 00 to >FF. This

number of nulls is transmitted after a CR is sent to the currently enabled EIA port (1 or 2). In the demo mode, the nulls are transmitted to the terminal before the CR is echoed to port 2.

/R Toggle size of register file. File sizes are 128 and 256, toggled for each entry of the /R command. File size is determined by processor and set at reset.

/W Change buffer timeout delay. This sets the delay after raising the handshake line or the receipt of the last character in a download buffer before the buffer is unloaded internally. This is the number of 40 ms delays and can be a number from 00 to >FF. Powerup default value is >20. When using the 'HS 1' (XON) and the line-wait feature of a terminal emulator, this number can be lowered to speed the download.

6.7.1 System Access Commands

Listed in Table 6-2 are the System Access commands. Each command is similar to the equivalent Monitor command except it is preceded by a "$" character, and each has the same parameters as the appropriate Monitor command.

The EVM firmware scales all memory address inputs to the >4000 to >7FFF range for internal use while displaying these addresses in the >C000 to >FFFF range, which is the range that the >4000 to >7FFF RAM will occupy after the memory bank swap that accompanies program execution. In this way, you work in the true microcomputer address range. (EVM memory map is shown in Figure 6-2.) The System Access commands provide a mechanism to bypass this address scaling for selected Monitor commands in order to utilize the entire memory map, including the EVM firmware that resides from >C000 to >FFFF when the Monitor is running.

For example, the Monitor MM command accesses addresses corresponding to the RAM of the microcomputer. The $MM command accesses addresses actually used by the EVM. As shown in Figure 6-2, these areas are separated by >8000. Thus, a value found by the MM command at >F010 will also be found by the $MM command at >7010.

Table 6-2. System Access Command Summary

COMMAND	DESCRIPTION:†
$CE	Compare EPROM (see page 9-3)
$DM/$DH	Display Memory
$DS	Display Machine State
$FB	Find Byte in Memory
$FM	Fill Memory
$HE	Help Menu
$LM	Load 7000 Object Code (see page 12-2)
$LT	Load Tektronix Object Code (see page 12-2)
$MM	Display/Modify Memory
$MR	Display/Modify Register File
$MP	Display/Modify Peripheral File from P0-P255
$MV	Move Memory
$PE	Program EPROM (see page 9-2)
$RE	Read EPROM (see page 9-4)
$RT	EVM Power Cycle Reset (see page 2-4)
$RU	Execute EVM Firmware in RAM (see page 12-2)
$SM	Save Tektronix Object to Audio Tape (see page 8-1)
$ST	Save 7000-Format Object to Audio Tape (see page 8-1)
$XA	For Creation of User Commands (see page 12-2)
$VE	Verify Erased PROM (see page 9-4)
$XL	For Creation of User Commands (see page 12-2)
$XP	For Creation of User Commands (see page 12-2)
$XR	For Creation of User Commands (see page 12-2)

†These are described in Section 6 except where noted otherwise.

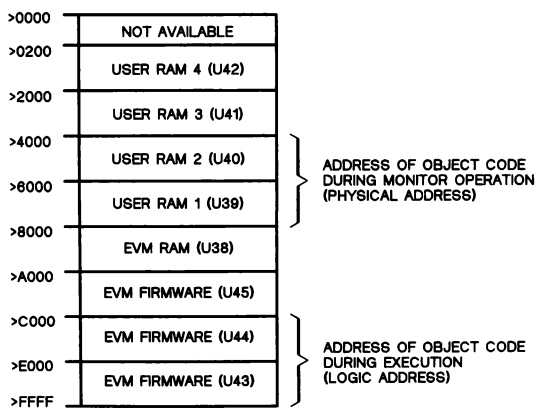


Figure 6-2. EVM Memory Map

6.8 Object Code Loading and Dumping

The EVM accepts object code in three formats (7000, 9900, and Tektronix) and outputs object code in two formats (7000 and Tektronix). Input files must be at absolute or load module level. For an explanation of each of these formats, see the appropriate assembler manual (a listing of these assembler manuals is given in Section 1.5). Object files other than load level may cause the EVM to generate an error for that file. Note that the EVM assemblers do not output tagged object, merely absolute values loaded at set memory locations.

6.8.1 7000 Dump Format

If a dump is performed from locations >FF34 to >FF46, then the 7000 dump format would appear as follows:

```
K0000PROGRAM 9FF34BDEADBDEADBDEADBDEADBDEADBDEADBDEADBDEADBDEAD7F0BFF
BDEAD*DE7FDC6F
:<CR>
```

If a dump is performed from locations >FF34 to >FF47, then the 7000 dump format would appear as follows:

```
K0000PROGRAM 9FF34BDEADBDEADBDEADBDEADBDEADBDEADBDEADBDEADBDEAD7F0BFF
BDEADBDEAD7FD29F
:<CR>
```

The 7000 format tags are explained below:

"K" 4-digit count of relocatable bytes followed by 8-character program identifier (this is a "0" for 9900 format).

Load address:

"9" 4-digit origin address follows the "9".

Object code data:

"B" 4-digit word follows a "B".

"*" 2-digit byte follows a "*" (7000 format only).

End of line:

"7" 4-digit checksum follows - compare the checksums.

"8" 4-digit checksum follows - ignore the checksum.

"F" ignore all characters until <CR>.

The 7000 checksum is the 2's complement of the 16-bit sum of the ASCII values of all characters on the line between carriage returns, excluding all control characters such as <CR>, <LF>, and <FF>.

6.8.2 Tektronix Dump Format

The two previous dumps would be represented in Tektronix format as follows:

```
/FF34130EDEADDEADDEADDEADDEADDEADDEADDEADDEADDEDD
/00000000<CR><LF>
```

and:

```
/FF34140FDEADDEADDEADDEADDEADDEADDEADDEADDEADDEADF4
/00000000<CR><LF>
```

respectively.

The Tektronix line starts with a "/" that is never included in checksum calculations. The next four digits are the address to store the first byte of data on that line. The next two bytes are the number bytes of data on that line. The next two bytes is the first checksum, an 8-bit value consisting of the sum of the hex values of each digit in the address and byte count fields. The data bytes follow, equal to the count given at the start of the line, followed by another checksum, an 8-bit value equal to the sum of the hex values of each digit of the data bytes only. The dump is terminated with a line of zeros in the first three fields.

6.9 Software Breakpoint TRAP 0

The TRAP 0 location is "borrowed" at execution time and used to re-enter the Monitor from a program. In the process of borrowing the vector, the contents of >FFFE and >FFFF are saved in EVM system RAM and replaced by the address of the breakpoint handler. When a breakpoint is encountered, the original vector contents are restored. This is not the case for the execute without breakpoints command (RU). Since this command provides emulation of 100% of the instruction set in real time, it does not borrow the user's TRAP 0 vector contents. Execution commands other than the RU command will execute a TRAP 0 in the user's program but will require some processing time to determine that a breakpoint was not encountered.

6.10 The Stack

The breakpoint routine pushes the PC and ST onto the stack, which allows the Monitor to save the state of the interrupted program. The PC and ST define this state and require three stack locations. Therefore, in addition to the stack locations implemented in the program, three additional stack bytes must be reserved for Monitor execution of breakpoints and/or single-steps. The Monitor uses these three additional stack bytes as temporary storage to transfer the current PC and ST registers of the program to the Monitor's local storage. This does not affect the execution with the stack, since the correct stack pointer value is restored and displayed by the Monitor. The next three register locations beyond the stack cannot be used as work registers, since their contents will be written over by the Monitor upon execution of a breakpoint or single-step.

For example, if the program is known to use 12 stack locations, and the Stack Pointer is initialized to R1, the registers R14, R15, and R16 should be reserved for Monitor use, as shown in Figure 6-3. Figure 6-4 demonstrates the procedure for placing the stack at the end of the Register File.

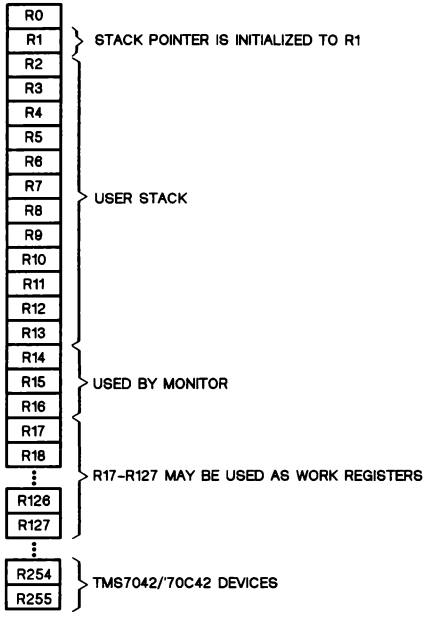


Figure 6-3. Example of Setting the Stack

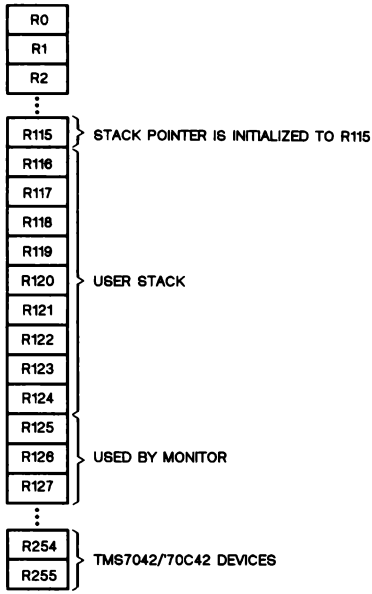


Figure 6-4. Example of Setting the Stack to the Register File Limit

6.11 Reset

The EVM is designed to distinguish between a "cold" reset (one performed at powerup) and various types of "warm" resets (performed at various points during EVM operation). The reset switch may be toggled at any time during program execution to halt the program, causing the Program Counter at the time of the reset to be displayed:

```
STOP AT PC=xxxx
```

where xxxx is Program Counter value at reset.

The disadvantage to this approach is that the $\overline{\text{RESET}}$ interrupt initializes the Stack Pointer to R1 and stores the interrupted Program Counter in R0 and R1. Therefore, you cannot start executing from the point of the $\overline{\text{RESET}}$ interrupt. To make the best use of the displayed PC, the program can be executed again with a breakpoint equal to the interrupted PC. When this "warm" reset occurs, after the PC is displayed the Monitor will set the PC and the SP and ST registers to their default values (shown

in Section 6.5). The advantage of the warm reset is that the program can be stopped at any time to check for proper execution to that point.

Distinguished from the $\overline{\text{RESET}}$ interrupt, which forces execution of TRAP 0 after initializing the processor, is the execution of the TRAP 0 instruction opcode (>FF). If executed within the legal range of addresses during emulation (PC default to >FFFF), it is emulated like any other instruction. If executed outside the legal address range, emulation halts and the "STOP AT PC=xxxx" message is issued, where xxxx is the address of the >FF opcode. This message is also issued any time a PC value outside the legal range is detected prior to execution.

6.12 Reset During EVM Operation

If reset is performed during Monitor command or Assembler execution, control will immediately return to the Monitor top level and the "?" prompt will be printed. A subsequent reset will require a <CR> from the terminal and print the Monitor banner message.

If reset is performed during Text Editor command execution, control will immediately return to the Text Editor top level and the "*" prompt will be printed. A subsequent reset will require a <CR> from the terminal and print the Monitor banner message. Except for the first reset after powerup and execution of $RT (see Section 2.6.1), reset never affects the contents of the Text Editor.

6.13 Monitor Errors

ERROR

This is the general error indicator used when an error condition exists that is not covered by a specified error message:

1) Presence of an invalid tag character or control character during LM or LT execution. ERROR is not fatal and loading continues. The approximate address of the error is sent to the EIA port that is not used for input.

2) Entry of a parameter that is out of legal range. Example is entry of an input or output port for a load or dump that is out of the range 0 to 3. This is a fatal error.

INPUT ERROR

This error is issued immediately after a character has been input that is not within the legal range for the operation involved:

1) Attempting to input a character other that 0-9 or A-F when hexadecimal input is expected or 0-9 when decimal input is expected will cause immediate INPUT ERROR.

2) During LM, LT, or LS execution, only hexadecimal data is expected. If characters other than 0-9, A-F are encountered, INPUT ERROR is issued. In this case, the error is fatal.

ADDRESS ERROR This error is issued immediately after entry of start and stop address parameters in a command string when the start address is greater than the stop address. Also, failure to enter all required address parameters or entry of an address that is out of the legal range for that command will cause ADDRESS ERROR.

During LM and LT execution, occurrence of a load address outside the range of PC default for the current device type to >FFFF will cause ADDRESS ERROR. In this case, the error is fatal.

CHECKSUM ERROR This error is a checksum error for LM and LT commands. The error is not fatal and loading continues. Along with the error message, the approximate address of the error is printed. The address in this case is only approximate because any character on the line that was input could have caused CHECKSUM ERROR.

TAPE ERROR This error is the checksum error for the audio tape read operation. Any time audio tape is used, this error indicates that the checksum written at the end of a block of tape has disagreed with the checksum generated when the block is read. This error is not fatal, and loading will continue. This error can also occur when executing the Text Editor from tape or assembling a file from tape.

7. In-Circuit Emulation

The EVM is designed to emulate a program in Single-Chip mode with the program being executed from the onboard RAM. Emulation is achieved through a detachable 40-pin emulation cable connecting the EVM with your prototype. The emulation cable has 40-pin connectors at both ends equipped with test points for each line. Use of a low profile socket at the target end as a pin protector is recommended. Figure 7-1 shows connector and connector-pin orientation on the board.

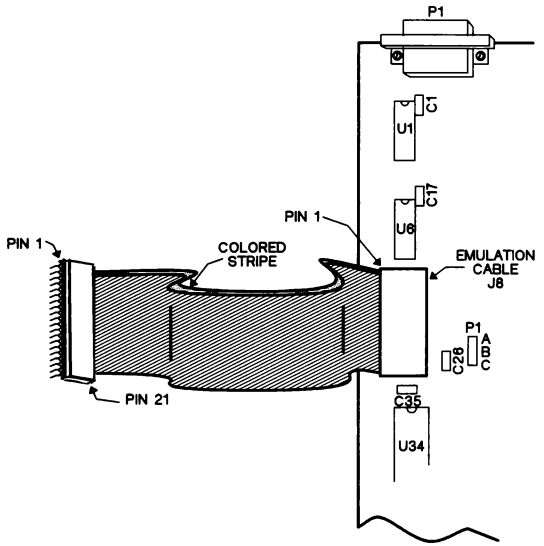


Figure 7-1. In-Circuit Emulation Cable and Connectors

7.1 In-Circuit Emulation Hardware

During in-circuit emulation, the onboard TMS70xx is functioning in Full Expansion mode, thus disabling Ports C and D and the most significant nibble (MSN) of Port B. Any ROM on the part used on the EVM will be ignored since the Mode Control pin is tied to +5 V. The disabled Ports C and D I/O pins are replaced by a RAM Input/Output Timer (RIOT) chip (Rockwell 6522 or 6532 on the NMOS EVM, National NSC810 on the CMOS EVM). Port B pins that are disabled in the Full Expansion mode are replaced by a 4-bit D-type register (74LS175 on the NMOS EVM, 74C175 on the CMOS EVM), so that the most significant nibble of Port B is a TTL output for the NMOS EVM. This will cause a drive-capability difference between the emulated TMS70xx and the actual onboard TMS70xx. All other features are emulated exactly, because the onboard TMS70xx is doing the emulation.

7.2 Powering a Target with the EVM

Pin 25 of the emulation cable is the +5 V target V_{CC}. (Pin numbers are shown on cable connectors -- see Figure 7-1.) This pin is not directly connected to +5 V on the EVM. The B-C connection on hardware jumper P1 connects the EVM +5 V to the target V_{CC} to allow the EVM to power a small prototype. If the prototype supplies its own power, the A-B connection on P1 will isolate the EVM +5 V from the target V_{CC}.

For the CMOS EVM, the target V_{CC} must be supplied to the EVM via pin 25 of the emulation cable because the buffer chips are actually powered by the target, allowing V_{CC} to vary over its full range.

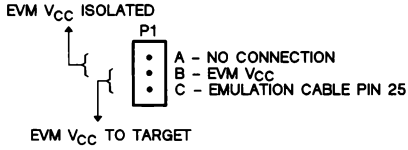


Figure 7-2. EVM/Prototype Power Isolation (P1)

7.3 Connecting an External Clock to the EVM

Hardware jumper P2 selects the clock input for the TMS70xx on the EVM as shown in Figure 7-3. When positions B and C are connected, the EVM is clocked by the onboard crystal. When positions A and B are connected, the EVM is clocked by the TTL clock signal input at pin 17 on the emulation cable.

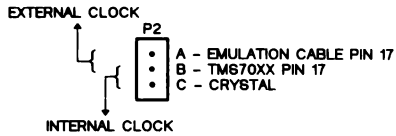


Figure 7-3. Internal/External Clock Configuration (P2)

Whether the EVM is clocked internally or externally, the frequency at the P2 jumper must not be below 1 MHz for the NMOS EVM in order to satisfy the minimum frequency requirements of the Rockwell 6532 RIOT chip.

7.4 Making a New Monitor EPROM

During emulation with internal clock selected, the clock used for operation is the same one that drives the Monitor EPROMs. If a different internal clock frequency is needed, the crystal on the EVM must be changed and the >E000 to >FFFF EPROM (U43) must be reprogrammed with the new values in the frequency-dependent constant table (these are in Appendix D). This can be accomplished by transferring the contents of the EPROM to RAM with the $MV command, changing the values in the table, and programming a new EPROM. After putting this EPROM in place of the original, change the crystal; the EVM is then ready to run at the new frequency. For example:

```
?$MV E000 FFFF 4000          MOVE TO RAM
?$MM 5FBx                    MODIFY IN RAM
   .
   .
   .
?$PE 0 1FFF 4000             PROGRAM BLANK EPROM
PROGRAMMING COMPLETE
?                            REPLACE NEW EPROM
```

8. Audio Tape System

An audio tape peripheral can be attached to the EVM to record the following types of files on cassette tape:

- Source programs
- Object programs in TMS7000 format
- Object programs in Tektronix format
- Machine-state data (EVM current register values)

8.1 File Names and Audio Tape Commands

File type (listed on 8-1) is set by the command used to create it. Files in each file type can be given a single-letter (A to Z) file name. To avoid errors, file names should be different within each file type. Because files are identified by type as well as name, files of different types can have the same name without confusion (i.e., both a TMS7000 source and TMS7000 object program on the same cassette could be named "A" without creating a tape search problem). Table 8-1 lists the file types and the commands used to access them. Table 8-2 lists the commands used with the audio tape system.

Table 8-1. Tape File Types and Associated Commands

FILE TYPE	COMMAND TO CREATE AND SAVE A FILE ON TAPE	COMMAND TO READ A FILE FROM TAPE
Source	"Q 3" (Quit Editor)	"XE 3" (Execute Editor) "XA 3 1" (Execute Assm)
7000 Object	"SM 3" (Save 7000 Object)	"LM 3" (Load 7000/9900 Object)
Tek Object	"ST 3" (Save Tek Object)	"LT 3" (Load Tek Object)
Machine State	"DS 3" (Display State)	"LS 3" (Load State)

Commands that prompt form "FILENAME" when dealing with Port 3 will allow more than one character but will use the last character entered as the file name. Commands that read from tape will also accept "*", which will load the first file encountered of the proper type. For example:

```
?XE 3<CR>
EVM TEXT EDITOR
FILENAME: *<CR>
A
16432
*
```

The Text Editor is entered with initial text input from the cassette tape. The first SOURCE file encountered ("*") is to be loaded. The name of the file found ("A") is displayed after load is complete.

Table 8-2. Audio Tape Command Summary

COMMAND	M/E†	FUNCTION	PAGE
DR 1 or 2	M	Display directory of files on tape	6-14
DS 3	M	Save machine state on audio tape	6-15
LM 3	M	Load 7000 object format from audio tape	6-26
LS 3	M	Load machine state from audio tape	6-27
LT 3	M	Load Tektronix format from audio tape	6-28
MO	M	Cassette motor control circuit is closed to allow control at cassette keys	6-30
Q 3	E	Quit edit, store file on audio tape	4-11
SM 3	M	Save 7000 object format on audio tape	6-36
ST 3	M	Save Tektronix object format on audio tape	6-38
XA 3	M	Execute Assembler, input source file from audio tape	6-44
XE 3	M	Execute Text Editor, input file from audio tape	6-45

† M = Monitor command, E = Text Editor command

8.2 Audio Tape Connection

Connections from the EVM to audio cassette recorder are through Port 3 which consists of the following jacks:

	JACK NOMENCLATURE		
EVM JACK	AT EVM	AT RECORDER	USE
J3	Motor Control	REMote	Cassette recorder motor on/off
J4	OUT	MIC	Data in
J5	IN	EAR	Data out

Connectors can be built with two submini plugs for J3 and two mini plugs each for J4 and J5. These plugs are carried by most home electronics vendors. Connect tip-to-tip and barrel-to-barrel. There is no requirement for shielded or twisted pair.

8.3 Cassette Recorder and Level Settings

A cassette recorder that is the equivalent of Radio Shack series CTR-XX is recommended. At first, experiment by recording and playing back a practice file until you obtain the best settings of the recorder's VOLUME and TONE controls. Start with the controls approximately two-thirds high. The optimum setting should be within the bandwidth between the high and low settings that cause an error message when the practice file is read back from the recorder. Because settings vary from machine to machine, each has to be calibrated separately. Mark or note the optimum settings for further reference.

8.4 Cassette Tapes

High-quality conventional cassette tapes of 20 minutes are recommended; however, tapes up to 60 minutes can be used. Longer tapes are thinner and provide less fidelity. Some special-formula tapes such as chromium oxide and metal-particle tapes are not recommended. *Use leaderless data-grade tape.*

Make sure tapes are not write protected (i.e., be sure the write protect tab on the bottom has not been removed; if it has, cover the hole with a piece of tape).

8.5 Aborting a Tape Session

To stop the recorder in the middle of a playback or recording session, use the EVM RESET switch. Following this, the Monitor prompt is displayed (or the Text Editor prompt with the XE command executing). Use of this feature in the directory operation is discussed in Section 8.9.

8.6 Other Considerations

For recording, usually the PLAY and RECORD cassette keys are pressed at the same time. Make sure that files are not accidently written over; this can happen when the MO command is issued while the recorder keys are still in the record mode.

Be sure the VOLUME and TONE dials are at their optimum settings. Run through several test record/playback sessions to verify settings before making a lengthy recording.

The CHECKSUM ERROR and TAPE ERROR messages occur when the computed and the read checksums for a record differ during a read operation. These are discussed in Section 6.13.

8.7 Creating Audio Tape Files

When a command is executed with output port 3, a file is created and the EVM responds with:

```
FILENAME:
```

A single alphabetic character filename must be entered, followed by a <CR>. The EVM will allow more than one character to be entered, but will accept only the last character entered before the <CR> as the file name. Next, the EVM will turn on the tape motor, giving a chance to position the tape and/or press the record button. The prompt is:

```
READY TO RECORD?
```

At this time, any keyboard input will continue with command execution. Dumping of data to tape begins after a time delay to allow the motors to get up to speed (approximately 1.5 seconds).

Two LEDs, DATA and RLY, support the tape operation. The former is the data indicator and the latter indicates the status of the motor. In the record mode, the RLY LED is illuminated while the motor is on, and the DATA LED is illuminated while the data is being output. After the dump is finished, control will return to the Monitor command handler.

8.8 Loading Audio Tape Files

When a command is executed with input port 3, a file is loaded and the EVM responds with:

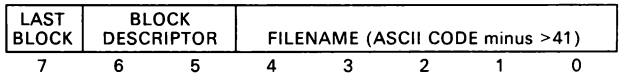
FILENAME:

A single alphabetic character file name must be entered, followed by a <CR>. The EVM will allow more than one character to be entered, but will accept only the last character entered before the <CR> as the file name. After the <CR>, the EVM begins searching the tape for a file with the given name that is of the type required by the command being executed (see Table 8-1). If the file is not found, reset will have to be performed. If reset is performed during a tape operation, command returns immediately to the Monitor and the "?" prompt will appear. If the Text Editor is being entered from Port 3, control remains in the Text Editor and the "*" prompt is displayed. The "FILENAME" entry can be responded to with a "*", which is the wildcard file name. This will cause the first file encountered of the proper type to be loaded.

During the load operation, the RLY LED will act the same way as in the dump operation above, but the DATA LED will light only when the file in question has been found and loading data from the tape into the buffer has begun. Since the tape load routine starts to look for the file beginning with the current position of the tape, file names of the same name and type can be stored on the tape and the tape can be positioned past the first file, either by the Directory operation or using the tape counter, before starting the load operation. Also, since a given load operation looks for a file type in addition to file name, a source file and its assembled version can be stored to tape with the same name for reference. The XE command will only respond to a "SOURCE" file, while the LM command will look for a "MLP OBJECT" file.

8.9 Audio Tape Directory

The Tape Directory command DR allows you to keep a record of files stored to tape. Executing DR with Port 2 produces hard copy for storage with the tape. All blocks of data stored to tape contain a block descriptor byte that is written just before the first data byte. This byte contains the file name, the file type, and whether the block is the last block of the file:



where BLOCK DESCRIPTOR for each type is:

00 (Machine State)
01 (Tek Object)
10 (7000 Object)
11 (Source)

During a Directory operation, the file name and type are printed out as they are first encountered. The number of blocks in the file is printed out last, as the last block is encountered. In this way, the Directory command can be used to position the tape at the logical end-of-tape for storage of a file.

As the block count of the last file is printed, watch the DATA LED, which is lit while the Directory command is reading a block. As soon as this LED goes out, that portion of the file is now completely past the read heads of the cassette player. If a short file

is written over a part of a longer file, the Directory command will terminate the longer file entry with the current block count and initialize a new entry with the new file name. When the shorter file is past, the remainder of the old, longer file will be displayed. Since no motor control other than ON/OFF is available to the EVM, the Monitor has no way of maintaining logical end-of-tape marks for purposes of terminating the Directory command; it must be terminated with a reset. When a reset is performed from the Directory, control is passed directly to the Monitor command handler, and a "?" is printed to the terminal.

Example 8-1. Example of Directory Output

```
AUDIO TAPE DIRECTORY

A     SOURCE          05
A     7000            03
A     STATE           02
D     TEK             01
F     SOURCE          03
F     7000            02<RESET>
?
```

8.10 Motor Control Utility Command (MO)

Since the cassette motor is under program control, it is only enabled when the EVM is trying to create or load a file. To enable the motor without disconnecting the motor control cable, the Monitor MO command enables the motor until another key is pressed. This provides a direct way to rewind tapes for storage or to fast-forward a tape past the clear leader prior to storing a file to the beginning of a tape.

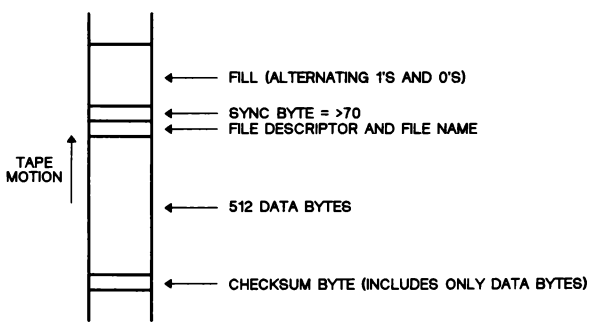


Figure 8-1. Typical Audio Tape Data Block

8.11 Error Messages

ERROR

This is the general error indicator used when an error condition exists that is not covered by a specified error message:

1) Presence of an invalid tag character or control character during LM or LT execution. ERROR is not fatal and loading continues. The approximate address of the error is sent to the terminal.

2) Entry of a parameter that is out of legal range. Example is entry of an input or output port for a load or dump that is out of the range 0 to 3.

INPUT ERROR

This error is issued immediately after a character has been input that is not within the legal range for the operation involved:

1) Attempting to input a character other that 0-9 or A-F when hexadecimal input is expected or 0-9 when decimal input is expected will cause immediate INPUT ERROR.

2) During LM, LT, or LS execution, only hexadecimal data is expected. If characters other the 0-9, A-F are encountered, INPUT ERROR is issued. In this case, the error is fatal.

ADDRESS ERROR

This error is issued immediately after entry of start and stop address parameters in a command string when the start address is greater than the stop address. Also, failure to enter all required address parameters or entry of an address that is out of the legal range for that command will cause ADDRESS ERROR.

During LM and LT execution, occurrence of a load address outside the range of PC default for the current device type to >FFFF will cause ADDRESS ERROR. In this case, the error is fatal.

CHECKSUM ERROR

This error is a checksum error for LM and LT commands. The error is not fatal and loading continues. Along with the error message, the approximate address of the error is printed. The address in this case is only approximate because any character on the line that was input could have caused the CHECKSUM ERROR.

TAPE ERROR

This error is the checksum error for the audio tape read operation. Any time audio tape is used, issuance of this error indicates that the checksum written at the end of a block of tape

has disagreed with the checksum generated when the block is read. This error is not fatal, and loading will continue. This error can also occur when executing the Text Editor from tape or assembling a file from tape.

9. EPROM Programmer

The EVM is equipped to program the following EPROMs:

- TMS2732 (4K x 8 EPROM)
- TMS2764 (8K x 8 EPROM)
- TMS27128 (16K x 8 EPROM)
- TMS27C64 (8K x 8 CMOS EPROM)
- TMS27C128 (16K x 8 CMOS EPROM)
- TMS7742 (4K x 8 microcomputer)
- SEEQ 72710 (1K x 8 microcomputer)
- SEEQ 72720 (2K x 8 microcomputer)

It is good practice to check the voltage at the terminal strip power connector (V_{PP} Out) periodically to ensure good programming. The Monitor command "21" is provided to aid in V_{PP} calibration. Entering the command will set V_{PP} to that value until any keyboard input, which will return V_{PP} to +5 volts.

The LED near the EPROM sockets (PGM) is lit anytime a V_{PP} other than +5 volts is applied to the programming sockets.

Caution:

1. Do not insert an EPROM if the PGM LED is lit.

2. Do not have a device in both U19 and U20 at the same time as signals are enabled to both sockets at the same time.

9.1 EPROM Programmer Commands

The EPROM programmer commands are similar in format.

For each command, the EPROM and memory parameters assume leading zeros and will use the last four digits if more than four are entered. Entry is in hexadecimal, and the range restriction on the RAM start address allows entry of an even page boundary. For example, for device type 1 (TMS7020 and '70C20) with default PC = >F806, the lowest legal RAM start address is >F800. Execution of any EPROM command will automatically terminate if the memory address increments from >FFFF to >0000.

Each command also contains a destination parameter that specifies the device that is to be programmed. This parameter enables an out-of-range address check of the EPROM start and stop locations. In all cases, this parameter has a default. The default value in stored in EPROM at location >FFB0 and can be changed to any of the legal device types in Table 9-1. The value is initially >04 (TMS2764) and if it is changed to an illegal value, then >04 is assumed. Section 9.7 describes how to program the EPROM.

Table 9-1 lists the destination-code parameters and the EPROM start and stop minimum and maximum values used by the EPROM programmer commands described below for that destination.

Table 9-1. EPROM Destination Parameters

DESTINATION CODE	EPROM	EPROM START ADDRESS	EPROM STOP ADDRESS	SOCKET
>02	TMS7742	>0000	>0FFF	U19*
>04	TMS2764	>0000	>1FFF	U19
>08	TMS27128	>0000	>3FFF	U19
>C4	TMS27C64	>0000	>1FFF	U19
>C8	TMS27C128	>0000	>3FFF	U19
>42	TMS7742	>0000	>0FFF	U20†
>71	72710‡	>0000	>03FF	U20
>72	72720‡	>0000	>07FF	U20

*Requires adapter RTC/PGM2764-06
†Rev. A EVM boards only
‡ SEEQ microcomputer.

9.1.1 Program EPROM from Memory (PE)

FORMAT: PE <EPROM start> <EPROM stop> <mem start> <dest>

PARAMETERS:
1) EPROM start address
2) EPROM stop address
3) Memory start address of data to be programmed
4) Destination (default=4; see Table 9-1)

Purpose: to program the specified device.

Locations are compared immediately after programming. When a miscompare occurs, programming can be aborted with <CR> or continued by pressing any other key. Section 9.8 discusses the compare error format and procedure.

EXAMPLES:

1) ?PE 1800 1FFF F800 <CR> COMPLETE
   ?

   This example shows successful programming of a TMS2764. If errors occurred, the display could show the following (EPROM address and value followed by corresponding memory address and value):

2) ?PE 1800 1FFF F800<CR>
   ERROR E:19FE 80 M:F9FE D1<SP>
   ERROR E:1A00 00 M:FA00 FF<SP>
   PROGRAM COMPLETE<SP>
   ERROR E:19FE 80 M:F9FE D1<SP>
   ERROR E:1A00 00 M:FA00 FF<SP>
   COMPARE COMPLETE
   ?

9.1.2 Compare EPROM to Memory (CE)

FORMAT: CE <EPROM start> <EPROM stop> <mem start> <dest>

PARAMETERS:
1) EPROM start address
2) EPROM stop address
3) Memory start address
4) Destination (default=4; see Table 9-1)

Purpose: to compare the contents of the specified device with memory.

When locations do not compare successfully, the routine prints out both addresses and both values and stops. At this point, <CR> terminates the compare operation, and hitting any other key continues with the compare operation. Section 9.8 discusses the compare error format and procedure.

EXAMPLES:

1) ?CE 1800 1FFF F800<CR>
   COMPARE COMPLETE
   ?

   The contents of a TMS2764 EPROM were successfully compared to the contents of memory.

2) ?CE 0000 0FFF F000 8<CR>
   ERROR E:0019 CD M:F019 CA<SP>

```
ERROR   E:0110 FF M:F110 00<SP>
COMPARE COMPLETE
?
```

The compare operation found two bad comparisons.

9.1.3 Read EPROM to RAM (RE)

FORMAT: RE <EPROM start> <EPROM stop> <mem start> <dest>

PARAMETERS:
1) EPROM start address
2) EPROM stop address
3) Memory start address
4) Destination (default=4; see Table 9-1)

Purpose: to read the contents of the specified device into RAM.

EXAMPLE:

```
?RE 1800 1FFF F800 4<CR>
COMPLETE
?
```

The upper 2K bytes of a TMS2764 are read into RAM starting at address >F800.

9.1.4 Verify EPROM Erased (VE)

FORMAT: VE <EPROM start> <EPROM stop> <dest>

PARAMETERS:
1) EPROM start address
2) EPROM stop address
3) Destination (default=4; see Table 9-1)

Purpose: to verify whether the specified device is empty (>FF in all locations).

Section 9.8 discusses the verify error format and procedure.

EXAMPLES:

1)
```
?VE 0 1FFF<CR>
COMPLETE
?
```

A TMS2764 is successfully verified.

2)
```
?VE 0 1FFF<CR>
ERROR   E:0019 CD<CR>
?
```

The VE operation is aborted when the first location not containing >FF was found. An <SP> instead of the <CR> would have continued the verify operation.

9.2 TMS2764 and TMS27128 EPROM Programming

The TMS2764 and TMS27128 EPROMS use the 28-pin programming socket (U19). The programming algorithm provides for 50 ms programming pulses and will skip over any location which is to be programmed with the value >FF, the erased state.

Locations are compared immediately after programming, showing errors as they occur. If any errors are detected, the location in the EPROM and the data at that location will be printed along with the corresponding location in memory and its data, and the cursor will stop at the end of the line. To continue the program/compare process, press <SP>. If a <CR> is entered, the process is terminated. Repeating this procedure will show all errors. The "PROGRAMMING COMPLETE" message will be printed at completion whether or not any errors occurred.

9.3 Programming the TMS7742 with the RTC/PGM2764-06

The TMS7742 can be programmed with the aid of the programming adapter module (RTC/PGM2764-06 or TMDS7080001). The module plugs into the the EPROM programming socket and treats the TMS7742 as a TMS2732A EPROM. (A rev. A board with 2.X software (or later) does not need the adapter; otherwise the adapter is necessary.) To compare the EPROM contents to RAM immediately after programming, several on-adapter switch settings must be made. Before any command is executed (except PE), place the adapter module switch in the READ position. Before programming, place the switch in the PGM position. Prompts will remind you to make these settings.

When programming a TMS7742, jumper P7 must be set to "77X2".

After programming is comptete and the PROGRAMMING COMPLETE message issued, place the switch in the READ position (also reminded by a prompt) in order to proceed with the compare operation. Along with the switch prompts, a terminal key must be pressed to continue command execution.

9.4 Programming the SEEQ 727x0

The EVM is designed to program the 727x0 EEPROM devices in U20, the 40-pin socket (socket U19 is for TMS2764/'27128 devices). The same commands used to program TMS2764/'27128 devices (Section 9.1.1 to Section 9.1.4) are used to program 727x0 devices. Section 9.4.1 covers erasing the 727x0.

When programming the 727x0, jumper P7 must be set to "727X0".

Examples of programming commands:

?VE 0 3FF 71<CR>	(verify all locations = >FF)
?PE 0 7FF F800 C4<CR>	(program 2K algorithm from >F800 to >FFFF into 72720)
?CE 0 3FF FC00 C4<CR>	(compare 72720 contents with memory addr >FC00 to >FFFF)
?RE 0 7FF F800 72<CR>	(read contents of 72720 into memory addr >F800 to >FFFF)

The above examples assume that the 72720 security lock at P127 is not set at program execution. This feature prohibits programming or reading the program

memory of the microcomputer. To remove this security lock, it is necessary to erase the entire 72720 memory which can be done with the BC command described below. If the security lock is not set, the EEPROM memory can be programmed and read at will on a byte-by-byte basis as is the case with all EEPROM devices.

9.4.1 Clearing the 727x0 (BC)

FORMAT: BE

PARAMETERS: None

Purpose: to remove the security lock at P127 and erase the entire EEPROM memory contents of the 727x0 (perform a "block clear").

Before 727x0 memory can be written to or read, the security lock must be removed. This command (whether or not previously set) the security lock, erases entire memory, and also disables a 727x0 locked into microprocessor mode. This operation requires that +12 V be applied to the 727x0 for 50 ms. Calibrating this voltage is described in Section 9.5

9.4.2 Assembling a Subroutine to Program the 727x0

A subroutine that uses the PRG instruction to program the SEEQ 727X0 microcomputers can be assembled, but not executed, on the EVM. The PRG instruction writes the data in Register A to the location specified by a register pair Rn and Rn-1 (using indirect addressing). The format is:

[LABEL] <PRG> <*register no.> [comment]

EXAMPLE:

```
LABEL    PRG    *R6   CONTENTS OF REGISTER A TO R5/R6 ADDR
```

The above example programs the contents of Register A into the address pointed to by R5 (MSB) and R6 (LSB). Note that the PRG instruction can only be assembled -- *not executed* -- on the EVM board.

9.5 Calibrating VPP Voltage (12 or 21)

VPP programming voltages of 12 V and 21 V are selected for the specific device according to the destination code in Table 9-1. These voltages can be calibrated with the "12" and "21" commands for the optimum setting in these two voltage areas.

1) Attach a voltmeter to the two rightmost posts (GND and VPP) of power connector J6 (shown in upper right of Figure 1-1).

2) Enter the command "12" for 12 V or "21" for 21 V calibration.

3) R32 is located in the center of the board (3 inches below connector J5). Now adjust this for the optimum voltage.

4) Hit any key. The VPP reading returns to 5 V, and control returns to the Monitor.

For example, to adjust the 12 V power:

```
?12                              (enter 12 V calibrate command)
ARE YOU SURE? (N) Y              (enter Y, adjust R32 for VPP)
<any key>                        (any key = return to Monitor)
?
```

The same procedure can be used for 21 V adjustment with the "21" command.

Caution:

Do not insert an EPROM if the PGM LED is lit.

9.6 Copy Monitor EPROMS (43, 44, 45)

FORMAT: 43 or 44 or 45

PARAMETERS: None

Purpose: to copy EPROM contents at EVM sockets U43, U44, or U45 into an erased TMS2764 EPROM in programming socket U19.

COMMAND	REPLACES COMMAND	PURPOSE
43	$PE 0 1FFF E000	Copy U43 EPROM into U19 EPROM
44	$PE 0 1FFF C000	Copy U44 EPROM into U19 EPROM
45	$PE 0 1FFF A000	Copy U45 EPROM into U19 EPROM

9.7 Making a New Monitor EPROM

This section repeats parts of Appendix D. Sometimes a modified version of the Monitor is needed with the new software reprogrammed into EPROM and inserted in U43 (or U44 or U45 as shown in Figure 2-2). This can be accomplished by reading the EPROM contents into memory, modifying the addresses needed, then programming the modified code back into an erased EPROM.

For example, if a different internal clock frequency is needed for your operation, the crystal on the EVM must be changed and the >E000 to >FFFF EPROM (U43) must be reprogrammed with the new values in the frequency-dependent constant table (described in Section 2.9). Do this by transferring the contents of the EPROM to RAM with the $MV command, changing the values in the table, and programming the results into a new EPROM. Then replace the original EPROM with the new version. For example:

```
?$MV E000 FFFF 4000              (move to RAM)
?$MM 5FBx                        (modify in RAM)
   .
   .
   .
?$PE 0 1FFF 4000                 (program blank EPROM)
PROGRAMMING COMPLETE
?                                (replace new EPROM)
```

This same procedure is followed for changing the EVM default values described in Section 2.

9.8 EPROM Programmer Errors

When an error occurs during EPROM operations, execution stops with the cursor at the end of the error display. Entering a <CR> will terminate the process immediately. Entering anything else will continue the process until it is completed or until the next error is encountered.

The format of a verify error is:

```
ERROR    E:XXXX xx
```

where "XXXX" is the EPROM address and "xx" is the data at that location.

The format of a compare error is:

```
ERROR    E:XXXX xx    M:YYYY yy
```

where "XXXX" is the EPROM address and "xx" is the data at that location, and "YYYY" is the memory address and "yy" is the data at that location.

10. I/O Port Reconstruction

As shown on the EVM schematic (in Appendix A), numerous addresses are decoded within the Peripheral File address range to reconstruct Ports B, C, and D, and to construct the ports used by the EVM for firmware operation (P250-P254). Table 10-1 lists all decoded addresses and functions in the Peripheral File.

Table 10-1. Decoded Peripheral File I/O Ports

PORT	FUNCTION
P6	Reconstructed top half of Port B
P8-P11	Reconstructed C/CDR and D/DDR ports
P248	Spare
P249	Spare
P250	Port I/O: Read/Write (Lsb) IO0 - EIA Port Select (0=Port 2, 1=Port 1) IO1 - $\overline{\text{DTR}}$ Out to EIA IO2 - FSK Data Out to Port 3 IO3 - EIA Data Out IO4 - Tape Motor (0=OFF, 1=ON) IO5 - EIA Data In IO6 - $\overline{\text{DSR}}$ In from EIA (Msb) IO7 - FSK Data In from Port 3
P251	Port Z: Write only, EPROM programmmer Address (Z0-Z5) EPROM Programmer V_{PP} control (Z6, Z7)
P252	Port Y: Write only, EPROM programmer address - LSB
P253	Port X: Read/Write, EPROM programmer data
P254	Port W: Write only, Control W0 - Bank Select (0=EPROM, 1=RAM) W1 - DATA LED W2 - EPROM Programmer $\overline{\text{OE}}$ W3 - EPROM Programmer $\overline{\text{PGM}}$ Data Window W4 - EPROM Programmer $\overline{\text{PGM}}$ W5 - V_{PP} Control (0=OFF, 1=ON) W6 - EPROM Programmer 727X0 Reset W7 - EPROM Programmer 727X0 Mode
P255	Spare

11. Diagnostic Input/Output Utilities

Subroutine calls can be embedded in a program to access terminal I/O routines during emulation. This allows messages to be sent or flags and calculation results to be printed to the terminal without stopping program execution. During execution of fixed display debug commands in the Monitor, user I/O is disabled.

Access to the I/O utilities is with the statement

```
CALL  @>4000
```

in the program. The function performed is dependent on the contents of Registers A and B, and in the case of utility number 5, the contents of the stack.

B = >00 (or any value not listed below) Print the contents of the A register to the terminal. The contents of Register A is assumed to be an ASCII value.

B = >01 Convert the contents of Register A from two hex nibbles to two ASCII characters and print the result to the terminal.

B = >02 Stop program execution until a key on the terminal is struck. When this is done, load the ASCII value of the key into Register A.

B = >03 Print a new line (<CR><LF>) to the terminal.

B = >04 Print a space to the terminal.

B = >05 Print a message to the terminal on a new line. The address of the message is pushed onto the stack by the program prior to the subroutine call. The subroutine call pops both values (even if disabled during fixed display debug), restoring the stack to its original content.

Example 11-1. Sample Output Utilities

```
(a)         MOV  %>35,A       LOAD ASCII "5"
            CLR  B            PRINT "5"
            CALL @>4000       TO THE TERMINAL
                 .
                 .
                 .

(b)         MOV  %>35,A       LOAD HEX >35
            MOV  %1,B         PRINT "35"
            CALL @>4000       TO THE TERMINAL
                 .
                 .
                 .

(c)         MOVD %MESG,B      LOAD ADDRESS OF MESSAGE
            PUSH A            PUSH MSB
            PUSH B            PUSH LSB
            MOV  %5,B         INDICATE MESSAGE
            CALL @>4000       PRINT "TEST 1" ON A NEW LINE
                 .
                 .
                 .
       MESG TEXT 'TEST 1'     MESSAGE BODY
            BYTE 0            MESSAGE TERMINATOR
                 .
                 .
                 .
```

Example 11-2. Sample Input Utility

```
       WAIT MOV  %2,B         STOP PROGRAM
            CALL @>4000       EXECUTION AND WAIT
            CMP  %'Q',A       FOR ENTRY OF
            JNE  WAIT         "Q"
                 .
                 .
                 .
```

12. Creating Monitor Commands

The Monitor nas ten commands that branch to entry points in EPROM socket U45, allowing you to create Monitor commands. These commands are designated "Ux" where "x" is a number from 0 - 9. Any command preceded by "$" will be treated as a System Access command (no address restriction).

When the command is entered, it loads and branches to an address stored in the appropriate two bytes (determined by the value of "x") in the top 20 bytes of the EPROM. Table 12-1 lists the commands and entry point vector locations in EPROM.

Table 12-1. Monitor Command-Location Vectors in Expansion EPROM (U45)

COMMAND	ENTRY POINT VECTOR LOCATION
U0	>BFFF LSB >BFFE MSB
U1	>BFFD LSB >BFFC MSB
U2	>BFFB LSB >BFFA MSB
U3	>BFF9 LSB >BFF8 MSB
U4	>BFF7 LSB >BFF6 MSB
U5	>BFF5 LSB >BFF4 MSB
U6	>BFF3 LSB >BFF2 MSB
U7	>BFF1 LSB >BFF0 MSB
U8	>BFEF LSB >BFEE MSB
U9	>BFED LSB >BFEC MSB

12.1 Monitor Re-Entry Points

Table 12-2 lists the locations of Monitor entry point addresses located for easy Monitor re-entry from a command executing in expansion EPROM. These addresses must be loaded and branched to with a routine like the following:

```
LDA     @>FFAB          LOAD LSB OF BANNER ENTRY
XCHB    A               PUT LSB IN B
LDA     @>FFAA          LOAD MSB OF BANNER ENTRY
BR      *B              BRANCH THRU A/B PAIR
```

Table 12-2. Monitor Entry Points

ADDRESS	ENTRY POINT
>FFA0 MSB >FFA1 LSB	Spare
>FFA2 MSB >FFA3 LSB	Spare
>FFA4 MSB >FFA5 LSB	ADDRESS ERROR
>FFA6 MSB >FFA7 LSB	INPUT ERROR
>FFA8 MSB >FFA9 LSB	ERROR
>FFAA MSB >FFAB LSB	Banner
>FFAC MSB >FFAD LSB	?

12.2 Monitor Command Development Aids

Several development aids are built in to the EVM firmware to allow use of the EVM Text Editor and Assembler to develop custom Monitor commands:

$LM	Load 7000 object code
$LT	Load Tektronix object code
$RU	Execute from address specified
$XA	Execute Assembler without address restriction
$XL	Execute LBLA without address restriction
$XP	Execute Patch Assembler without address restriction
$XR	Execute Reverse Assembler

12.2.1 $LM and $LT

The EVM object code load commands each have system access counterparts allowing the EVM to load up to 16K bytes of object code originating at any address. These commands will store incoming data in RAM between >4000 and >7FFF in a location relative to the true origin address.

12.2.2 $RU

The format of the $RU command is:

$RU <addr>

where the address parameter is the address of the routine to be executed (defaults to >4000). $RU differs from the Monitor command RU in that no bank select of RAM occurs, leaving the Monitor in control, and no restoration of the TMS70x0 state is performed. The command is a simple branch, executing the subsequent code as though it were EVM firmware.

When a command has been written and debugged, it can be programmed into an EPROM with its entry address in two of the top 20 bytes in the EPROM and subseqently executed as "Ux", where "x" is a number that indicates the entry point vector location (see Table 12-1).

12.2.3 $XA, $XL, XP, and $XR

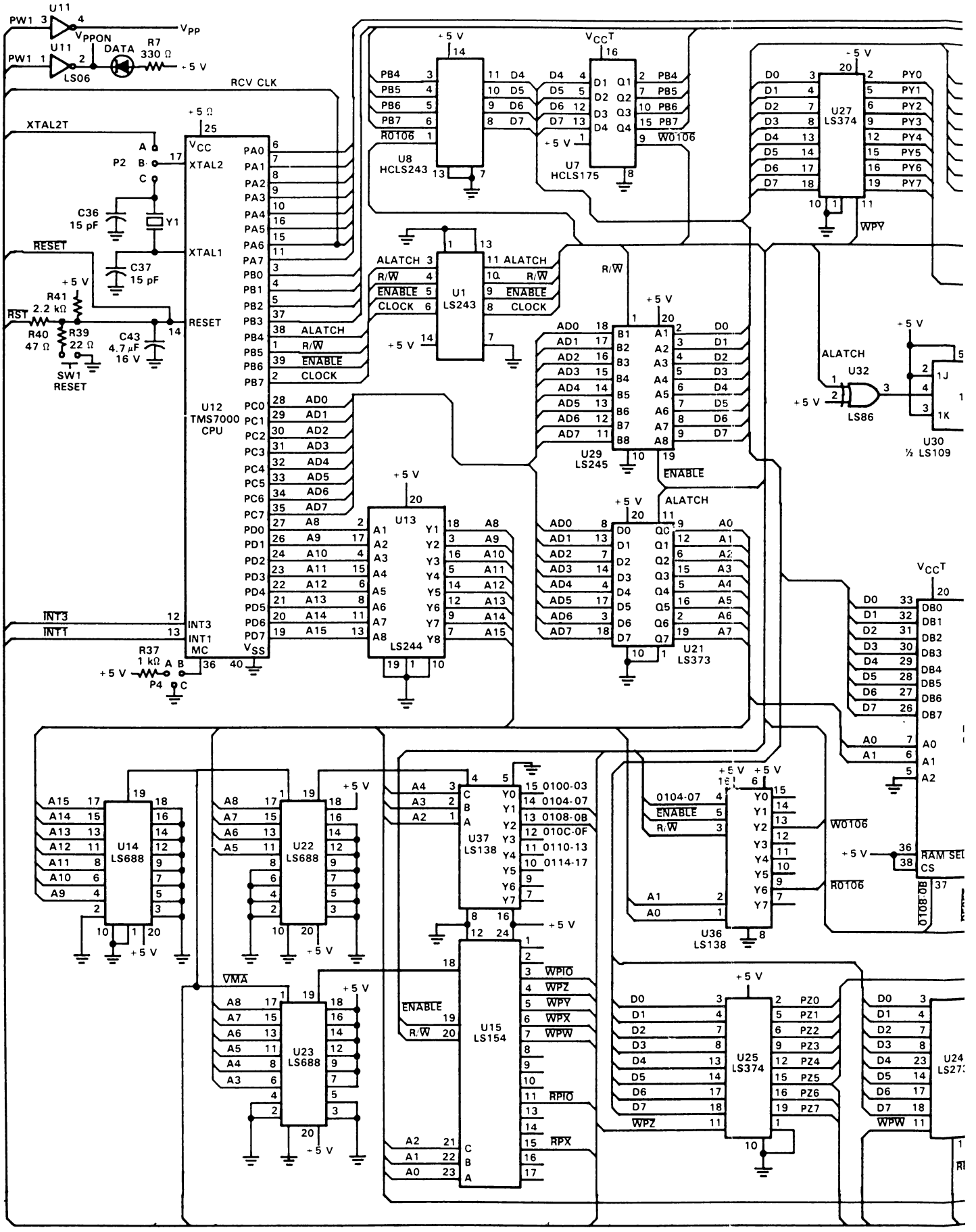
The $XA command has the same parameters as the Monitor XA command. The difference between the two assemblers is the disabling of the address format conversion used by the Assembler to store code in the >4000 to >7FFF RAM address range while creating code executable at the >C000 to >FFFF address range. By removing this restriction, the Assembler can create code executable at the RAM address range for execution with the $RU command. The AORG for the $XA and $XL commands defaults to >4000. The $XL, $XP, and $XR commands are otherwise identical to the XL, XP, and XR commands respectively except for the removal of address restrictions.

The Monitor TRAP vectors are useful during creation of Monitor commands. These TRAPs are used by the Monitor for various I/O functions in command entry and execution. Table 12-3 lists these TRAPs, their functions, and the registers needed to use them. To determine the registers used by each TRAP and how all routines interact, an assembled source listing is required.

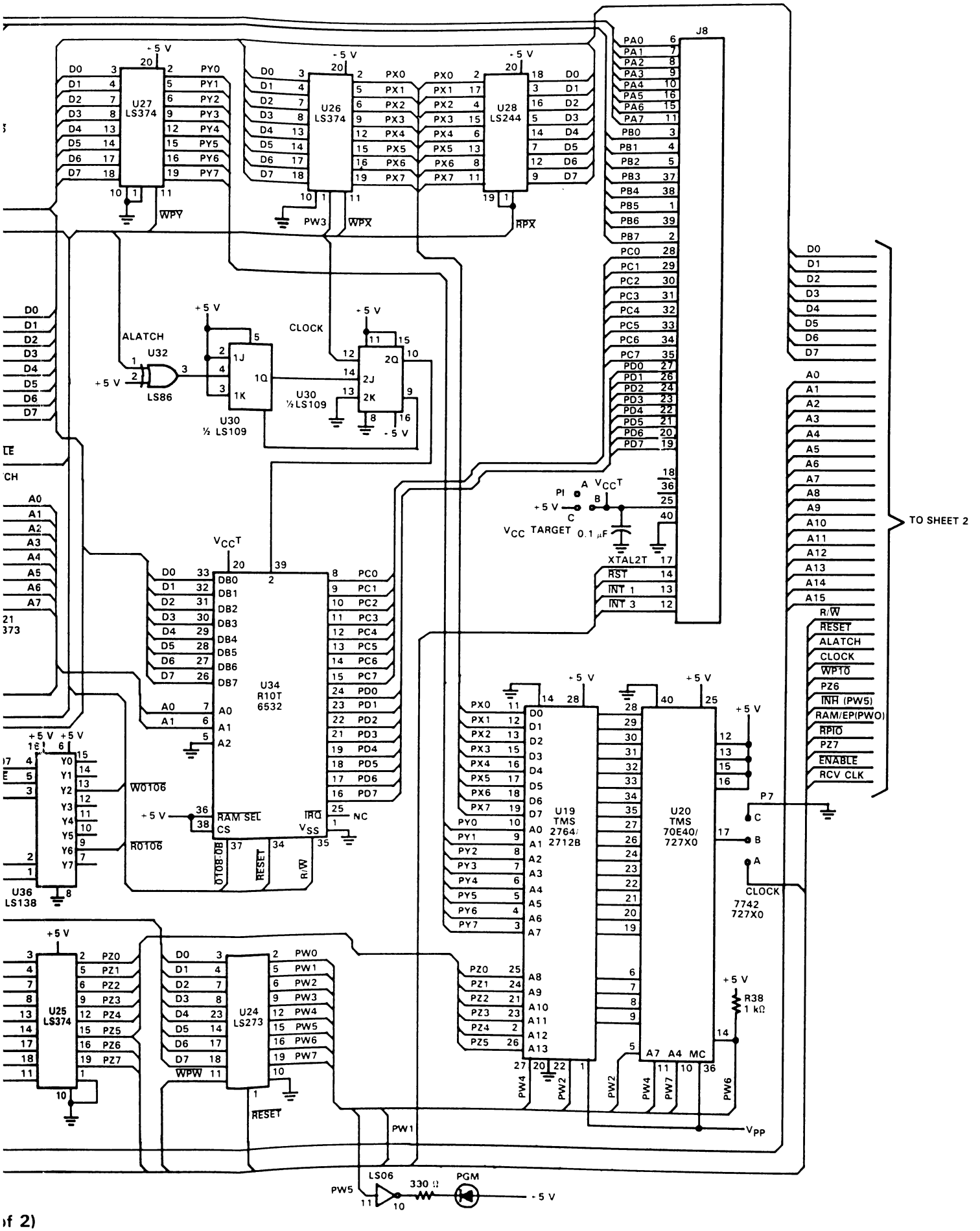
Table 12-3. Monitor TRAP Vectors

TRAP	FUNCTION
TRAP 23	Convert nibble to ASCII and print
TRAP 22	Print <CR>
TRAP 21	Wait for "=" in input stream
TRAP 20	Print <SP>
TRAP 19	Print 16-bit address
TRAP 18	Print 8-bit data, <SP>, wait for input
TRAP 17	Print <SP>, wait for input
TRAP 16	Print message
TRAP 15	Input start and end address pair
TRAP 14	Load Register A from memory
TRAP 13	Input 8-bit decimal number (data)
TRAP 12	Reserved for assembler
TRAP 11	Reserved for assembler
TRAP 10	Input hex characters until <CR> or <SP>
TRAP 9	Accumulate MLP checksum
TRAP 8	Print <CR><LF>
TRAP 7	Convert byte to 2 nibbles
TRAP 6	Load ASCII character to Register A
TRAP 5	Print ASCII character in Register A
TRAP 4	Store Register A to memory
TRAP 3	Load default Program Counter/AORG
TRAP 2	Bell
TRAP 1	Input hexadecimal character after error
TRAP 0	Reset

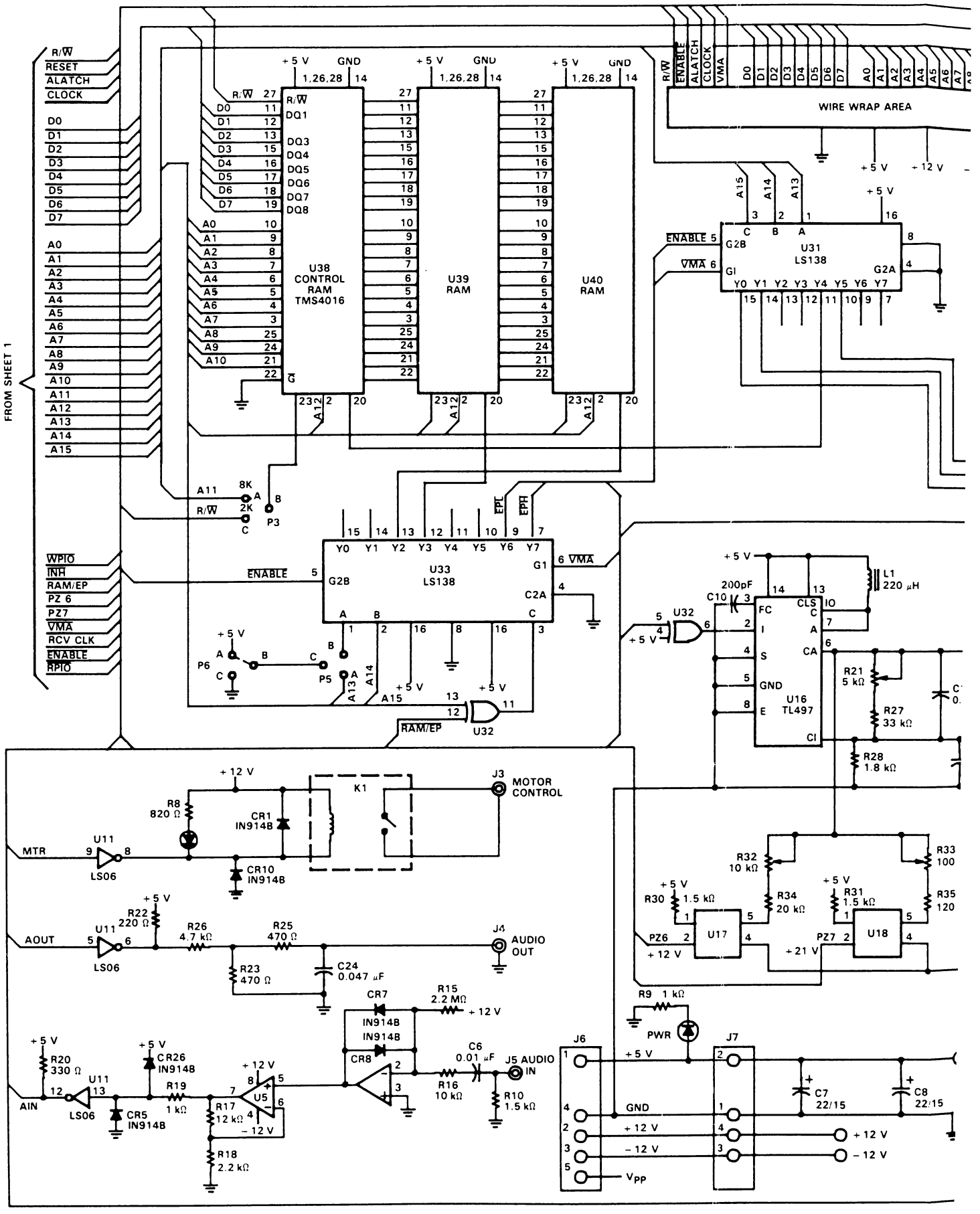
A. Schematic



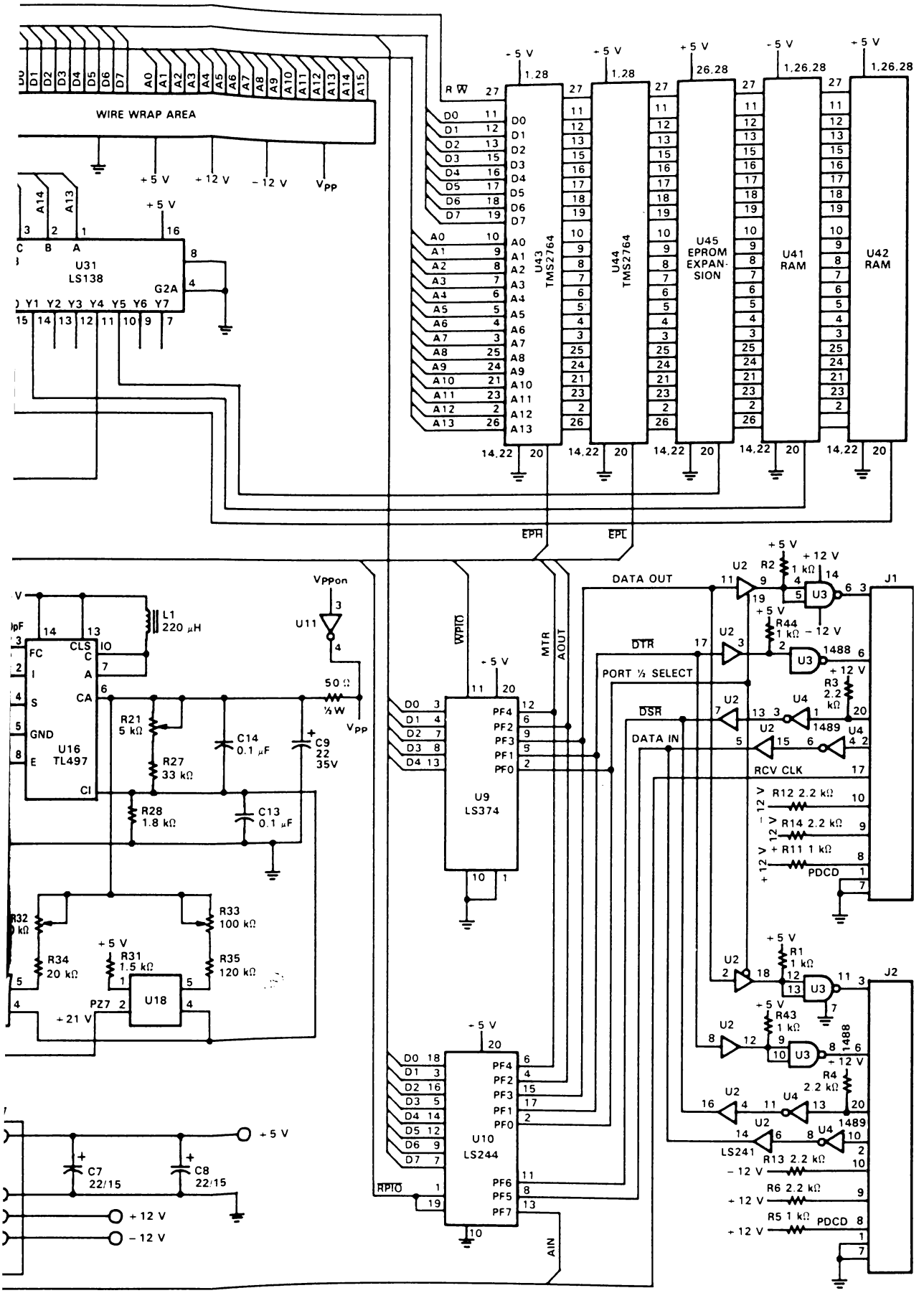
RTC/EVM7000N-1 (Page 1 of 2)

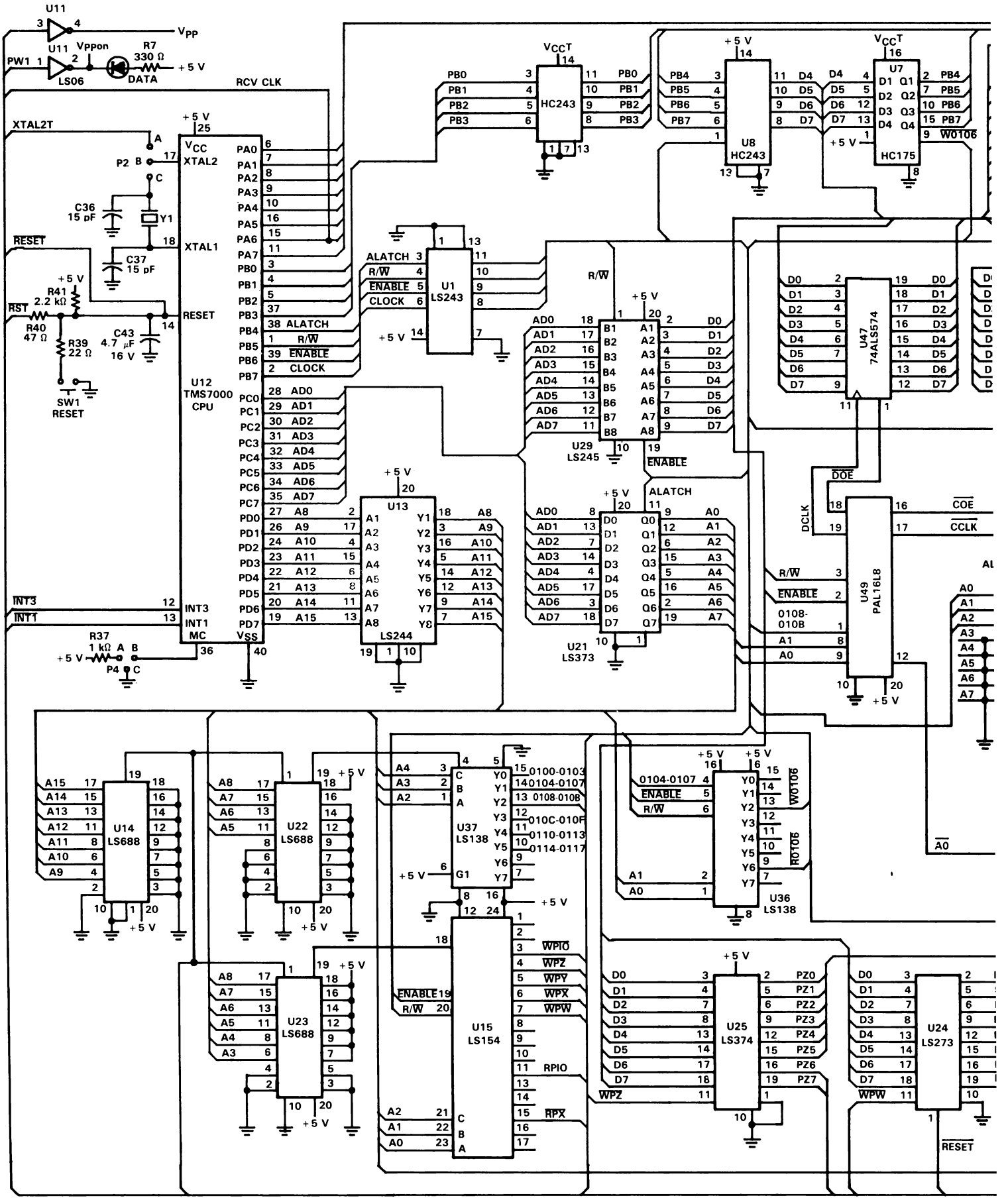

U27 LS374
U26 LS374
U28 LS244
J8
U32 LS86
U30 ½ LS109
ALATCH
CLOCK
U34 R10T 6532
U36 LS138
U19 TMS 2764/ 2712B
U20 TMS 70E40/ 727X0
U25 LS374
U24 LS273
XTAL2T
RST
INT 1
INT 3
R/W
RESET
ALATCH
CLOCK
WP10
PZ6
INH (PW5)
RAM/EP(PWO)
RPIO
PZ7
ENABLE
RCV CLK
TO SHEET 2
RAM SEL
W0106
R0106
WPY
WPX
RPX
PW3
PW1
PW2
PW4
PW6
PW7
LS06
330 Ω
PGM
R38 1 kΩ
0.1 μF
CLOCK 7742 727X0

of 2)

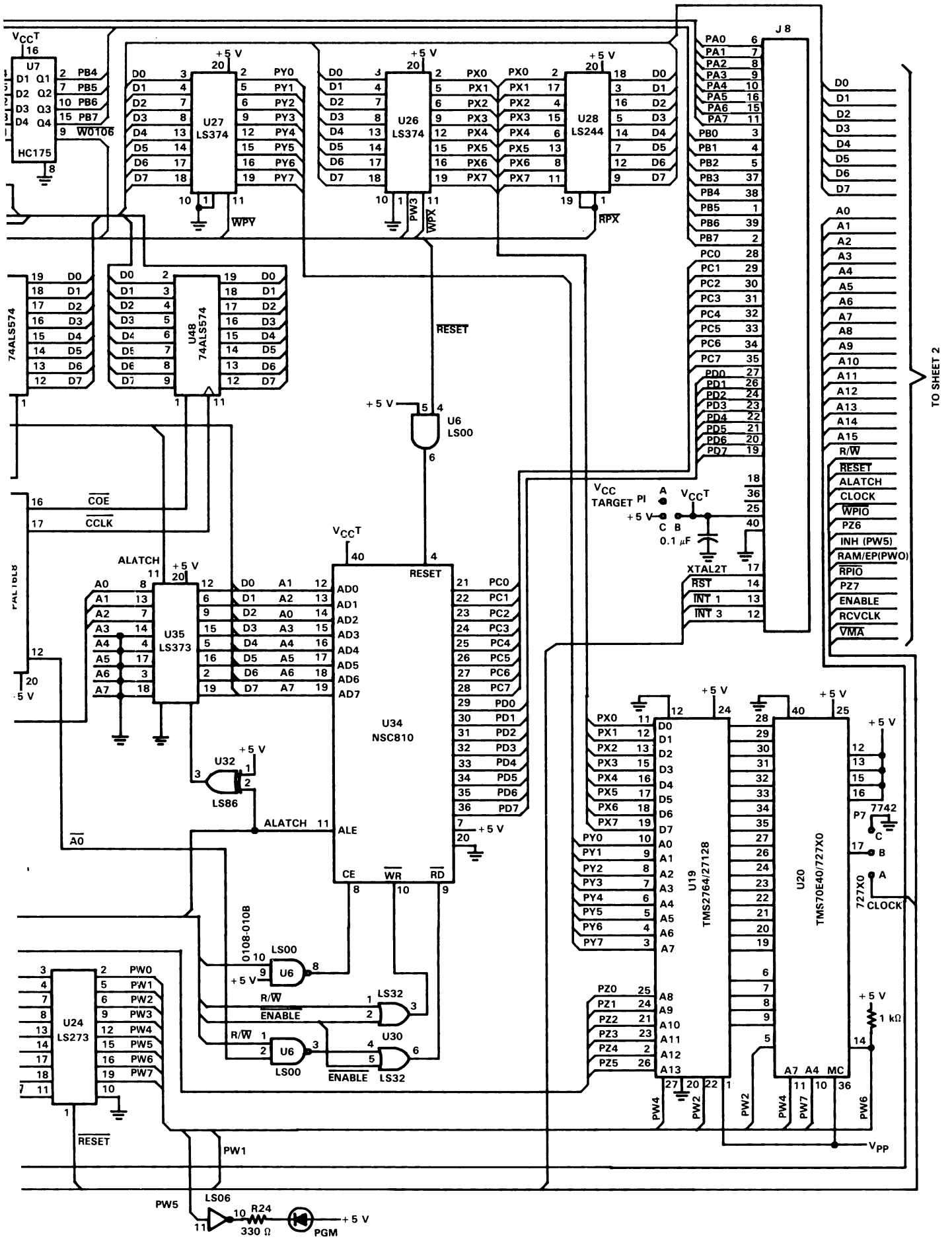


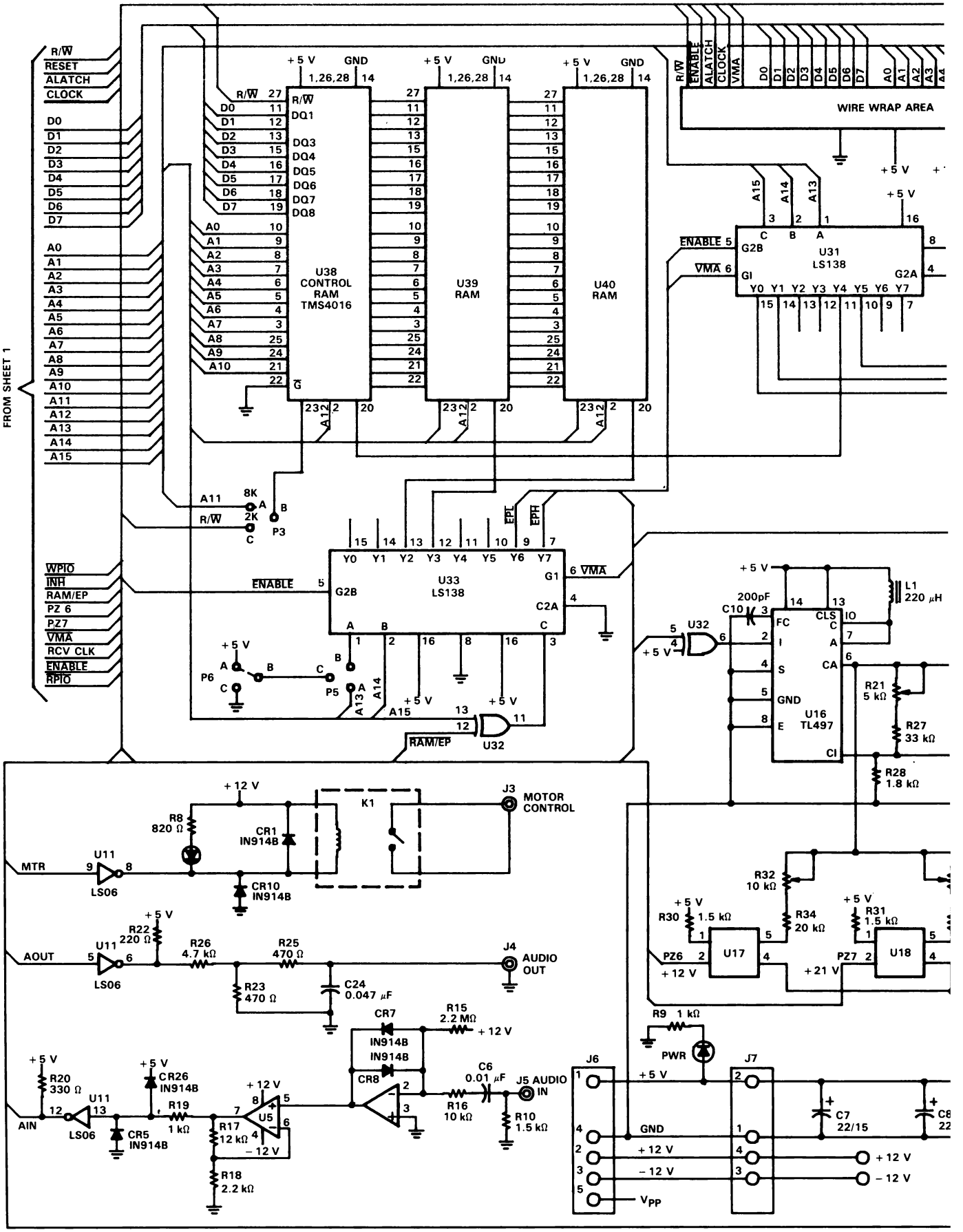
RTC/EVM7000N-1 (Page 2 of 2)


WIRE WRAP AREA
+5 V
+12 V
-12 V
VPP
U31
LS138
U43
TMS2764
U44
TMS2764
U45
EPROM
EXPAN-
SION
U41
RAM
U42
RAM
EPH
EPL
U16
TL497
L1
220 µH
R21
5 kΩ
R27
33 kΩ
R28
1.8 kΩ
C14
0.1 µF
C13
0.1 µF
C9
22
35V
50 Ω
½W
VPPon
U11
R32
R33
100 kΩ
R34
20 kΩ
R31
1.5 kΩ
R35
120 kΩ
U18
PZ7
+21 V
C7
22/15
C8
22/15
U9
LS374
U10
LS244
WPIO
RPIO
MTR
AOUT
AIN
DATA OUT
DTR
PORT ½ SELECT
DSR
DATA IN
RCV CLK
R2
1 kΩ
R44
1 kΩ
R3
2.2 kΩ
R12 2.2 kΩ
R14 2.2 kΩ
R11 1 kΩ
PDCD
R1
1 kΩ
R43
1 kΩ
R4
2.2 kΩ
R13 2.2 kΩ
R6 2.2 kΩ
R5 1 kΩ
1488
1489
LS241
J1
J2

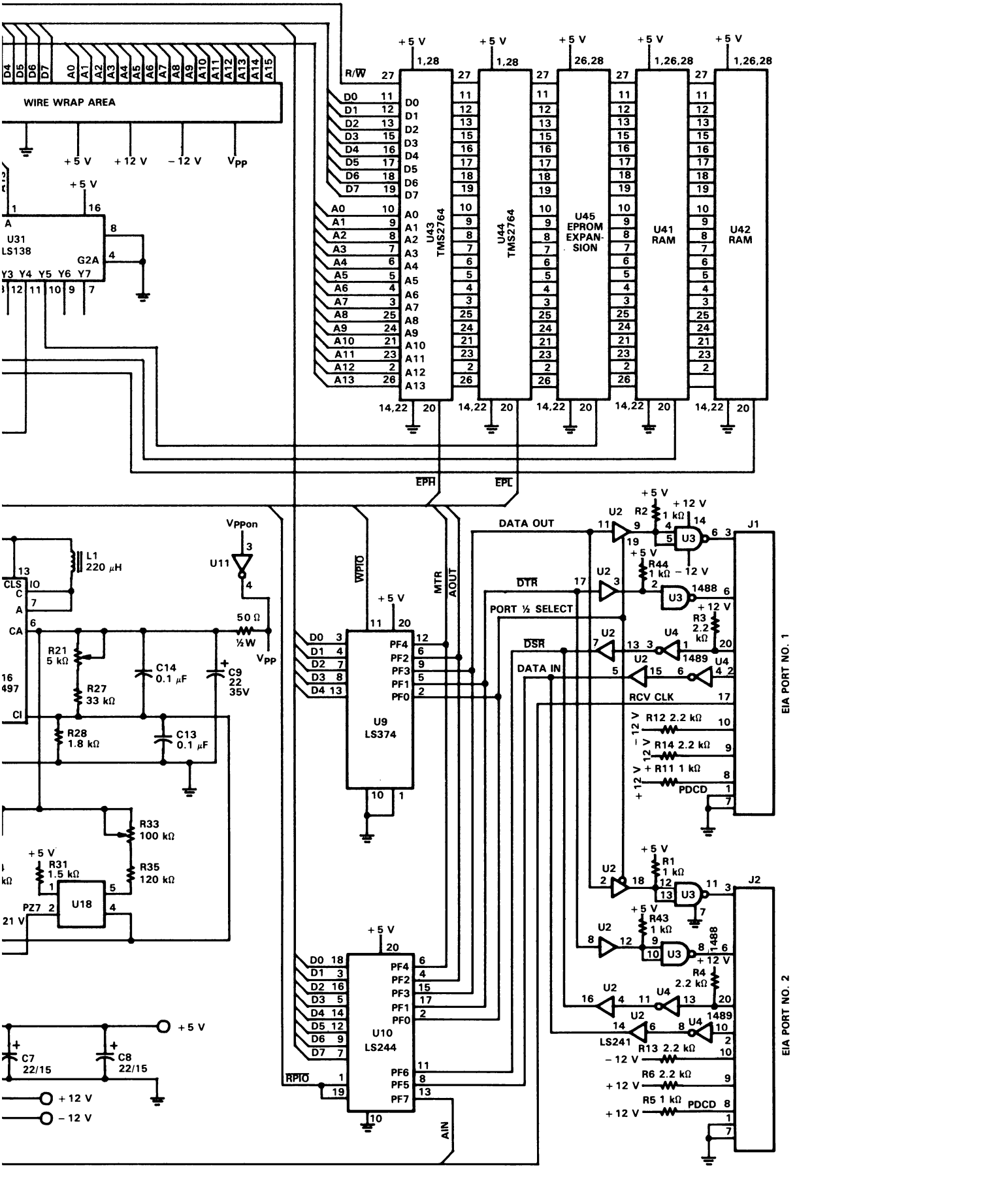


RTC/EVM7000C-1 (Page 1 of 2)


U7
HC175
U27
LS374
U26
LS374
U28
LS244
J 8
TO SHEET 2
U48
74ALS574
U6
LS00
U35
LS373
U34
NSC810
U32
LS86
U24
LS273
U19
TMS2764/27128
U20
TMS70E40/727X0
7742
XTAL2T
RESET
ALATCH
CLOCK
WPIO
PZ6
INH (PW5)
RAM/EP(PWO)
RPIO
PZ7
ENABLE
RCVCLK
VMA
R/W
COE
CCLK
ALE
CE
WR
RD
WPY
WPX
RPX
PW3
0108-0108
R24
330 Ω
1 kΩ
PGM
LS06
LS32
LS273
+5 V
VCC TARGET PI
0.1 μF
CLOCK



RTC/EVM7000C-1 (Page 2 of 2)


WIRE WRAP AREA
+5 V
+12 V
−12 V
VPP
U31 LS138
G2A
R/W
U43 TMS2764
U44 TMS2764
U45 EPROM EXPANSION
U41 RAM
U42 RAM
EPH
EPL
VPPon
U11
L1 220 µH
50 Ω ½W
R21 5 kΩ
R27 33 kΩ
C14 0.1 µF
C9 22 35V
R28 1.8 kΩ
C13 0.1 µF
R33 100 kΩ
R31 1.5 kΩ
R35 120 kΩ
U18
C7 22/15
C8 22/15
WPIO
MTR
AOUT
U9 LS374
U10 LS244
RPIO
AIN
DATA OUT
DTR
PORT ½ SELECT
DSR
DATA IN
RCV CLK
PDCD
R1 1 kΩ
R2 1 kΩ
R3 2.2 kΩ
R4 2.2 kΩ
R5 1 kΩ
R6 2.2 kΩ
R11 1 kΩ
R12 2.2 kΩ
R13 2.2 kΩ
R14 2.2 kΩ
R43 1 kΩ
R44 1 kΩ
1488
1489
LS241
J1
J2
EIA PORT NO. 1
EIA PORT NO. 2

B. Command Format Summary

Appendix B

B.1 Debug Monitor Command Format Summary

In all Monitor command strings, the Monitor prints the space immediately after the command to indicate recognition of the command. All other spaces are user-entered. Default values are given in parentheses.

B.1.1 General Utilities

COMMAND	FUNCTION
AR <number> <number>	+/- Hex Arithmetic
CL *cursor-left*<cursor-left><CR>	Display/Modify Cursor-Left
CU *cursor-up*<cursor-up><CR>	Display/Modify Cursor-Up
DC <decimal number><CR,SP>	Decimal-Hex Byte Conversion
DV *device type index* <device index><CR,SP>	Display/Modify Device Type
HC <hex number>	Hex-Decimal Word Conversion
HE {**M**,E} {port **1**,2}	Help
HS *current value* {0=Disable, 1=XON, 2=ACK, 3=Both}	Display/Modify Software Handshake

B.1.2 Memory Load/Dump Commands

- Address entries are in hexadecimal notation
- Port selection can be defaulted by entering <CR> or <SP> in place of a port number

COMMAND	FUNCTION
DS {port **1**,2,3}	Display/Save Machine State
LM {port 1,2,**3**}	Load Memory - 7000 Format
LS {port 1,2,**3**}	Load Machine State
LT {port 1,2,**3**}	Load Memory - Tektronics Format
SM <start addr> <stop addr> {port 1,2,**3**}	Save Memory - 7000 Format
ST <start addr> <stop addr> {port 1,2,**3**}	Save Memory - Tektronics Format

B.1.3 General Memory/Register Manipulation Commands

- Address entries are in hexadecimal notation
- Default values are given in parentheses

COMMAND	FUNCTION
DM <start addr> <stop addr> {port **1**,2,3}	Display Memory
FB <start addr> <stop addr> <value><CR,SP>	Find Byte in Memory
FM <start addr> <stop addr> <value><CR,SP>	Fill Memory
FR <start reg> <stop reg> <value><CR,SP>	Fill Register File
IO {port **1**,2}	Display I/O Status
MV <start addr> <stop addr> <dest addr><CR,SP>	Move Memory
NP	Fill Memory with NOPs

B.1.4 Register Modify/Display Commands

- Register/peripheral entries are in decimal notation
- Program Counter (PC) and Status (ST) entries are in hexadecimal notation
- Stack Pointer (SP) entries are in decimal (register) notation
- Memory entries are in hexadecimal notation

After the command is executed:

- <SP> accesses subsequent locations
- "+" accesses subsequent locations (except MA, MS)
- "-" accesses previous locations (except MA, MS)
- <CR> returns to the Monitor

COMMAND	FUNCTION
CP	Clear Processor Status
DP	Display Processor Status
MA	Display/Modify A Register
MB	Display/Modify B Register
MM <start addr><CR,SP>	Display/Modify Memory
MP <peripheral register><CR,SP>	Display/Modify Peripheral File
MR <register><CR,SP>	Display/Modify Register File
MS/PC/SR/SP	Display/Modify PC, ST, and SP

B.1.5 Program Support Commands

COMMAND	FUNCTION
BT <vector> ... <vector><CR>	Set Breakpoints on TRAP
B1 <addr><CR,SP>	Set Breakpoint 1
B2 <addr><CR,SP>	Set Breakpoint 2
CB	Clear Breakpoints
CT {vector numbers,A}	Clear Breakpoint on Trap
C1	Clear Breakpoint 1
C2	Clear Breakpoint 2
DB	Display Breakpoints
DT	Display Breakpoint on Trap
EF <display type> [event count]<CR>	Execute Program with Fixed Display
ET [event count]<CR,SP>	Execute Program with Breakpoints/Trace
EX [event count]<CR,SP>	Execute Program with Breakpoints
FS <display type> <step count><CR,SP>	Single-Step Program with Fixed Display
LA <line number><CR,SP>	Show Address of Line
LL <line number> <count><CR,SP>	List Line(s) from Editor
LN <addr><CR,SP>	Show Line at Address
L1 <line number><CR,SP>	Set Breakpoint 1 by Line Number
L2 <line number><CR,SP>	Set Breakpoint 2 by Line Number
RT	Reset Target Processor
RU	Execute Program Without Breakpoints
SS <count> {port **1**,2}	Single-Step Program
TC	Configure Single-Step Trace
TR	Display Trace Line
TS <count> {port **1**,2}	Single-Step Program with Trace
T0	Load Program Counter with TRAP 0 Vector

B.1.6 EIA Support Command

The Port 1 baud rate is determined automatically at power-up/reset by entering <CR>. Port 2 baud rate defaults to 9600 at power-up.

COMMAND	FUNCTION
BR *(br1,br2)* <port> <index>	Display/Modify Baud Rate

B.1.7 Audio Tape Commands

COMMAND	FUNCTION
DR {port **1**,2}	Audio Tape Directory
MO	Enable Cassette Motor

B.1.8 EPROM Programmer Commands

- Address entries are in hexadecimal notation
- Destination is '4' for TMS2764 (default) and '8' for TMS27128.

COMMAND	FUNCTION
CE <EPROM start> <EPROM stop> <mem start> {**4**,8}<CR,SP>	Compare EPROM
PE <EPROM start> <EPROM stop> <mem start> {**4**,8}<CR,SP>	Program EPROM
RE <EPROM start> <EPROM stop> <mem start> {**4**,8}<CR,SP>	Read EPROM
VE <EPROM start> <EPROM stop> {**4**,8}<CR,SP>	Verify EPROM
BC	Clear 727x0 EPROM
12	Calibrate 12 VPP
21	Calibrate 21 VPP
4x	Copy U4x into U19

B.1.9 TMS7000 Assembler Support Commands

COMMAND	FUNCTION
AT {port **1**,2}	Display Assembler Label Table
XA {port **0**,1,2,3} {port 0,**1**,2} [N]	Execute Assembler
XL	Execute Line-by-Line Assembler
XP	Execute Patch Assembler

B.1.10 TMS7000 Text Editor Support Command

COMMAND	FUNCTION
XE {port **0**,1,2,3} [N]	Execute Text Editor

B.2 Text Editor Command Format Summary

In all Text Editor Command Strings, the Text Editor prints the space immediately after the command to indicate recognition of the command. All other spaces are user-entered.

B.2.1 General Utilities

COMMAND	FUNCTION
H {port **1**,2}	Help
I {port 1,2,**3**}	Input File to the Editor
M	Display Free RAM Remaining
Q {port **0**,1,2,3} [N	Quit Edit and Save File
T *current tab value* <tab value><CR,SP>	Display/Modify Tab
Z	Initialize Text Editor

B.2.2 Line Manipulation Commands

- Line numbers are optional
- If no line number is entered, then the current line is assumed (except "A", "R", and "F")
- For "A", the highest unused line number is assumed
- For "F" and "R", the lowest used line number is assumed

COMMAND	FUNCTION
+	Line Number Pointer to EOF
-	Line Number Pointer to BOF
=	Display Current Line Number
[line number]A	Autoincrement Line Number Mode
[line number]R <increment>	Resequence Line Numbers
<line number>L <# of lines><CR,SP>	List Line(s) to Terminal
<line number>C <line number>	Change Line Number
<line number>D <line number>	Duplicate Line
[line number]F <string>	Find Character String
<line number><CR>	Delete Line
<line number>E	Edit Line

B.2.3 Edit Line Commands

For Insert and Delete commands, 110 to 1200 baud manual Insert/Delete allows up to 9 characters. For 2400 to 9600 baud interactive Insert/Delete, only one character is allowed.

COMMAND	FUNCTION
<cursor-right>	Cursor-Right Character
<cursor-left>	Cursor-Left Character
<home>	Cursor Home Character
<tab>	Tab Right
<back tab>	Tab Left
CNTL-E	Undo Line
CNTL-N{<ESC>,1-9}	Insert Character(s)
CNTL-D<# or characters>	Delete Character(s)
<RUB> or <DEL>	Delete Previous Character
<CR>	Save Line
<cursor-down>	Save Line/Edit Next Line
<cursor-up>	Save Line/Edit Previous Line

C. EVM Walkthrough

This section helps you install and operate the EVM. Topics covered include installation and initialization, command operations, program development and debug, and EPROM programming.

Appendix C

C.1 Notation

The syntax conventions described below are used throughout this section to simplify the use of the commands.

- Parentheses () placed before "Enter" statements are provided to check off the steps as you progress.
- Angle brackets (<>) mean keys to press or parameters to enter:

  UPPER CASE letters = press indicated key (e.g., <CR> means press carriage return)

  lower case letters = parameter to enter (e.g., <reg> means enter a register number).
- The "¬" symbol represents the position of the cursor in the display.
- A parameter *in italics* indicates that the current value of the parameter is displayed.
- Information to be entered from the keyboard is preceded by "Enter:". If this entry requires a space or carriage return as the final character, the <SP> or <CR> symbol will follow the text to be entered.
- Information following the "Display:" banner should appear on your monitor after entering commands and parameters.

The following are some examples of formats which will be used throughout this section.

CMD
CMD *previous value* <new value>
CMD <addr 1> <addr 2> <value>

For example, a Fill Memory (FM) command used to fill RAM locations >F600 to >FA00 with the value >33. The following is the format:

FM <addr 1> <addr 2> <value><CR>

The command you would enter:

```
FM F600 FA00 33<CR>
```

Note:

The space between the command (e.g., FM) and the first parameter (e.g., F600) will be provided by the EVM Monitor for all commands which require additional parameters. Do not enter this space.

C.2 Walkthrough Exercises

The following Walkthrough Exercises introduce you to the TMS7000 EVM and to some of the following commonly used features of the EVM system:

- Installation
- Device initialization
- Status commands
- Memory manipulation commands
- Arithmetic logic commands
- Program development
  1) Editing and assembling
  2) Breakpoint and trace functions
- EPROM control commands

C.2.1 EVM Installation

Use the following procedure to power up the EVM (see Section 2.5 for suggested power supply and Figure 1-1 for the port configuration).

( ) Connect power supply to EVM port J7 or J6.

( ) Connect terminal to EVM Port 1 (J1) using a standard RS-232-C cable.

( ) Turn on power supply. The "Power" LED (next to J7) should light if the power supply is properly connected.

( ) Turn on video terminal.

( ) Toggle the EVM RESET switch (SW1).

( ) Enter: <CR>

Display:

```
TMS7000 DEBUG MONITOR REV 2.X
DEVICE TYPE = 2 (704X)
SYSTEM RAM = 32256 BYTES
HELP   : HE /H $HE
MODIFY : MM/IM MR/IR MP/IP   A/MA B/MB MS-P/PC,SP
DISPLAY: DM/DH DS IO DV
DEVICE : C/CP D/DP   EI DI   DV
STATE  : C/CP D/DP xC,xN,xZ,xI (x=C/S) EI DI SR
SAVE   : SM ST DS
LOAD   : LM LT LS
MOVE   : MV
FIND   : FB
FILL   : FM NP   FR
RESET  : RT TO
BRKPT  : B1 B2 C1 C2 CB DB   BT CT DT
TRACE  : TO TF IT PT IS   TC TR
DEBUG  : SS TS FS CS   CY GO EX ET EF   RU
EDITOR : XE   L1 L2 LL LA LN
ASSM   : XA XL XP   AT   XR
EPROM  : PE VE CE RE   BC   12 21 25   43 44 45
MATH   : HC DC AR
```

```
PORTS   : BR    DR MO
?
```

If this display does not appear, confirm proper terminal and power supply connections and execute the installation procedure again. If your system is a revision less than 1.4, the display will not appear.

C.2.2 Device Initialization

This section of the Walkthrough Exercises is designed to demonstrate the commands that are used to initialize the EVM.

DV (Device Type). This command is used to display or modify the device being emulated by the EVM.

( ) Enter: DV

Display: `DEVICE TYPE = x (7xxx) ¬`

The type (represented by "x" after the "=") should be 2 for a TMS704x device type. If the default value is not 2, changed it to 2 to emulate a TMS704x or a TMS70C4x (see Table C-1).

Table C-1. TMS7000 Family Device Types

INDEX	DEVICE	MEMORY START FOR PC/AORG	MINIMUM RAM REQUIRED
1	TMS7020,'70C20	>F806	8K bytes
2	TMS704x,'70C4x	>F006	16K bytes
3	Reserved	>E006	16K bytes
4	Reserved	>D006	16K bytes
5	Reserved	>C006	24K bytes

If the present default value is 2 then:

( ) Enter: <CR>

If the default value is not 2 then:

( ) Enter: 2<CR>

Display:
```
DV
DEVICE TYPE = x (7xxx) 2 <CR>
PC=F006  C=0 N=0 Z=0 I=0  SP=R1  A=00 B=00
```

If you wish to change the default values for the cursor-up and the cursor-left characters, continue with this section. Otherwise, go to Section C.2.3.

All alphanumeric and most control keys can be entered for cursor control. New characters entered will not be displayed (echoed) during entry. The next CU or CL command will display the ASCII code of current ASCII character(s) used. An error can cause the command to revert back to the original defaults of "0B" (vertical tab) for CU and "08" (back space) for CL.

CU (Display/Modify Cursor-Up Character). This command allows you to set the character or character sequence to be recognized by the terminal as a cursor up.

( ) Enter: CU

Display: `CU 0B000000 ¬`

The display shows the ASCII code of the current character used. You may change the code by entering one character or a string of up to three characters. "Arrow" keys are usually chosen for simplicity. If you don't want to change the cursor-up character, enter <CR>.

( ) Enter: <chosen character(s)> <CR>

CL (Display/Modify Cursor-Left Character). This command allows you to set the character or character sequence to be recognized by the terminal as a cursor left.

( ) Enter: CL

Display: `CL 08000000 ¬`

You may enter a single character or a string of three characters. "Arrow" keys are usually chosen for simplicity. If you don't want to change the cursor-left character, enter <CR>.

( ) Enter: <chosen character(s)> <CR>

C.2.3 Status Commands

This section demonstrates the commands that verify and manipulate processor status.

DP (Display Processor Status). This command displays the contents of the Program Counter (PC), Status Register (ST), Stack Pointer (SP), and Registers A and B of the presently selected default device.

( ) Enter: DP

Display:
```
PC=F006  C=0 N=0 Z=0 I=0  SP=R1  A=XX  B=XX
?¬
```

MS (Modify Processor Status). This command is used to display and modify the processor status (PC, ST, SP, and Registers A and B).

We will now change the values of the processor status:

( ) Enter: MS

Display: `PC=F006 ¬`

( ) Enter: F806<SP> (set PC to >F806)

Note:

If a space is entered at the end of an MS line, the next entry line will be displayed. If a <CR> is entered, control returns to the ? prompt.

Display: `SP=R1 ¬`

( ) Enter: 66<SP> (set Stack Pointer to R66)

Note: If more than two hex digits are entered, the Monitor will recognize only the last two digits entered.

Display: `A=00 ¬`

( ) Enter: 7C<SP> (set Reg. A to >7C)

Display: `B=00 ¬`

( ) Enter: 8<CR> (set Reg. B to 8)

If the <SP> is used instead of the <CR> to terminate MS command, the PC would be displayed again.

To verify the changes made to the processor status, re-execute the DP (Display Processor Status) command:

( ) Enter: DP

Display:
```
PC=F806  C=0 N=1 Z=0 I=1  SP=R66  A=7C  B=08
?¬
```

CP (Clear Processor Status). This command restores the PC, ST, SP, and Registers A and B to their default values.

( ) Enter: CP

Display:
```
PC=F006  C=0 N=0 Z=0 I=0  SP=R1  A=00  B=00
?¬
```

IO (Display IO Status). The IO command is used to display the values in the Peripheral File locations P0 to P31, in hex and binary.

( ) Enter: IO<CR>

Your display may differ from what is shown.

Display:

```
?IO

PC=F006  C=0 N=0 Z=0 I=0  SP=R1  A=00  B=00

P0= 00 (00000000)       P16=00 (00000000)
P1= 00 (00000000)       P17=05 (00000101)
P2= 00 (00000000)       P18=00 (00000000)
P3= 00 (00000000)       P19=00 (00000000)
P4= 3F (00111111)       P20=08 (00001000)
P5= 00 (00000000)       P21=FF (11111111)
P6= FF (11111111)       P22=00 (00000000)
P7= FF (11111111)       P23=F6 (11110110)
P8= FF (11111111)       P24=F7 (11110111)
P9= 00 (00000000)       P25=F6 (11110110)
P10=FF (11111111)       P26=F7 (11110111)
P11=00 (00000000)       P27=F6 (11110110)
P12=F6 (11110110)       P28=F7 (11110111)
P13=F7 (11110111)       P29=F6 (11110110)
P14=F6 (11110110)       P30=F7 (11110111)
P15=F7 (11110111)       P31=FF (11111111)

?¬
```

C.2.4 Memory Manipulation Commands

This section demonstrates the commands that display and/or modify the processor memory locations.

FR (Fill Register File). This command is used to fill registers in the Register File with a hex value.

Fill Registers 0 through 127 with >AA:

( ) Enter: FR 0 127 AA<CR>

MA (Modify Register A). This command is used to display/modify the contents of Register A (also called Register 0).

( ) Enter: MA

Display: `A=AA ¬`

( ) Enter: A8<CR> (set RA to >A8)

MB (Modify Register B). This command is used to display/modify the contents of Register B (also called Register B).

( ) Enter: MB

Display: `B=AA ¬`

( ) Enter: 3C<CR> (set RB to >3C)

Now, display the processor status to see the changes made by the MA and MB instructions:

( ) Enter: DP

```
Display:    PC=F006  C=0 N=0 Z=0 I=0  SP=R1  A=A8  B=3C
            ?¬
```

MR (Modify/Display Register File). This command is used to display/modify internal registers.

Change the value in Registers 37 and 38:

```
( ) Enter:  MR 37<CR>

Display:    ?MR 37
            R37=AA (10101010) ¬

( ) Enter:  7A<SP>                          (set R37 to >7A)
```

If <SP> is used, the value >7A will be entered into R37 and the display will prompt for R38. If the <CR> is used instead of <SP>, the MR function will be exited.

```
Display:    ?MR 37
            + R37=AA (10101010) 7A  <SP>
              R38=AA (10101010) ¬

( ) Enter:  34<CR>                          (set R38 to >34)

Display:    ?MR 37
            + R37=AA (10101010) 7A  <SP>
              R38=AA (10101010) 34  <SP>
            ?¬
```

MP (Display/Modify Peripheral File). This command is used to display and optionally modify any peripheral register in the range of P0 to P31.

Enter a value in timer 1 data latch (P2):

```
( ) Enter:  MP 2<CR>

Display:    ?MP 2
            P2=00 (00000000) ¬

( ) Enter:  3F<CR>                          (set P2 to >3F)

Display:    ?MP 2
            P2=00 (00000000) 3F  <CR>
            ?¬
```

NP (Fill Memory with NOPs). This command is used to fill the program memory with the NOP opcode (>00).

```
( ) Enter:  NP

Display:    ?NP
            ARE YOU SURE? (N) ¬
( ) Enter:  Y
```

DM (Display Memory). This command is used to display the hexidecimal contents of memory.

Display the contents of memory between >F006 and >F024 inclusive:

( ) Enter: DM F006 F024<CR>

Display:

```
F000  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
F010  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
F020  00 00 00 00  00
```

```
?┐
```

As you can see, the memory display does not start at >F006, but at >F000. This is because the memory dumps always start on the zero nibble boundary of the start address (>XXX0). Therefore, when >F006 was entered, >F000 was interpreted as the first address. However, the DM command will stop on whatever address is given as the stop address in the command, and not necessarily on the zero nibble boundary.

FM (Fill Memory). This command is used to fill program memory locations with any hex value.

Fill memory between >F006 and >F020 with >41:

( ) Enter: FM F006 F020 41<CR>

To see the changes made by the FM command, display the memory dump by entering the following:

( ) Enter: DM F006 F030<CR>

Display:

```
F000  XX XX XX XX  XX XX 41 41  41 41 41 41  41 41 41 41
F010  41 41 41 41  41 41 41 41  41 41 41 41  41 41 41 41
F020  41 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
F030  00
```

```
?┐
```

MV (Move/Copy Memory). This command is used to move (copy) a block of memory specified by the start address and the stop address, inclusive, to memory specified by the destination start address.

Copy the memory between >F006 and >F030 inclusive to memory starting at >F806:

( ) Enter: MV F006 F030 F806<CR>

To see if the MV command worked correctly, display memory with the DM command:

( ) Enter: DM F806 F840<CR>

Display:

```
F800  00 00 00 00  00 00 41 41  41 41 41 41  41 41 41 41
F810  41 41 41 41  41 41 41 41  41 41 41 41  41 41 41 41
F820  41 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
F830  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
F840  00

?┐
```

MM (Modify Memory). This command is used to display and optionally modify program memory.

Modify memory location >F810.

( ) Enter: MM F810<CR>

Display:
```
?MM F810
  F810=41 (01000001) ┐
```

( ) Enter: 42<SP>

Note: If a <SP> or <+> is used as as the entry terminator, the command continues on the next memory location. If a <-> is used, the command continues to the previous memory location. If the <CR> is used, the command will be terminated after performing the operation.

Display:
```
?MM F810
+ F810=41 (01000001) 42
  F811=41 (01000001) ┐
```

( ) Enter: 43<CR>

Display:
```
?MM F810
+ F810=41 (01000001) 42
  F811=41 (01000001) 43 <CR>
?┐
```

To see the changes made by the MM command, display memory using the following:

( ) Enter: DM F800 F812<CR>

Display:

```
F800  00 00 00 00  00 00 41 41  41 41 41 41  41 41 41 41
F810  42 43 41

?┐
```

FB (Find Byte). This command is used to find the occurance(s) of a value in the memory limits specified in the command parameters.

Find the hex value >42 between >F800 and >F840:

( ) Enter: FB F800 F840 42<CR>

Display:
```
?FB F800 F840 42
  F810=42 (01000010) ┐
```

The value at >F810 can now be changed:

( ) Enter: 44<CR>

Display:
```
?FB F800 F840 42
  F810=42 (01000010) 44
¬
```

When the FB command is entered, the command displays the first location of the byte value. Subsequent <CR> entries will display further locations of the byte (to address >F840).

C.2.5 Arithmetic Logic Commands

This section of the Walkthrough Exercises shows the commands that provide arithmetic operations.

HC (Hex-Decimal Word Conversion). This command converts a hex value (up to four digits in length) to decimal format.

Convert >D3A8 to decimal:

( ) Enter: HC D3A8<CR>

Display:
```
?HC D3A8
  >D3A8=54184
?¬
```

DC (Decimal-Hex Byte Conversion). This command converts a decimal number (up to 255) to hex format.

Convert 183 to hex:

( ) Enter: DC 183<CR>

Display:
```
?DC 183
183=>B7
?¬
```

AR (Hex Arithmetic +/-). This command displays the sum and the difference of numbers entered. The numbers are in hex and have up to four digits.

Hexidecimally add and subtract >34C1 and >6F2A:

( ) Enter: AR 34C1 6F2A<CR>

Display:
```
?AR 34C1 6F2A
34C1+6F2A=A3EB   34C1-6F2A=C597
?¬
```

C.2.6 Use Text Editor to Develop Program

This section of the Walkthrough Exercises demonstrates commands for developing, editing, and saving TMS7000 programs. The sample program is a real-time clock that counts seconds, minutes, and hours, and will be used to demonstrate the breakpoint and trace functions of the EVM.

XE (Execute Text Editor). This command is used to enter the EVM Text Editor. To exit from the Editor back to the Monitor, press <ESC>. Enter the EVM Text Editor:

( ) Enter: XE<CR>

Display:
```
EVM TEXT EDITOR      (Text Editor banner)
22016                (Bytes available for text storage)
*¬                   (* = Text Editor command prompt)
```

Z (Initialize Text Editor). This Text Editor command clears all text from memory and initializes the Text Editor workspace pointers.

Initialize the Text Editor:

( ) Enter: Z

Display:
```
*Z
ARE YOU SURE? (N) ¬
```

( ) Enter: Y

Display:
```
*Z
ARE YOU SURE? (N) Y
22016
*¬
```

> **Caution:**
>
> **The Z command clears text in memory. The XE (Execute Editor) command does not destroy values in memory. To exit the Text Editor and go to the Monitor, press "Q" and <CR>. To re-enter at the line addressed at exit, enter "XE<CR>", then "A".**

A (Autoincrement Line Number Mode). This Text Editor command allows entry of text beginning at a specified line number. A new number, incremented by a set value, is printed for each text line.

Enter the Autoincrement mode:

( ) Enter: A

Display:
```
*A
0010 ¬
```

Walkthrough Test Program. The following Walkthrough test program - a real-time clock which counts seconds, minutes, and hours - will now be entered. This program will demonstrate the editing, assembling, and debug commands. Comments have been added to help explain this program, although these comments do not need to be entered. The label, mnemonic, and opcode fields must be included. By omitting comments, the effective number of code lines will decrease from 75 to 31. The first line continues after the "A" Text Editor command was issued. The 0010 is the first line number. Enter the label as the first entry on a line; if no label, enter a <SP> or tab over to the mnemonic field, as applicable.

( ) Enter:

```
0010        IDT   'CLOCK'
0020        AORG  >F006              THIS IS WHERE THE ASSEMBLER
0030 *                                WILL ORIGINATE THIS PROGRAM.
0040 *
0050 *      *REORIENT THE STACK FROM R1 TO R100*
0060 *
0070        MOV   %>64,B             LOAD REGISTER B WITH >64.
0080        LDSP                     ORIENT STACK POINTER TO R100.
0090 *
0100 *      *INITIALIZE TIMERS AND REGISTER STORAGE AREAS:
0110 *            THE 'SECONDS' COUNT WILL BE STORED IN 'R3'.
0120 *            THE 'MINUTES' COUNT WILL BE STORED IN 'R4'.
0130 *            THE 'HOURS' COUNT WILL BE STORED IN 'R5'.
0140 *
0150 START MOVP  %>08,P0            MODE IS SINGLE CHIP
0160 *                                AND INTERRUPT 2
0170 *                                IS  CLEARED.
0180        MOVP  %>08,P16           INT4 DISABLED, ENABLE INT5.
0190        MOVP  %>63,P2            STORE >63 IN T1 DATA
0200 *                                REGISTER.
0210        MOVP  %>7C,P18           STORE >7C IN T2 DATAREG;
0220 *                                TIMERS ARE NOW INITIALIZED.
0230        CLR   R3                ERASE R3 AND R4 -- INITIALIZES
0240        CLR   R4                   MINUTE AND SECOND COUNTERS.
0250        MOV   %>12,R5           R5 INTIIALIZED FOR HOUR COUNT.
0260        MOV   %>3C,R6            TEST REGISTERS R6 AND R7
0270        MOV   %>3C,R7              TO EQUAL 60.
0280 *
0290 *      *LOAD TIMER PRESCALES, AND START TIMERS*
0300 *
0310        MOVP  %>98,P3            LOAD TIMER 1 PRESCALE WITH
0320 *                               >18, AND START TIMER 1.
0330        MOVP  %>A0,P19           START TIMER 2.
0340 *
0350 *      * BEGIN CHECKING FOR TIMER 2 INTERRUPT, WHICH WILL
0360 *      * SIGNAL THE LAPSE OF ONE (1) SECOND.
0370 *
0380 CHECK BTJZP %>08,P16,CHECK     TEST TO SEE WHEN TIMER 2
0390 *                                COUNTS DOWN TO ZERO (0).
0400        MOVP  %>08,P16           CLEAR TIMER 2 INT. FLAG.
0410 *
0420 *      * MODIFY THE 'SECONDS' STORAGE REGISTER (R3).
0430 *
0440 SEC    DAC   %>01,R3            ADD 1 TO REGISTER 3,
0450 *                                WHICH IS THE 'SECONDS'
0460 *                                STORAGE AREA.
0470        DJNZ  R6,CHECK           JUMP TO 'CHECK' UNTIL R6
0480 *                                HAS COUNTED DOWN FROM >3C.
0490        CLR   R3                 RESET THE 'SECONDS' REGISTER
0500 *                                TO ZERO (0).
0510        MOV   %>3C,R6            RELOAD R6 WITH >3C.
0520 *
0530 *      * MODIFY THE 'MINUTES' STORAGE REGISTER (R4).
```

```
0540 *
0550 MIN    DAC    %>01,R4           ADD DECIMAL 1 TO REGISTER 4,
0560 *                                 WHICH IS THE 'MINUTES'
0570 *                                 STORAGE REGISTER.
0580        DJNZ   R7,CHECK          JUMP TO 'CHECK' UNTIL R7
0590 *                                 HAS COUNTED DOWN FROM >3C.
0600        CLR    R4                RESET THE 'MINUTES' REGISTER
0610 *                                 TO ZERO (0).
0620        MOV    %>3C,R7           RELOAD R7 WITH >3C.
0630 *
0640 *      * MODIFY THE 'HOURS' STORAGE REGISTER (R5).
0650 *
0660 HOUR   DAC    %>01,R5           ADD DECIMAL 1 TO REGISTER
0670 *                               FIVE (5) WHICH IS THE 'HOURS'
0680 *                                 STORAGE REGISTER.
0690        CMP    %>13,R5           HAS THE 'HOURS' REGISTER
0700 *                                 COUNTED TO BCD >13?
0710        JNE    CHECK             IF NOT, JUMP TO 'CHECK'.
0720        MOV    %>01,R5           IF SO, CHANGE THE 'HOURS'
0730 *                                 REGISTER TO BCD >01.
0740 BACK   JMP    CHECK             JUMP TO 'CHECK'.
0750 END    END                      END OF PROGRAM.
0760 Q<CR>
```

Display: *⌐

Now return to the Monitor by entering a 'Q' and <CR>:

( ) Enter: Q<CR>

Display:
```
TMS7000 DEBUG MONITOR REV 2.X
?⌐
```

This program will be used in further command demonstrations.

C.2.7 Assemble Program

XA (Execute Assembler). This command is used to execute the assembler on a specified file.

Execute the assembler:

( ) Enter:
```
XA<CR>
INITIALIZE? (Y)<CR>
```

The following should appear after the scrolling of a successful assembly of the above program:

Display:
```
0740 F050 E0     BACK JMP CHECK      JUMP TO CHECK
     F051 D6
0750 F052        END  END            END OF PROGRAM

0 ERRORS

TMS7000 DEBUG MONITOR REV 2.X
?⌐
```

The assembled program will scroll on the Monitor. When the assembly is finished, the error statement should read zero if no errors occurred.

C.2.8 Program Debug

TC (Configure Single-Step Trace). This command sets values to be displayed by the Trace Step (TS) command. These values (up to six) may be registers, peripheral registers, or memory addresses.

Set up a trace line showing R5, R4, R3, P16, and >F048.

( ) Enter: TC

Display: ¬

( ) Enter with a space between each entry:

R5 R4 R3 P16 >F048<CR>

> **Note:**
>
> When entering an address, it must be preceded by a hex symbol (>) and be in the device memory default range (>F006 to >FFFF for the TMS7041).

Display:

```
R5        R4        R3        P16       >F048
R5=xx     R4=xx     R3=xx     P16=xx    F048=xx
<CR>
?¬
```

Now that traces have been entered, you will run the clock program in the single-step mode, requiring the use of the TS command. You will also run the program without breakpoints. To accomplish this, an RU, EX, or ET command must be entered. You will use the RU command in this demonstration.

TS (Single-Step Program with Trace). This command is used to display the trace line while single-stepping through the user program.

Begin the single-step program with trace:

( ) Enter: TS<CR>

Display:

```
LAST INST---> 0010  IDT 'CLOCK'
CYCLE COUNT = 000007
NEXT INST---> 0080  LDSP   ORIENT STACK POINTER TO R100.
PC=F008  C=0 N=0 Z=0 I=0  SP=R1 A=00 B=F0
R5=AA (10101010)   R4=00 (00000000)   R3=09 (00001001)
P16=08 (00001000)  >F048=7D (01111101)  ¬
```

Continue to step through the program by pressing the space bar. Do this until the

```
0380   CHECK   BTJZP   %>08,P16,CHECK
```

instruction appears under the "LAST INST--->" banner. To escape from the TS command hit <CR>.

( ) Enter: <CR>

RU (Execute Program without Breakpoints). This command is used to execute a program while ignoring any previously entered breakpoints.

If you would like to check the accuracy of this program, look at a stop watch or clock with a second hand and note the time when you strike the <CR> after 'RU' command. First, however, the processor should be reset to its default values. To do this, use the (CP) Clear Processor Status command.

( ) Enter: CP

Display: `PC=F006  C=0 N=0 Z=0 I=0  SP=R1  A=00  B=00`

Now, start the program:

( ) Enter: RU<CR>

Display: `?RU`
`¬`

To stop the timer, toggle the RESET switch on the EVM. If you are timing this program, note the time when you toggle the RESET switch.

( ) Enter: Toggle the RESET switch.

Display: `STOP AT PC=Fxxx`
`?¬`

To display the traces, enter TC again:

( ) Enter: TC

Display: `R5=xx  R4=xx  R3=xx  P16=xx  >F048=7D`
`<CR>`
`?¬`

The display will list the BCD contents of the hours in R5 (starting with hour 12), minutes in R4, seconds in R3, condition of the Timer 2 control register in P16, and contents of >F048. Unless the program has run for an hour or more, R5 should contain >12.

Now you need to enter breakpoints. Set breakpoints using the B1 and B2 (Set Breakpoint on Address) commands. However, there are two other ways to set breakpoints. To see how, refer to the L1, L2, and BT commands in this manual.

B1 and B2 (Set Breakpoint on Address 1 and 2). Use these commands to set the two breakpoints on addresses. The address entered must be within the range of the default values of the device being emulated. In this case (TMS704x), the range is >F006 to >FFFF.

Set up first breakpoint on address:

( ) Enter: B1 F03D<CR>

Display: `BP1=F03D  BP2=0000`
`?¬`

Set up second breakpoint on address:

( ) Enter: B2 F045<CR>

Display:
```
BP1=F03D  BP2=F045
?
```

DB (Display Breakpoints). This command is used to display the current values of the two breakpoints.

Display the breakpoints:

( ) Enter: DB

Display:
```
?DB
BP1=F03D  BP2=F045
?
```

Now execute the program using breakpoints. The first breakpoint is at >F03D, which is the point where minutes are counted. Execute the program and list the trace line after the breakpoint is recognized. To do this, use the EF command.

EF (Execute Program with Fixed Display). This command is used to execute the program to a breakpoint and update a display in a fixed place on the terminal screen.

Execute the EF command. For this example, the breakpoint was set on the one minute boundary. Therefore, it will take approximately one minute for this example to end.

( ) Enter: EF<CR>

After the breakpoint is recognized, the following should be displayed on your terminal:

Display:

```
NEXT INST---> 0580  DJNZ R7,CHECK  JUMP TO 'CHECK' UNTIL R7
PC=F03D  C=0 N=0 Z=0 I=0  SP=R100 A=F0 B=64  BP1
R5=12 (00010010)  R4=01 (00000001)  R3=00 (00000000)
P16=00 (00000000) >F048=7D (01111101)
```

Enter <CR> to enable the EVM to accept another command; then enter the CP command to set the PC back to the default value of >F006:

( ) Enter: <CR>

( ) Enter: CP

Display:
```
PC=F006  C=0 N=0 Z=0 I=0  SP=R1  A=00  B=00
?
```

You will now change Breakpoint 1 to >F032 (seconds counter) and set up an event counter to cause the breakpoint to be ignored until it has occurred a number of times equal to the event counter.

( ) Enter: B1 F032<CR>

Display:
```
?B1
BP1=F032  BP2=F045
?
```

ET (Execute Program with Breakpoints/Trace). This command displays a trace line whenever a breakpoint is detected. An event count feature of this command is used to cycle over a breakpoint for a specified number of times.

Execute of the ET command. Set the event count to 8, which will cause the processor to recognize the breakpoint after the breakpoint's eighth occurrence.

( ) Enter: ET 8<CR>

The following should appear on your monitor:

Display:

```
NEXT INST---> 0470  DJNZ R6,CHECK   JUMP TO 'CHECK' UNTIL R6
PC=F032  C=0 N=0 Z=0 I=0  SP=R100  A=00  B=64      EV1
R5=12 (00010010)     R4=00 (00000000)     R3=08 (00001000)
P16=00 (00000000)     >F048=7D (01111101)   ┐
```

( ) Enter: <CR>

Please refer to Section 6 to obtain more information on the parameter options available for the breakpoint and trace commands.

C.2.9 EPROM Control Commands

This section of the Walkthrough Exercises is designed to show the uses of the EPROM programmer functions of the TMS7000 EVM. When complete with this section, an EPROM should have been programmed with the program you entered in the last section.

To program an EPROM, one must first be inserted into the 28-pin Zero Insertion Force socket (U19). Any standard TMS2764 EPROM will work.

VE (Verified EPROM Erased). This command is used to verify whether the specified device is empty (>FF in all locations).

Verify that the EPROM is erased:

( ) Enter: VE 0000 1FFF<CR>

Display:
```
VERIFY COMPLETE
?┐
```

If the EPROM is empty, the COMPLETE message will appear in a few seconds. If an ERROR message appears, insert a new EPROM or erase the present EPROM, and verify it again.

PE (Program EPROM from Memory). This command is used to program the specified EPROM.

Begin programming the EPROM:

( ) Enter: PE 1000 1FFF F000<CR>

Display:
```
PROGRAM COMPLETE
VERIFY COMPLETE
?┐
```

The EPROM has now been programmed with the total contents of the EVM designated memory.

CE (Compare EPROM to Memory). This command is used to compare the contents of the specified device with the EVM memory.

Compare the contents of the EPROM to the contents of memory:

( ) Enter: CE 1000 1FFF F000<CR>

Display:
```
COMPARE COMPLETE
?
```

The COMPLETE statement will appear if the contents of the EPROM compare exactly with the contents of the EVM memory. If an ERROR message appears, the contents do not compare, and the EPROM must be erased and reprogrammed.

This completes the TMS7000 EVM Walkthrough Exercises.

D. Crystal Frequency Dependent Constants

Crystal frequency dependent constants used in the EVM are listed in the following tables. These values reside in EPROM starting at location >FFB2 and are accessible with the Monitor command $MM described in Section 6.7. If the crystal frequency is changed, the appropriate values for that frequency must be placed in the table by creating a new EPROM.

The table starts with the software UART triplets stored in registers R122-R127 during Monitor operation.

The three constants called "ONEHI", "ZERHI", and "MIDCYC" are for the audio tape timing. If these values are not changed with the crystal frequency, the EVM will still be able to read tapes that it generates, but may not keep tape compatibility with other EVMs.

The constant "EPCNT" is used in the EPROM programming routines to insure proper delay during programming cycles.

The length of the machine cycle that the EVM is executing is determined by the crystal frequency divided by the built-in divider (either 2 or 4). As the machine cycle gets longer, with a lower frequency crystal or a divide-by-4 part, the resolution of the software UART and the audio tape interface are affected and at certain points rendered inoperable. Table entries for audio tape constants are given as "XX" when the audio tape will not accurately work for the given crystal or divider combination used. Likewise, software UART table entries that will not function properly are given as:

```
XXXX    BYTE    >01
XXXX    BYTE    >01
XXXX    BYTE    >01
```

These entries must be set to >01 so that the autobaud routine can pass them to select one of the lower baud rates.

D.1 Making a New Monitor EPROM

During emulation with internal clock selected, the clock used for operation is the same one that drives the Monitor EPROMs. If a different internal clock frequency is needed, the crystal on the EVM must be changed and the >E000 to >FFFF EPROM (U43) must be reprogrammed with the new values in the frequency-dependent constant table.

For example, suppose that a system was upgraded by substituting a TMS70Cx0AN2L for a TMS70Cx0N2L (on a non-Rev. A board). The crystal frequency is raised from 3.579 MHz to 5 MHz. Replace the crystal and reprogram PROM U43 with the values listed in the applicable software UART table, Table D-7.

This can be accomplished by transferring the contents of the EPROM to RAM with the $MV command, changing the values in the table, and programming a new EPROM (similar to that for changing the Monitor in Section 9.7.) After putting this EPROM in place of the original, change the crystal; the EVM is then ready to run at the new frequency. For example:

```
?$MV E000 FFFF 4000          (move to RAM)
?$MM 5FBx                    (modify in RAM)
     .
     .
     .
?$PE 0 1FFF 4000             (program blank EPROM)
PROGRAMMING COMPLETE
?                            (replace with new EPROM)
```

This procedure is similar to that described in Section 9.7 on 9-7 and used to change the EVM default values listed in Section 2 (e.g., change EPROM programmer destination in Section 2.12 and change Port 2 default baud rate in Section 2.13).

Table D-1. Crystal Dependent Constants: 1 MHz Crystal with Divide-by-2 Oscillator

```
    FFB2  BYTE  >C8            110 BAUD
    FFB3  BYTE  >02
    FFB4  BYTE  >76
    FFB5  BYTE  >91            150 BAUD
    FFB6  BYTE  >02
    FFB7  BYTE  >56
    FFB8  BYTE  >45            300 BAUD
    FFB9  BYTE  >02
    FFBA  BYTE  >27
    FFBB  BYTE  >1F            600 BAUD
    FFBC  BYTE  >02
    FFBD  BYTE  >11
    FFBE  BYTE  >0C           1200 BAUD
    FFBF  BYTE  >02
    FFC0  BYTE  >08
    FFC1  BYTE  >07           2400 BAUD
    FFC2  BYTE  >01
    FFC3  BYTE  >04
    FFC4  BYTE  >01           4800 BAUD
    FFC5  BYTE  >01
    FFC6  BYTE  >01
    FFC7  BYTE  >01           9600 BAUD
    FFC8  BYTE  >01
    FFC9  BYTE  >01
    FFCA  BYTE  >01           AUTO BAUD
    FFCB  BYTE  >01
CASSETTE TAPE TIME CONSTANTS
    FFCC  BYTE  >XX   ZERHI   OUTPUT WAIT COUNT FOR ZERO HIGH
    FFCD  BYTE  >XX   ONEHI   OUTPUT WAIT COUNT FOR ONE HIGH
    FFCE  BYTE  >XX   MIDCYC MID CYCLE THRESH FOR BYTE READ
EPROM PROGRAMMER TIME DELAY VALUE
    FFCF  BYTE  >2D   EPCNT   PROGRAMMING DELAY COUNTER
```

Table D-2. Crystal Dependent Constants: 1 MHz Crystal with Divide-by-4 Oscillator

```
    FFB2  BYTE  >61           110 BAUD
    FFB3  BYTE  >02
    FFB4  BYTE  >3A
    FFB5  BYTE  >45           150 BAUD
    FFB6  BYTE  >02
    FFB7  BYTE  >2A
    FFB8  BYTE  >1F           300 BAUD
    FFB9  BYTE  >02
    FFBA  BYTE  >11
    FFBB  BYTE  >0C           600 BAUD
    FFBC  BYTE  >02
    FFBD  BYTE  >08
    FFBE  BYTE  >01          1200 BAUD
    FFBF  BYTE  >01
    FFC0  BYTE  >01
    FFC1  BYTE  >01          2400 BAUD
    FFC2  BYTE  >01
    FFC3  BYTE  >01
    FFC4  BYTE  >01          4800 BAUD
    FFC5  BYTE  >01
    FFC6  BYTE  >01
    FFC7  BYTE  >01          9600 BAUD
    FFC8  BYTE  >01
    FFC9  BYTE  >01
    FFCA  BYTE  >01          AUTO BAUD
    FFCB  BYTE  >01
CASSETTE TAPE TIME CONSTANTS
    FFCC  BYTE  >XX   ZERHI  OUTPUT WAIT COUNT FOR ZERO HIGH
    FFCD  BYTE  >XX   ONEHI  OUTPUT WAIT COUNT FOR ONE HIGH
    FFCE  BYTE  >XX   MIDCYC MID CYCLE THRESH FOR BYTE READ
EPROM PROGRAMMER TIME DELAY VALUE
    FFCF  BYTE  >16   EPCNT  PROGRAMMING DELAY COUNTER
```

Table D-3. Crystal Dependent Constants: 2 MHz Crystal with Divide-by-2 Oscillator, or 4 MHz Crystal with Divide-by-4 Oscillator

```
    FFB2  BYTE  >CA            110 BAUD
    FFB3  BYTE  >04
    FFB4  BYTE  >ED
    FFB5  BYTE  >C5            150 BAUD
    FFB6  BYTE  >03
    FFB7  BYTE  >AE
    FFB8  BYTE  >60            300 BAUD
    FFB9  BYTE  >03
    FFBA  BYTE  >52
    FFBB  BYTE  >45            600 BAUD
    FFBC  BYTE  >02
    FFBD  BYTE  >27
    FFBE  BYTE  >1F           1200 BAUD
    FFBF  BYTE  >02
    FFC0  BYTE  >11
    FFC1  BYTE  >0C           2400 BAUD
    FFC2  BYTE  >02
    FFC3  BYTE  >08
    FFC4  BYTE  >07           4800 BAUD
    FFC5  BYTE  >01
    FFC6  BYTE  >04
    FFC7  BYTE  >01           9600 BAUD
    FFC8  BYTE  >01
    FFC9  BYTE  >01
    FFCA  BYTE  >01           AUTO BAUD
    FFCB  BYTE  >01
```

CASSETTE TAPE TIME CONSTANTS

```
    FFCC  BYTE  >XX   ZERHI   OUTPUT WAIT COUNT FOR ZERO HIGH
    FFCD  BYTE  >XX   ONEHI   OUTPUT WAIT COUNT FOR ONE HIGH
    FFCE  BYTE  >XX   MIDCYC MID CYCLE THRESH FOR BYTE READ
```

EPROM PROGRAMMER TIME DELAY VALUE

```
    FFCF  BYTE  >59   EPCNT   PROGRAMMING DELAY COUNTER
```

Table D-4. Crystal Dependent Constants: 3.579 MHz Crystal (Color Burst) with Divide-by-2 Oscillator

```
    FFB2  BYTE  >FF            110 BAUD
    FFB3  BYTE  >06
    FFB4  BYTE  >FF
    FFB5  BYTE  >D5            150 BAUD
    FFB6  BYTE  >05
    FFB7  BYTE  >FF
    FFB8  BYTE  >83            300 BAUD
    FFB9  BYTE  >04
    FFBA  BYTE  >9B
    FFBB  BYTE  >55            600 BAUD
    FFBC  BYTE  >03
    FFBD  BYTE  >4D
    FFBE  BYTE  >28           1200 BAUD
    FFBF  BYTE  >03
    FFC0  BYTE  >25
    FFC1  BYTE  >1B           2400 BAUD
    FFC2  BYTE  >02
    FFC3  BYTE  >12
    FFC4  BYTE  >0A           4800 BAUD
    FFC5  BYTE  >02
    FFC6  BYTE  >08
    FFC7  BYTE  >05           9600 BAUD
    FFC8  BYTE  >01
    FFC9  BYTE  >04
    FFCA  BYTE  >09           AUTO BAUD
    FFCB  BYTE  >01
```

CASSETTE TAPE TIME CONSTANTS

```
    FFCC  BYTE  >3B   ZERHI   OUTPUT WAIT COUNT FOR ZERO HIGH
    FFCD  BYTE  >1C   ONEHI   OUTPUT WAIT COUNT FOR ONE HIGH
    FFCE  BYTE  >CC   MIDCYC MID CYCLE THRESH FOR BYTE READ
```

EPROM PROGRAMMER TIME DELAY VALUE

```
    FFCF  BYTE  >A1   EPCNT   PROGRAMMING DELAY COUNTER
```

Table D-5. Crystal Dependent Constants: 3.579 MHz Crystal (Color Burst) with Divide-by-4 Oscillator

```
    FFB2  BYTE  >B5            110 BAUD
    FFB3  BYTE  >04
    FFB4  BYTE  >D4
    FFB5  BYTE  >B0            150 BAUD
    FFB6  BYTE  >03
    FFB7  BYTE  >9B
    FFB8  BYTE  >55            300 BAUD
    FFB9  BYTE  >03
    FFBA  BYTE  >4A
    FFBB  BYTE  >28            600 BAUD
    FFBC  BYTE  >03
    FFBD  BYTE  >23
    FFBE  BYTE  >18           1200 BAUD
    FFBF  BYTE  >02
    FFC0  BYTE  >0F
    FFC1  BYTE  >0A           2400 BAUD
    FFC2  BYTE  >02
    FFC3  BYTE  >08
    FFC4  BYTE  >05           4800 BAUD
    FFC5  BYTE  >01
    FFC6  BYTE  >04
    FFC7  BYTE  >01           9600 BAUD
    FFC8  BYTE  >01
    FFC9  BYTE  >01
    FFCA  BYTE  >01           AUTO BAUD
    FFCB  BYTE  >01
```

CASSETTE TAPE TIME CONSTANTS

```
    FFCC  BYTE  >XX   ZERHI   OUTPUT WAIT COUNT FOR ZERO HIGH
    FFCD  BYTE  >XX   ONEHI   OUTPUT WAIT COUNT FOR ONE HIGH
    FFCE  BYTE  >XX   MIDCYC MID CYCLE THRESH FOR BYTE READ
```

EPROM PROGRAMMER TIME DELAY VALUE

```
    FFCF  BYTE  >50   EPCNT   PROGRAMMING DELAY COUNTER
```

Table D-6. Crystal Dependent Constants: 4 MHz Crystal with Divide-by-2 Oscillator, or 8 MHz Crystal with Divide-by-4 Oscillator

```
    FFB2  BYTE  >EA            110 BAUD
    FFB3  BYTE  >07
    FFB4  BYTE  >FF
    FFB5  BYTE  >AA            150 BAUD
    FFB6  BYTE  >07
    FFB7  BYTE  >FF
    FFB8  BYTE  >C5            300 BAUD
    FFB9  BYTE  >03
    FFBA  BYTE  >AB
    FFBB  BYTE  >60            600 BAUD
    FFBC  BYTE  >03
    FFBD  BYTE  >52
    FFBE  BYTE  >45           1200 BAUD
    FFBF  BYTE  >02
    FFC0  BYTE  >27
    FFC1  BYTE  >1F           2400 BAUD
    FFC2  BYTE  >02
    FFC3  BYTE  >11
    FFC4  BYTE  >0C           4800 BAUD
    FFC5  BYTE  >02
    FFC6  BYTE  >08
    FFC7  BYTE  >07           9600 BAUD
    FFC8  BYTE  >01
    FFC9  BYTE  >04
    FFCA  BYTE  >09           AUTO BAUD
    FFCB  BYTE  >01
CASSETTE TAPE TIME CONSTANTS
    FFCC  BYTE  >43   ZERHI   OUTPUT WAIT COUNT FOR ZERO HIGH
    FFCD  BYTE  >20   ONEHI   OUTPUT WAIT COUNT FOR ONE HIGH
    FFCE  BYTE  >C6   MIDCYC MID CYCLE THRESHOLD FOR BYTE READ
EPROM PROGRAMMER TIME DELAY VALUE
    FFCF  BYTE  >B5   EPCNT   PROGRAMMING DELAY COUNTER
```

Table D-7. Crystal Dependent Constants: 5 MHz Crystal with Divide-by-2 Oscillator, or 10 MHz Crystal with Divide-by-4 Oscillator

```
    FFB2  BYTE  >EF            110 BAUD
    FFB3  BYTE  >09
    FFB4  BYTE  >FF
    FFB5  BYTE  >F9            150 BAUD
    FFB6  BYTE  >06
    FFB7  BYTE  >FF
    FFB8  BYTE  >B9            300 BAUD
    FFB9  BYTE  >04
    FFBA  BYTE  >D7
    FFBB  BYTE  >7E            600 BAUD
    FFBC  BYTE  >03
    FFBD  BYTE  >67
    FFBE  BYTE  >B4           1200 BAUD
    FFBF  BYTE  >01
    FFC0  BYTE  >32
    FFC1  BYTE  >53           2400 BAUD
    FFC2  BYTE  >01
    FFC3  BYTE  >16
    FFC4  BYTE  >23           4800 BAUD
    FFC5  BYTE  >01
    FFC6  BYTE  >09
    FFC7  BYTE  >05           9600 BAUD
    FFC8  BYTE  >02
    FFC9  BYTE  >05
    FFCA  BYTE  >09           AUTO BAUD
    FFCB  BYTE  >02
CASSETTE TAPE TIME CONSTANTS
    FFCC  BYTE  >53   ZERHI   OUTPUT WAIT COUNT FOR ZERO HIGH
    FFCD  BYTE  >28   ONEHI   OUTPUT WAIT COUNT FOR ONE HIGH
    FFCE  BYTE  >B7   MIDCYC MIC CYCLE THRESH FOR BYTE READ
EPROM PROGRAMMER TIME DELAY VALUE
    FFCF  BYTE  >E1   EPCNT   PROGRAMMING DELAY COUNTER
```

Table D-8. Crystal Dependent Constants: 5 MHz Crystal with Divide-by-4 Oscillator

```
FFB2  BYTE  >CB          110 BAUD
FFB3  BYTE  >05
FFB4  BYTE  >FF
FFB5  BYTE  >F8          150 BAUD
FFB6  BYTE  >03
FFB7  BYTE  >DA
FFB8  BYTE  >7E          300 BAUD
FFB9  BYTE  >03
FFBA  BYTE  >67
FFBB  BYTE  >B4          600 BAUD
FFBC  BYTE  >01
FFBD  BYTE  >32
FFBE  BYTE  >53         1200 BAUD
FFBF  BYTE  >01
FFC0  BYTE  >16
FFC1  BYTE  >23         2400 BAUD
FFC2  BYTE  >01
FFC3  BYTE  >09
FFC4  BYTE  >05         4800 BAUD
FFC5  BYTE  >02
FFC6  BYTE  >05
FFC7  BYTE  >01         9600 BAUD
FFC8  BYTE  >01
FFC9  BYTE  >01
FFCA  BYTE  >03         AUTO BAUD
FFCB  BYTE  >01
```

CASSETTE TAPE TIME CONSTANTS

```
FFCC  BYTE  >XX   ZERHI   OUTPUT WAIT COUNT FOR ZERO HIGH
FFCD  BYTE  >XX   ONEHI   OUTPUT WAIT COUNT FOR ONE HIGH
FFCE  BYTE  >XX   MIDCYC MID CYCLE THRESH FOR BYTE READ
```

EPROM PROGRAMMER TIME DELAY VALUE

```
FFCF  BYTE  >70   EPCNT   PROGRAMMING DELAY COUNTER
```

Table D-9. Crystal Dependent Constants: 6 MHz Crystal with Divide-by-2 Oscillator, or 12 MHz Crystal with Divide-by-4 Oscillator

```
FFB2  BYTE  >5F          110 BAUD
FFB3  BYTE  >0A
FFB4  BYTE  >FF
FFB5  BYTE  >E0          150 BAUD
FFB6  BYTE  >08
FFB7  BYTE  >FF
FFB8  BYTE  >DF          300 BAUD
FFB9  BYTE  >04
FFBA  BYTE  >FF
FFBB  BYTE  >DD          600 BAUD
FFBC  BYTE  >02
FFBD  BYTE  >82
FFBE  BYTE  >D8         1200 BAUD
FFBF  BYTE  >01
FFC0  BYTE  >40
FFC1  BYTE  >66         2400 BAUD
FFC2  BYTE  >01
FFC3  BYTE  >1F
FFC4  BYTE  >2E         4800 BAUD
FFC5  BYTE  >01
FFC6  BYTE  >0F
FFC7  BYTE  >11         9600 BAUD
FFC8  BYTE  >01
FFC9  BYTE  >06
FFCA  BYTE  >1F         AUTO BAUD
FFCB  BYTE  >01
FFCC  BYTE  >XX
FFCD  BYTE  >XX
FFCE  BYTE  >XX
FFCF  BYTE  >F0         EPROM PROGRAMMER
```

Table D-10. Crystal Dependent Constants: 7.158 MHz Crystal with Divide-by-2 Oscillator, or 14.316 MHz Crystal with Divide-by-4 Oscillator

```
FFB2  BYTE  >F4          110 BAUD
FFB3  BYTE  >0C
FFB4  BYTE  >FF
FFB5  BYTE  >EE          150 BAUD
FFB6  BYTE  >09
FFB7  BYTE  >FF
FFB8  BYTE  >D5          300 BAUD
FFB9  BYTE  >05
FFBA  BYTE  >FF
FFBB  BYTE  >B0          600 BAUD
FFBC  BYTE  >03
FFBD  BYTE  >9B
FFBE  BYTE  >81         1200 BAUD
FFBF  BYTE  >02
FFC0  BYTE  >4D
FFC1  BYTE  >7C         2400 BAUD
FFC2  BYTE  >01
FFC3  BYTE  >25
FFC4  BYTE  >39         4800 BAUD
FFC5  BYTE  >01
FFC6  BYTE  >12
FFC7  BYTE  >17         9600 BAUD
FFC8  BYTE  >01
FFC9  BYTE  >08
FFCA  BYTE  >28         AUTO BAUD
FFCB  BYTE  >01
FFCC  BYTE  >XX
FFCD  BYTE  >XX
FFCE  BYTE  >XX
FFCF  BYTE  >FF         EPROM PROGRAMMER
```

Table D-11. Crystal Dependent Constants: 7.5 MHz Crystal with Divide-by-2 Oscillator, or 15 MHz Crystal with Divide-by-4 Oscillator

```
FFB2  BYTE  >F0          110 BAUD
FFB3  BYTE  >0D
FFB4  BYTE  >FF
FFB5  BYTE  >FA          150 BAUD
FFB6  BYTE  >09
FFB7  BYTE  >FF
FFB8  BYTE  >E0          300 BAUD
FFB9  BYTE  >05
FFBA  BYTE  >FF
FFBB  BYTE  >B8          600 BAUD
FFBC  BYTE  >03
FFBD  BYTE  >A3
FFBE  BYTE  >88         1200 BAUD
FFBF  BYTE  >02
FFC0  BYTE  >50
FFC1  BYTE  >83         2400 BAUD
FFC2  BYTE  >01
FFC3  BYTE  >27
FFC4  BYTE  >3C         4800 BAUD
FFC5  BYTE  >01
FFC6  BYTE  >13
FFC7  BYTE  >18         9600 BAUD
FFC8  BYTE  >01
FFC9  BYTE  >08
FFCA  BYTE  >2A         AUTO BAUD
FFCB  BYTE  >01
FFCC  BYTE  >XX
FFCD  BYTE  >XX
FFCE  BYTE  >XX
FFCF  BYTE  >FF         EPROM PROGRAMMER
```

Table D-12. Crystal Dependent Constants: 8 MHz Crystal with Divide-by-2 Oscillator, or 16 MHz Crystal with Divide-by-4 Oscillator

```
FFB2  BYTE  >FC          110 BAUD
FFB3  BYTE  >0D
FFB4  BYTE  >FF
FFB5  BYTE  >F0          150 BAUD
FFB6  BYTE  >0A
FFB7  BYTE  >FF
FFB8  BYTE  >EF          300 BAUD
FFB9  BYTE  >05
FFBA  BYTE  >FF
FFBB  BYTE  >C5          600 BAUD
FFBC  BYTE  >03
FFBD  BYTE  >AE
FFBE  BYTE  >91         1200 BAUD
FFBF  BYTE  >02
FFC0  BYTE  >56
FFC1  BYTE  >8C         2400 BAUD
FFC2  BYTE  >01
FFC3  BYTE  >2A
FFC4  BYTE  >40         4800 BAUD
FFC5  BYTE  >01
FFC6  BYTE  >14
FFC7  BYTE  >1B         9600 BAUD
FFC8  BYTE  >01
FFC9  BYTE  >09
FFCA  BYTE  >2D         AUTO BAUD
FFCB  BYTE  >01
FFCC  BYTE  >XX
FFCD  BYTE  >XX
FFCE  BYTE  >XX
FFCF  BYTE  >FF         EPROM PROGRAMMER
```

Index

A

B

C

D

E

P

R

S

T

U

V

X

TMS7000 Evaluation Module User's Guide

Please use this form to communicate your comments about this document, its organization and subject matter, for the purpose of improving technical documentation.

1) Is the Installation section clear and complete? If not, why? _____

2) Is the Debug Monitor section clear and complete? If not, why? _____

3) Is the In-Circuit Emulation section clear and complete? If not, why? _____

4) Is the Audio Tape section clear and complete? If not, why? _____

5) What additions do you think would enhance the structure and subject matter? _____

6) What deletions could be made without affecting overall usefulness? _____

7) Is there any incorrect or misleading information? _____

8) How would you improve this document? _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 6189 HOUSTON, TX

POSTAGE WILL BE PAID BY ADDRESSEE

Texas Instruments Incorporated
M/S 640
P.O. Box 1443
Houston, Texas 77001

MONITOR COMMANDS (Concluded)

COMMAND	DESCRIPTION
MP	Display/Modify Peripheral File
MR	Display/Modify Register File
MS	Display/Modify PC, SP, Registers A and B
MV	Move (Copy) Memory
NP	Fill Memory with NOPs
PC	Display/Modify PC, SP, Registers A and B
PE	Program EPROM from Memory
PT	Print Instruction Trace Memory
RE	Read EPROM to RAM
RT	Reset Target Processor
RU	Execute Program Without Breakpoints
SA	Save 7000 Format Object to Port
SC	Set SR Carry Bit
SI	Set SR Interrupt Bit
SM	Save Memory - 7000 Format
SN	Set SR Negative Bit
SP	Display/Modify PC, SP, Registers A and B
SR	Display Status Register
SS	Single Step Program
ST	Save Memory - Tektronix Format
SZ	Set SR Zero Bit
TC	Configure Single-Step Trace
TF	Turn Off Instruction Trace
TO	Turn On Instruction Trace
TR	Display Trace Line
TS	Single Step Program With Trace
T0	Load PC With TRAP 0 Vector
VE	Verify EPROM Erased
XA	Execute Assembler
XE	Execute Text Editor
XL	Execute Line Assembler (new symbol table)
XP	Execute Line Assembler (old symbol table)
XR	Execute Reverse Assembler

ASSEMBLER DIRECTIVES

AORG — **Absolute Origin**
AORG allows a program or piece of a program to be placed at a specified location in memory.

```
AORG >XXXX
```

BES — **Block Ending with Symbol**
BES reserves a block of memory, and the label is assigned the value of the location of the last byte in the block.

```
[label] BES <value>
```

BSS — **Block Starting with Symbol**
BSS reserves a block of memory, and the label is assigned the value of the location of the first byte in the block.

```
[label] BSS <value>
```

BYTE — **Initialize Byte**
BYTE loads a one-byte constant into the next byte of memory.

```
BYTE <value>
```

DATA — **Initialize Word**
DATA loads a two-byte constant into the next two bytes of memory.

```
DATA <value>
```

END — **Program End**
END prints unresolved labels, out-of-range labels, and the total error count, and returns control to the Monitor.

```
END
```

EQU — **Define Assembly-Time Constant**
EQU assigns a label to a decimal or hex value, register, or peripheral register.

```
<label> EQU {value, register, or peripheral register}
```

TEXT — **Initialize Text**
TEXT places the characters of a string in successive bytes of program memory.

```
TEXT '<string>'
```

TMS7000 EVALUATION MODULE
Quick Reference Card

For help with the TMS7000 EVM, contact the TI Atlanta Regional Technology Center.

Atlanta Regional Technology Center
5515 Spalding Drive
Norcross, GA 30092
(404) 662-7945

MONITOR COMMANDS

COMMAND	DESCRIPTION
AR	+/- Hex Arithmetic
AT	Display Assembler Label Table
BC	Clear 727x0 EEPROM in U20
BR	Display/Modify Baud Rate
BT	Set Breakpoint on Trap
B1,B2	Set Breakpoints 1 and 2
CB	Clear Breakpoints
CC	Clear SR Carry Bit
CE	Compare EPROM to Memory
CI	Clear SR Interrupt Bit
CL	Display/Modify Cursor-Left
CN	Clear SR Negative Bit
CP	Clear Processor Status
CS	Cycle Count Single Step
CT	Clear Breakpoint on TRAP
CU	Display/Modify Cursor-Up
CY	Cycle Count Display/Clear
CZ	Clear SR Zero Bit
C1,C2	Clear Breakpoints Individually
DB	Display Breakpoints
DC	Decimal-Hex Byte Conversion
DI	Clear All SR Bits
DM	Display Memory
DP	Display Processor Status
DR	Audio Tape Directory
DS	Display Machine State
DT	Display Breakpoint on Trap
DV	Select TMS7000 Family Device
EF	Execute to Breakpoint with Terminal Fixed Display
EI	Set All SR Bits
ET	Execute to Breakpoint with Trace
EX	Execute to Breakpoints
FB	Find Byte in Memory
FM	Fill Memory
FR	Fill Register File
FS	Single Step with Fixed Display
GO	Go Execute at User Program
HC	Hex-Decimal Word Conversion
HE	Help
IO	Display I/O Status
IS	Inspect Instruction Trace Count
IT	Inspect Instruction Trace Samples
LA	Show Address of Line
LL	List Lines from Text Editor
LM	Load Memory - 7000 Format
LN	Show Editor Line at Address
LS	Load Machine State
LT	Load Memory - Tektronix Format
L1,L2	Set Breakpoint 1 or 2 by Line Number
MA,MB	Display/Modify Registers A and B
MM	Display/Modify Memory
MO	Audio Tape Motor On

ASCII CHARACTER CODES

CHARACTER	BINARY CODE	HEX CODE
Space	010 0000	20
!	010 0001	21
"	010 0010	22
#	010 0011	23
$	010 0100	24
%	010 0101	25
&	010 0110	26
' (single quote)	010 0111	27
(	010 1000	28
)	010 1001	29
*	010 1010	2A
+	010 1011	2B
, (comma)	010 1100	2C
-	010 1101	2D
.	010 1110	2E
/	010 1111	2F
0	011 0000	30
1	011 0001	31
2	011 0010	32
3	011 0011	33
4	011 0100	34
5	011 0101	35
6	011 0110	36
7	011 0111	37
8	011 1000	38
9	011 1001	39
:	011 1010	3A
;	011 1011	3B
<	011 1100	3C
=	011 1101	3D
>	011 1110	3E
?	011 1111	3F
@	100 0000	40
A	100 0001	41
B	100 0010	42
C	100 0011	43
D	100 0100	44
E	100 0101	45
F	100 0110	46
G	100 0111	47
H	100 1000	48
I	100 1001	49
J	100 1010	4A
K	100 1011	4B
L	100 1100	4C
M	100 1101	4D
N	100 1110	4E
O	100 1111	4F
P	101 0000	50
Q	101 0001	51
R	101 0010	52
S	101 0011	53
T	101 0100	54
U	101 0101	55
V	101 0110	56
W	101 0111	57
X	101 1000	58
Y	101 1001	59
Z	101 1010	5A
[	101 1011	5B
\	101 1100	5C
]	101 1101	5D
^	101 1110	5E
_	101 1111	5F

ASCII CHARACTER CODES (Concluded)

CHARACTER	BINARY CODE	HEX CODE
`	110 0000	60
a	110 0001	61
b	110 0010	62
c	110 0011	63
d	110 0100	64
e	110 0101	65
f	110 0110	66
g	110 0111	67
h	110 1000	68
i	110 1001	69
j	110 1010	6A
k	110 1011	6B
l	110 1100	6C
m	110 1101	6D
n	110 1110	6E
o	110 1111	6F
p	111 0000	70
q	111 0001	71
r	111 0010	72
s	111 0011	73
t	111 0100	74
u	111 0101	75
v	111 0110	76
w	111 0111	77
x	111 1000	78
y	111 1001	79
z	111 1010	7A
{	111 1011	7B
|	111 1100	7C
}	111 1101	7D
~	111 1110	7E

SYSTEM UTILITIES

COMMAND	DESCRIPTION
/A	Reset Assembler Label Table
/B	Recover Monitor Registers After Reset
/C	Reset Cursor Up and Cursor Left to Default Values
/D	Toggle Demo Mode
/E	Reset Assembly from Text Editor Flag
/H	Display System Utilities
/M	Reset Parameter Default Values
/N	Display/Modify Nulls Transmitted After <CR>
/R	Toggle Register File Size
/W	Change Buffer Timeout Delay

TEXT EDITOR COMMANDS

COMMAND	DESCRIPTION
A	Autoincrement Line Number Mode
C	Change Line Number
D	Duplicate Line
E	Edit Line
F	Find Character String
H	Help
I	Input File to Text Editor
L	List Line(s) to Terminal
M	Display Free RAM Remaining
Q	Quit Edit and Save File
R	Resequence Line Numbers
T	Display/Modify Tab
Z	Initialize Text Editor
<CR>	Delete Line
+	Line Number Pointer to EOF
-	Line Number Pointer to BOF
=	Display Current Line Number

Revision *
Printed in U.S.A.

TEXAS INSTRUMENTS

SPNU007