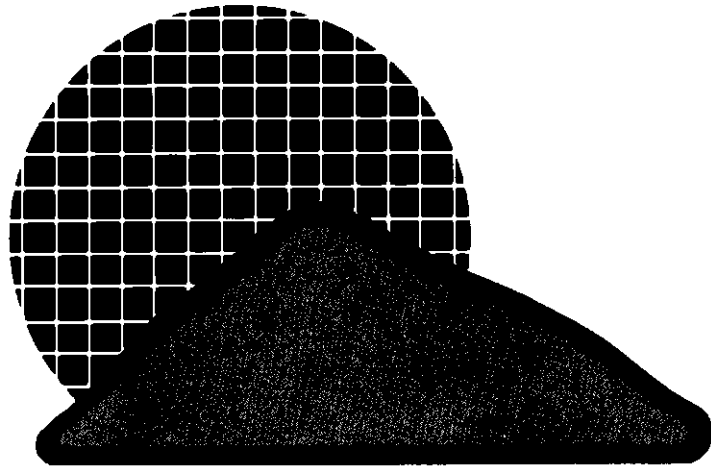


DCT11-EM Evaluation Module User's Guide



DCT11-EM Evaluation Module User's Guide

Prepared by Educational Services
of
Digital Equipment Corporation

© Digital Equipment Corporation 1983.

All Rights Reserved.

Printed in U.S.A.

The material in this manual is for informational purposes and is subject to change without notice.

Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

The manuscript for this book was created on a DIGITAL Word Processing System and, via a translation program, was automatically typeset on DIGITAL's DECset Integrated Publishing System. Book production was done by Educational Services Development and Publishing in Marlboro, MA.

The following are trademarks of Digital Equipment Corporation:

digital	MASSBUS	TOPS-20
DEC	PDP	UNIBUS
DECmate	P/OS	VAX
DECsystem-10	Professional	VMS
DECSYSTEM-20	Rainbow	VT
DECUS	RSTS	Work Processor
DECwriter	RSX	
DIBOL	TOPS-10	

CONTENTS

	Page
PREFACE	
CHAPTER 1 OVERVIEW	
1.1 INTRODUCTION.....	1-1
1.2 THE DCT11-EM AT A GLANCE.....	1-1
CHAPTER 2 INSTALLATION	
2.1 INTRODUCTION.....	2-1
2.2 POWER SUPPLY REQUIREMENTS.....	2-1
2.3 CONSTRUCTING A POWER SUPPLY CABLE.....	2-1
2.4 POWERING UP AND VERIFYING CORRECT OPERATION.....	2-3
2.5 JUMPERING.....	2-4
CHAPTER 3 KEYPAD OPERATION	
3.1 INTRODUCTION.....	3-1
3.2 NUMBER SYSTEM.....	3-2
3.3 EXAMINING REGISTERS.....	3-2
3.4 ALTERING REGISTERS.....	3-4
3.5 EXAMINING MEMORY LOCATIONS.....	3-7
3.6 ALTERING MEMORY LOCATIONS.....	3-10
3.7 ENTERING AND EXECUTING A SIMPLE PROGRAM.....	3-12
3.8 WORKING WITH A LONGER PROGRAM.....	3-16
3.8.1 Using the Single-Step Function.....	3-17
3.8.2 Using Breakpoints.....	3-19
3.8.3 Using the Watchpoint.....	3-22
3.9 USING THE HALT SWITCH.....	3-24
3.10 THE INT SWITCH.....	3-25
3.11 THE USER LED.....	3-25
3.12 PERFORMING KEYPAD SPECIAL FUNCTIONS.....	3-26
3.12.1 Function 0 – Cancel Breakpoints.....	3-27
3.12.2 Function 1 – Go With LEDs.....	3-27
3.12.3 Function 2 – Start Console.....	3-28
3.12.4 Function 3 – Set Baud Rates.....	3-28
3.12.5 Function 4 – Release Protection.....	3-30
3.13 KEYPAD COMMAND SUMMARY.....	3-32
CHAPTER 4 CONSOLE OPERATION	
4.1 INTRODUCTION.....	4-1
4.2 TRANSFERRING CONTROL BETWEEN THE CONSOLE TERMINAL AND THE KEYPAD.....	4-2



CONTENTS (Cont)

4.3	GENERAL RULES FOR CONSOLE INPUT	4-2
4.4	CONSOLE CONTROL COMMANDS	4-3
4.5	EXPRESSIONS	4-3
4.6	EXAMINING REGISTERS	4-5
4.7	ALTERING REGISTERS	4-5
4.8	EXAMINING AND ALTERING MEMORY LOCATIONS - INTRODUCTORY CONCEPTS	4-5
4.8.1	The Address Pointer	4-6
4.8.2	Output Formatting Modes	4-6
4.9	EXAMINING MEMORY LOCATIONS	4-7
4.10	ALTERING MEMORY LOCATIONS	4-10
4.11	INSTRUCTION FORMAT	4-11
4.12	TYPING IN A PROGRAM	4-12
4.13	USING SYMBOLS	4-13
4.13.1	Symbol Names	4-13
4.13.2	Commands That Define and Manipulate Symbols	4-14
4.14	EXECUTING AND DEBUGGING PROGRAMS	4-15
4.14.1	Using the Watchpoint	4-16
4.14.2	Using the Single-Step Function	4-17
4.14.3	Using Breakpoints	4-19
4.15	DCT11-EM DIRECTIVES	4-20
4.16	MISCELLANEOUS CONSOLE FUNCTIONS	4-21
4.17	LOADING THE DCT11-EM FROM A HOST COMPUTER	4-22
4.17.1	Configuring the DCT11-EM for Host Loading	4-22
4.17.2	The HOST Command	4-23
4.17.3	The TALK Command	4-23
4.17.4	The LOAD Command	4-24
4.17.5	Preparing DCT11-EM Loadable Programs on a Host	4-25
4.17.6	An Illustration of the Host Loading Process	4-28
4.18	CONSOLE COMMAND SUMMARY	4-29
4.19	CONTROL COMMAND SUMMARY	4-32

CHAPTER 5 SOFTWARE

5.1	INTRODUCTION	5-1
5.2	THE MONITOR	5-1
5.2.1	The Stack Pointer	5-2
5.3	MONITOR SUBROUTINES	5-2
5.3.1	Keypad/LED Subroutines	5-2
5.3.1.1	Using the LEDs and Keypad	5-4
5.3.1.2	Using Keypad/LED Subroutines - Example Programs	5-5
5.3.2	Console Subroutines	5-8
5.3.2.1	Console Character Manipulation	5-10
5.3.2.2	Using Console Subroutines - Sample Program	5-10
5.4	DCT11-EM ADDRESS SPACE	5-13
5.4.1	Standard RAM Space	5-15
5.4.2	Expansion RAM Space	5-15
5.4.3	Monitor Space	5-15
5.4.4	I/O Space	5-15
5.4.5	Space Available for External Hardware	5-16
5.4.6	Reserved Space	5-16

CONTENTS (Cont)

CHAPTER 6 HARDWARE

6.1	INTRODUCTION.....	6-1
6.2	DCT11-AA MICROPROCESSOR.....	6-3
6.3	RAM.....	6-3
6.4	MONITOR EPROM.....	6-4
6.5	PERIPHERAL CHIPS AND DRIVERS.....	6-4
6.5.1	Parallel Port (8255A).....	6-4
6.5.2	DLART Console Serial Line.....	6-5
6.5.3	Auxiliary Serial Line (8251A).....	6-5
6.6	INTERNAL DATA BUS BUFFER.....	6-5
6.7	USER BUFFERS.....	6-5
6.8	PROCESSOR CYCLE DECODING.....	6-5
6.9	ADDRESS LATCHING AND RANGE DECODING.....	6-5
6.10	PERIPHERAL CHIP ADDRESS DECODING.....	6-6
6.11	INTERRUPTS AND TRAPS.....	6-6
6.12	TIMING.....	6-7
6.13	MISCELLANEOUS CONTROL LOGIC.....	6-7
6.14	KEYPAD/LEDS.....	6-7
6.15	HALT AND INT SWITCHES.....	6-7
6.16	CONNECTORS.....	6-8
6.17	HARDWARE EXPANSION – EXAMPLE.....	6-11

APPENDIX A MONITOR LISTING

APPENDIX B SCHEMATIC DRAWINGS

APPENDIX C ERROR MESSAGES

FIGURES

Figure No.	Title	Page
2-1	Attaching Pin to Wire.....	2-1
2-2	Inserting Pin into Connector.....	2-2
2-3	Power Supply Connector Pin Assignments.....	2-2
2-4	Power Supply Connector.....	2-3
3-1	DCT11-EM Keypad, Switches, and Displays.....	3-1
4-1	Connecting a Console Terminal to the DCT11-EM.....	4-1
4-2	Connecting a Host Computer to the DCT11-EM.....	4-22
5-1	LED Segment Assignments.....	5-4
5-2	SEGBUF Assignments.....	5-4
5-3	DCT11-EM Memory Map.....	5-14
5-4	Map of I/O Space.....	5-16
6-1	DCT11-EM System Block Diagram.....	6-2
6-2	Address Range Decoding.....	6-6
6-3	60-Pin Connector Pin Assignments.....	6-11
6-4	16 KB Memory Module.....	6-12

TABLES

Table No.	Title	Page
4-1	Control Command Summary	4-3
4-2	DCT11-EM Directives.....	4-21
6-1	Reserved I/O Locations for Peripheral Chips	6-4
6-2	Interrupt Vector Locations	6-7
6-3	DCT11-EM Connector Pin Assignments.....	6-8
C-1	Console Messages.....	C-3

PREFACE

This book explains how to use the DCT11-EM evaluation module. Although the book presents many topics in simplified form, some technical background is assumed on your part. It is assumed, for example, that you are familiar with the basics of microprocessor software and hardware. However, you need not be familiar with microprocessors manufactured by Digital Equipment Corporation.

To understand software details such as those contained in the monitor listing (Appendix A), you should be familiar with the MACRO-11 language. To understand hardware details, you should be able to read and interpret schematic diagrams and data sheets.

The book is organized as follows.

Chapter 1 is an introduction to the DCT11-EM and contains a specification summary for your reference.

Chapter 2 explains how to install and power up your DCT11-EM.

Chapter 3 shows you how to operate your DCT11-EM using the on-board keypad and switches.

Chapter 4 shows you how to operate your DCT11-EM using a console terminal.

Chapter 5 provides software information and describes the use of monitor subroutines.

Chapter 6 gives an overview of the DCT11-EM hardware and contains information you will find useful if you plan to design external circuitry.

Appendices A and B contain the DCT11-EM monitor listing and schematics, respectively. Appendix C contains a summary of error messages.

You will find that this book requires a high degree of interaction between you and the DCT11-EM. Chapters 3 and 4, in particular, are structured so that examples are an integral part of the text.

CHAPTER 1 OVERVIEW

1.1 INTRODUCTION

The DCT11-EM is a standalone, single-board computer. It is designed as a tool to introduce you to the DCT11-AA microprocessor CPU chip and the PDP-11 architecture. It contains supporting hardware and software which greatly simplify the task of evaluating the DCT11-AA for your application.

The DCT11-EM gives you considerable flexibility in designing your own software and hardware. You can write programs to run on the DCT11-EM which are substantially the same as those written in the versatile PDP-11 assembly language. You can also design your own hardware interfaces to use with the DCT11-EM. In fact, your hardware can connect directly to the DCT11-EM's internal data paths through an on-board, 60-pin male connector.

This chapter summarizes the features of the DCT11-EM. You will find more detail in the chapters that follow.

1.2 THE DCT11-EM AT A GLANCE

Central Processor

- DCT11-AA microprocessor
- 133 ns cycle time (standard)

Memory

- 4 K bytes of RAM, expandable to 8 K bytes
- 16 K bytes of EPROM which contains the DCT11-EM monitor

Input/Output

- Two serial ports, one for a console terminal and the other for an auxiliary RS232-C device
- One 8-bit parallel port
- One 60-pin connector for use by expansion hardware; allows access to DCT11-EM internal data paths

Host Communication

- Host computer can be connected to the auxiliary serial port for downline loading of programs
- Character format is asynchronous, 8 bits, ASCII, no parity, one stop bit
- Available baud rates are: 300, 600, 1200, 2400, 4800, 9600, 19200

Software

- Software is EPROM resident
- Keypad monitor for performing operations with the on-board keypad
- Console monitor for performing operations with a console terminal and with a host computer

On-Board Control Hardware

- 20-key keypad (4 × 5 array)
- HALT push-button switch

On-Board Displays

- Two rows of 6-digit octal LEDs
- User LED

Interrupts

- Four levels of interrupt priorities
- On-board interrupt push-button switch

Diagnostics

- Contained in EPROM, automatically run at power-up time

Documentation (provided with DCT11-EM)

- *DCT11-EM User's Guide*
- *DCT11-AA (MICRO/T-11) User's Guide*
- *DC319-AA DLART Data Sheet*

Physical

- Width: 20.3 cm (8 in)
- Length (including connectors): 27.9 cm (11 in)
- Height: 3.8 cm (1.5 in)
- Weight: 450 gm (15 oz)

Environment

- Noncaustic environment required
- Operating temperature: 10 to 40 degrees C
- Relative humidity: 10% to 90% (noncondensing)
- Altitude: Up to 2.4 km (8000 ft)

DC Power Requirements

- For keypad operation only:
+5 V (±5%) @ 2 A
- For console terminal operation:
+5 V (±5%) @ 2 A
+12 V (±10%) @ 50 mA
-12 V (±10%) @ 50 mA

CHAPTER 2 INSTALLATION

2.1 INTRODUCTION

DCT11-EM installation consists of attaching a power supply, powering up, and verifying that the board is operating properly.

2.2 POWER SUPPLY REQUIREMENTS

The DCT11-EM requires a +5 Vdc supply ($\pm 5\%$) capable of delivering up to 2 A. If you plan to use the console or auxiliary serial ports, you also need a +12 Vdc supply ($\pm 10\%$) and a -12 Vdc supply ($\pm 10\%$) each capable of delivering up to 50 mA.

WARNING

To protect yourself against injury from contact with circuits that carry high energy, we suggest your power supply outputs be limited to no more than 240 VA. This is in accordance with standards for data processing equipment.

CAUTION

We recommend that you fuse your power supply outputs at 5 A or less to provide overcurrent protection to circuits and remote wiring.

2.3 CONSTRUCTING A POWER SUPPLY CABLE

The DCT11-EM is packaged with a power supply connector and pins with which you can assemble a power supply cable. To do this, perform the following steps.

1. Select a 1 foot length of medium gauge (10 to 14 gauge) wire and strip the wire about 1/4 inch from each end. If you are using multiple-strand wire, twist the strands together at each stripped end.
2. Insert one end of the wire into a pin as shown in Figure 2-1. Crimp the pin to the wire with a pair of pliers or a crimping tool and solder the wire.

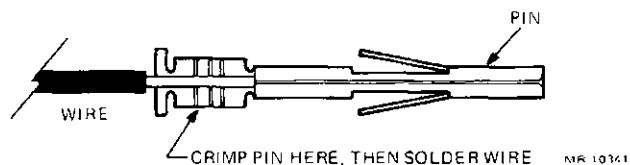


Figure 2-1 Attaching Pin to Wire

3. Insert the pin in the connector as shown in Figure 2-2.

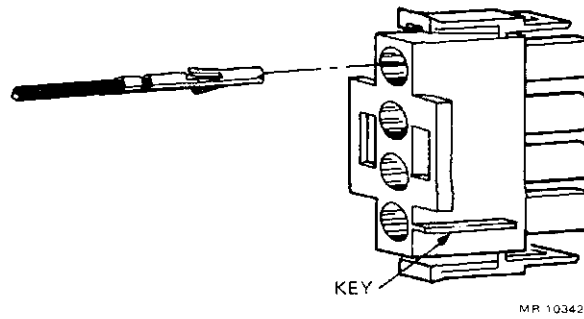


Figure 2-2 Inserting Pin into Connector

4. Repeat steps 1, 2, and 3 for each wire in your connector. (Use different colored wires for easy identification.)
5. Attach the wires to a power supply, making sure they correspond to the correct voltages as shown in Figure 2-3.

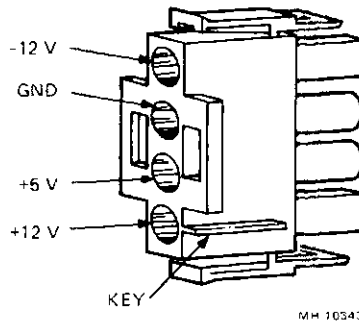


Figure 2-3 Power Supply Connector Pin Assignments

6. Turn on the power supply and use a voltmeter to check that the correct voltages appear at the connector pins.
7. Turn off the power supply and attach the connector to the DCT11-EM as shown in Figure 2-4.

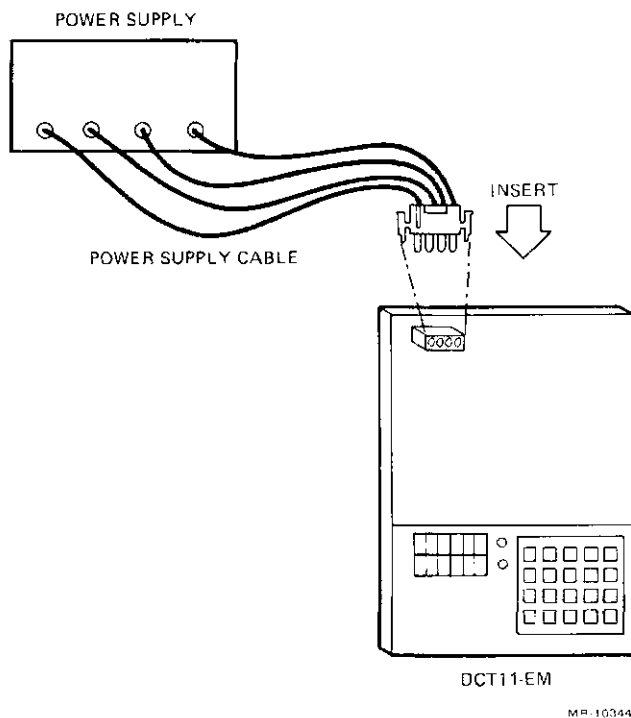


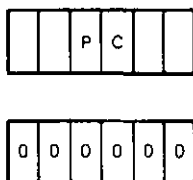
Figure 2-4 Power Supply Connector

You are now ready to power up your DCT11-EM.

2.4 POWERING UP AND VERIFYING CORRECT OPERATION

Make sure your DCT11-EM is on a nonmetallic surface and positioned so that it does not come in contact with any conducting objects. Then turn on your power supply.

When you apply power, the DCT11-EM runs its diagnostics automatically. The diagnostics: (1) perform checksum tests for the on-board RAM and PROM, and (2) write and read back data to and from the DLART. If the diagnostics run successfully, the LEDs display the following.



This shows that the keypad monitor is active and ready to accept your input. Further information on using the keypad is given in Chapter 3.

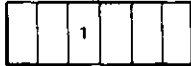

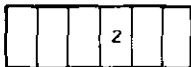
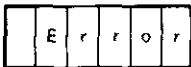
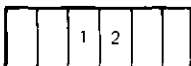
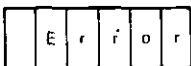
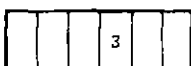

In addition to activating the keypad monitor, the DCT11-EM also sets these initial conditions:

- All user RAM is cleared
- The symbol table is cleared
- The stack pointer is set to 7400
- Both serial port baud rates are set to 9600.

You will learn more about what these initial conditions mean in later chapters.

If the two rows of LEDs are blank after you power up, your DCT11-EM probably has major problems.

The LEDs may display a flashing diagnostic error message after you power up. The meaning of the error messages is summarized below.

UPPER DISPLAY	LOWER DISPLAY	MEANING
		RAM checksum test failed. Try replacing RAM chips.
		ROM checksum test failed. Try replacing ROM chips.
		RAM and ROM checksum tests failed. Board may have major problems.
		DLART feedback test failed. Board may have major problems.

If you suspect your board has major problems, contact Digital Equipment Corporation or the authorized distributor from whom you purchased your DCT11-EM. The DCT11-EM is warranted against defects in workmanship for a period of ninety (90) days from the date of delivery.

2.5 JUMPERING

There are two metal posts located above the upper left corner of the DCT11-EM keypad. Install a jumper between these posts if you want to disable the generation of DCT11-AA microprocessor cycle slips. If you do this, you must also change the DCT11-AA's 7.5 MHz clock frequency to a frequency between 3 MHz and 6 MHz. To change the clock frequency, remove the 7.5 MHz crystal from the board and install a crystal of the desired frequency.

This is the only jumper that may be installed on the DCT11-EM.

CHAPTER 3 KEYPAD OPERATION

3.1 INTRODUCTION

This chapter explains how to operate the DCT11-EM using the DCT11-EM's on-board keypad and switches. When you complete this chapter, you will be able to do the following.

- Examine and alter the contents of registers in the DCT11-AA register file.
- Examine and alter the contents of memory locations.
- Enter and execute programs.
- Debug your programs through the use of single steps, breakpoints, and watchpoints.
- Perform halts and interrupts.
- Perform a number of other functions.

The DCT11-EM keypad, switches, and displays are arranged as shown in Figure 3-1.

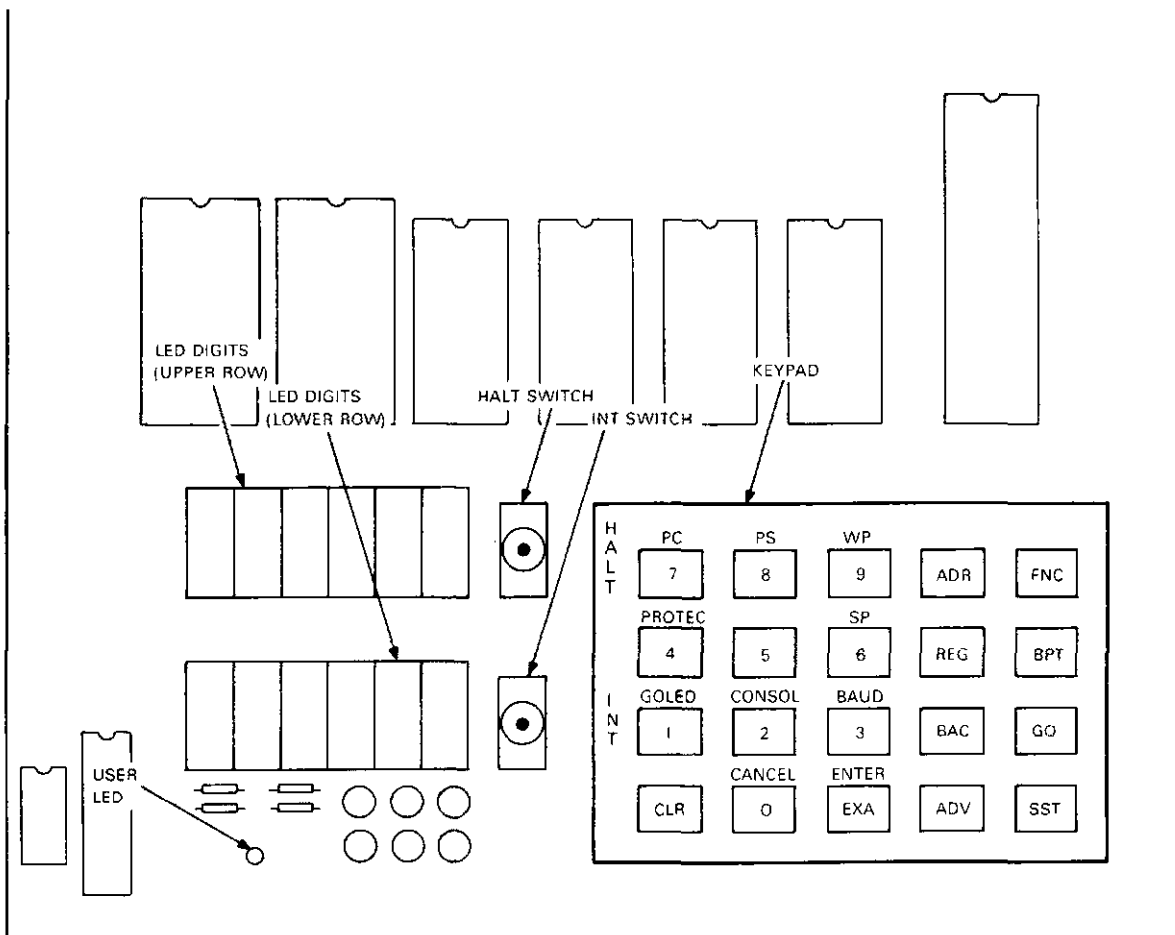


Figure 3-1 DCT11-EM Keypad, Switches, and Displays

MR 10345

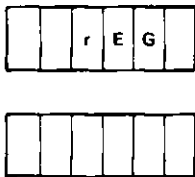
Figure 3-1 shows two rows of six LED digits which display data, addresses, and other information. Below the two rows of LED digits is a single LED (called the user LED) which you can use as an indicator in your programs. To the right of the LEDs are the HALT and INT push-button switches. These switches perform the functions their names imply; that is, they can halt and interrupt DCT11-EM operation. To the extreme right is a 4 × 5 array of switches called the keypad. The keypad is used to enter commands, instructions, and other information to the DCT11-EM. Pressing keys will generally cause responses to appear on the LED displays.

3.2 NUMBER SYSTEM

Information that is input or output via the keypad/LED combination must be represented in octal notation. Since the DCT11-AA is a 16-bit processor, the acceptable range of octal inputs and outputs is 000000 through 177777.

3.3 EXAMINING REGISTERS

To examine the contents of a DCT11-AA internal register, you must first place the DCT11-EM in register mode. Do this by pressing the REG key. The LEDs display:



Now you may examine a register by pressing any of the following keys:

0 through 9

or

CLR, which is equivalent to key 0.

In general, the name of the register being examined appears on the upper row of LEDs, and the contents of the register appears on the lower row of LEDs. The exception to this is when the watchpoint (WP) is examined. In that case, an address called the watchpoint (WP) address is displayed on the upper row of LEDs, and the WP contents are displayed on the lower row.

Keys 0 through 7 select R0 through R7, the eight registers in the DCT11-AA register file. Registers 6 and 7 are dedicated and are referred to as the stack pointer (SP) and program counter (PC), respectively.

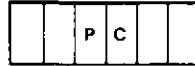
Key 8 selects the processor status (PS) word. Only the lower 8 bits of the PS word are meaningful. The PS word contains the current processor priority, the processor condition codes (N, Z, V, and C), and the T-bit.

Key 9 selects the watchpoint (WP). The WP is useful in connection with program debugging and is discussed in Paragraph 3.8.3.

NOTE

If you have just powered up your DCT11-EM or if you have halted its operation, it is not necessary to press the REG key to get into register mode.

The upper row of LEDs will display:



The lower row of LEDs will display the current value of the PC, and you can simply press one of the keys 0 through 9 or CLR to select a register.

Examining Registers – Example 1

You want to examine the contents of various registers. It is assumed that the registers still contain their initial (power-up) values.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
REG	r E G		Get into register mode.
0	r 0	0 0 0 0 0 0	
7	P C	0 0 0 0 0 0	
6	S P	0 0 6 5 0 0	
CLR	r 0	0 0 0 0 0 0	
8	P S	0 0 0 0 0 0	
9	0 0 0 0 0 0	0 0 0 0 0 0	

The ADV and BAC keys allow you to examine register contents sequentially. Pressing the ADV key allows you to examine registers in ascending order. You cannot advance beyond the WP. Pressing the BAC key allows you to examine registers in descending order. You cannot back up beyond R0.

If you want to examine registers nonsequentially after you have pressed ADV or BAC, you must first press the REG key to reinitialize register mode. Otherwise, you will alter the contents of the register you are currently examining.

Examining Registers - Example 2

You want to examine more registers. You begin by examining them sequentially. It is assumed that the registers still contain their initial (power-up) values.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
REG	r E G		Get into register mode.
ADV	r 0	0 0 0 0 0 0	
ADV	r 1	0 0 0 0 0 0	
ADV	r 2	0 0 0 0 0 0	
BAC	r 1	0 0 0 0 0 0	
BAC	r 0	0 0 0 0 0 0	
4	r 0	0 0 0 0 0 4	You wanted to examine R4 but you altered R0 instead.
CLR	r 0	0 0 0 0 0 0	Clear the incorrect entry.
REG	r E G		This is how to examine R4.
4	r 4	0 0 0 0 0 0	
BAC	r 3	0 0 0 0 0 0	Now you're sequentially examining again.

3.4 ALTERING REGISTERS

To alter the contents of a DCT11-AA internal register, you must first examine the register as described in Paragraph 3.3. To review, this means you must place the DCT11-EM in register mode by pressing the REG key and examine the desired register by pressing one of the keys 0 through 9. Once you have done this, press the EXA key and alter the contents of the selected register by pressing keys 0 through 7 in the desired order.

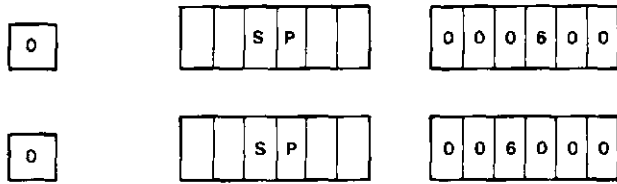
The previous contents of the register will be cleared and you will see the new contents being shifted in from the right side of the lower row of LEDs as you press keys.

If you wish to clear the contents of a register, press the CLR key after you have pressed the EXA key. Pressing the CLR key also clears a register while you are entering data.

Altering Registers - Example 1

You want to set register R3 to 000232 and the SP to 6000. It is assumed that the registers still contain their initial (power-up) values.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
REG	r E G		Get into register mode.
3	r 3	0 0 0 0 0 0	Examine R3.
EXA	r 3	0 0 0 0 0 0	Enter 232.
2	r 3	0 0 0 0 0 2	
3	r 3	0 0 0 0 2 3	
2	r 3	0 0 0 2 3 2	
CLR	r 3	0 0 0 0 0 0	You change your mind, and clear R3 instead.
REG	r E G		Get back into register mode to change another register.
6	S P	0 0 7 4 0 0	Examine R6 (the SP). It currently = 7400.
EXA	S P	0 0 7 4 0 0	Set SP = 6000.
6	S P	0 0 0 0 0 6	
0	S P	0 0 0 0 6 0	



The ADV and BAC keys allow you to alter register contents sequentially. Pressing the ADV key allows you to alter registers in ascending order. You cannot advance beyond the WP. Pressing the BAC key allows you to alter registers in descending order. You cannot back up beyond R0.

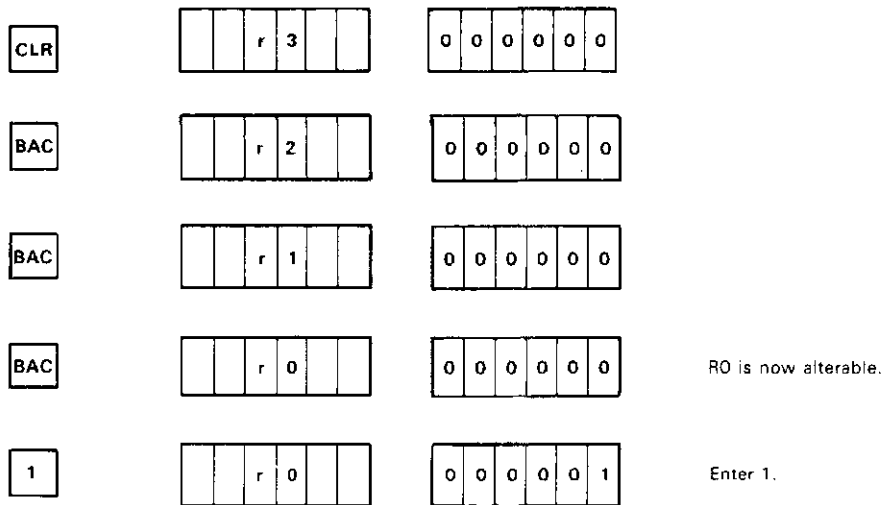
Press the ADV or BAC keys until you arrive at the register you want to alter, then press the desired sequence of keys 0 through 7 to enter the data. If you want to clear the register, press CLR. If you want to alter registers nonsequentially after you have pressed the ADV or BAC keys, you must first press the REG key to reinitialize register mode. Then press 0 through 9 to select the desired register, followed by EXA, followed by the data you wish to enter.

It is generally easier to stay in sequential mode and step forward or backward to the desired register than to perform nonsequential alterations.

Altering Registers – Example 2

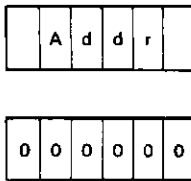
You have just completed example 1 and now want to sequentially change some registers.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
BAC	r 5	0 0 0 0 0 0	R5 is now alterable.
4	r 5	0 0 0 0 0 4	Enter 43.
3	r 5	0 0 0 0 4 3	
BAC	r 4	0 0 0 0 0 0	R4 is now alterable.
BAC	r 3	0 0 0 0 0 0	R3 is now alterable.
1	r 3	0 0 0 0 0 1	Enter 1.
ADV	r 4	0 0 0 0 0 0	R4 is now alterable. But now you want to clear R3 and alter R0.
BAC	r 3	0 0 0 0 0 1	

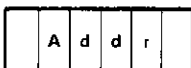


3.5 EXAMINING MEMORY LOCATIONS

To examine a memory location, you must first place the DCT11-EM in address mode. Do this by pressing the ADR key. The first time you do this, the LEDs display:



Subsequently, when you press the ADR key, the upper row of LEDs displays:



and the lower row of LEDs displays the most recently examined address.

Enter the address of the memory location you wish to examine by pressing the appropriate sequence of keys 0 through 7 .

The previous address will be cleared and you will see the address you are entering being shifted in from the right side of the lower row of LEDs as you press keys.

If you make a mistake entering your address, you can press the CLR key to clear your entry and try again.

Once you are satisfied with the address you have entered, press the EXA key. The address of the location you are examining (the address you just entered) appears on the upper row of LEDs, and the contents of this location appear on the lower row of LEDs. Only even addresses can be examined. If you enter an odd address, it is rounded down when you press the EXA key.

Examining Memory Locations - Example 1

You have just powered up your DCT11-EM and want to examine some memory locations. There is a program that resides in the DCT11-EM called the monitor which begins at address 140000. You will learn more about the monitor later, but for now, all we want to do is examine some of its locations.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
ADR	A d d r	0 0 0 0 0 0	Get into address mode.
1	A d d r	0 0 0 0 0 1	Enter 140000.
4	A d d r	0 0 0 0 1 4	
0	A d d r	0 0 0 1 4 0	
0	A d d r	0 0 1 4 0 0	
0	A d d r	0 1 4 0 0 0	
0	A d d r	1 4 0 0 0 0	
EXA	A d d r	0 0 0 1 6 7	

The ADV and BAC keys allow you to examine memory locations sequentially. Pressing the ADV key allows you to examine memory locations in ascending order. Advancing beyond address 177776 results in examining locations starting at 000000. Pressing the BAC key allows you to alter registers in descending order. Backing up beyond 000000 results in examining locations starting at 177776.

If you want to examine memory locations nonsequentially after you have pressed ADV or BAC, you must first press the ADR key to reinitialize address mode. Otherwise, you will change the contents of the memory location you are currently examining.

Examining Memory Locations - Example 2

You have just finished example 1 and now want to examine more locations in the monitor. You start by examining them sequentially.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
ADV	1 4 0 0 0 2	0 1 5 3 3 0	
ADV	1 4 0 0 0 4	0 0 0 1 6 7	
ADV	1 4 0 0 0 6	1 7 6 7 4 2	
BAC	1 4 0 0 0 4	0 0 0 1 6 7	
ADR	A d d r	1 4 0 0 0 4	Reinitialize address mode for nonsequential examine.
1	A d d r	0 0 0 0 0 1	You try to examine 140101.
4	A d d r	0 0 0 0 1 4	
0	A d d r	0 0 0 1 4 0	
1	A d d r	0 0 1 4 0 1	
0	A d d r	0 1 4 0 1 0	
1	A d d r	1 4 0 1 0 1	
EXA	1 4 0 1 0 0	0 0 0 1 6 7	The address is rounded down and you examine 140100.
BAC	1 4 0 0 7 6	1 7 3 3 0 6	Now you're sequentially examining again.

3.6 ALTERING MEMORY LOCATIONS

To alter the contents of a memory location, you must first examine the memory location as described in Paragraph 3.5. To review, this means you must place the DCT11-EM in address mode by pressing the ADR key and examine the desired location by pressing the appropriate sequence of keys 0 through 7. Then press the EXA key.

The contents of the location you are examining can now be changed by pressing keys 0 through 7 in the desired sequence. The previous contents of the location will be cleared and you will see the new contents being shifted in from the right side of the lower row of LEDs as you press keys.

Pressing the CLR key will clear the location before or during data entry.

Altering Memory Locations – Example 1

You have just powered up your DCT11-EM and want to alter some memory locations. Locations 2000 through 2006 are in the user RAM area and will be used in the following examples. Later on, you will learn more about the availability of DCT11-EM memory and address space, but for now the discussion will be limited to these locations only.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
ADR	A d d r	0 0 0 0 0 0	Get into address mode.
2	A d d r	0 0 0 0 0 2	Enter 2004
1	A d d r	0 0 0 0 2 1	You made a mistake and entered 1 instead of 0.
CLR	A d d r	0 0 0 0 0 0	Clear entry.
2	A d d r	0 0 0 0 0 2	Try again – enter 2004.
0	A d d r	0 0 0 0 2 0	
0	A d d r	0 0 0 2 0 0	
4	A d d r	0 0 2 0 0 4	
EXA	0 0 2 0 0 4	0 0 0 0 0 0	2004 is currently empty.
3	0 0 2 0 0 4	0 0 0 0 0 3	Enter 32.

2	0 0 2 0 0 4	0 0 0 0 3 2	
ADR	A d d r	0 0 2 0 0 4	Back to address mode to alter another location. Note – most recent address examined is displayed.
2	A d d r	0 0 0 0 0 2	Enter 2000.
0	A d d r	0 0 0 0 2 0	
0	A d d r	0 0 0 2 0 0	
0	A d d r	0 0 2 0 0 0	
EXA	0 0 2 0 0 0	0 0 0 0 0 0	2000 is currently empty.
1	0 0 2 0 0 0	0 0 0 0 0 1	Enter 1.

The ADV and BAC keys allow you to alter memory locations sequentially. Pressing the ADV key allows you to alter memory locations in ascending order. Pressing the BAC key allows you to alter memory locations in descending order.

Press the ADV or BAC keys until you arrive at the memory location you want to alter and press the desired sequence of keys 0 through 7 to enter the data. If you want to clear the memory location, press CLR.

Altering Memory Locations – Example 2

You have just completed example 1 and now want to sequentially alter memory locations.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
ADV	0 0 2 0 0 2	0 0 0 0 0 0	Location 2002 is now alterable.
4	0 0 2 0 0 2	0 0 0 0 0 4	Enter 47.
7	0 0 2 0 0 2	0 0 0 0 4 7	
ADV	0 0 2 0 0 4	0 0 0 0 3 2	Location 2004 is now alterable.

5	0 0 2 0 0 4	0 0 0 0 0 5	Enter 5.
ADV	0 0 2 0 0 6	0 0 0 0 0 0	Location 2006 is now alterable.
2	0 0 2 0 0 6	0 0 0 0 0 2	Enter 22
2	0 0 2 0 0 6	0 0 0 0 2 2	
BAC	0 0 2 0 0 4	0 0 0 0 0 5	Location 2004 is now alterable.
CLR	0 0 2 0 0 4	0 0 0 0 0 0	Clear it.

3.7 ENTERING AND EXECUTING A SIMPLE PROGRAM

In this paragraph, we will tie together the material presented thus far by entering and running a simple program. It is assumed you are familiar with an assembly language instruction set. A full description of the instruction set used in the following example is found in Chapter 6 of the *DCT11-AA User's Guide* (EK-DCT11-UG).

The following program takes a number in register R1, adds it to a number in register R2, and puts the result in register R3.

Octal Code	Instruction	Comments
010103	MOV R1,R3	Get first term.
060203	ADD R2,R3	Add second term.
000000	HALT	Halt program execution.

We will place this program in user RAM, starting at address 2000.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
ADR	A d d r	Most recently examined address shown on lower LEDs.	
2	A d d r	0 0 0 0 0 2	Examine 2000.
0	A d d r	0 0 0 0 2 0	
0	A d d r	0 0 0 2 0 0	

0	A d d r	0 0 2 0 0 0
---	---------	-------------

EXA	0 0 2 0 0 0	Current contents of location 2000 are shown on lower LEDs.
-----	-------------	--

1	0 0 2 0 0 0	0 0 0 0 0 1
---	-------------	-------------

Enter first instruction.
You don't need to enter leading zeroes.

0	0 0 2 0 0 0	0 0 0 0 1 0
---	-------------	-------------

1	0 0 2 0 0 0	0 0 0 1 0 1
---	-------------	-------------

0	0 0 2 0 0 0	0 0 1 0 1 0
---	-------------	-------------

3	0 0 2 0 0 0	0 1 0 1 0 3
---	-------------	-------------

ADV	0 0 2 0 0 2	Current contents of 2002 are shown on lower LEDs.
-----	-------------	---

Examine next address.

6	0 0 2 0 0 2	0 0 0 0 0 6
---	-------------	-------------

Enter second instruction.

0	0 0 2 0 0 2	0 0 0 0 6 0
---	-------------	-------------

2	0 0 2 0 0 2	0 0 0 6 0 2
---	-------------	-------------

0	0 0 2 0 0 2	0 0 6 0 2 0
---	-------------	-------------

3	0 0 2 0 0 2	0 6 0 2 0 3
---	-------------	-------------

ADV	0 0 2 0 0 4	Current contents of 2004 are shown on lower LEDs.
-----	-------------	---

Examine next address.

CLR	0 0 2 0 0 4	0 0 0 0 0 0
-----	-------------	-------------

Enter third instruction.
HALT is entered by clearing the location.

Now place some numbers in R1 and R2 that the program can work with. Also make sure R3 is cleared.

REG	r E G		Get into register mode.
1	r 1	Current contents of R1 shown on lower LEDs.	
EXA	r 1	Current contents of R1 remain on lower LEDs.	
2	r 1	0 0 0 0 0 2	Enter 2.
ADV	r 2	Current contents of R2 shown on lower LEDs.	
3	r 2	0 0 0 0 0 3	Enter 3.
ADV	r 3	Current contents of R3 shown on lower LEDs.	
CLR	r 3	0 0 0 0 0 0	Clear R3.

Now go back and check what you have just entered.

ADR	A d d r	0 0 2 0 0 4	Get into address mode.
2	A d d r	0 0 2 0 0 2	Examine 2000.
0	A d d r	0 0 0 0 2 0	
0	A d d r	0 0 0 2 0 0	
0	A d d r	0 0 2 0 0 0	
EXA	0 0 2 0 0 0	0 1 0 1 0 3	

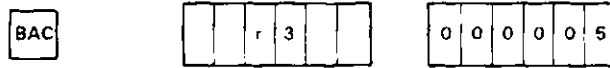
ADV	0 0 2 0 0 2	0 6 0 2 0 3	Examine 2002.
ADV	0 0 2 0 0 4	0 0 0 0 0 0	Examine 2004.
REG	r E G		Get into register mode.
1	r 1	0 0 0 0 0 2	Examine registers.
2	r 2	0 0 0 0 0 3	
3	r 3	0 0 0 0 0 0	Everything O.K.

Now run the program. Set the PC to the starting address of the program (2000) and press the GO key to begin program execution. Since you are currently in register mode, press the ADV key until you get to the PC.

ADV	P C	Current contents of PC shown on lower LEDs.	
2	P C	0 0 0 0 0 2	Enter 2000.
0	P C	0 0 0 0 2 0	
0	P C	0 0 0 2 0 0	
0	P C	0 0 2 0 0 0	
GO	P C	0 0 2 0 0 6	Execute the program.

The program is now halted. The current PC (2006) is displayed on the lower row of LEDs.

Check R3 to find out if the program executed properly. Since you are in register mode, press the BAC key until you get to R3:



and the result is correct.

3.8 WORKING WITH A LONGER PROGRAM

Consider a program that calculates the sum of all the numbers from 0 to n, where you specify n. That is, the program determines:

$$\sum_{x=0}^n x$$

You place the value of n in R0, and the program deposits the sum it calculates in memory location 2044.

Use the keypad to enter the following summation program starting at location 2000.

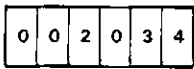
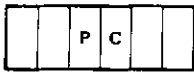
Location	Octal Code	Instruction	Comments
2000	005001	CLR R1	Set R1 = 0.
2002	020001	CMP R0,R1	R0 contains n.
2004	003413	BLE 2034	Illegal input (n ≤ 0)?
2006	005037	CLR @#2044	No, start operation by clearing 2044.
2010	002044		
2012	010037	MOV R0,@#2044	Place first value of n in 2044.
2014	002044		
2016	005300	DEC R0	n = n-1
2020	060037	ADD R0,@#2044	Add next value (n-1) to 2044.
2022	002044		
2024	102404	BVS 2036	Overflow?
2026	020001	CMP R0,R1	No overflow.
2030	003372	BGT 2016	Continue operation if n > 0.
2032	000000	HALT	Operation complete if n = 0.
2034	005000	CLR R0	Illegal input. Clear R0, 2044.
2036	005037	CLR @#2044	Overflow. Clear 2044.
2040	002044		
2042	000000	HALT	Illegal input or overflow occurred.
2044	xxxxxx		Initial contents of 2044 irrelevant.

To test the operation of the program, deposit a value of 10 (octal) into R0. The program should sum $10 + 7 + 6 + 5 + 4 + 3 + 2 + 1$ (octal) in memory location 2044 to produce 44 (octal). Note that this is equivalent to having a value of 8 (decimal) in R0 and letting the program sum $8 + 7 + 6 + 5 + 4 + 3 + 2 + 1$ (decimal) to produce 36 (decimal).

Set R0 = 10
SP = 7400
PC = 2000

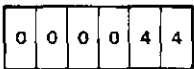
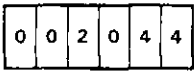
Press the GO key.

The result should be:



The program halted at location 2032. Now examine location 2044.

The result should be:



3.8.1 Using the Single-Step Function

The single-step function is a powerful program analysis tool. It allows you to execute your program one instruction at a time. At the end of each instruction, you may examine various registers and memory locations to determine how the program has affected them.

To perform the single-step function, press the SST key.

Result: the instruction currently pointed to by the PC is executed.

Let us now single step through the preceding summation program. First set up your initial conditions:

Set R0 = 10
SP = 7400
PC = 2000

Then key in the following.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
SST	PC	002002	2000 executed.
SST	PC	002004	2002 executed.
SST	PC	002006	2004 executed.
SST	PC	002012	2006 executed.
SST	PC	002016	2012 executed.
ADR	Addr	Most recently examined address shown on lower LEDs.	Has R0 been moved to 2044?
2	Addr	000002	
0	Addr	000020	
4	Addr	000204	
4	Addr	002044	
EXA	002044	000010	Yes.
SST	PC	002020	2016 executed.
REG	REG		Did R0 get decremented?
0	r0	000007	Yes.

SST	PC	002024	2020 executed.
ADR	Addr	002044	Has R0 been added to 2044?
EXA	002044	000017	Yes.
SST	PC	002026	2024 executed.
SST	PC	002030	2026 executed.
SST	PC	002016	Program branched correctly.
SST	PC	002020	2016 executed.
REG	REG		Did R0 get decremented?
O	r0	000006	Yes.
SST	PC	002024	2020 executed.
ADR	Addr	002044	Did R0 get added to 2044?
EXA	002044	000025	Yes.

You can continue executing the program one instruction at a time by repetitively pressing the SST key.

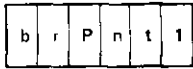
3.8.2 Using Breakpoints

The use of breakpoints is another means of analyzing and debugging your programs. Breakpoints cause your program to halt whenever they are encountered. When the program is halted, you can then examine various registers and memory locations to determine how the program has affected them.

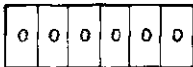
If there were no debugging aids, you would have to insert breakpoint (BPT) instructions in your program and write a breakpoint trap handling routine if you desired breakpoints. The DCT11-EM, however, *does* contain debugging aids and provides a convenient means of inserting up to four breakpoints via the keypad. Do not attempt to set breakpoints by inserting BPT instructions in your program. If you do this, you will get an error message when you try to execute the program (see Appendix C).

To insert breakpoints, press the BPT key.

The upper row of LEDs displays:



and the lower row of LEDs displays the current address of breakpoint 1. If breakpoint 1 is not currently set, the lower row of LEDs displays:



You can then press the appropriate sequence of keys 0 through 7 to enter the address of breakpoint 1.

You can set up to four breakpoints in this way. Use the ADV and BAC keys to examine breakpoints 1 through 4 and press the appropriate sequence of keys 0 through 7 to enter an address for each breakpoint.

If you press BAC while you are examining breakpoint 1, you are examining something called the watchpoint address. The use of the watchpoint is explained in Paragraph 3.8.3.

Breakpoints are recognized only if you use the GO key or the "Go With LEDs" special function (Paragraph 3.12.2) to execute your program from the keypad. Breakpoints are disabled when you single step through a program.

When you execute a program with breakpoints in it (by pressing the GO key, for example), your program halts at the address of your first breakpoint without executing the instruction at that address. The instruction at the breakpoint address is executed when you resume program execution (e.g., by pressing the GO or SST keys).

Using Breakpoints - Example

We want to set two breakpoints in the summation program shown in Paragraph 3.8. Set the first breakpoint at address 2020 to make sure R0 has been decremented properly. Then set the second breakpoint at address 2024 to check the results of the cumulative addition.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
BPT	b r P n t 1	0 0 0 0 0 0	Assume no breakpoints have been previously set.
2	b r P n t 1	0 0 0 0 0 2	Enter 2020.
0	b r P n t 1	0 0 0 0 2 0	
2	b r P n t 1	0 0 0 2 0 2	

0	b r P n t 1	0 0 2 0 2 0
ADV	b r P n t 2	0 0 0 0 0 0
2	b r P n t 2	0 0 0 0 0 2
0	b r P n t 2	0 0 0 0 2 0
2	b r P n t 2	0 0 0 2 0 2
4	b r P n t 2	0 0 2 0 2 4

Set breakpoint 2.

Enter 2024.

Set your initial conditions as before (R0 = 10, SP = 7400, PC = 2000) and execute the program.

GO	P C	0 0 2 0 2 0
REG	r E G	
0	r 0	0 0 0 0 0 7
GO	P C	0 0 2 0 2 4

First breakpoint encountered.

Did R0 get decremented?

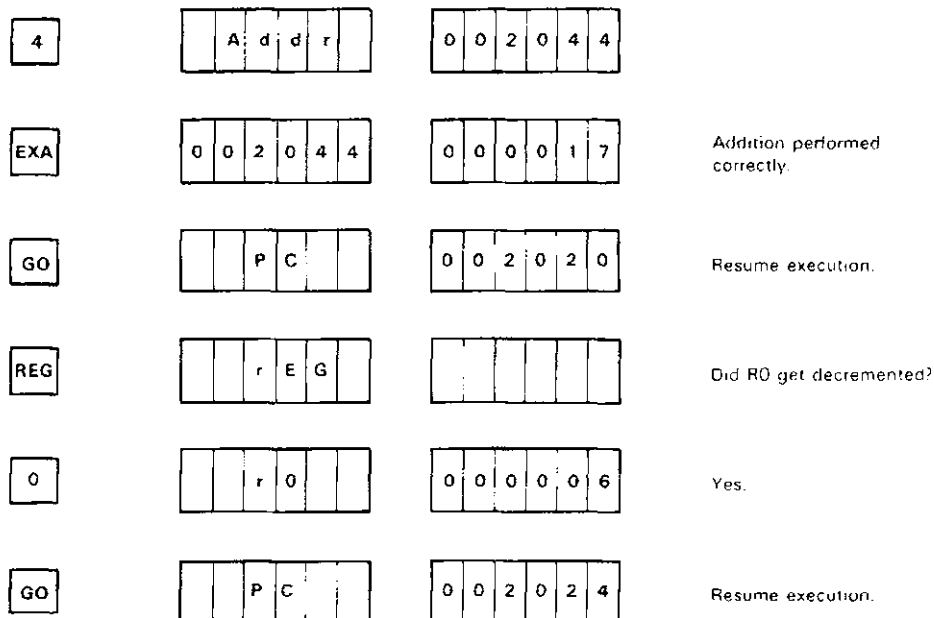
Yes

Resume execution.

Examine address 2044 to see if the first addition was performed correctly.

ADR	A d d r	Most recently examined address shown on lower LEDs.
2	A d d r	0 0 0 0 0 2
0	A d d r	0 0 0 0 2 0
4	A d d r	0 0 0 2 0 4

Examine 2044.



3.8.3 Using the Watchpoint

In Paragraph 3.3 we identified the watchpoint (WP) and showed how it could be examined. In this section, you will learn how to use the WP for program analysis and debugging.

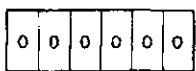
The watchpoint is a convenient means of monitoring and altering a memory location. Once you set the watchpoint address to the address of a memory location, you can examine and alter that location just as you would a register. To set a watchpoint address, press the BPT key, then the BAC key.

The upper row of LEDs displays:



and the lower row of LEDs displays the current WP address.

If the WP is not set, the lower row of LEDs displays:



You may now enter the new WP address by pressing the desired sequence of keys 0 through 7. You can clear the WP by setting its address to zero.

Using the Watchpoint – Example 1

In our summation program in Paragraph 3.8, we used location 2044 to deposit our accumulated result. Since we would like to examine (and possibly alter) this location frequently, we set our watchpoint address at 2044.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
BPT	b r P n t 1	0 0 2 0 0 2	Current value of BP1 displayed on lower LEDs.
BAC	P n t A d r	0 0 0 0 0 0	
2	P n t A d r	0 0 0 0 0 2	Enter watchpoint address of 2044.
0	P n t A d r	0 0 0 0 2 0	
4	P n t A d r	0 0 0 2 0 4	
4	P n t A d r	0 0 2 0 4 4	

Now we can examine and alter the contents of location 2044 just as we would a register. If you run the summation program in Paragraph 3.8 and want to inspect the result, do the following.

REG	r E G		Get into register mode.
9	0 0 2 0 4 4	Current contents of 2044 shown on lower LEDs.	Examine WP.
EXA	0 0 2 0 4 4	Current contents of 2004 shown on lower LEDs.	
CLR	0 0 2 0 4 4	0 0 0 0 0 0	Clear it.

You have just cleared the contents of location 2044.

The watchpoint can be used in conjunction with the single-step function to continuously monitor the contents of a memory location. When this is done, the contents of the watchpoint appear on the upper row of LEDs, and the current PC appears on the lower row of LEDs.

Using the Watchpoint – Example 2

You are still working with the summation program in Paragraph 3.8. You have established a watchpoint at location 2044, as shown in the previous example, and want to continuously monitor the contents of this location as you single step through the program.

Set your initial conditions as before (R0 = 10, SP = 7400, PC = 2000) and key in the following.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
SST	0 0 0 0 0 0	0 0 2 0 0 2	2000 executed. 2044 contents shown on upper LEDs.
SST	0 0 0 0 0 0	0 0 2 0 0 4	2002 executed.
SST	0 0 0 0 0 0	0 0 2 0 0 6	2004 executed.
SST	0 0 0 0 0 0	0 0 2 0 1 2	2006 executed.
SST	0 0 0 0 1 0	0 0 2 0 1 6	2012 executed. 2044 has changed.
SST	0 0 0 0 1 0	0 0 2 0 2 0	2016 executed.
SST	0 0 0 0 1 7	0 0 2 0 2 4	2020 executed. 2044 has changed.

And so on. When you finish executing the program (e.g., after you have executed instruction 2032 in the example above), the upper row of LEDs displays:



and the lower row of LEDs displays the current PC.

3.9 USING THE HALT SWITCH

Pressing the HALT switch causes you to unconditionally halt the operation of your program and return control to the DCT11-EM keypad. When you press the HALT switch, the upper row of LEDs displays:



and the lower row of LEDs displays the current PC.

You can use the HALT switch, for example, to escape from an infinite loop your program is executing or to inhibit hardware problems which are interrupting the proper operation of the DCT11-EM.

After pressing HALT, you are in register mode and can readily examine the contents of the DCT11-EM registers as described in Paragraph 3.3.

3.10 THE INT SWITCH

Pressing the INT switch causes you to generate a power-fail (PF) nonmaskable interrupt. Pressing this switch will have no effect on the operation of the DCT11-EM unless you specifically write a program that recognizes input from it. The program in the following example shows you how to use this switch to experiment with interrupts. Also refer to Chapter 6 for more information on interrupts.

Using the INT Switch – Example

This program simply loops on itself at location 1510 until you press the INT switch. Then the current PC and PS are pushed onto the hardware stack and the DCT11-EM goes to locations 24 and 26 for its new PC and PS, respectively. Location 24 is called the power-fail vector and usually points to a power-fail service routine. The first line of your program, however, changes location 24 to point to your own “power-fail service routine.” This routine, which you specify to start at location 1514, is not a power-fail service routine at all, but a routine that sets R0 to the number of times (in octal) that the interrupt switch has been pressed.

Enter the following.

Location	Octal Code	Instruction	Comments
1500	012737	MOV #1514,@#24	
1502	001514		Set power-fail vector to “service routine.”
1504	000024		
1506	005000	CLR R0	Initialize R0.
1510	000777	BR 1510	Await interrupt.
1512	000000	HALT	
1514	005200	INC R0	Start of “service routine.”
1516	000002	RTI	Restore old PC and PS.

Start the program at 1500 and press the INT switch a number of times. The upper and lower row of LEDs will be blank while you are pressing the INT switch. Each time you press INT, you execute the “service routine” which increments R0. When the routine is done, it returns to 1510 to await another INT switch press. Press the HALT switch to suspend operation and examine R0.

3.11 THE USER LED

The user LED is a general-purpose indicator lamp that your program can use to indicate the status of some condition. You can turn on the user LED by writing 000011 to port C of the 8255A parallel port chip (see Paragraph 6.5.1). Port C of the 8255A chip has an address of 177444. Writing 000010 to port C of the 8255A chip turns off the user LED once it has been turned on by your program.

User LED – Example

The following is a short program that turns the user LED on and off. Use the keypad to enter this program starting at location 1000.

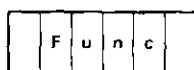
Location	Octal Code	Instruction	Comments
1000	012737	MOV #11,@#177444	Write 11 to port C.
1002	000011		
1004	177444		
1006	012700	MOV #100000,R0	Put a large number in R0.
1010	100000		
1012	077001	SOB R0,1012	Loop for a while.
1014	012737	MOV #10,@#177444	Write 10 to port C.
1016	000010		
1020	177444		
1022	012700	MOV #100000,R0	Put a large number in R0.
1024	100000		
1026	077001	SOB R0,1026	Loop for a while.
1030	000137	JMP @#1000	Loop infinitely.
1032	001000		

3.12 PERFORMING KEYPAD SPECIAL FUNCTIONS

There are five types of special functions you can perform through the use of the DCT11-EM keypad.

Function Number	Function Name
0	Cancel breakpoints
1	Go with LEDs
2	Start console
3	Set baud rates
4	Release protection

To perform one of these special functions, you must first place the DCT11-EM in function mode. To do this, press the FNC key. The LEDs display:



and you can now select a function by pressing any of the keys 0 through 4 or CLR, which is equivalent to 0. When you have done this, the number of the selected function appears on the bottom row of LEDs. The selected special function is then activated by pressing the EXA key.

The following paragraphs describe the operation of these special functions.

3.12.1 Function 0 – Cancel Breakpoints

Function 0 clears all four breakpoint addresses and the watchpoint address. When function 0 is activated, the upper row of LEDs displays:



and the lower row of LEDs displays the current value of the PC. The DCT11-EM is left in register mode.

Function 0 – Example

You are debugging a program you just entered and want to clear all the breakpoints and watchpoints associated with an old program.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
FNC	F u n c		Get into function mode.
0	F u n c	0	Select function 0.
EXA	P C	Current PC shown on lower LEDs.	
BPT	b r P n t 1	0 0 0 0 0 0	Verify breakpoint 1 cleared.
BAC	P n t A d r	0 0 0 0 0 0	Verify watchpoint address cleared.
ADV or BPT	b r P n t 1	0 0 0 0 0 0	Breakpoint 1 again.
ADV	b r P n t 2	0 0 0 0 0 0	Verify breakpoint 2 cleared.

3.12.2 Function 1 – Go With LEDs

Within the DCT11-EM are a number of routines your program can call to display values on the LEDs (see Paragraph 5.3.1). For these routines to operate correctly, you must explicitly turn on the LEDs before you call a routine. One way to turn on the LEDs is to include a `MOV #7,@#177444` instruction in your program (see Paragraph 5.3.1.1). Another way is to execute your program by activating function 1 rather than by pressing the GO key. Activating function 1 turns on the LEDs and starts your program.

3.12.3 Function 2 – Start Console

If you connect a console terminal to your DCT11-EM, you will use function 2 frequently. Activating function 2 starts a program called the console monitor. The console monitor resides in the on-board PROMs and allows you to enter, execute, and debug programs through the use of a console terminal rather than the keypad. The console monitor also allows you to perform other operations and is described in more detail in Chapter 4.

Before activating function 2, connect a console terminal to the DCT11-EM and select an appropriate baud rate by activating function 3 (see Paragraph 3.12.4).

When function 2 is activated, the upper and lower LEDs are blank and the message:

```
TEM CONSOLE MONITOR V1.0
TEM>
```

appears on the console terminal.

3.12.4 Function 3 – Set Baud Rates

The DCT11-EM has two serial ports: a console terminal port and an auxiliary serial port. Activating function 3 allows you to select baud rates for these ports. When the DCT11-EM is powered up, it automatically sets both baud rates to 9600.

When you first activate function 3, the upper row of LEDs displays:

C	o	n	s	o	l
---	---	---	---	---	---

and the lower row of LEDs displays:

0	9	6	0	0
---	---	---	---	---

This tells us that the console terminal port is currently set at 9600 baud. Note that the baud rate is displayed as a decimal number.

Pressing the ADV and BAC keys steps you backward and forward among the various baud rate values available for the console terminal port. The available baud rates for the console terminal port are: 300, 600, 1200, 2400, 4800, 9600, and 19200.

You can now set the baud rate of the auxiliary serial port by pressing the EXA key. The first time you do this, the upper row of LEDs displays:

A	P	o	r	t
---	---	---	---	---

and the lower row of LEDs displays:

0	9	6	0	0
---	---	---	---	---

This tells us that the auxiliary serial port is currently set at 9600 baud. Repetitively pressing any of the keys CLR, 0 through 9, ADV, or BAC causes you to alternate between two available baud rates. The two available baud rates for the auxiliary serial port are: the current baud rate of the console terminal port and one-fourth the current baud rate of the console terminal port. Pressing EXA again puts you into register mode.

Function 3 - Example

You want to use function 3 to experiment with setting baud rates for the console terminal serial port and the auxiliary serial port. You have not altered these baud rates since powering up.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
FNC	F u n c		Get into function mode.
3	F u n c	3	Select function 3.
EXA	C o n S o l	0 9 6 0 0	Baud rate for console terminal port = 9600.
ADV	C o n S o l	1 9 2 0 0	
BAC	C o n S o l	0 9 6 0 0	
BAC	C o n S o l	0 4 8 0 0	
BAC	C o n S o l	0 2 4 0 0	
BAC	C o n S o l	0 1 2 0 0	
EXA	A . P o r t	0 1 2 0 0	Baud rate for auxiliary port = terminal port.
0	A . P o r t	0 0 3 0 0	300 = one fourth of 1200.
8	A . P o r t	0 1 2 0 0	Back to 1200.
ADV	A . P o r t	0 0 3 0 0	Back to 300.
EXA	P C	Current PC shown on lower LEDs.	Now in register mode. Console port = 1200 Auxiliary port = 300.

3.12.5 Function 4 – Release Protection

There is an area of RAM that the DCT11-EM normally prevents you from directly altering from the keypad. This area is called the keypad monitor scratchpad and occupies the memory space from addresses 7400 to 7777. Refer to Paragraphs 5.2 and 5.4.1 for more information about the keypad monitor and this protected area.

Additionally, if you have a console terminal connected to your DCT11-EM, there are two other areas of RAM that are protected. These areas are the console monitor scratchpad and the symbol table. They occupy the memory space from addresses 6500 to 7377. Again, more information is provided in Chapter 5. Activating function 4 releases the protection associated with these memory spaces. Note that “protection” refers to protection from direct alteration via the keypad or console terminal. Programs that you write can and will alter these locations. Therefore, when you write your programs make sure these spaces are not inadvertently violated.

Protection is restored when you press the HALT switch, when a HALT instruction is encountered, or whenever you restart the DCT11-EM.

When you activate function 4, the upper row of LEDs displays:



and the lower row of LEDs displays the current value of the PC.

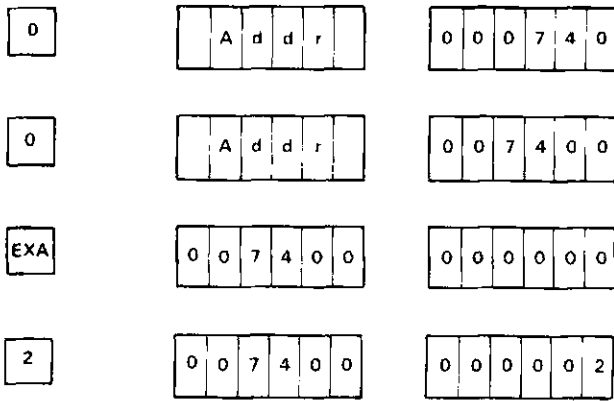
Function 4 – Example

You want to alter location 7400, a normally protected location.

KEY	UPPER DISPLAY	LOWER DISPLAY	COMMENTS
FNC	F u n c		Get into function mode.
4	F u n c	4	Select function 4.
EXA	P C	Current PC shown on lower LEDs'	Protection off.

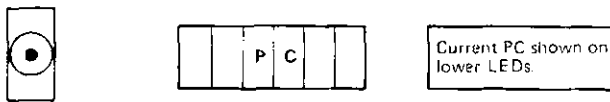
Alter the location.

ADR	A d d r	Most recently examined address shown on lower LEDs.	
7	A d d r	0 0 0 0 0 7	Alter 7400
4	A d d r	0 0 0 0 7 4	



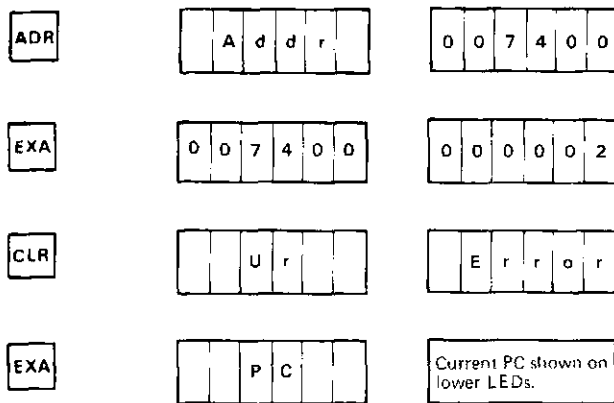
Enter 2.

Restore protection by pressing the HALT switch.



Protection restored.

Now try to alter location 7400.



When you try to clear 7400 you get a flashing error message.

Now in register mode.

See Appendix C for more information on error messages.

3.13 KEYPAD COMMAND SUMMARY

OPERATION	PRESS	ENTER	PRESS	ENTER	REFERENCE
ALTER MEMORY	ADR	LOCATION ADDRESS	EXA	NEW CONTENTS	3.6
ALTER REGISTER	REG	REGISTER NUMBER	EXA	NEW CONTENTS	3.4
CANCEL BREAKPOINTS	FNC	0	EXA		3.12.1
EXAMINE MEMORY	ADR	LOCATION ADDRESS	EXA		3.5
EXAMINE REGISTER	REG	REGISTER NUMBER			3.3
EXECUTE PROGRAM	GO				3.7
GO WITH LEDs	FNC	1	EXA		3.12.2
RELEASE PROTECTION	FNC	4	EXA		3.12.5
SET BAUD RATES	FNC	3	EXA		3.12.4
SET BREAKPOINT	BPT	BREAKPOINT ADDRESS			3.8.2
SET WATCHPOINT	BPT		BAC	WATCHPOINT ADDRESS	3.8.3
SINGLE STEP	SST				3.8.1
START CONSOLE	FNC	2	EXA		3.12.3

CHAPTER 4 CONSOLE OPERATION

4.1 INTRODUCTION

This chapter explains how to operate the DCT11-EM using a console terminal. Using a console terminal simplifies program development because it gives you access to the DCT11-EM's symbolic assembler/disassembler. In addition, a console terminal allows you to perform several functions not accessible with the keypad, such as the ability to load programs from a host computer.

When you complete this chapter, you will be able to use a console terminal to:

- Examine and alter the contents of registers in the DCT11-AA register file
- Examine and alter the contents of memory locations
- Enter and execute programs
- Debug your programs through the use of single steps, breakpoints, and watchpoints
- Load programs from a host computer
- Perform a variety of other console functions.

You can connect an RS232-C compatible console terminal to your DCT11-EM as shown in Figure 4-1.

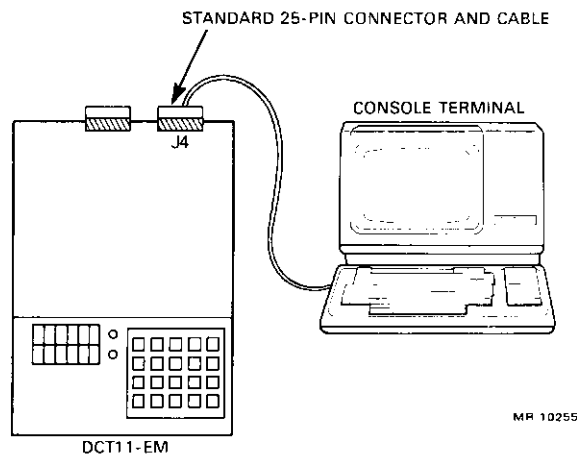


Figure 4-1 Connecting a Console Terminal to the DCT11-EM

No other wiring or jumpering is required. Use the DCT11-EM keypad to set the correct baud rate by activating function 3 (see Paragraph 3.12.4). Also, make sure your console terminal is set up to transmit and receive characters in the following format: asynchronous, ASCII, 8 bits (eighth bit is always space), no parity, one stop bit.

Then activate function 2 as described in Paragraph 3.12.3. The terminal displays:

```
TEM CONSOLE MONITOR V1.0  
TEM>
```

and the console monitor is ready to accept your input.

The following default conditions are set when the console monitor is started:

1. The stack pointer is set to 6500
2. The PASS2, NOHOST, INSTRU, SYMBOL, and VTOFF operating modes are set.

More information about the stack pointer is given in Paragraph 5.2.1. The operating modes are described in detail elsewhere in this chapter and are summarized in Paragraph 4.18.

4.2 TRANSFERRING CONTROL BETWEEN THE CONSOLE TERMINAL AND THE KEYPAD

Either the console or the keypad can control the DCT11-EM, but not both at the same time.

From the keypad, your primary means of transferring control to the console is by activating function 2 (see Paragraph 3.12.3). You can regain keypad control by pressing the HALT switch. When you press the HALT switch, DCT11-EM operation is stopped and the LEDs display the current PC.

From the console, you can transfer control back to the keypad by typing in the EXIT command or by executing a HALT instruction.

4.3 GENERAL RULES FOR CONSOLE INPUT

Information you input via the console terminal is typically in the form of commands, instructions, or data.

In general, your input should conform with the following guidelines.

1. Your input line must not exceed 96 characters in length. If you exceed this limit, the console terminal does not accept further input until you type in a CTRL/C or CTRL/Y. Then, you must reinput the line.
2. Any number of commands and instructions can be strung together on a line as long as the line does not exceed 96 characters.
3. Commands or instructions strung together on a line should be separated with a space or a comma. Extra separation characters (space, tab, line feed, form feed) between commands or instructions on a line are ignored.
4. If a command or an instruction in a one-line string causes an error, the commands or instructions following it are ignored.
5. Extra separation characters (space, tab, line feed, form feed) inserted between parts of a command or instruction are ignored.
6. Information preceded by a semicolon on a line is interpreted as a comment and is ignored by the DCT11-EM.

7. Pressing the DELETE key deletes characters on a line. As characters are deleted, they are echoed (preceded by a slash character) on the console terminal if you are in VTOFF mode. The console monitor defaults to VTOFF mode when it is started or restarted. Type in the VTON command if you are using a video terminal as a console device. VTON mode eliminates the echoing of deleted characters. Typing in VTOFF returns you to VTOFF mode from VTON mode.
8. A line is terminated by pressing the RETURN key. Characters on a line are ignored until you press RETURN. In all the examples in this chapter, a RETURN is used to end your input line unless otherwise noted.

A summary of the various forms of acceptable console input is given in Paragraph 4.18. Information about specific commands and instruction formats is given elsewhere in this chapter.

4.4 CONSOLE CONTROL COMMANDS

The DCT11-EM recognizes several control commands that allow you to easily perform a variety of operations from the console terminal. These commands and their typical uses are listed in Table 4-1.

Table 4-1 Control Command Summary

Command	Meaning	Typical Use
CTRL/U CTRL/X	Delete line.	You made several typing errors in your input line. You want to cancel the line and try again.
CTRL/R	Display line.	You used the DELETE key several times while typing your input line. You want to see a clean version of the line you are editing.
CTRL/S	Suspend execution.	You want to suspend the operation of the console monitor or the execution of a currently running program.
CTRL/Q	Resume execution.	You have previously used the CTRL/S command and want to resume operation.
CTRL/C CTRL/Y	Abort operation.	You want to abort the current operation of the DCT11-EM and input a new line.
CTRL/O	Ignore host message.	You are in HOST mode (see Paragraph 4.17.1) and want to ignore information currently being sent by your host computer.
CTRL/A	Select option.	You are in TALK or LOAD modes (see Paragraphs 4.17.3 and 4.17.4) and want to select one of these four options: exit, resume, send break, send CTRL/A.

4.5 EXPRESSIONS

Several commands and instructions accept expressions as parameters. This paragraph familiarizes you with expressions and provides the rules for their definition.

If you wish to ignore the use of expressions for the moment, you can advance to the paragraph that describes the command or function you are interested in. You can return to this paragraph when you need more detail on how to define an expression for a particular command or instruction.

An expression is simply a user-definable combination of one or more components called terms that represents a 16-bit number. There are three types of terms allowable in a DCT11-EM expression:

- Numeric
- Literal
- Symbolic

Numeric terms are either octal or decimal numbers. Octal numbers can be any sequence of the digits 0 through 7. Decimal numbers can be any sequence of the digits 0 through 9 followed by a decimal point (.). Do not attempt to define decimal fractions.

Examples of Numeric Terms

100702
377
39874.

A literal term is an ASCII character you use as data. Precede an ASCII character with an apostrophe (') to form a literal term.

Examples of Literal Terms

'E
'*
'4

Symbolic terms are values symbolized by names which are defined according to the rules in Paragraph 4.13.

Examples of Symbolic Terms

R1
P\$PORC
SUM

Terms are joined together in an expression by binary operators. (The word binary in this case means that at least two terms are involved.) The DCT11-EM allows six types of binary operators:

+ (addition)
- (subtraction)
* (multiplication)
/ (division)
& (logical AND)
! (logical OR)

The unary (one term) operators, + (plus sign) and - (minus sign), can only be used at the beginning of an expression (or parenthesized term). Multiple unary operators are not valid.

Expressions are evaluated from left to right with no operator hierarchy rules. Two types of brackets can be used to nest expressions up to seven levels deep. Use angle brackets (< >) as you would arithmetic parentheses and use square brackets ([]) if you want the DCT11-EM to substitute the enclosed information with the contents of the memory word it addresses. Note that if the information in square brackets represents an odd address, it is first rounded down.

A quick way to evaluate the octal value of an expression is to type in the expression followed by a question mark. This command has the form:

expression?

Expressions – Examples

The following are examples of valid DCT11-EM expressions. You can verify the octal values of these expressions by using the question mark command.

Expression	Octal Value
65209	177271
7077+189.+’E	007501
-2*<300/25>+70	000046
120.*<-8.>	176100
177777&25+3	000030

4.6 EXAMINING REGISTERS

There are two ways to examine registers. One way is to type in the appropriate register symbol followed by a question mark. This command has the general format:

register symbol?

where the register symbol (R0, R1, etc.) is as defined in Paragraph 4.13.1.

Another way to examine registers is to type in the SHOWRE command. The SHOWRE command displays the values of all the registers and the PS in ascending order. If a watchpoint has been set, it is also displayed. Paragraph 4.14.1, which describes program debugging, gives you an illustration of the SHOWRE command.

4.7 ALTERING REGISTERS

Registers are altered by using an equal sign to set the register to the desired value. The general format for this is:

register symbol=expression

where the register symbol (R0, R1, etc.) is as defined in Paragraph 4.13.1 and the register is set to the value of the expression. (See Paragraph 4.5 for expression formats and conventions.)

Altering Registers – Example

Alter and examine the values of R0 and the SP.

```
TEM> R0=65.  
TEM> SP=6500  
TEM> R0? SP?  
000101  
006500  
TEM>
```

4.8 EXAMINING AND ALTERING MEMORY LOCATIONS – INTRODUCTORY CONCEPTS

There are several ways to examine and alter memory locations. Before discussing the specific techniques, there are some introductory concepts you should be familiar with.

4.8.1 The Address Pointer

The address pointer specifies your current *open location*, i.e., the address of the location currently available to you for alteration. It is important to keep track of the address pointer because it determines where in memory you deposit information. As you will discover later, most commands which alter and examine memory locations are closely linked to the value of the address pointer. This paragraph gives an overview of how the address pointer can be affected.

Since the DCT11-EM recognizes a period (.) as the symbol for the address pointer, you can manipulate the value of the address pointer just as you would any other symbol (see Paragraph 4.13.2). For example, to set the value of the address pointer use an equal sign:

```
.=expression
```

where the value of the address pointer is determined by the expression (see Paragraph 4.5). To determine the current value of the address pointer, use a question mark:

```
.?
```

Another way to set the address pointer is to use the backslash (\) character. The general format for a backslash command is:

```
expression\
```

where the value of the address pointer is determined by the expression. This command also displays the memory location specified by the expression (see Paragraph 4.9).

There are two other commands which affect the address pointer. These commands are the caret (^) command and the RETURN (or null line input) command. The effects of these commands depend upon which output format mode the DCT11-EM is in (see Paragraph 4.8.2). A description of their effects is given in Paragraphs 4.9 and 4.10.

When you type in a program, the DCT11-EM automatically increments the address pointer for each instruction or data object you input. Note that whenever you power up, the address pointer is initialized to zero.

Do not confuse the address pointer with the PC (program counter). They are unrelated. As you type in a program, the address pointer is incremented, but the PC is unaffected. And the execution of a program will affect the PC but not necessarily the address pointer.

4.8.2 Output Formatting Modes

When you examine memory locations, the values the DCT11-EM displays on the console depend upon your current output formatting mode. There are five output formatting modes: INSTRU, WORD, BYTE, SYMBOL, and ABSOLU.

The INSTRU (or instruction disassembly) mode causes memory locations to be displayed as disassembled DCT11-AA instructions. INSTRU is one of the modes that the DCT11-EM is initialized to when you power up.

The WORD and BYTE modes cause memory locations to be displayed octally. Set the WORD mode (by typing in the WORD command) if you want the values of memory locations to be displayed as 16-bit words. Set the BYTE mode (by typing in the BYTE command) if you want the values of memory locations to be displayed as individual bytes. If you are in WORD or BYTE mode, type in the INSTRU command to return to instruction disassembly mode. Notice that the INSTRU, WORD, and BYTE commands are mutually exclusive. To summarize:

Mode	Memory Locations Are Displayed As
INSTRU	Disassembled DCT11-AA instructions
BYTE	8-bit octal bytes
WORD	16-bit octal words

Odd addresses (i.e., upper bytes) can only be examined in BYTE mode. If you try to examine an odd address in INSTRU or WORD mode, an error message results.

The SYMBOL and ABSOLU modes determine how the symbols you have defined in your program are to be displayed. You will usually use SYMBOL and ABSOLU in conjunction with instruction disassembly (INSTRU) mode.

If your program contains user-defined symbols (see Paragraph 4.13), these symbols will be displayed only if you are in SYMBOL mode. SYMBOL is one of the modes the DCT11-EM is initialized to when you power up.

If you want user-defined symbols to be displayed as octal values, type in the ABSOLU command to set the ABSOLU mode. Typing in SYMBOL restores the SYMBOL mode. Note that SYMBOL and ABSOLU are mutually exclusive. To summarize:

Mode	User-Defined Symbols Are Displayed As
SYMBOL	Symbols
ABSOLU	Octal values

4.9 EXAMINING MEMORY LOCATIONS

This paragraph summarizes the commands with which you can examine memory locations. The examples in this paragraph are intended to give you a general idea of what to expect when you use these commands in different output formatting modes.

One way to examine a memory location is to type in the desired memory location followed by a backslash. The general format for this command is:

```
expression\
```

where the memory location you examine is determined by the value of the expression (see Paragraph 4.5). The backslash command also sets the address pointer, so a location examined with a backslash becomes the current open location.

Examining Memory Locations – Example 1

Use the backslash command to examine a memory location. Try this in different output formatting modes. You have not altered any memory locations from their initial (power-up) values.

```
TEM> WORD
TEM> 2000\
002000: 000000
TEM> 100\
000100: 135676
TEM> BYTE
TEM> 100\
000100: 276
TEM> INSTRU
TEM> 140000\
140000: JMP 155334
140004:
TEM>
```

Examining Memory Locations – Example 2

You use the backslash command to try to examine location 1001 without first typing in the BYTE command. This causes an error message to be generated.

```
TEM> INSTRU
TEM> 1001\
001001:
///
ODD ADDRESS
TEM>
```

Another way to examine memory locations is to simply press the RETURN key. When you press the RETURN key as a command input, the result depends on both your output formatting mode and the value of the address pointer. If you are in INSTRU mode, RETURN displays the next eight instructions relative to the address pointer. In WORD mode, the contents of the next eight words relative to the address pointer are displayed. In BYTE mode, the contents of the next eight bytes relative to the address pointer are displayed. Note that the RETURN command is ignored when you are loading a program from a host computer (see Paragraph 4.17).

Examining Memory Locations – Example 3

You want to examine memory locations by pressing the RETURN key. You have not altered these locations since you have powered up. Start with location 1000.

```
TEM> INSTRU
TEM> 1000\
001000: HALT
001002:
TEM>
001000: HALT
001002: HALT
001004: HALT
001006: HALT
001010: HALT
001012: HALT
001014: HALT
001016: HALT
001020:
TEM> WORD
TEM>
001000: 000000    000000    000000    000000    000000    000000    000000    000000
TEM>
001020: 000000    000000    000000    000000    000000    000000    000000    000000
TEM> BYTE
TEM>
001020: 000      000      000      000      000      000      000      000
TEM>
001030: 000      000      000      000      000      000      000      000
TEM>
```

A one-line string of bytes or words is numbered from left to right.

Note that consecutive RETURN keypresses increment the address pointer but nonconsecutive ones do not. For example, in WORD mode you incremented the address pointer from 1000 to 1020 by pressing RETURN twice in a row. In BYTE mode you incremented the address pointer from 1020 to 1030 by pressing RETURN twice in a row. Pressing RETURN only once (as you did in INSTRU mode) had no effect on the address pointer.

A way to examine a range of memory locations is to use the underscore (___) character. The general format for this command is:

expression1___expression2

where the beginning of the range is determined by the value of expression1 and the end of the range is determined by the value of expression2. See Paragraph 4.5 for expression formats.

Expression1 should be less than expression2. If it is not, only one location is displayed: the beginning of range location.

The use of the underscore command has no effect on the address pointer.

Examining Memory Locations – Example 4

You want to examine groups of memory locations by using the underscore command. Try this command in INSTRU, WORD, and BYTE modes. These memory locations have not been altered since you have powered up.

```
TEM> INSTRU
TEM> 2000\
002000: HALT
002002:
TEM> 1004__1012
001004: HALT
001006: HALT
001010: HALT
001012: HALT
001014:
TEM> WORD
TEM> 1002__1006
001002: 000000      000000      000000
TEM> BYTE
TEM> 1005__1020
001005: 000      000      000      000      000      000      000
001015: 000      000      000
TEM> INSTRU
TEM>
002000: HALT
002002: HALT
002004: HALT
002006: HALT
002010: HALT
002012: HALT
002014: HALT
002016: HALT
002020:
TEM>
```

Notice that the address pointer value of 2000 which was set at the beginning of the example is still 2000 at the end of the example.

4.10 ALTERING MEMORY LOCATIONS

You can deposit data into a memory location by setting the address pointer to the address of the location you wish to alter and typing in the information. The address pointer is automatically incremented as you deposit instructions and data to consecutive memory locations (as when you type in a program, for example).

If you make an error during data entry, you can use the caret (^) command to decrement the address pointer and retype the input line. In INSTRU and WORD modes, the address pointer is decremented by one word for each caret command, and in BYTE mode it is decremented by one byte for each caret command.

Altering Memory Locations – Example 1

You type in a short program starting at address 1000. You make an error while typing in the program and use the caret (^) command to help you correct the error. You then examine the program. The program adds R1 to R2 and places the result in R3. The final line restarts the console monitor so that you will get a TEM> prompt after you execute the program.

	Comments
TEM> INSTRU	
TEM> .=1000	
TEM> MOV R1,R3	
TEM> ADD R2,R4	;you really wanted to type ADD R2,R3
TEM> ^	;decrement address pointer by one word
TEM> ADD R2,R3	;type in correct line
TEM> JMP @#140010	;continue typing in program
TEM> 1000\ 001000: MOV R1,R3	;now examine your program
001002:	
TEM>	
001000: MOV R1,R3	
001002: ADD R2,R3	
001004: JMP @#140010	
001010: HALT	
001012: HALT	
001014: HALT	
001016: HALT	
001020:	
001022:	
TEM>	

Altering Memory Locations – Example 2

You type in some data starting at location 2000. You type several values on a line, remembering to include separation characters (in this case, spaces) between them. You then examine the data you have typed in.

	Comments
TEM> .=2000	
TEM> 1 2 3 4	;type in four data values
TEM> ?	;the address pointer is incremented
002010	
TEM> .=2000	;reinitialize the address pointer
TEM> WORD	
TEM> ..+.10	;examine the data
002000: 000001 000002 000003 000004 000000	
TEM>	

4.11 INSTRUCTION FORMAT

This paragraph provides some guidelines for formatting instructions for input to the DCT11-EM. A complete description of the instruction set is found in the *DCT11-AA User's Guide* (EK-DCT14-UG).

An acceptable instruction consists of the instruction mnemonic followed by the appropriate string of operands (if any). A separation character (space, tab, feed) must be used to separate a mnemonic from its first operand (if any). A comma must appear between two operands in a two-operand instruction.

Expressions can be used as operands, if desired. Refer to Paragraph 4.5 for expression formats.

If you wish, you may enter your instructions in octal format, rather than in assembly language format. Refer to Chapter 6 of the *DCT11-AA User's Guide* for information on constructing octal equivalents of instructions.

4.12 TYPING IN A PROGRAM

Set the address pointer to 2000 and type in the summation program from Paragraph 3.8 as shown below.

```
TEM> .=2000
TEM> CLR R1
TEM> CMP R0,R1
TEM> BLE 2034
TEM> CLR @#2044
TEM> MOV R0,@#2044
TEM> DEC R0
TEM> ADD R0,@#2044
TEM> BVS 2036
TEM> CMP R0,R1
TEM> BGT 2016
TEM> HALT
TEM> CLR R0
TEM> CLR @#2044
TEM> HALT
TEM>
```

Now verify you have typed in the program correctly by setting the address pointer to 2000 and repeatedly pressing the RETURN key. If you are in INSTRU mode, the following will be displayed.

```
TEM> .=2000
TEM>
002000: CLR          R1
002002: CMP          R0,R1
002004: BLE          002034
002006: CLR          @#002044
002012: MOV          R0,@#002044
002016: DEC          R0
002020: ADD          R0,@#002044
002024: BVS          002036
002026:
TEM>
002026: CMP          R0,R1
002030: BGT          002016
002032: HALT
002034: CLR          R0
002036: CLR          @#002044
002042: HALT
002044: HALT
002046: HALT
002050:
TEM>
```

The address pointer is automatically incremented each time you type in an instruction.

You may want to use symbolic names to represent values such as 2044 (the location of the accumulated sum) to increase the readability of the program. The next few paragraphs tell you how to define and manipulate symbols.

If you want to execute the program and learn how to use the DCT11-EM's console debugging facilities, advance to Paragraph 4.14.

4.13 USING SYMBOLS

The DCT11-EM allows you to define symbols in your programs. This paragraph describes the formats required for DCT11-EM symbols and explains the use of the commands which allow you to define and manipulate symbols.

4.13.1 Symbol Names

The following characters may be used in symbol names.

- A through Z
- 0 through 9
- . (Period)
- \$ (Dollar sign)

A symbol name must not begin with 0 through 9, however.

A symbol name may have any length; however, only the first six characters are stored by the DCT11-EM. Thus, to avoid ambiguities, make the first six characters of your symbol names unique.

The following predefined symbol names are reserved.

R0	Register 0
R1	Register 1
R2	Register 2
R3	Register 3
R4	Register 4
R5	Register 5
SP	Stack pointer (register 6)
PC	Program counter (register 7)
PS	Processor status word
WP	Watchpoint address
.	Address pointer
B1	Breakpoint 1
B2	Breakpoint 2
B3	Breakpoint 3
B4	Breakpoint 4

These predefined symbols must not be used as user-defined symbol names, although they can be used as arguments in expressions, instructions, or commands.

DCT11-EM directives, commands, and DCT11-AA instruction mnemonics (e.g., MOV, ADD, JMP) are also reserved and must not be used as symbol names.

4.13.2 Commands That Define And Manipulate Symbols

There are two ways you can define a symbol. One way is to use the equal sign (=) to equate a symbol to a specific value. The general format of this is:

symbol=expression

where the value of the symbol is determined by the expression.

The other way of defining a symbol is to use a colon (:). The colon defines a symbol as the label of the current address. The general format of this is:

symbol:

where the value of the symbol is defined as the value of the address pointer.

Once you have defined a symbol, you cannot redefine it unless you first delete it. Symbol deletion is accomplished with the DELETE command. This command has the general format:

DELETE symbol

where the name of the symbol you wish to delete follows the command. The DELETE command causes the DCT11-EM to display the symbol and its value as it performs the deletion.

Predefined symbols (see Paragraph 4.13.1) cannot be deleted with the DELETE command. Predefined symbols can be redefined, however, by using the equal sign.

When you define a symbol, it is stored in an area called the user symbol table. There is sufficient space in the user symbol table to store up to 66 symbols. If you try to define more than 66 symbols, you will cause an error message to be generated.

If you want to display your user symbol table, type in the SHOWSY command. The symbols are displayed in the order in which you defined them, starting with the most recently defined symbol.

If you want to delete your entire symbol table, type in the CLEAR command. The CLEAR command is typically used at the beginning of a long program or a program you are loading in from a host computer (see Paragraph 4.17). CLEAR gives the program plenty of room in the symbol table for symbols and avoids conflicts between symbol names in the program and any symbol names you may have previously defined.

If you want to display the value of an individual symbol, type in the symbol followed by a question mark (?). The general format for this is:

symbol?

Defining and Manipulating Symbols – Example 1

You set the address pointer to 1000 and define the symbols FIRST, SECOND, and THIRD. You experiment with displaying and deleting these symbols.

```
TEM> CLEAR
TEM> .=1000
TEM> FIRST:
TEM> SECOND=FIRST+100
TEM> THIRD=SECOND+300
TEM> SHOWSY
THIRD = 001400
SECOND= 001100
FIRST = 001000
TEM> DELETE FIRST
FIRST = 001000
TEM> SHOWSY
THIRD = 001400
SECOND= 001100
TEM> CLEAR
TEM> SHOWSY
TEM>
```

SECOND is set equal to 1100 because FIRST equals 1000 at the time of definition. If you delete FIRST after you delete SECOND SECOND will be unaffected.

4.14 EXECUTING AND DEBUGGING PROGRAMS

Refer to the summation program you typed in (Paragraph 4.12). You will make some minor changes to this program and use the DCT11-EM's debugging facilities to observe its operation.

First, replace memory location numbers with symbolic names (see Paragraph 4.13).

```
TEM> ILLIN=2034
TEM> SUM=2044
TEM> OVFL0=2036
TEM> LOOP=2016
TEM>
```

Instead of having the program HALT when it completes execution (which returns control to the keypad), you want it to restart the console monitor and give you a TEM> prompt. You do this by placing a JMP @#140010 instruction at location 2032 which restarts the console monitor when executed.

```
TEM> .=2032
TEM> JMP @#140010
TEM>
```

Now set your initial conditions as follows.

```
TEM> R0=100
TEM> PC=2000
TEM> SP=6500
TEM>
```

Then execute the program by typing in the GO command.

```
TEM> GO
TEM>
```

To check the result, examine SUM.

```
TEM> SUM\
SUM  : 004040
TEM>
```

This is the correct sum in octal of the numbers from 0 to 100 (octal).

4.14.1 Using The Watchpoint

The DCT11-EM recognizes WP as the symbol for the watchpoint. Set the watchpoint address by using the equal sign command. The general format for this is:

```
WP=expression
```

where the watchpoint address is determined by the value of the expression.

For example, to monitor the accumulated sum in your program, set your watchpoint at the location SUM.

```
TEM>WP=SUM
TEM>
```

Now reset your initial conditions (set R0=100) and execute the program. After you receive the TEM> prompt, use the SHOWRE command to examine your registers.

```
TEM> SHOWRE
R0      = 000000
R1      = 000000
R2      = 000000
R3      = 000000
R4      = 000000
R5      = 000000
SP      = 006500
PC      = 002000
PS      = 000004
SUM     : 004040
TEM>
```

Notice your watchpoint appears as well. Another way to examine the watchpoint is to use the question mark command. WP? displays numerically the watchpoint address. [WP]? displays numerically the watchpoint contents.

```
TEM> WP?
002044
TEM> [WP]?
004040
TEM>
```

The watchpoint can be cleared by setting it to zero or by typing in the CANCEL command, which not only clears the watchpoint but the breakpoints as well.

4.14.2 Using the Single-Step Function

Single stepping is accomplished by typing in the STEP command. STEP executes the instruction at the current PC and displays: (1) the new values of your registers and watchpoint, and (2) the next instruction to be executed.

To experiment with the STEP command, reset the initial conditions for your program (set R0=100) and type in STEP. You can then continue single stepping by repeatedly pressing the RETURN key.

```
TEM> STEP
R0   = 000100
R1   = 000000
R2   = 000000
R3   = 000000
R4   = 000000
R5   = 000000
SP   = 006500
PC   = 002002
PS   = 000024
SUM  : 004040
002002 : CMP  R0,R1
002004 :
TEM>
R0   = 000100
R1   = 000000
R2   = 000000
R3   = 000000
R4   = 000000
R5   = 000000
SP   = 006500
PC   = 002004
PS   = 000020
SUM  : 004040
002004 : BLE  ILLIN
002006 :
TEM>
R0   = 000100
R1   = 000000
R2   = 000000
R3   = 000000
R4   = 000000
R5   = 000000
SP   = 006500
PC   = 002006
PS   = 000020
SUM  : 004040
002006 : CLR  @#SUM
002012 :
TEM>
R0   = 000100
R1   = 000000
R2   = 000000
R3   = 000000
R4   = 000000
R5   = 000000
```

```

SP    = 006500
PC    = 002012
PS    = 000024
SUM   : 000000
002012 : MOV  R0,@#SUM
LOOP  :
TEM>
R0    = 000100
R1    = 000000
R2    = 000000
R3    = 000000
R4    = 000000
R5    = 000000
SP    = 006500
PC    = LOOP
PS    = 000020
SUM   : 000100
LOOP  : DEC  R0
002020 :
TEM>
R0    = 000077
R1    = 000000
R2    = 000000
R3    = 000000
R4    = 000000
R5    = 000000
SP    = 006500
PC    = 002020
PS    = 000020
SUM   : 000100
002020 : ADD  R0,@#SUM
002024 :
TEM>
R0    = 000077
R1    = 000000
R2    = 000000
R3    = 000000
R4    = 000000
R5    = 000000
SP    = 006500
PC    = 002024
PS    = 000020
SUM   : 000177
002024 : BVS  OVFL0
002026 :
TEM>

```

The program is functioning properly; R0 is getting decremented and added to SUM.

4.14.3 Using Breakpoints

The use of breakpoints with a console terminal is similar to the use of breakpoints with the keypad (see Paragraph 3.8.2).

The DCT11-EM recognizes B1, B2, B3, and B4 as the symbols for the four breakpoints. Set a breakpoint by using the equal sign command. The general format for this is:

```
breakpoint symbol=expression
```

where the breakpoint is determined by the value of the expression.

For example, to set breakpoints at locations 2020 and 2024 in the summation program (as you did with the keypad), type in:

```
TEM> B1=2020
TEM> B2=2024
TEM>
```

Now reset your initial conditions (set R0=100, PC=2000) and execute the program.

```
TEM> GO
BREAK POINT ENCOUNTERED
002020: ADD R0,@#SUM
002024:
TEM>
```

You get a message which indicates that your first breakpoint (B1) was encountered by the console monitor. To determine the current state of your program, type in the SHOWRE command.

```
TEM> SHOWRE
R0 = 000077
R1 = 000000
R2 = 000000
R3 = 000000
R4 = 000000
R5 = 000000
SP = 006500
PC = 002020
PS = 000000
SUM : 000100
TEM>
```

You see that R0 has been correctly decremented. When you type in GO again:

```
TEM> GO
BREAK POINT ENCOUNTERED
002024: BVS  OVFLO
002026:
TEM>
```

the console message appears (this time because of B2), and you again examine the state of your program.

```
TEM> SHOWRE
R0  = 000077
R1  = 000000
R2  = 000000
R3  = 000000
R4  = 000000
R5  = 000000
SP  = 006500
PC  = 002024
PS  = 000000
SUM : 000177
TEM>
```

You see that the first addition was performed correctly. You can continue to observe the operation of the program by typing in the GO and SHOWRE sequence as just shown.

To examine a breakpoint, type in B1?, B2?, B3?, or B4?. Breakpoints can be cleared by setting them to zero or by typing in the CANCEL command, which not only clears the breakpoints but the watchpoint as well.

4.15 DCT11-EM DIRECTIVES

As a convenience for users who are familiar with assemblers (and in particular the PDP-11 MACRO-11 assembler), the DCT11-EM recognizes several directives which are similar to those used in the MACRO-11 language. These directives are listed in Table 4-2 with a brief explanation of their operation. For detailed information about how to use these directives in your programs, refer to the *PDP-11 MACRO-11 Language Reference Manual* (AA-5075A-TC).

Table 4-2 DCT11-EM Directives

Directive Form	Operation
ASCII /string/	Generates a block of data containing the ASCII equivalent of the character string (enclosed in delimiting characters), one character per byte. The delimiting characters can be slashes or any other visible character. The ASCII code for each character is stored in successive bytes of memory starting at the location indicated by the current address pointer. The address pointer is advanced by the length (in characters) of the enclosed string.
BLKB expr	Reserves a block of storage space whose length in bytes is determined by the specified expression. The current address pointer is advanced by the value of the expression.
BLKW expr	Reserves a block of storage space whose length in words is determined by the specified expression. The current address pointer is advanced by two times the value of the expression.
BYTE expr1, expr2, . . .	Generates successive bytes of data; each byte contains the value of the corresponding specified expression. The lower eight bits of each expression are stored in successive bytes of memory, starting at the location indicated by the current address pointer. The address pointer is advanced by the number of expressions given. The last expression must not be followed by a comma.
EVEN	Ensures that the current address pointer contains an even address by adding 1 if it is odd.
WORD expr1, expr2, . . .	Generates successive words of data; each word contains the value of the corresponding specified expression. The value of each expression is stored in successive words of memory starting at the location indicated by the current address pointer. The address pointer is advanced by two times the number of expressions given. The last expression must not be followed by a comma.

4.16 MISCELLANEOUS CONSOLE FUNCTIONS

The DCT11-EM has a HELP facility which displays all the commands available to you. To invoke this facility, type in HELP.

You can display the current operating modes of the DCT11-EM by typing in the SHOWMO command. When you first power up, typing in the SHOWMO command displays the default operating modes:

```
PASS2 NOHOST INSTRU SYMBOL VTOFF
```

The operating modes that can be displayed are: PASS1, PASS2, HOST, NOHOST, INSTRU, WORD, BYTE, SYMBOL, ABSOLU, VTON, and VTOFF. See the command summary in Paragraph 4.18 for brief descriptions of these modes. The command summary also contains references to paragraphs that explain the modes in detail.

The DCT11-EM has a REPEAT function which allows you to repeatedly input a line of commands or instructions. The form of this command is:

```
REPEAT expression,input line
```

where the input line is executed the number of times specified by the value of the expression. One application of the REPEAT command is to fill memory. For example, the REPEAT 1000,NOP command will fill 1000 (octal) locations with NOP instructions. The REPEAT command also lets you single step without having to press the RETURN key repeatedly (see Paragraph 4.14.2). The REPEAT 7, STEP command, for example, single steps through your program seven times.

Repeat Command – Example

A third application of the REPEAT command is to move blocks of memory. You must specify the beginning and ending addresses of the range to be moved (the source range) and the beginning address of the destination range. The general format for a command sequence which moves a block of memory is:

```
TEM> .=destination
TEM> REPEAT <end-start>/2+1,[.+start-destination]
```

where start and end are the beginning and ending addresses (respectively) of the source range, and destination is the beginning address of the destination range. Both ranges must begin at even addresses, and only words of memory can be transferred.

For example, if you want to move the contents of locations 1000 through 1010 to 2030 through 2040, type in the following.

```
TEM> .=2030
TEM> REPEAT <1010-1000>/2+1,[.+1000-2030]
```

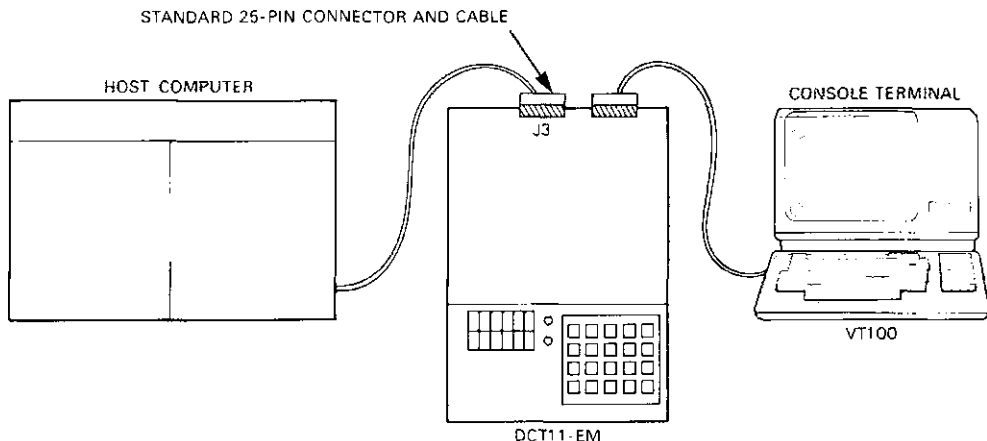
4.17 LOADING THE DCT11-EM FROM A HOST COMPUTER

This paragraph describes how to load information (usually a program) from a host computer into your DCT11-EM. Two broad areas are covered. First, you are familiarized with the console monitor commands that initiate and control the host loading process. Second, you are shown how to prepare a DCT11-EM loadable program on your host.

4.17.1 Configuring the DCT11-EM for Host Loading

Since host loading is accomplished through the use of console monitor commands, you must have your DCT11-EM connected to a console terminal.

Your host computer must have a serial communication line capable of sending data to the DCT11-EM in ASCII text format. Connect the serial line from the host to the DCT11-EM auxiliary serial port connector (J3) as shown in Figure 4-2.



MR 10254

Figure 4-2 Connecting a Host Computer to the DCT11-EM

No other wiring or jumpering is required. Set the baud rate of your console terminal equal to the baud rate of your host line. Use the DCT11-EM keypad to set the same baud rate for the console serial port by activating function 3 as described in Paragraph 3.12.4. Also make sure you are set up to transmit and receive characters in the following format: asynchronous, ASCII, 8 bits (eighth bit is always space), no parity, one stop bit.

You need not set the baud rate of the auxiliary serial port. When you communicate with a host, the DCT11-EM automatically sets the baud rate of the auxiliary serial port equal to the baud rate of the console serial port.

Activate function 2 as described in Paragraph 3.12.3 to start the console monitor. When the console displays a prompt, type in:

```
TEM> HOST
```

The HOST command places you in HOST mode and is a prerequisite for performing any operation with a host computer.

HOST mode only allows you to receive messages from the host. If you want to communicate directly with the host, use the TALK command in addition to HOST. If you want to load information from the host, use the LOAD command in addition to HOST. The TALK and LOAD commands are explained in Paragraphs 4.17.3 and 4.17.4.

The NOHOST command cancels HOST mode.

4.17.2 The HOST Command

Typing in the HOST command has two effects:

1. It defines the auxiliary serial port as a host communication line.
2. It creates an input buffer to hold information received from the host.

A HOST> prompt precedes any messages you receive from your host while in HOST mode.

Host Mode – Example

You are in HOST mode and you receive a message from the host before you finish typing an input line. If you simply continue typing the line, the console monitor restores the interrupted line and appends your new material to it. Messages from the host are ignored as console input.

	Comments
TEM> HOST	
TEM> MOV R0,R1	
TEM> MOV R	:You were typing MOV R0,R2
HOST>	
This is a message from the host ...	
TEM> MOV R0,R2	;You type 0,R2 and the previous line is restored.
TEM>	

4.17.3 The TALK Command

Typing in the TALK command allows you to communicate directly with the host just as if the DCT11-EM were not there. It has the same effect as connecting the serial line from the host directly to your console terminal.

Typically, you might use the TALK mode to prepare a DCT11-EM loadable file with your host's editing facilities (see Paragraph 4.17.5).

The user LED is illuminated in TALK mode.

There are two ways to cancel TALK mode. One way is to press the BREAK key on the console. The DCT11-EM responds:

```
CONSOLE BREAK
TEM>
```

and you not only cancel TALK mode but HOST mode as well. The other way is to type in CTRL/A. The DCT11-EM responds:

```
E (EXIT), R (RESUME), B (SEND BREAK), ^ A (SEND ^A) ?
```

If you type in the letter E, TALK mode is canceled but HOST mode is not.

Note that typing in CTRL/A (^A) in TALK mode gives you three other options. You can send a break or CTRL/A character to the host by typing in B or CTRL/A, respectively. Typing in an R or any characters other than E, B, or CTRL/A, causes TALK mode to be resumed.

4.17.4 The LOAD Command

Use the LOAD command only after you have prepared a DCT11-EM loadable program on your host as explained in Paragraph 4.17.5.

Typing in the LOAD command allows you to communicate directly with the host just as the TALK command does. In LOAD mode, however, the DCT11-EM constantly monitors the characters received from the host and begins loading if it sees the .START directive.

To initiate host loading while in LOAD mode, therefore, simply enter a command (which depends on the particular host you are using) that types out your program on the console terminal. For example, if you were using a VAX/VMS host, you would use TYPE PROG.TXT to type out the file PROG.TXT on your terminal. Since your DCT11-EM loadable program always has the .START directive as its first line, typing the program causes the DCT11-EM to start loading.

NOTE

During the loading process, the DCT11-EM accepts input from both serial ports simultaneously. As a rule, do not press keys on the console terminal during loading since this interferes with your host's input. The only exception to this rule is the use of the CTRL/O command (see below).

Once loading begins, it continues until the .END directive is received or until an error in the incoming program is detected. In both cases, when loading stops, you are returned to HOST mode. If loading stops as a result of an error in your program, the portion of the program that follows the error will be displayed on your console terminal as a "message" from the host.

If loading is terminated by an error in your program, you can suppress further display of the program by typing in the CTRL/O command. CTRL/O causes the DCT11-EM to ignore messages from the host until you press the RETURN key. If you want to terminate output from the host, enter CTRL/O followed by the TALK command followed by the appropriate interrupt control sequence your host recognizes (such as CTRL/Y or CTRL/C).

Note that the user LED is illuminated in LOAD mode. Also note that pressing BREAK or typing in CTRL/A has the same effect in LOAD mode as in TALK mode.

4.17.5 Preparing DCT11-EM Loadable Programs on a Host

In general, you will find it convenient to use your host's editing facilities to create programs. This is particularly true if the programs are long or require extensive alteration. In order to prepare a program the DCT11-EM will accept, keep the following in mind.

Make sure the .START directive precedes your program. The DCT11-EM will not be able to load your program unless it first sees the .START directive.

Next, realize that when your program is loaded into the DCT11-EM, the DCT11-EM will behave as if each line of your program were being entered by you directly from the console terminal. This means that the rules and conventions of console monitor input apply also to each line of the program you create on your host. The one exception is that the null line command (i.e., pressing RETURN to examine successive memory locations) is disabled during host loading.

If your program contains undefined symbols or expressions (for example, a "forward reference"), prepare your program as two identical segments. Precede the first segment with the PASS1 command and the second segment with the PASS2 command. The idea behind the PASS1 and PASS2 commands is as follows.

Normally, the DCT11-EM does not allow you to input undefined symbols. This mode of operation is called the PASS2 mode (PASS2 is one of the power-up default modes of the DCT11-EM). The use of undefined symbols in your programs, however, is commonly desirable. For example, consider the following program segment.

```
CMP #177700,R1
BNE .+14
INC R2
CMP R2,#200
BLO .+4
CLR R2
MOV R2,R3
```

In this example, you are branching to a defined location (i.e., the location that contains the MOV R2,R3 instruction), but are doing so in a cumbersome way. You need to perform a calculation for each branching instruction. An easier way to write this program segment would be:

```
      CMP #177700,R1
      BNE THERE
      INC R2
      CMP R2,#200
      BLO THERE
      CLR R2
THERE: MOV R2,R3
```

But if the DCT11-EM is in a mode in which it rejects undefined symbols, an error results when you try to input BNE THERE.

The DCT11-EM solves this problem by having a mode of operation in which undefined symbols are tolerated called the PASS1 mode.

In the PASS1 mode, the DCT11-EM substitutes the value zero for each undefined symbol encountered in expressions except when the undefined symbol is the operand of a branch instruction. In that case, the value of the current address pointer is substituted for the value of the branch operand. If an undefined symbol is later defined through the use of an equal sign (=) or colon (:), the symbol is entered into the symbol table with its proper value.

NOTE

Undefined expressions involving the address pointer are never tolerated, not even in PASS1. For example if A, B, and C are undefined, the following are unacceptable:

```
.=A
.BLKB B
.BLKW C
```

Make sure the .END directive terminates your program.

A DCT11-EM Loadable Program – Example 1

Use your host's editing facilities to create the summation program presented in Paragraph 3.8. Comments from the original program are shown in lowercase, and comments added for the benefit of the example are shown in uppercase.

```
.START           ;THIS LINE MUST APPEAR FIRST.
. = 2000         ;ENTER THE PROGRAM STARTING AT LOCATION 2000.
CLR R1          ;Set R1 = 0.
CMP R0,R1       ;R0 contains n.
BLE 2034        ;Illegal input (n ≤ 0)?
CLR @#2044      ;No, start operation by clearing 2044.
MOV R0,@#2044   ;Place first value of n in 2044.
DEC R0          ;n = n-1
ADD R0,@#2044   ;Add next value (n-1) to 2044.
BVS 2036        ;Overflow?
CMP R0,R1       ;No.
BGT 2016        ;Continue operation if n > 0.
HALT           ;Operation complete if n = 0.
CLR R0          ;Yes, illegal input. Clear R0, 2044.
CLR @#2044      ;Yes, overflow. Clear 2044 only.
HALT           ;Illegal input or overflow occurred.
.END           ;STOP HOST LOADING.
```


A DCT11-EM Loadable Program – Example 2

In this example, the summation program above is changed slightly such that forward references to the symbols ILLIN and OVFL0 are made. Since these symbols are undefined when first referenced, you must create the program in two segments to make it acceptable to the DCT11-EM.

```
.START                                ;THIS LINE MUST APPEAR FIRST.
CLEAR                                ;DELETE ALL PREVIOUSLY DEFINED
                                      ;SYMBOLS.
PASS1                                ;PERFORM FIRST PASS INPUT OF PROGRAM.
                                      ;ALLOW UNDEFINED SYMBOLS AND SET
                                      ;THEIR VALUES TO ZERO.
. = 2000                              ;INPUT THE PROGRAM STARTING AT
                                      ;LOCATION 2000.
BEGIN:    CLR R1                      ;Set R1 = 0.
          CMP R0,R1                   ;R0 contains n.
          BLE ILLIN                   ;Illegal input (n ≤ 0)?
          CLR @#2044                  ;No, start operation by clearing 2044.
          MOV R0,@#2044               ;Place first value of n in 2044.
LOOP:    DEC R0                       ;n = n-1
          ADD R0,@#2044               ;Add next value (n-1) to 2044.
          BVS OVFL0                  ;Overflow?
          CMP R0,R1                   ;No.
          BGT LOOP                   ;Continue operation if n > 0.
          HALT                       ;Operation complete if n = 0.
ILLIN:   CLR R0                       ;Yes, illegal input. Clear R0, 2044.
OVFL0:   CLR @#2044                  ;Yes, overflow. Clear 2044 only.
          HALT                       ;Illegal input or overflow occurred.
PASS2                                ;PERFORM SECOND PASS ENTRY OF
                                      ;PROGRAM.
                                      ;SYMBOLS ARE NO LONGER UNDEFINED AND
                                      ;ARE ASSIGNED CORRECT VALUES.
                                      ;UNRESOLVED ADDRESSES ARE RESOLVED.
. = 2000                              ;INPUT THE PROGRAM AGAIN STARTING AT
                                      ;LOCATION 2000.
BEGIN:    CLR R1                      ;Set R1 = 0.
          CMP R0,R1                   ;R0 contains n.
          BLE ILLIN                   ;Illegal input (n ≤ 0)?
          CLR @#2044                  ;No, start operation by clearing 2044.
          MOV R0,@#2044               ;Place first value of n in 2044.
LOOP:    DEC R0                       ;n = n-1
          ADD R0,@#2044               ;Add next value (n-1) to 2044.
          BVS OVFL0                  ;Overflow?
          CMP R0,R1                   ;No.
          BGT LOOP                   ;Continue operation if n > 0.
          HALT                       ;Operation complete if n = 0.
ILLIN:   CLR R0                       ;Yes, illegal input. Clear R0, 2044.
OVFL0:   CLR @#2044                  ;Yes, overflow. Clear 2044 only.
          HALT                       ;Illegal input or overflow occurred.
.END                                    ;STOP HOST LOADING.
```

4.17.6 An Illustration of the Host Loading Process

This paragraph steps you through a session with the DCT11-EM during which you load the summation program you created on your host (see example 1, Paragraph 4.17.5).

Assuming you are in NOHOST mode, the first step is to get into LOAD mode by typing in:

```
TEM> HOST LOAD
```

Press RETURN until your host responds with its usual prompting character(s).

Then direct your host to type or otherwise display on the console terminal the program you wish to load. This command might be in the form:

```
TYPE SUMM
```

where TYPE is the the command your host recognizes to display the program and SUMM is the name you gave the program. When the host starts typing the program, the following appears on your console terminal. (This program does not contain any undefined symbols and requires only one pass.)

```
      .START
TEM>                                ;THIS LINE MUST APPEAR FIRST.
TEM> CLEAR                          ;DELETE ALL PREVIOUSLY DEFINED
TEM>                                ;SYMBOLS.
TEM> . = 2000                        ;ENTER THE PROGRAM STARTING AT
TEM>                                ;LOCATION 2000.
TEM> CLR R1                          ;Set R1 = 0.
TEM> CMP R0,R1                      ;R0 contains n.
TEM> BLE 2034                       ;Illegal input (n ≤ 0)?
TEM> CLR @#2044                     ;No, start operation by clearing 2044.
TEM> MOV R0,@#2044                 ;Place first value of n in 2044.
TEM> DEC R0                         ;n = n-1
TEM> ADD R0,@#2044                 ;Add next value (n-1) to 2044.
TEM> BVS 2036                       ;Overflow?
TEM> CMP R0,R1                     ;No.
TEM> BGT 2016                       ;Continue operation if n > 0.
TEM> HALT                          ;Operation complete if n = 0.
TEM> CLR R0                         ;Yes, illegal input. Clear R0, 2044.
TEM> CLR @#2044                    ;Yes, overflow. Clear 2044 only.
TEM> HALT                          ;Illegal input or overflow occurred.
TEM> .END                          ;STOP HOST LOADING.
HOST>
<your host's prompt appears on the last line>
```

The DCT11-EM successfully loaded your program and you are now back in HOST mode. The console monitor can now respond to your input. If you type in 2000, you can examine the program you just loaded by pressing RETURN repeatedly.

If your program is successfully being loaded into the DCT11-EM, each line is preceded by the TEM> prompt. If a program line causes a console error message, loading stops but typing continues (without the TEM> prompt).

4.18 CONSOLE COMMAND SUMMARY

General Form	Operation
expr	Deposit the expression in RAM unless the expression is part of another command. Advance the address pointer accordingly.
instr	Assemble and deposit the instruction at the location indicated by the address pointer. Advance the address pointer accordingly.
expr1__expr2	Display the contents of the locations from expr1 through expr2. If expr1 is greater than expr2, only expr1 contents displayed.
expr?	Display numerically the value of the expression.
expr\ null line	Set the address pointer to the value of the expression. Display the contents of the location specified by the expression.
symbol=expr	Display, relative to the address pointer, the next eight instructions if in INSTRU mode, the next eight words if in WORD mode, or the next eight bytes if in BYTE mode. If single stepping, single step again.
symbol:	Define the symbol to be equal to the expression.
^	Define the symbol to be equal to the current value of the address pointer.
	Decrement the address pointer by one byte if you are in BYTE mode or by one word if you are in WORD or INSTRU mode.

Form	Paragraph Reference	Operation
ABSOLU	4.8.2	Sets the ABSOLU mode. Causes user-defined symbols in programs to be displayed as numeric values.
.ASCII /string/	4.15	Generates a block of data containing the ASCII equivalent of the character string (enclosed in delimiting characters), one character per byte.
.BLKB expr	4.15	Reserves a block of storage space whose length in bytes is determined by the specified expression.
.BLKW expr	4.15	Reserves a block of storage space whose length in words is determined by the specified expression.

.BYTE expr1, expr2, . . .	4.15	Generates successive bytes of data; each byte contains the value of the corresponding specified expression.
BYTE	4.8.2	Sets the BYTE mode for memory examination and alteration operations. Causes memory to be displayed as three digit bytes.
CANCEL	4.14.1 4.14.3	Clears all four breakpoints and the watchpoint.
CLEAR	4.13.2	Clears the user symbol table.
DELETE symbol	4.13.2	Deletes user-defined symbol from the user symbol table.
.END	4.17.4	Used at the end of programs that are to be loaded into the DCT11-EM from a host. Causes the DCT11-EM to terminate host loading if in LOAD mode.
.EVEN	4.15	Ensures that the current address pointer contains an even address by adding 1 if it is odd.
EXIT	4.2	Transfers console control to the keypad.
GO	4.14	Executes the user program at the current PC value.
HELP	4.16	Displays a table of commands.
HOST	4.17.1	Sets the HOST mode. Allows the DCT11-EM to receive messages from a host.
INSTRU	4.8.2	Sets the INSTRU mode. Data is displayed as DCT11-AA assembly language instructions. Is set by default when the DCT11-EM is powered up.
LOAD	4.17.1	Sets the LOAD mode. Establishes host communication. Allows the DCT11-EM to begin loading if it encounters the .START command.
NOHOST	4.17.1	Clears the HOST mode. Is set by default when the DCT11-EM is powered up.
PASS1	4.17.5	Sets the PASS1 mode. Allows the use of undefined symbols.
PASS2	4.17.5	Sets the PASS2 mode. Requires all symbols to be defined. Is set by default when the DCT11-EM is powered up.

REPEAT expr,cmd	4.16	Repeatedly performs the specified command(s) by the number of times specified by the expression.
SHOWMO	4.16	Displays the current operating modes of the DCT11-EM.
SHOWRE	4.6	Displays the contents of user registers and the watchpoint if the watchpoint is set.
SHOWSY	4.13.2	Displays the contents of the user symbol table.
START	4.17.5	Used at the beginning of programs that are to be loaded into the DCT11-EM from a host. Causes the DCT11-EM to start host loading if in LOAD mode.
STEP	4.14.2	Executes one instruction of a user program at the current value of the PC. Displays the contents of user registers and the watchpoint if the watchpoint is set. Displays the next instruction to be executed.
SYMBOL	4.8.2	Sets the SYMBOL mode. Causes user-defined symbols in programs to be displayed as symbols rather than as numeric values. Is set by default when the DCT11-EM is powered up.
TALK	4.17.1	Sets the TALK mode. Establishes host communication. Allows you to communicate directly with a host as if the DCT11-EM were not there. Typically used for program development on a host.
VTOFF	4.3	Sets the VTOFF mode. Causes deleted characters to be echoed (preceded by a slash) on the console terminal when you press the DELETE key. Is set by default when the DCT11-EM is powered up.
VTON	4.3	Sets the VTON mode. Characters you delete by pressing the DELETE key are not echoed in this mode. Typically used if you have a video terminal for a console.
.WORD expr1, expr2, . . .	4.15	Generates successive words of data; each word contains the value of the corresponding specified expression.

4.19 CONTROL COMMAND SUMMARY

Command	Meaning	Typical Use
CTRL/U CTRL/X	Delete line.	You made several typing errors in your input line. You want to cancel the line and try again.
CTRL/R	Display line.	You used the DELETE key several times while typing your input line. You want to see a clean version of the line you are editing.
CTRL/S	Suspend execution.	You want to suspend the operation of the console monitor or the execution of a currently running program.
CTRL/Q	Resume execution.	You have previously used the CTRL/S command and want to resume operation.
CTRL/C	Abort operation.	You want to abort the current CTRL/Y operation of the DCT11-EM and input a new line.
CTRL/O	Ignore host message.	You are in HOST mode (see Paragraph 4.17.1) and want to ignore information currently being sent by your host computer.
CTRL/A	Select option.	You are in TALK or LOAD modes (see Paragraphs 4.17.3 and 4.17.4) and want to select one of these four options: exit, resume, send break, send CTRL/A.

CHAPTER 5 SOFTWARE

5.1 INTRODUCTION

This chapter provides some information about the DCT11-EM monitor and introduces you to monitor subroutines you can use in your programs. The chapter also describes how address space is allocated in the DCT11-EM and shows you which areas are available and unavailable for use.

This material is summary in nature. More detailed information can be found in the DCT11-EM monitor listing (see Appendix A) and in the DCT11-EM schematics (see Appendix B).

5.2 THE MONITOR

The monitor is a PROM resident program that controls the operation of the DCT11-EM. Its main purpose is to provide an environment in which you can develop and run programs. The portion of the monitor that allows you to operate the DCT11-EM with the keypad is called the keypad monitor and the portion that lets you use a console terminal is called the console monitor. When you learned how to operate the DCT11-EM with the keypad and console in Chapters 3 and 4, you were really learning how to get the monitor to do things for you.

When you execute a program, the monitor relinquishes control of the DCT11-EM to your program. Control is returned to the monitor (i.e., the monitor is restarted) by any of the following ways:

1. Pressing the HALT switch
2. Executing a HALT instruction
3. Encountering a breakpoint or executing a BPT instruction
4. Attempting to execute an illegal instruction
5. Performing a single step
6. Sending a BREAK from a console terminal
7. Executing a JMP @#140010 instruction
8. Sending a CTRL/C or CTRL/Y command from a console terminal

The first two ways (1 and 2) always return control to the keypad monitor. The remaining ways (3 through 8) return control to the keypad monitor if you are using the keypad or the console monitor if you are using a console terminal. When control is returned to the keypad monitor, the PC is displayed on the LEDs and you are in register mode. (If a flashing error message appears on the LEDs, press the HALT switch or any key to display the PC.) When control is returned to the console monitor, a TEM> prompt is displayed on your console terminal. Refer to Appendix A for details about how the HALT and BPT instructions interact with the monitor. Also refer to Appendix A for information on the effects of setting the T-bit.

When you return control to the monitor, the DCT11-EM registers contain the values they had at the time your program stopped with the following exceptions.

1. If you execute a JMP @#140010 instruction, the PC is restored to the value it had when your program started. This allows easy restarting of the program.
2. If you type CTRL/C or CTRL/Y from a console terminal, all the DCT11-EM registers are restored to the values they had when your program started.
3. If you encounter a breakpoint, the stack pointer is set to the value it had just before the breakpoint occurred.

5.2.1 The Stack Pointer

The monitor initializes your stack pointer to 7400 when you power up. The monitor also checks to make sure your stack pointer is in the range 4 through 7400, inclusive (for the keypad monitor), or 4 through 6500, inclusive (for the console monitor), before it allows your program to execute. It is up to you, however, to make sure the stack pointer has a reasonable value before you attempt to run a program.

For example, it is a good idea to make sure your stack pointer and the information you push onto the stack do not overlap your program. To decrease the chance of this happening, set the stack pointer well above the area in which your program resides.

5.3 MONITOR SUBROUTINES

Several portions of the DCT11-EM monitor are accessible to your programs as subroutines. If you want to use one of these subroutines in your program, simply include a JSR instruction in the form:

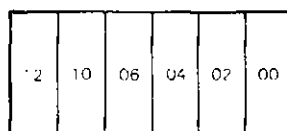
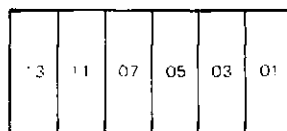
```
JSR PC,@#subroutine
```

where subroutine refers to the address of the desired subroutine.

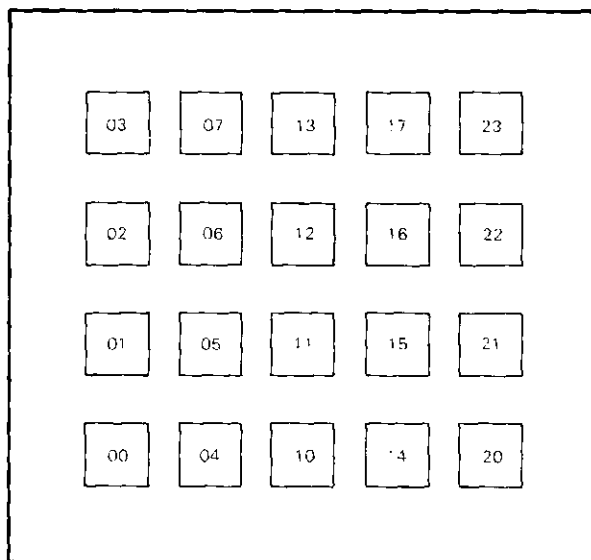
Subroutine mnemonics are defined within the monitor only, so make sure your program defines these mnemonics before it uses them as symbols.

5.3.1 Keypad/LED Subroutines

Address	Mnemonic	Description
140014	DGDISP	Display Hex Digit Takes the value contained in the lower four bits of R0 and displays it as a hexadecimal number on the LED digit indexed by R1. Values of R1 correspond to LED digits as follows. Uses R0 and R1.



Address	Mnemonic	Description
140020	UPDISP	Display Octal Number on Upper LEDs Takes the value in R2 and displays it as a six-digit octal number on the upper row of LEDs. Uses R0 and R1.
140024	LODISP	Display Octal Number on Lower LEDs Takes the value in R2 and displays it as a six-digit octal number on the lower row of LEDs. Uses R0 and R1.
140030	DEC DSP	Display Decimal Number Takes the value in R1 and displays it as a five-digit decimal number on the row of LEDs indexed by R2. R2 = 12 for the lower row, R2 = 13 for the upper row. Uses R0, R1, R2, and R3.
140034	UPBLNK	Clear Upper Row Blanks out the upper row of LEDs. Uses R0 and R1.
140040	LOBLNK	Clear Lower Row Blanks out the lower row of LEDs. Uses R0 and R1.
140044	GETKEY	Get Key Number Turns on LEDs and waits for a keypress. When a keypress occurs, takes the number of the key and stores it in R1. Keys are numbered as follows.



140050	BCDKEY	Convert Key Number to BCD Takes the key number in R1 and converts it to a BCD value for the keys 0 through 9. Returns 12 for the CLR key and 13 for the EXA key. Returns -1 (i.e., 177777) for any other key. Stores the result in R1.
--------	--------	---

Address	Mnemonic	Description
140054	GETNUM	Input Number From Keypad Displays the value in R2 as a six-digit octal number on the lower row of LEDs. Enables the user to modify the value of R2 using keys 0 through 7 and CLR. Stores modified value in R2 when the EXA key is pressed. Uses R0, R1, and R2.

5.3.1.1 Using the LEDs and Keypad – This paragraph contains some information you may find useful when writing programs that use the LEDs and keypad. This is intended as introductory material only. For further detail, refer to the monitor listing in Appendix A and the DCT11-EM schematics in Appendix B.

You can create your own characters to display on the LED digits. First, form bytes that represent the segment patterns of the characters you wish to display as shown in Figure 5-1.

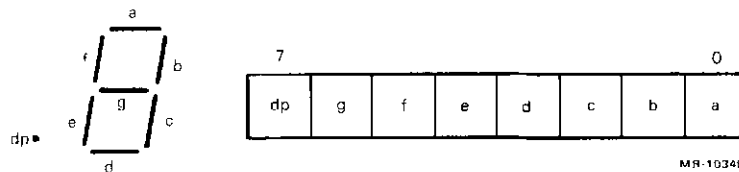


Figure 5-1 LED Segment Assignments

For example, 01110100 (or 164 octal) corresponds to the letter *h* and 01001111 (or 117 octal) corresponds to the numeral 3.

Then, move the bytes containing the segment patterns to locations 7712 through 7725. These locations (collectively called SEGBUF) are used by the LED interrupt service routine in the monitor to display segment patterns on the LEDs. Each location (byte) corresponds to an LED digit as shown in Figure 5-2.

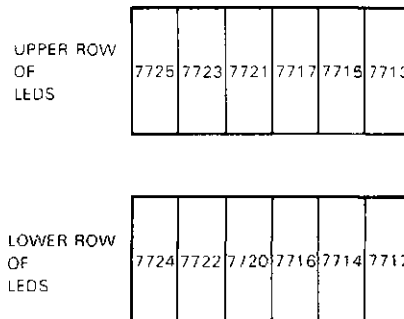


Figure 5-2 SEGBUF Assignments

For example, if you moved 164 (octal) to location 7721, 117 (octal) to location 7717, and 0 to all the other SEGBUF locations, you would display *h3* centered in the upper row of LEDs.

Finally, turn on the LEDs by writing 000007 to port C of the 8255A parallel port chip (address 177444). Make sure your program keeps the LEDs on long enough for you to see them. One way of leaving the LEDs on is to incorporate one or more delay loops in your program of the form:

```
CLR R5
SOB R5,
```

The LED interrupt (INTA) is disabled (i.e., the LEDs are turned off) by writing 000010 to port C of the 8255A chip. This also disables the user LED.

To turn on the user LED (which, incidentally, also disables the LED digits), write 000011 to port C of the 8255A chip.

Examples of displaying values on the LEDs are provided in Paragraph 5.3.1.2. You can also create your own interrupt service routines to respond to keypresses. When a key is pressed, program control is transferred to the routine pointed to by 7676. If 7676 is zero, keypresses are ignored. The number of the key pressed is available to your routine through R1. See the description of the GETKEY subroutine in Paragraph 5.3.1 for key numbers.

In order for a keypress to be sensed, the LEDs must be enabled. This can be done by writing 000007 to location 177444 as explained above or by executing your program from the keypad with the “Go With LEDs” special function (see Paragraph 3.12.2).

At the end of your keypress service routine, include the sequence:

```
MOV (SP)+,R2
MOV (SP)+,R1
MOV (SP)+,R0
RTI
```

to restore the state of your interrupted program. As an example, here is a program which loops on itself and increments R3 each time a key is pressed. To check the operation of the program, start it at location 1000 and press keys. Press HALT and examine R3.

Address	Data	Symbolic Representation	Comments
1000	012737	MOV #7,@#177444	;enable keypad
1002	000007		
1004	177444		
1006	012737	MOV #1500,@#7676	;attach interrupt routine
1010	001500		
1012	007676		
1014	000774	BR 1006	;loop infinitely
1500	005203	INC R3	;interrupt service routine
1502	012602	MOV (SP)+,R2	
1504	012601	MOV (SP)+,R1	
1506	012600	MOV (SP)+,R0	
1510	000002	RTI	

5.3.1.2 Using Keypad/LED Subroutines – Example Programs – The following examples show you how to use keypad/LED subroutines.

Example 1 – Countdown Program This program decrements a number at a rate inversely proportional to the number’s size. The program begins by displaying the word “EntEr” on the upper row of LEDs. You enter a number (which is displayed on the lower row of LEDs) and press the EXA key. The number is displayed as it is decremented. The program initializes itself at the end to allow you to enter another number.

There are undefined symbols in this program, so if you enter the symbolic version via the console terminal, make sure you incorporate the PASS1 and PASS2 commands as described in Paragraph 4.17.5.

Address	Data	Symbolic Representation	Comments
		SEGBUF = 7712 LODISP = 140024 GETNUM = 140054	;LED segment buffer ;octal number on lower LEDs ;input number from keypad
		. = 1000	;position program at location ;1000
1000	012701	TEST: MOV #SEGBUF+1,R1	;point to rightmost
1002	007713		;digit of upper row of LEDs
1004	012700	MOV #6,R0	;set digit count = six digits
1006	000006		
1010	116021	TEST1: MOVB ENTER-1(R0),(R1)+	;transfer character to
1012	001063		;digit in upper row of LEDs
1014	005201	INC R1	;advance pointer to next
			;digit in upper row of LEDs
1016	077004	SOB R0,TEST1	;loop to transfer all six
			;characters
1020	004737	TEST2: JSR PC,@#GETNUM	;input an octal number from
1022	140054		;the keypad
1024	005702	TEST4: TST R2	;reject zero as an input
1026	001774	BEQ TEST2	;if zero, get a new number
1030	012701	MOV #3,R1	;call delay routine three
1032	000003		;times
1034	004767	TEST3: JSR PC,DELAY	;wait a while
1036	000012		
1040	077103	SOB R1,TEST3	
1042	005302	DEC R2	;decrement entered number
1044	004737	JSR PC,@#LODISP	;display new value
1046	140024		
1050	000765	BR TEST4	;see if we have zero yet

;DELAY loops a number of times inversely proportional to the size of R2. Thus there is a smaller delay when R2 is large.

1052	012700	DELAY: MOV #-1,R0	;set dividend
1054	177777		
1056	160200	DELAY 1: SUB R2,R0	;divide by successive
1060	103376	BCC DELAY1	;subtraction; keep
			;subtracting until
			;underflow occurs
1062	000207	RTS PC	

; ENTER is a list of bytes corresponding to the segment patterns ; "EntEr".

1064	052171	ENTER: .BYTE 171,124,170,171,120,0	
1066	074570		
1070	000120		

Example 2 – Clock Program

This program displays a minutes and seconds clock on the lower row of LEDs. The values for minutes and seconds are obtained from a timer maintained by the monitor. You can use the INT switch to set minutes. Note that the upper row of LEDs and the left-most digit of the lower row display the current contents of the LED segment buffer for those digits.

All symbols referenced in this program are defined. The symbolic version of the program will run if entered as follows. As an exercise, try to modify the program so that the upper row of LEDs and the left-most digit of the lower row of LEDs are blank when the program is running.

Address	Data	Symbolic Representation	Comments
		PFVEC = 24	;INT switch vector
		TIME = 7730	;upper six bits count seconds
		SEGBUF = 7712	;LED segment buffer
		MINUTS = 776	;location to hold minutes count
		DECDSP = 140030	;display decimal number
		P\$PORC = 177444	;LED/keypad I/O port on 8255A
		SETIME = 1200	;subroutine to set minutes
		= 1000	;position program at loc. 1000
1000	012706	CLOCK: MOV #6500,SP	;initialize stack pointer
1002	006500		
1004	106427	MTPS #0	;set low priority
1006	000000		
1010	012737	MOV #SETIME,@#PFVEC	;connect minute set
1012	001200		;routine to INT switch
1014	000024		
1016	005037	CLR @#TIME	;clear seconds and second
1020	007730		;fractions
1022	005067	CLR MINUTS	;clear minutes
1024	177750		
1026	112737	MOVB #7,@#P\$PORC	;turn on LEDS
1030	000007		
1032	177444		
1034	032737	LOOP: BIT #1000,@#TIME	;wait for seconds to
1036	001000		;change
1040	007730		
1042	001774	BEQ LOOP	
1044	032737	LOOP1: BIT #1000,@#TIME	
1046	001000		
1050	007730		
1052	001374	BNE LOOP1	
1054	013701	SHOWIT: MOV @#TIME,R1	;show time
1056	007730		
1060	000301	SWAB R1	;get seconds
1062	006201	ASR R1	
1064	006201	ASR R1	
1066	042701	BIC #177700,R1	;mask out fractions of
1070	177700		;seconds
1072	001010	BNE .+22	;not a new minute (BNE CLK1)
1074	005267	INC MINUTS	;else count minutes

Address	Data	Symbolic Representation	Comments
1076	177676		
1100	026727	CMP MINUTS,#60.	;overflow?
1102	177672		
1104	000074		
1106	103402	BLO .+6	;no (BLO CLK1)
1110	005067	CLR MINUTS	
1112	177662		
1114	016702	CLK1: MOV MINUTS,R2	;get minutes
1116	177656		
1120	012700	MOV #3,R0	;shift minutes over three
1122	000003		;digits
1124	006302	LOOP2: ASL R2	;multiply R2 by 10 to shift
1126	010246	MOV R2,-(SP)	;one decimal digit to the left
1130	006302	ASL R2	
1132	006302	ASL R2	
1134	062602	ADD (SP)+,R2	
1136	077006	SOB R0,LOOP2	;loop three times to make × 1000
1140	060201	ADD R2,R1	;add minutes to seconds
1142	012702	MOV #10,R2	;index lower display
1144	000010		
1146	004767	JSR PC,@#DECDSP	;display time
1150	136656		
1152	112737	MOVB #100,@#SEGBUF+4	;display dash between
1154	000100		;minutes and seconds
1156	007716		
1160	000725	BR LOOP	
		. = 1200	
1200	005037	SETIME: CLR @#TIME	;reset seconds to increment
1202	007730		;minutes
1204	012706	MOV #6500,SP	;reinitialize stack
1206	006500		
1210	106427	MTPS #0	;reinitialize priority
1212	000000		
1214	000717	BR SHOWIT	;display new time

As an exercise, try to modify the program so that the upper row of LEDs and the left-most digit of the lower row of LEDs are blank when the program is running.

5.3.2 Console Subroutines

Address	Mnemonic	Description
140060	GETLIN	Get Input Line Types an angle bracket and space (>) as prompting characters and awaits input from the console terminal. Echoes characters as they are typed in. Stores these characters (up to 96 per line) in a character buffer. Uses R0, R3.

Address	Mnemonic	Description
140070	GETCH	<p>Get Character</p> <p>Retrieves the character indexed by R3 from the character buffer. R3 must have a value from 0 to 137 octal (95 decimal). Stores the character in the lower byte of R0 and increments R3. Lowercase characters are converted to uppercase. Sets the N condition code bit if the character is a separator (space, tab, feed). Sets the C condition code bit if the character has been converted from lowercase to uppercase. Uses R0, R3.</p>
140100	GETCHC	<p>Get Character, Preserve Case</p> <p>Retrieves a character with its case intact from the character buffer. Uses R3 as an index to the character. R3 must have a value from 0 to 137 octal (95 decimal). Stores the character in the lower byte of R0 and increments R3. Uses R0, R3.</p>
140104	GETNXT	<p>Get Character, Ignore Separators</p> <p>Retrieves the next character in the character buffer which is not a separator (space, tab, feed). Uses R3 as a starting point index. R3 must have a value from 0 to 137 octal (95 decimal). Stores the character in the lower byte of R0 and increments R3 in accordance with the number of separators encountered. Lowercase characters are converted to uppercase. Sets the C condition code bit if a character has been converted from lowercase to uppercase. Uses R0, R3.</p>
140120	CRLF	<p>Type Carriage Return, Line Feed</p> <p>Types a carriage return and line feed on the console.</p>
140124	CHROUT	<p>Type One or Two ASCII Characters</p> <p>Takes the value in the lower byte of R0 and types it as an ASCII character on the console. If the upper byte of R0 has a nonzero value, it is also typed as an ASCII character on the console. Uses R0.</p>
140134	CHRO2	<p>Send Byte to Auxiliary Port</p> <p>Takes the value in the lower byte of R0 and sends it to the auxiliary serial port. Uses R0.</p>
140140	ASCOUT	<p>Type One ASCII Character or String</p> <p>Takes the value in the lower byte of R0 and types it on the console either as an ASCII character or as an ASCII string of characters. ASCII strings that can be typed include <ESC>, <SP>, , and ^A through ^_ where the caret denotes a control character. Uses R0.</p>
140144	PRINTA	<p>Type ASCII String</p> <p>Takes a string of bytes starting at the location pointed to by R1 and types them on the console as ASCII characters until until a byte whose value is zero is encountered. Uses R0, R1.</p>

Address	Mnemonic	Description
140160	TYPOCT	Type Octal Number Types the value in R1 on the console as a six-digit octal number if in the INSTRU or WORD output formatting modes. Types the lower byte of R1 on the console as a three-digit octal number if in BYTE mode. Uses R0, R1.
140164	TYPDEC	Type Decimal Number Types the value in R1 on the console as a signed five-digit decimal number followed by a period. Uses R0, R1.

5.3.2.1 Console Character Manipulation – The console subroutines just described provide most of the functions you will require to manipulate characters via the console. If you require other functions, refer to the monitor listing in Appendix A.

Keep the following in mind when using the GETCH, GETCHC, and GETNXT subroutines.

When you use these subroutines, characters you input via the console terminal are stored in a 96-character buffer maintained by the monitor called BUFFER. All three subroutines use R3 as a pointer to determine which character within BUFFER is retrieved. Since BUFFER is 96 characters in length, R3 can have a value from 0 through 137 (octal), inclusive. For your convenience, the subroutines automatically increment R3 to point to the next character to be retrieved.

5.3.2.2 Using Console Subroutines – Sample Program

```

;
; NIM for the TEM
;
; In this version of NIM, there are 21 sticks. The game requires 2
; players (the user and the TEM). The players take turns picking up
; 1, 2, or 3 sticks. The object of the game is to leave 1 stick for
; the other player to pick up in the last turn. The algorithm used
; by the TEM ensures that it will win if it goes second, or if it goes
; first and the other player leaves it an opportunity to win.
;
; Operation of the game is explained by prompts. All responses by the
; user must be terminated with a carriage return. However, several
; responses may be typed together on a line, separated by commas, and
; will be taken as needed by upcoming prompts for input.
;
; There are undefined symbols in this program, so if you enter the
; symbolic version via the console terminal, make sure you incorporate
; the PASS1 and PASS2 commands as described in Paragraph 4.17.5.

```


;***** SYMBOL DEFINITIONS *****

CLEAR ;empty symbol table

MONITR = 140010 ;reentry point for monitor
GETLIN = 140060 ;get a line from the console
GETNXT = 140104 ;get next significant character from BUFFER
CRLF = 140120 ;send <CR>,<LF> to console
CHROUT = 140124 ;send a character to console
PRINTA = 140144 ;print an ASCII string
TYPDEC = 140164 ;type a decimal number on the console

CR = 15 ;<CR>
LF = 12 ;<LF>
SPACE = 40 ;space
STICKS = 21 ;initial number of sticks

;***** START OF PROGRAM *****

. = 1000

NIM: MOV #GREET,R1
 JSR PC,@#PRINTA ;print greeting message on console

NEWGAM:CLR R5 ;show no sticks taken yet
 ;(number of sticks left = STICKS - R5)

NEW1: CLR R3
 MOV #MEORU,R1
 JSR PC,@#PRINTA ;ask who plays first
 JSR PC,@#GETLIN ;wait for response from console
 JSR PC,@#GETNXT ;get first meaningful character
 CMPB R0,#'M ;m(e) means TEM goes first
 BEQ TEMGO
 CMPB R0,#'Y ;y(ou) means user goes first
 BEQ USERGO
 CMPB R0,#'Q ;q(uit) means return to monitor
 BNE NEW1 ;if user has not entered a valid response,
 JMP @#MONITR ;prompt for response again.

USERGO:	MOV #LEFT1,R1 JSR PC,@#PRINTA MOV #STICKS,R1 SUB R5,R1 MOV R1,R2 JSR PC,@#TYPDEC MOV #LEFT2,R1 JSR PC,@#PRINTA JSR PC,@#CRLF	;print header for number of sticks left ;calculate number of sticks left ;save a copy of result ;type decimal value on the console ;finish message ;start a new line
USER1:	MOV #400*SPACE+'/',R0 JSR PC,@#CHROUT SOB R2,USER1 JSR PC,@#GETNXT CMPB R0,#', BEQ NEXT	;put in R0 the character pair "/" ;type a "stick" on the screen ;loop to type the appropriate number ;of sticks ;get next character ;is it a comma? ;yes, get next move from line.
USER2:	MOV #HOWMNY,R1 JSR PC,@#PRINTA JSR PC,@#GETLIN	;ask how many sticks the user wants ;get a line
NEXT:	JSR PC,@#GETNXT CMPB R0,#'1 BLO USER2 CMPB R0,#'3 BHI USER2 SUB #'0,R0 ADD R0,R5 CMP R5,#STICKS-1 BLO TEMGO BHI USER4 MOV #ILOSE,R1 JSR PC,@#PRINTA JMP NEWGAM	;get first/next meaningful character ;on line ;illegal response, throw away line and ;try again ;illegal response ;convert digit in R0 into a number ;add to sticks already taken away ;is only 1 stick left? ;no, it's TEM's turn ;user took too many sticks ;announce the loss ;ask about another game
USER4:	MOV #GOOFED,R1 JSR PC,@#PRINTA JMP NEWGAM	;tell the idiot that he goofed up ;and ask about another game
TEMGO:	MOV #MYMOVE,R1 JSR PC,@#PRINTA MOV R5,R1 NEG R1 BIC #17774,R1 BNE TEM1 MOV #2,R1	;type beginning of TEM's move on ;console ;get number of sticks taken so far ;implement algorithm to select move: ;if R5 MOD 4 = 0, take 2 sticks ; R5 MOD 4 = 1, take 3 sticks ; R5 MOD 4 = 2, take 2 sticks

```

TEM1:   ADD R1,R5           ; R5 MOD 4 = 3, take 1 stick
        JSR PC,@#TYPDEC   ;show how many sticks TEM is taking
        MOV #MYMOV1,R1
        JSR PC,@#PRINTA   ;finish message
        CMP R5,#STICKS-1  ;is there only 1 stick left?
        BLO USERGO       ;no, it's the user's turn
        MOV #IWIN,R1
        JSR PC,@#PRINTA   ;announce the victory
        JMP NEWGAM        ;and ask about a new game

GREET:  .BYTE CR,LF
        .ASCII /NIM for the TEM/
        .BYTE CR,LF,0

MEORU:  .BYTE CR,LF
        .ASCII /Who goes first? Me (M) or you (Y)? (Q to quit)/
        .BYTE 0

LEFT1:  .BYTE CR,LF
        .ASCII /There are /
        .BYTE 0

LEFT2:  .ASCII / sticks left./
        .BYTE 0

HOWMNY: .BYTE CR,LF
        .ASCII /How many sticks will you take? (1, 2, or 3)/
        .BYTE 0

MYMOVE: .BYTE CR,LF
        .ASCII /I take /
        .BYTE 0

MYMOV1: .ASCII / stick(s)./
        .BYTE 0

ILOSE:  .BYTE CR,LF,LF
        .ASCII /You win!!! I demand a rematch!/
        .BYTE 0

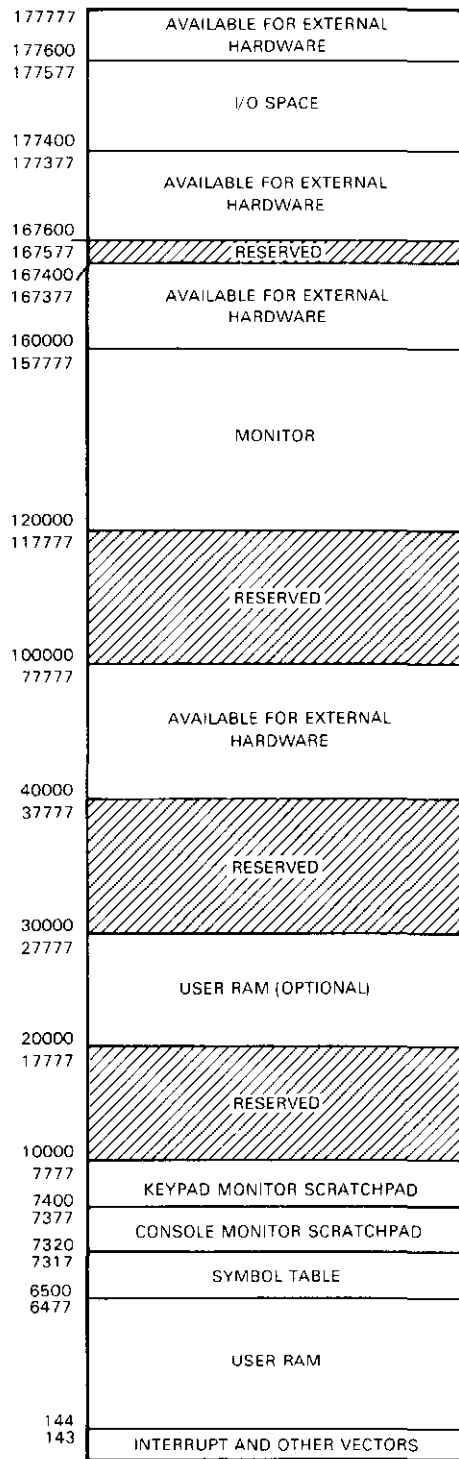
IWIN:   .BYTE CR,LF,LF
        .ASCII /I win!!! Want to try again?/
        .BYTE 0

GOOFED: .BYTE CR,LF,LF
        .ASCII /You goofed up! Can you get it right this time?/
        .BYTE 0

```

5.4 DCT11-EM ADDRESS SPACE

DCT11-EM address space is organized as shown in Figure 5-3. Functionally, the address space consists of: standard RAM space, expansion RAM space, monitor space, I/O space, space available for external hardware, and reserved space.



MR 10350

Figure 5-3 DCT11-EM Memory Map

5.4.1 Standard RAM Space

Addresses 000000 through 007777 are allocated to the RAM that comes standard with the DCT11-EM. This RAM space is utilized as follows.

Addresses	Use
000000-000143	Interrupt and Other Vectors. Contains vectors that point to service routines for various interrupt, trace, and trap conditions. See Chapter 6 for specific vector addresses.
000144-006477	User RAM Space. Space available for user programs.
006500-007317	Symbol Table (Console Monitor Only). Can contain up to 66 user defined symbols.
007320-007377	Console Monitor Scratchpad. Area for use by the console monitor.
007400-007777	Keypad Monitor Scratchpad. Area for use by the keypad monitor and console monitor.

If you are using only the keypad, the symbol table and console monitor scratchpad areas are available to you as user RAM.

The monitor ordinarily protects certain areas of RAM from direct alteration. If you are using the keypad only, the keypad monitor scratchpad is the only area of protected RAM. If you are using a console terminal, however, the symbol table, console monitor scratchpad, and keypad monitor scratchpad are all protected. You can remove protection by activating special function 4 from the keypad (see Paragraph 3.12.5).

Note that "protection" refers only to protection from direct alteration, such as examining a location from the keypad or console and then trying to change the location. An executing program can violate any RAM location. To avoid unpredictable results, therefore, make certain your programs do not inadvertently write data into areas not available as user RAM.

5.4.2 Expansion RAM Space

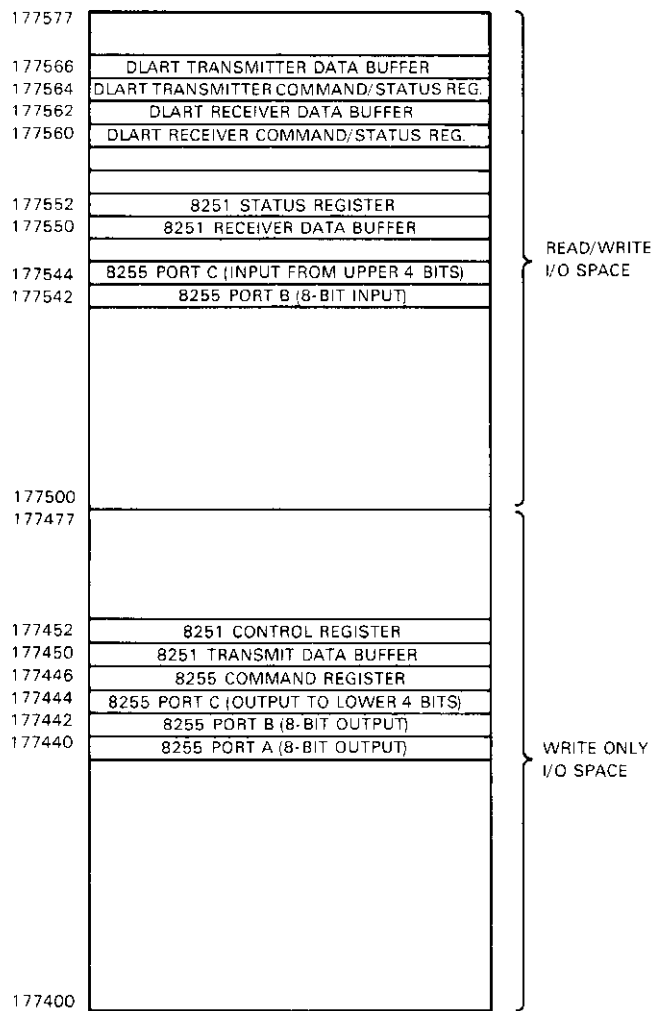
If you insert two 2 K × 8 RAMs into the expansion sockets (see Paragraph 5.3), another portion of user RAM space at locations 020000 through 027777 becomes available to you.

5.4.3 Monitor Space

The monitor space is the space occupied by the keypad and console monitors which reside in two 8 K × 8 PROMs at locations 120000 through 157777.

5.4.4 I/O Space

Locations 177400 through 177577 are designated as space for the DCT11-EM's I/O chips. The 8255A parallel port, the 8251A auxiliary serial line, and the DC319 DLART console serial line all use this space. The I/O space consists of locations that can be written only and locations that can be written or read. Figure 5-4 provides a detailed breakdown of how the I/O space is used.



MR 10351

Figure 5-4 Map of I/O Space

5.4.5 Space Available For External Hardware

The following space is available to you for any expansion hardware you may wish to design:

040000-077777	167600-177377
160000-167377	177600-177777

An example of a piece of external hardware that could use this space is a RAM board which responds to addresses 040000 through 057777. Further information on hardware expansion is provided in Chapter 6.

5.4.6 Reserved Space

The shaded space in Figure 5-3 is reserved and unavailable for use. The addresses affected are:

010000-017777	100000-117777
030000-037777	167400-167577

CHAPTER 6 HARDWARE

6.1 INTRODUCTION

The DCT11-EM hardware is organized as shown in Figure 6-1. In general, the hardware consists of:

- The DCT11-AA microprocessor
- RAM
- Monitor EPROM
- Peripheral chips and drivers
- An internal data bus buffer
- User buffers
- Microprocessor control signal decoding logic
- Address latching and range decoding logic
- Peripheral chip address decoding logic
- Interrupt logic
- Timing logic
- Miscellaneous control logic
- Keypad/LEDs and supporting logic
- The HALT and INT push-button switches
- Connectors

The paragraphs that follow provide information on each of these. Refer to the DCT11-EM schematics in Appendix B for further details.

Paragraph 6.17 contains a design for an expansion memory board you can build.

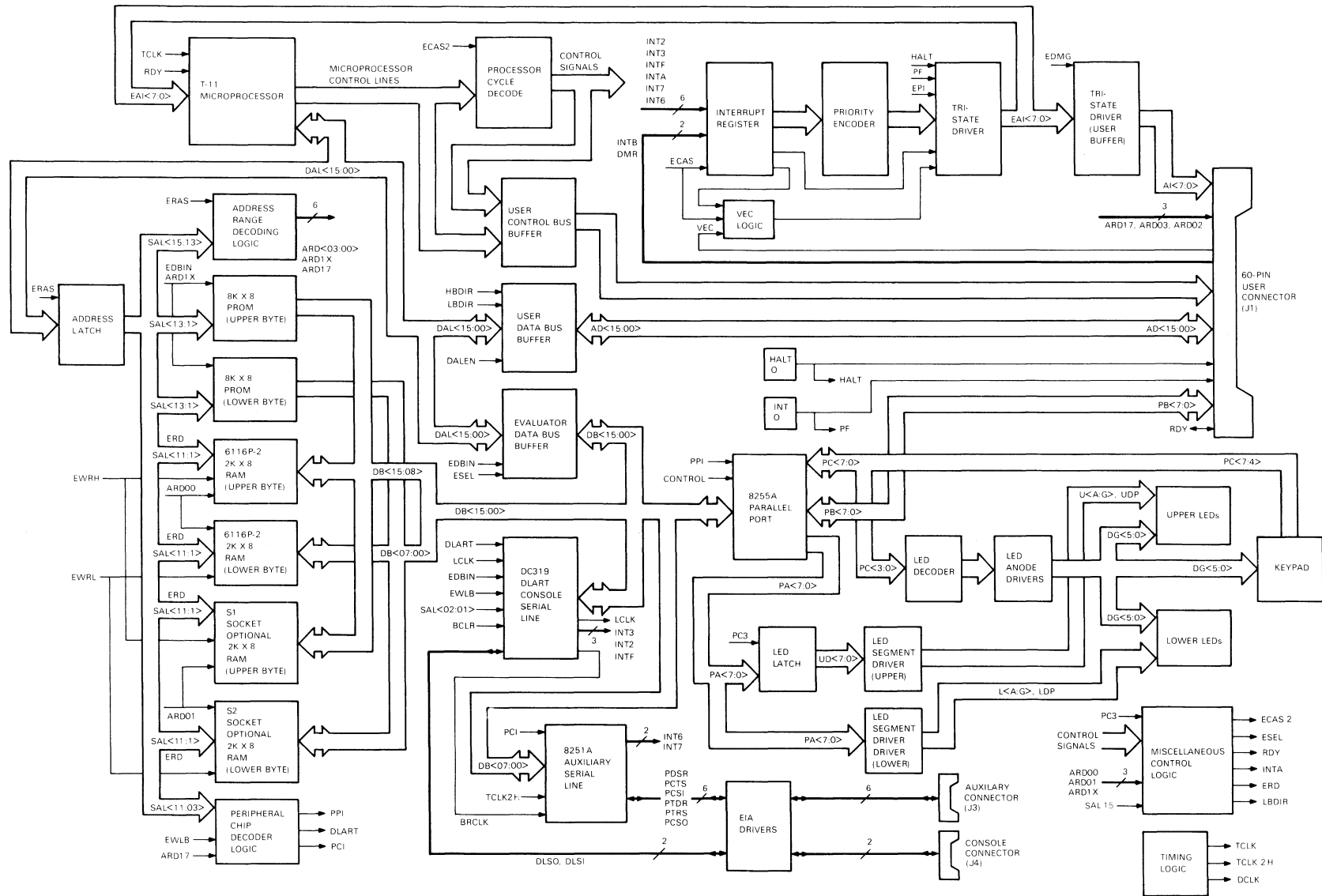


Figure 6-1 DCT11-EM System Block Diagram

6.2 DCT11-AA MICROPROCESSOR

The DCT11-AA microprocessor is the heart of the system. A detailed discussion of the operation and architecture of the DCT11-AA is beyond the scope of this book. Refer to the *DCT11-AA User's Guide* (EK-DCT11-UG) for this information.

At power-up or when a RESET instruction is executed, the DCT11-AA mode register bits are set to the following states and cannot be changed.

Mode Register Bits	State	Mode
<15:13>	000	Start address = 140000 Restart address = 140004
12	1	User
11	0	16-bit bus
10	1	4 K/16 K memory
9	0	Dynamic memory
8	0	Normal read/write
1	1	Standard microcycle
0	1	COUT = processor clock

This mode configuration has the following implications:

- Bit <11> = 1 – This allows you to use either 8-bit or 16-bit data paths in your expansion circuitry designs. If 8 bits, simply use the lower 8 bits of the 16-bit bus.
- Bits <10:09> = 10 – This allows you to use inexpensive 16 K dynamic RAMs for off-board memory expansion (see Paragraph 6.17).
- Bit <8> = 0 – This allows you to design expansion hardware with devices that respond to either normal or delayed read/write controls. If you use devices that respond to delayed read/write controls, include delay circuitry in your design.

NOTE

Do not use a RESET instruction in your program unless your program reconfigures (i.e., reinitializes) your peripheral chips. RESET causes the DCT11-AA to assert \bar{BCLR} which resets the peripheral chips.

6.3 RAM

There are two 2 K \times 8 CMOS static RAM chips in the basic DCT11-EM which provide 4 K bytes of memory space. These RAMs respond to addresses 000000 through 007777. Even numbered or low bytes reside in one RAM, and odd numbered or high bytes reside in the other.

There are two expansion sockets on the board which can accommodate two additional 2 K \times 8 static RAMs. The RAMs in these sockets respond to addresses 020000 through 027777. One RAM stores low bytes and the other stores high bytes.

Although the DCT11-AA generates refresh cycles as a result of being set to dynamic memory mode, this is for the benefit of off-board expansion with dynamic RAMs. The on-board static RAMs ignore the refresh cycles.

6.4 MONITOR EPROM

The DCT11-EM monitor is contained in two 8 K × 8 EPROM chips. The EPROMs respond to addresses 120000 through 157777.

Note that accesses to memory locations above 100000 cause a single cycle slip. This is not only for the benefit of the monitor EPROMs, but also for the peripheral chips and any slow expansion hardware which may be attached. Cycle slips are disabled if the jumper is installed (see Paragraph 2.5). If you install the jumper, make sure you also decrease the microprocessor's 7.5 MHz clock frequency to a value between 3 MHz and 6 MHz by changing the crystal.

6.5 PERIPHERAL CHIPS AND DRIVERS

The peripheral chips are the 8255A parallel port, the DC319-AA DLART console serial line, and the 8251A auxiliary serial line. The DLART and the 8251A have EIA drivers associated with them so that they can communicate with RS232-C devices. The locations reserved for use by the peripheral chips are listed in Table 6-1.

Table 6-1 Reserved I/O Locations for Peripheral Chips

Location Number	Location Type	Use
177566	Read/write	DLART transmitter data buffer
177564	Read/write	DLART transmitter command/status register
177562	Read/write	DLART receiver data buffer
177560	Read/write	DLART receiver command/status register
177552	Read/write	8251A status register
177550	Read/write	8251A receiver data buffer
177544	Read/write	8255A port C (input from upper 4 bits)
177542	Read/write	8255A port B (8-bit input)
177452	Write only	8251A control register
177450	Write only	8251A transmit data buffer
177446	Write only	8255A command register
177444	Write only	8255A port C (output to lower 4 bits)
177442	Write only	8255A port B (8-bit output)
177440	Write only	8255A port A (8-bit output)

6.5.1 Parallel Port (8255A)

The 8255A parallel port chip has two 8-bit ports and two 4-bit ports, all independently controllable. Port A (8 bits) is dedicated to supplying LED segment data to the LED segment drivers. Port A is configured as output only. Port B (8 bits) is initially configured as an input port and is available to your expansion hardware via the 60-pin connector. You can configure port B as either an input port or an output port. Port C is really two independent 4-bit ports. The upper four bits of port C are dedicated to sensing keypresses and are configured as input only. The lower four bits of port C are dedicated to turning on the LEDs and enabling the keypad. These lower four bits are configured as output only.

For detailed information on programming the 8255A, refer to the Intel Component Data Catalog (available from Intel Corp., 3065 Bowers Ave., Santa Clara, CA 95051). The order number is 210298-001.

6.5.2 DLART Console Serial Line

The DLART (DC319-AA) is an asynchronous receiver/transmitter which the DCT11-EM uses to communicate with a console terminal.

The DLART can generate receiver, transmitter, and break detection interrupt requests. See Paragraph 6.11 for more information on interrupts.

The 800 Hz clock output of the DLART is used to generate the INTA interrupt request which causes the LEDs and keypad to be activated. Again, see Paragraph 6.11.

Detailed information on programming the DLART is contained in the *DLART Data Sheet* (ED-23181).

6.5.3 Auxiliary Serial Line (8251A)

The 8251A auxiliary serial line chip is a universal synchronous/asynchronous receiver/transmitter (USART) which the DCT11-EM uses for general purpose asynchronous serial communications. Typically, the auxiliary serial line is used for communicating with a host computer.

The auxiliary serial line features modem control. The Data Carrier Detect and the SYNDET functions of the 8251A, however, are not used in the DCT11-EM.

For detailed information on programming the 8251A, refer to the Intel Component Data Catalog (available from Intel Corp., 3065 Bowers Ave., Santa Clara, CA 95051). The order number is 210298-001.

6.6 INTERNAL DATA BUS BUFFER

The internal data bus buffer is not accessible to you directly but is the buffer through which most of the data in the DCT11-EM passes. It buffers DAL<15:00>, the microprocessor's primary data bus.

6.7 USER BUFFERS

The DCT11-EM has three user buffers. One buffer provides access to DAL<15:00>, the main internal data bus. Another buffer provides access to AI<7:0>, the microprocessor's address interrupt lines. The third buffer provides access to various DCT11-EM control signals. These buffers are directly accessible through the 60-pin connector. See Paragraph 6.16 for the names of the specific buffered signals available.

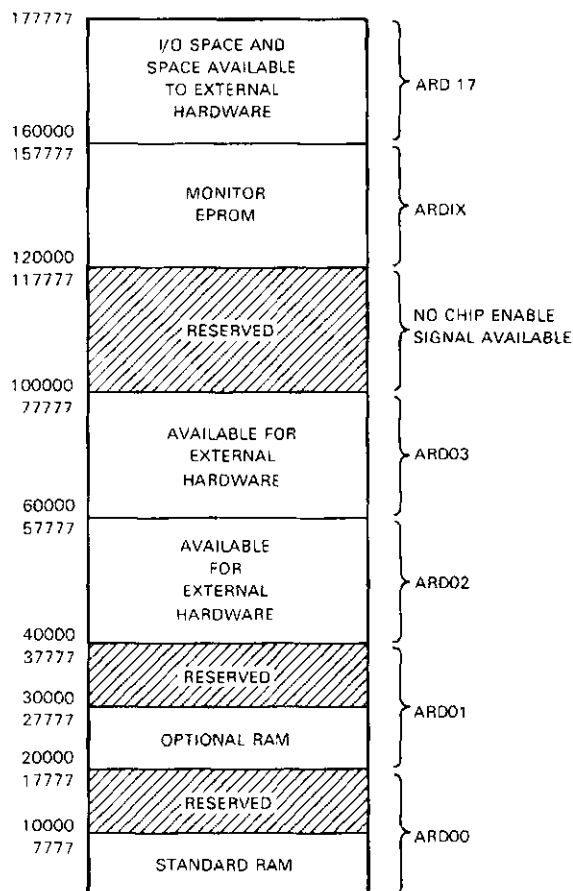
6.8 PROCESSOR CYCLE DECODING

There is some decoding logic that takes microprocessor control signals and combines them to direct the operation of various chips within the DCT11-EM. Some of these decoded signals are available to you through one of the user buffers. See Paragraph 6.16 for the names of these buffered control signals.

6.9 ADDRESS LATCHING AND RANGE DECODING

Memory addresses are latched from DAL<15:00> when the microprocessor asserts ERAS L. As shown in Figure 6-1, the upper three bits of the latched address are converted (decoded) to signals which enable pairs of on-board ROMs or RAMs.

Decoded address range signals are also available to your expansion hardware via the 60-pin connector. These signals are ARD02 L, ARD03 L, and ARD17 L. Figure 6-2 illustrates the address ranges to which these signals correspond.



MR 10353

Figure 6-2 Address Range Decoding

6.10 PERIPHERAL CHIP ADDRESS DECODING

The peripheral chip decoding logic is similar in function to the address range decoding logic (see Paragraph 6.9). The difference is that this logic generates peripheral chip enable signals instead of memory chip enable signals. Peripheral chips are enabled according to the addressing scheme shown in Paragraph 6.5. The peripheral chip decoding signals are internal to the DCT11-EM and are unavailable for expansion hardware.

6.11 INTERRUPTS AND TRAPS

Interrupt vectors are located as shown in Table 6-2.

Interrupts are encoded and sent to the DCT11-AA for handling. The DCT11-AA also handles DMA requests and external vector requests via the interrupt logic.

Some of the interrupt vectors above point to a "null" service routine. That is, they simply point to an RTI instruction. These include the power-fail, DLART transmitter, external, and 8251A transmitter interrupts. If you do not write your own service routines and change the interrupt vectors accordingly, these interrupts will have no effect.

Table 6-2 Interrupt Vector Locations

Interrupt Vector Address	Priority	Type of Interrupt
-	-	HALT switch and HALT instruction
000024	-	Nonmaskable power fail interrupt (caused by pressing INT switch)
000060	4	DLART receiver (INT3)
000064	4	DLART transmitter (INT2)
000100	6	External interrupt (INTB)
000104	6	Keypad/LED scanning (INTA)
000120	5	8251A receiver (INT7)
000124	5	8251A transmitter (INT6)
000140	7	DLART receiver break (INTF)

Other vectors include the following.

Vector Address	Description
000010	Illegal instruction trap
000014	Breakpoint and trace service

6.12 TIMING

Fundamental timing signals are generated by:

1. A 7.5 MHz crystal oscillator which drives the DCT11-AA microprocessor
2. A 614.4 KHz clock which drives the DLART.

The 7.5 MHz signal is also divided down to 1.875 MHz to drive the 8251A auxiliary serial line. These signals are unavailable for external use.

6.13 MISCELLANEOUS CONTROL LOGIC

There are pieces of logic spread throughout the DCT11-EM that generate miscellaneous control signals. The only signal that the miscellaneous control logic makes available to you is the signal RDY L, which, when asserted, causes a single cycle slip. Note that the installation of the jumper disables the on-board generation of RDY L.

6.14 KEYPAD/LEDS

The keypad/LEDs combination communicates with the rest of the DCT11-EM through the 8255A parallel port peripheral chip (see Paragraph 6.5.1). Note that the lower four bits of port C are decoded and are used to drive the LED anodes and keypad simultaneously. These lines are asserted sequentially in response to the INTA interrupt request.

6.15 HALT AND INT SWITCHES

The operation of the HALT and INT switches is explained in Paragraphs 3.9 and 3.10, respectively. To summarize, pressing the HALT switch asserts the HALT signal which restarts the keypad monitor and unconditionally stops whatever operation was in progress. Pressing the INT switch asserts the PF signal which causes a power-fail interrupt. Power-fail interrupts have no effect unless you write a service routine (vectored by location 24) to handle the interrupts.

Both HALT and PF are available to you via the 60-pin expansion connector.

6.16 CONNECTORS

There are four male connectors on the DCT11-EM as follows.

Connector	Number of Pins	Function
J1	60	Connector for expansion hardware
J2	4	Power supply connector
J3	25	Auxiliary serial line
J4	25	Console serial line

J1, J3, and J4 are all standard size connectors. J2 is the connector you assembled when you installed the DCT11-EM.

The pin assignments of these connectors are listed in Table 6-3.

Table 6-3 DCT11-EM Connector Pin Assignments

60-Pin Connector for Expansion Hardware (J1) (see Figure 6-3)

Pin Number	Signal Name	Description
1	AD00 H	AD<15:00> is the buffered version of DAL<15:00>, the bidirectional multiplexed data/address lines of the DCT11-AA.
2	AD01 H	
3	AD02 H	
4	AD03 H	
5	AD04 H	
6	AD05 H	
7	AD06 H	
8	AD07 H	
9	AD08 H	
10	AD09 H	
11	AD10 H	
12	AD11 H	
13	AD12 H	
14	AD13 H	
15	AD14 H	
16	AD15 H	
17	AI0 H	AI<7:0> is the buffered version of the DCT11-AA's AI<7:0> address interrupt lines. Output only. During DMA transactions, these lines are set to the high impedance state.
18	AI1 H	
19	AI2 H	
20	AI3 H	
21	AI4 H	
22	AI5 H	
23	AI6 H	
24	AI7 H	
25	GND	Buffered control signal. Output only. Corresponds to the DCT11-AA's -BCLR signal.
26	RESET L	
27	PB0 H	PB<7:0> are the bidirectional port B lines of the 8255A parallel port chip.
28	PB1 H	
29	PB2 H	
30	PB3 H	
31	PB4 H	
32	PB5 H	
33	PB6 H	
34	PB7 H	
35	HALT L	Asserted when HALT switch is pressed. Open collector.

Table 6-3 DCT11-EM Connector Pin Assignments (Cont)

60-Pin Connector for Expansion Hardware (J1) (see Figure 6-3)

Pin Number	Signal Name	Description
36	PF L	Asserted when INT switch is pressed. Open collector.
37	DMR L	External DMA request line. Input only.
38	INTB L	External interrupt request line. Input only.
39	VEC L	External vector request line. Input only.
40	PI H	Buffered control signal. Corresponds to the DCT11-AA's PI signal. Output only.
41	GND	
42	ARD02 H	Decoding signal for the address range 040000-057777. Output only.
43	ARD03 H	Decoding signal for the address range 060000-077777. Output only.
44	ARD17 L	Decoding signal for the address range 160000-177777. Output only.
45	IACK L	Buffered control signal. Interrupt acknowledge. Output only.
46	DMG L	Buffered control signal. DMA acknowledge. Output only.
47	SEL1 H	Buffered control signal. Corresponds to DCT11-AA's SEL1 signal. Output only.
48	SEL0 H	Buffered control signal. Corresponds to DCT11-AA's SEL0 signal. Output only.
49	WLB L	Buffered control signal. Corresponds to DCT11-AA's R/-WLB signal. Set to high impedance state during DMA. Input or output.
50	WHB L	Buffered control signal. Corresponds to DCT11-AA's R/-WHB signal. Set to high impedance state during DMA. Input or output.
51	GND	
52	RAS L	Buffered control signal. Corresponds to DCT11-AA's -RAS signal. Output only.
53	GND	
54	CAS L	Buffered control signal. Corresponds to DCT11-AA's -CAS signal. Output only.
55	GND	
56	DBIN L	Buffered control signal. Read enable line. Output only.
57	GND	
58	COU H	Buffered control signal. Corresponds to DCT11-AA's COU signal. Output only.
59	GND	
60	RDY L	Causes single cycle slip when asserted. Open collector.

Table 6-3 DCT11-EM Connector Pin Assignments (Cont)

Power Supply Connector (J2)

Pin Number	Signal Name	Description
1	+12VDC	Power signal
2	+5VDC	Power signal
3	GND	
4	-12VDC	Power signal

Auxiliary Serial Line Connector (J3)

Pin Number	Signal Name	Description
1	FGND	Frame ground
2	PCSO	Serial data out
3	PCSI	Serial data in
4	PRTS	Request to send
5	PCTS	Clear to send
6	PDSR	Data set ready
7	GND	Signal ground
8	-	Not used
9	-	Not used
10	-	Not used
11	-	Not used
12	-	Not used
13	-	Not used
14	-	Not used
15	-	Not used
16	-	Not used
17	-	Not used
18	-	Not used
19	-	Not used
20	PDTR	Data terminal ready
21	-	Not used
22	-	Not used
23	-	Not used
24	-	Not used
25	-	Not used

Table 6-3 DCT11-EM Connector Pin Assignments (Cont)

Console Serial Line Connector (J4)

Pin Number	Signal Name	Description
1	FGND	Frame ground
2	DLSO	Serial data out
3	DLSI	Serial data in
4	RTS	Request to send (always ON)
5	-	Not used
6	-	Not used
7	GND	Signal ground
8	-	Not used
9	-	Not used
10	-	Not used
11	-	Not used
12	-	Not used
13	-	Not used
14	-	Not used
15	-	Not used
16	-	Not used
17	-	Not used
18	-	Not used
19	-	Not used
20	DTR	Data terminal ready (always ON)
21	-	Not used
22	-	Not used
23	-	Not used
24	-	Not used
25	-	Not used

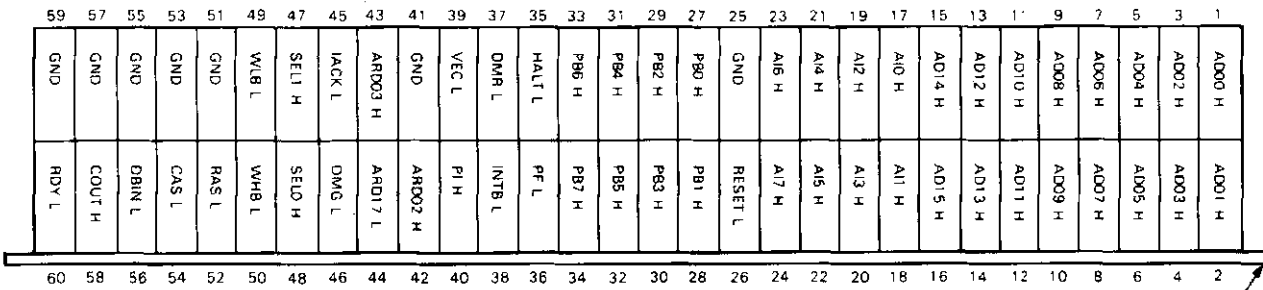


Figure 6-3 60-Pin Connector Pin Assignments

PC BOARD
MR 1037-4

6.17 HARDWARE EXPANSION - EXAMPLE

If you plan to develop and run large programs, one of the first expansion hardware circuits you may want to build is a memory module. Figure 6-4 shows a design for a 16 KB module.

The module responds to addresses 040000 through 077777. The address range decoding signals, ARD02 H and ARD03 H, are used to enable the module.

Note that the signal names on connector J1 of the memory module are in the opposite order as those shown on connector J1 of the DCT11-EM. This scheme eliminates the need to twist the interconnecting cable.

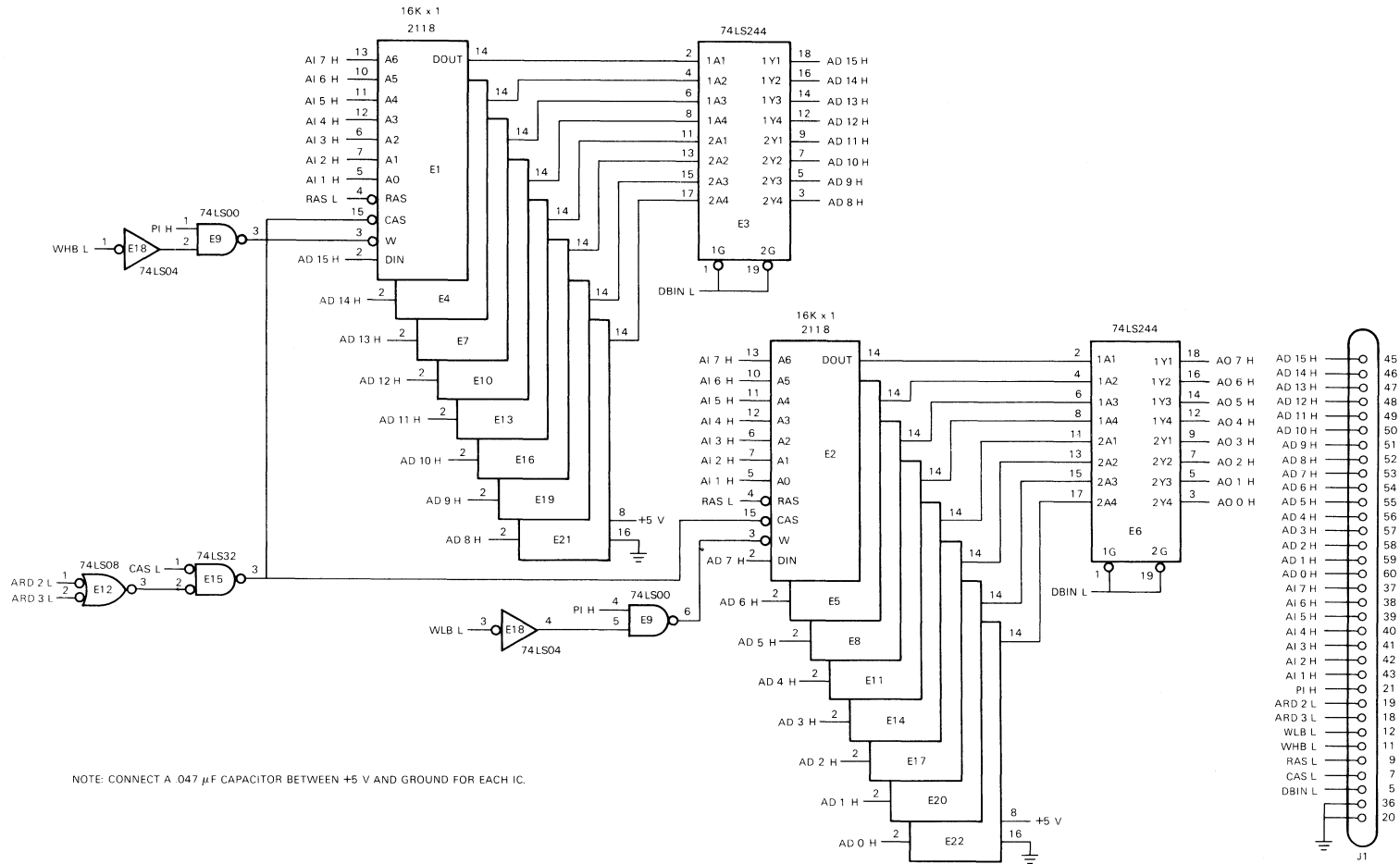


Figure 6-4 16 KB Memory Module

APPENDIX A MONITOR LISTING

T-11 Evaluation Module Monitor MACRO V05.00 Sunday 13-Mar-83 04:10
Table of contents

2-	4	Definitions
3-	2	Scratchpad locations in high RAM
5-	3	Console Serial Line Support Routines
11-	2	General I/O Support Routines
15-	2	Host Line Support Routines
17-	2	I/O Initialization and LED Driver
20-	2	LED Display Support Routines
27-	2	Error Condition and Startup Entry Code
32-	2	Entry Points
34-	2	Keypad Monitor
35-	2	Key Command Routines
36-	2	Special Keypad Monitor Functions
37-	2	Console Monitor
39-	2	Parsing Routines
44-	2	Directive and Verb Handling Routines
45-	2	Symbol and Label Handling Routines
48-	2	Arithmetic Routines
49-	2	Instruction Encode/Decode Routines
51-	2	Conversion and Timeout Routines
57-	2	Power-up Diagnostics
58-	2	Special Data Structures
59-	2	Spare Locations, Checksum, and ROM I.D.

```
1          .ENABL LC
2          ;
3          ;          T-11 EVALUATION MODULE MONITOR
4          ;
5          ;          Version 1.0      S.E.G. Hudson, Ma.
6          ;
7          ;
8          ;          COPYRIGHT (C) 1982 BY
9          ;          DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
10         ;
11         ; THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
12         ; ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
13         ; INCLUSION OF THE ABOVE COPYRIGHT NOTICE, THIS SOFTWARE OR ANY OTHER
14         ; COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
15         ; OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
16         ; TRANSFERRED.
17         ;
18         ; THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
19         ; AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
20         ; CORPORATION.
21         ;
22         ; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
23         ; SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
24         ;
```

```

1
2          .TITLE T-11 Evaluation Module Monitor
3          .IDENT /00.001/
4          .SBTTL Definitions
5
6          000000      R0      =X0          ;Registers
7          000001      R1      =X1
8          000002      R2      =X2
9          000003      R3      =X3
10         000004      R4      =X4
11         000005      R5      =X5
12         000006      SP      =X6
13         000007      PC      =X7
14
15         000024      PFVEC   ==024       ;power fail vector, called by INT button
16         000060      CONIN   ==060       ;DLART receiver interrupt vector
17         000064      CONOUT  ==064       ;DLART transmitter interrupt vector
18         000120      AUXIN   ==120       ;8251 receiver interrupt vector
19         000124      AUXOUT  ==124       ;8251 transmitter interrupt vector
20         000100      USERIV  ==100       ;uncommitted interrupt vector
21         000104      LEDIV   ==104       ;LED display interrupt vector
22         000140      CONBRK  ==140       ;DLART break interrupt vector
23
24
25         177440      P$PORA   ==177440   ;8255 port A (write only)
26         177542      P$PDIRI ==177542   ;port B (read only) available to user
27         177442      P$PORD  ==177442   ;port B (write only) available to user
28         177444      P$PDRC  ==177444   ;port C (write only) controls bits 3-0
29         177544      P$PORD  ==177544   ;port C (read only) reads bits 7-4
30         177446      P$CREG  ==177446   ;8255 command register (write only)
31
32         177550      A$RBUF   ==177550   ;8251 receiver data buffer (read only)
33         177450      A$XBUF   ==177450   ;8251 transmitter data buffer (write only)
34         177552      A$SREG   ==177552   ;8251 status register (read only)
35         177452      A$CREG   ==177452   ;8251 command register (write only)
36
37         177560      C$RCSR   ==177560   ;DLART receiver command/status register
38         177562      C$RBUF   ==177562   ;DLART receiver data buffer
39         177564      C$XCSR   ==177564   ;DLART transmitter command/status register
40         177566      C$XBUF   ==177566   ;DLART transmitter data buffer

```

Definitions

```

1
2          .SBTTL Scratchpad locations in high RAM
3
4 000000          .ASECT
5
6      006500      SCRDP1 ==      6500          ;7000          ;bottom of default console monitor scratch
7                                     ;default symbol table--66. symbol capacity
8      007320      TBLTOP ==      SCRDP1+620      ;top of symbol table, 2 words of name scratch
9
10     006500      %UR ==      6500          ;TBLTOP-<66.*6>
11
12
13          ;console monitor flag byte pairs--must start at even address
14
15     007324      LINFLG ==      TBLTOP+4          ;flags related to parsing of whole lines
16     007325      LASLIN ==      LINFLG+1          ;saved flags from previous line
17     000001      F.SST1 =001          ;line contained STEP command
18     000004      F.NULL =004          ;line is (functionally) empty
19     000010      F.SAVX =010          ;we have saved expr for double param commands
20     000020      F.COMM =020          ;we are in a comment, ignore characters
21
22     007326      CURCOM ==      LASLIN+1          ;current command flags
23     007327      LASCOM ==      CURCOM+1          ;saved flags from previous command
24     000001      F.MNEM =001          ;was an instruction mnemonic
25     000002      F.VALU =002          ;an undefined symbol or directive
26     000004      F.UNDF =004          ;expression doesn't contain undefined symbols
27     000010      F.USRD =010          ;was a user defined symbol or label
28
29     007330      THPMOD ==      LASCOM+1          ;current output mode flags
30     007331      PERMOD ==      THPMOD+1          ;permanent output mode flags
31     000002      F.ABS =002          ;absolute values, not symbolic output
32     004000      F.VT =<400*010>          ;vt mode on or off
33     000100      F.INST =100          ;instruction output
34                                     ;if neither ASCII nor instruction, then numeric
35     000200      F.BYTM =200          ;byte mode
36
37     007332      EXPFLG ==      PERMOD+1          ;current expression flags
38
39     007333      LEVEL ==      EXPFLG+1          ;parenthesis level in expression

```

```

1
2
3
4      007334      ADVADR ==      LEVEL+1      ;due to the varied uses of this location, a
5
6      ;complete list is in order:
7      ;DOCR: When dumping locations in response to a <CR>, the next address
8      ;      after that which has been dumped is saved in case the next
9      ;      command line requests another dump. The value of ADVADR need
10     ;      only be kept intact between two successive empty command lines.
11     ;$.BYTE, $.WORD, GETINS, subroutines of GETINS:
12     ;      These routines use ADVADR as a running copy of the current
13     ;      address. The current address, as stored in @$$ADDR, must not
14     ;      be changed as successive values are deposited into RAM by a
15     ;      single instruction or directive, because this would alter the
16     ;      value of the dot (.) symbol (current location reference).
17     ;DOSYMB:uses ADVADR to save R2, which points to the name to be defined,
18     ;      while the value to be equated is parsed. The value saved in
19     ;      R2 is an index to the name on the left of the equals sign, if
20     ;      the name is one of the predefined symbols or a user symbol that
21     ;      has already been defined.
22     ;TYPREG:uses ADVADR as a pointer to the list of registers to be typed.
23     ;EFND: uses ADVADR to save the upper search range limit before parsing
24     ;      the value and mask parameters.
25
26     007336      SAVEXP ==      ADVADR+2      ;saved expr value for double parameter commands
27
28     007340      COUNT1 ==      SAVEXP+2      ;counter for upper level routines
29     007341      COUNT2 ==      COUNT1+1      ;counter for lower level routines
30
31     007342      MODE ==      COUNT2+1      ;addressing mode scratch--must be even address
32     007343      OPERAT ==      MODE+1        ;save operator here in expressions
33
34     007344      CNTLC ==      OPERAT+1      ;CNTLC<0> set to 1 when console types ^C and
35     ;F.APPL is set, meaning that the
36     ;currently running program has a ^C handler
37
38     007345      DELIM ==      CNTLC+1      ;the .ASCII directive stores the delimiter here
39
40     ;now at 7346, bottom of console monitor stack
41     007346      BOTTOM ==      DELIM+1      ;console monitor has 38 word stack
42     ;NOTE: interrupts need up to 11 stack locations
43
44     007400      SCRPAD ==      BOTTOM+32     ;now at 7400, bottom of keypad monitor scratch
45     007462      STACK ==      SCRPAD+62     ;25 word stack for keypad monitor
46
47     007462      TBLBOT ==      STACK        ;pointer to bottom of symbol table space in use
48     007464      MFB251 ==      STACK + 2    ;8251 mode flag
49     007466      REPEAT ==      STACK + 4    ;console monitor command line repeat counter
50     007470      BUFFER ==      REPEAT+2     ;96 character console line input buffer
51     007630      BP ==      BUFFER+140      ;buffer back pointer, must be even address
52     007631      FP ==      BP+1            ;buffer forward pointer, must follow BP
53     007632      HOSTBF ==      FP+1         ;32 character host input buffer
54     007672      HFP ==      HOSTBF+40      ;buffer forward pointer
55     007673      HSTFLG ==      HFP+1       ;host flag
56     000200      F.LOAD =200                ;1 = taking input from host
57     000100      F.LDST =100                ;1 = looking for .START to begin loading

```

Scratchpad locations in high RAM

58	000001	F.BYTB	=001	i1 = ignoring message from host (byte bucket)
59				
60	007674	PWRFVS	== HSTFLG+1	i1 save user's power fail vector here
61	007676	KEYPIV	== PWRFVS+2	i1 keypress service vector
62	007700	KEYPVS	== KEYPIV+2	i1 save user's keypress vector here
63	007702	USERVS	== KEYPVS+2	i1 save user's interrupt vector here
64	007704	DEBNCE	== USERVS+2	i1 keypad debounce resistor
65	007706	FLAGSI	== DEBNCE+2	i1 flag bits, defined as follows
66	000001	F.DATA	=000001	i1 = data entry mode
67	000002	F.CHAN	=000002	i1 = numeric key pressed after opening location
68	000004	F.ADDR	=000004	i1 = address entry mode
69	000010	F.REG	=000010	i1 = register selection mode
70	000020	F.BRKS	=000020	i1 = breakpoint selection mode
71	000040	F.FUNC	=000040	i1 = special function selection mode
72	000100	F.USER	=000100	i1 = executing user code
73	000200	F.SST	=000200	i1 = single stepping
74	000400	F.BRKA	=000400	i1 = breakpoints active (installed)
75	001000	F.BRKG	=001000	i1 = currently stepping over a breakpoint
76	002000	F.KEYP	=002000	i1 = running console monitor
77	004000	F.MOST	=004000	i1 = accepting console commands from both ports
78	010000	F.APPL	=010000	i1 = running program with ^C handler
79	020000	F.TBAS	=020000	i1 = print address break point is at
80	040000	F.PROT	=040000	i1 = monitor scratch protection disabled
81	100000	F.PAS1	=100000	i1 = console monitor PASS1 mode active
82				
83	007710	CONFLG	== FLAGSI+2	i1 console line flag, must be even address
84	000002	F.STOP	=002	i1 control S pressed, execution suspended
85	000010	F.CPB0	=010	i1 programmable baud rate stored here
86	000020	F.CPB1	=020	
87	000040	F.CPB2	=040	
88	000200	F.A64X	=200	i1 aux port mode word bit<0>, selects 16X or 64X
89				
90	007711	AUXFLG	== CONFLG+1	i1 copy of what has been written to A\$CREG
91				
92	007712	SEGBUF	== AUXFLG+1	i1 6 word segment data buffer for LED displays
93				i1 high byte for upper display, low for lower
94	007726	SPCLBF	== SEGBUF+14	i1 word buffer for special 2 character display
95	007730	TIME	== SPCLBF+2	i1 upper 6 bits count 60 seconds, lower 10 count
96				i1 eighthundredths of seconds
97	007732	\$R0	== TIME+2	i1 save user context here
98	007734	\$R1	== \$R0+2	
99	007736	\$R2	== \$R1+2	
100	007740	\$R3	== \$R2+2	
101	007742	\$R4	== \$R3+2	
102	007744	\$R5	== \$R4+2	
103	007746	\$SP	== \$R5+2	
104	007750	\$PC	== \$SP+2	
105	007752	\$PS	== \$PC+2	
106	007754	\$WATCH	== \$PS+2	i1 watchpoint address
107	007756	\$ADDR	== \$WATCH+2	i1 current address
108	007760	BRKFIL	== \$ADDR+2	i1 file of four breakpoint address, contents pairs


```

1
2
3          .SBTTL Console Serial Line Support Routines
4
5          133000          ,      =      133000
6
7
8          ;++
9          ; LINEIN answers console input interrupts and buffers the incoming characters.
10         ; The characters ^C, ^D, ^O, ^S, and ^Y are trapped in this routine.
11         ;--
12
13 133000 010046          LINEIN::MOV      R0,-(SP)          ;preserve some registers for scratch
14 133002 010146          MOV          R1,-(SP)
15 133004 113700 177560          1$:  MOVB   @#C#RCSR,R0          ;read console receiver status register
16 133010 100375          BPL          1$          ;in polled mode, wait for a character
17 133012 013700 177562          MOV      @#C#RBUF,R0          ;read console receiver
18 133016 100005          BPL          2$          ;no error bits set
19 133020 012700 000007          5$:  MOV      #007,R0          ;on error, output <BELL>
20 133024 004767 000510          JSR      PC,CHRD1
21 133030 000457          BR          9$          ;ignore character, continue
22
23 133032 042700 177600          2$:  BIC      #177600,R0          ;clear all but 7 bits
24 133036 001454          BEQ      9$          ;ignore ASCII nulls
25 133040 120027 000003          CMPB   R0,#003          ;is it ^C?
26 133044 001454          BEQ      6$          ;yes
27 133046 120027 000031          CMPB   R0,#031          ;is it ^Y?
28 133052 001461          BEQ      7$          ;yes
29 133054 132737 000002 007710          BITB   #F,STOP,@#CONFLG;are we waiting for ^Q?
30 133062 001407          BEQ      3$          ;no
31 133064 120027 000021          CMPB   R0,#021          ;is it ^Q?
32 133070 001345          BNE      1$          ;no, wait for one
33 133072 142737 000002 007710          BICB   #F,STOP,@#CONFLG;clear ^S flag
34 133100 000433          BR          9$          ;ignore character, continue
35
36 133102 120027 000023          3$:  CMPB   R0,#023          ;is it ^S?
37 133106 001006          BNE      4$          ;no
38 133110 152737 000002 007710          BISB   #F,STOP,@#CONFLG;set ^S flag to suspend execution
39 133116 106427 000200          MTPS   #200          ;accept other interrupts while suspended
40 133122 000730          BR          1$          ;wait for ^Q
41
42 133124 120027 000017          4$:  CMPB   R0,#017          ;is it ^Q?
43 133130 001004          BNE      8$          ;no
44 133132 152737 000001 007673          BISB   #F,BYTB,@#HSTFLG;turn on host line byte bucket
45 133140 000413          BR          9$
46
47 133142 113701 007631          8$:  MOVB   @#FP,R1          ;set buffer input pointer
48 133146 042701 177400          BIC      #177400,R1          ;prevent sign extend
49 133152 120127 000140          CMPB   R1,#140          ;is buffer full? (96 characters)
50 133156 103320          BHIS   5$          ;yes, send <BELL>
51 133160 110061 007470          MOVB   R0,BUFFER(R1)          ;no, store character
52 133164 105237 007631          INCB   @#FP          ;advance pointer
53
54 133170 012601          9$:  MOV      (SP)+,R1          ;restore registers
55 133172 012600          MOV      (SP)+,R0
56 133174 000002          RTI          ;and continue
57

```

```

58 133176 032737 030000 007706 6%: RIT  #F,APPL+F,TRAS,@#FLAGS1 ;stop on ^C?
59 133204 001404 BEQ  7%          ;yes, not running program with ^C handler
60 133206 152737 000001 007344 BISS #1,@#CNTLC   ;no, just set flag for application program
61 133214 000765 BR   9%          ;and return
62
63 133216 012700 054536      7%: MOV  #400#131+136,R0 ;^C and ^Y terminate execution
64 133222 004767 000302 JSR  PC,CHROUT    ;type ^Y
65 133226 042737 000300 007706 BIC  #F,USER+F,SST,@#FLAGS1 ;clear user program flags
66 133234 000167 004330 JMP  $MONIT      ;remove breakpoints, enter monitor
  
```

```

1
2
3          ;++
4          ; GETLIN prompts for a line from the console, and returns with the line in
5          ; BUFFER and R3 cleared. Upon entry, BP should index the buffer position
6          ; just beyond the last character previously taken from the buffer (if typeahead
7          ; characters are desired). To throw away the typeahead characters, set BP and
8          ; FP to zero.
9          ; The characters typed on the console (and any earlier typeahead characters)
10         ; are echoed to the console following the prompt, "> ". Note that the ECHO
11         ; routine, which was specialized for use with monitor command input, echoes
12         ; control characters as ""<character>" until a semicolon is found on the line.
13         ; Control characters to the right of a semicolon (i.e. within a 'comment') are
14         ; echoed invisibly so that the user can echo control sequences to the console
15         ; terminal.
16         ;
17         ; Uses:      BP Index to BUFFER location just beyond last fetched character
18         ;              (which is the location that contains the first character in
19         ;              the new line)
20         ;
21         ; Returns:
22         ; BP      Cleared
23         ; R3      Cleared
24         ; BUFFER Locations indexed by entry value of BP through the end of the
25         ;          buffer replace contents of locations at the beginning of the
26         ;          buffer
27         ; FP      Adjusted to index the new next available location in the buffer
28         ;--
29
30 133240 105037 007324 GETLIN:;CLRB @#LINFLG ;reset line parsing flags for new line
31 133244 004767 000200 JSR  PC,PURGE  ;purge bottom of buffer, clear running pointer
32
33 133250 004767 000250 1%: JSR  PC,CRLF  ;start a new line
34 133254 012700 020076 MOV  #400#040+076,R0 ;type > <SPACE>
35 133260 004767 000244 JSR  PC,CHROUT
36 133264 123703 007631 2%: CMPB @#FP,R3    ;see if a character has been typed
37 133270 001775 BEQ  2%          ;no, wait for one
38 133272 004767 000644 JSR  PC,ECHO   ;echo character to console
39 133276 126327 007467 000015 CMPB BUFFER-1(R3),#015 ;is last character a <CR>?
40 133304 001367 BNE  2%          ;no, wait for one
41 133306 105037 007324 CLRB @#LINFLG  ;reset line flags from ECHO
42 133312 005003 CLR  R3         ;set running pointer to beginning of line
43 133314 000207 RTS  PC        ;return to caller
  
```

```

1
2      ;++
3      ; GETCH retrieves characters from the input buffer using R3 as an index.
4      ; The character is placed in the lower 7 bits of R0. Lower case characters
5      ; are folded to upper case. The N flag is set if the character is a separator
6      ; (space, horizontal tab, vertical tab, line feed, or form feed) and the C flag
7      ; is set if the character has been folded. Upon entry, R3 must index a byte
8      ; within the console input buffer (R3 < 140 octal or 96 decimal).
9      ;
10     ; Uses: R3      Index to next character in console input buffer (BUFFER)
11     ;
12     ; Returns:
13     ;      R0      Contains character fetched from BUFFER in lower 7 bits,
14     ;              upper bits cleared
15     ;      R3      Set to previous value + 1
16     ;      N,C     See comments above
17     ;--
18
19 133316 116300 007470   GETCH:: MOVB  BUFFER(R3),R0   ;yes, set a character--bit 7 should be clear
20 133322 005203         INC      R3
21 133324 120027 000011   CMPB   R0,#011   ;is it less than <TAB>?
22 133330 103423         BLD     3%      ;yes, not a separator
23 133332 120027 000040   CMPB   R0,#040   ;is it a space?
24 133336 001403         BEQ     2%      ;yes, a separator
25 133340 120027 000014   CMPB   R0,#014   ;is it <FF> or lower?
26 133344 101003         BHI     4%      ;no, check for lower case
27 133346 000270         2%:   SEN      ;flag a separator
28 133350 000241         CLC
29 133352 000207         RTS     PC
30
31 133354 120027 000141   4%:   CMPB   R0,#141   ;is it lower case 'a'?
32 133360 103407         RLD     3%      ;no, don't fold
33 133362 120027 000172   CMPB   R0,#172   ;is it lower case 'z'?
34 133366 101004         BHI     3%      ;no, don't fold
35 133370 162700 000040   SUB     #040,R0   ;fold to upper case
36 133374 000261         SEC      ;flag folded character
37 133376 000401         BR     5%
38 133400 000241         3%:   CLC      ;flag a normal character
39 133402 000250         5%:   CLN
40 133404 000207         RTS     PC
41

```

```

1
2           .ENABL LSB
3       ;++
4       ;   GETLCH recovers the most recently gotten character.
5       ;
6       ;   Uses:   R3       Index to location in BUFFER just beyond character to be
7       ;           returned
8       ;
9       ;   Returns:
10      ;           R3       Equal to previous value
11      ;           R0       Contains fetched character in lower 7 bits; upper bits cleared
12      ;           N,C      (See header of GETCH routine)
13      ;--
14
15 133406 005303   GETLCH::DEC   R3
16 133410 000742           BR     GETCH
17
18      ;GETCHC is like GETCH except it doesn't fold lower case.
19 133412 004767 177700   GETCHC::JSR   PC,GETCH
20 133416 103002           2#:   BCC   1#
21 133420 062700 000040           ADD   #040,R0
22 133424 000207           1#:   RTS   PC
23
24
25      ;++
26      ;   GETNXT is like GETCH except it doesn't return separators.
27      ;
28      ;   (See usage information of GETCH)
29      ;--
30
31 133426 004767 177664   GETNXT::JSR   PC,GETCH
32 133432 100775           BHI   GETNXT
33 133434 000207           RTS   PC
34           .DSABL LSB
35
36
37      ;++
38      ;   GETBP places the back pointer in R3 and prevents sign extension. This is
39      ;   useful to refetch characters from a line (providing BP still indexes the
40      ;   beginning of the line).
41      ;
42      ;   Uses:   BP       Index to be transferred to R3
43      ;
44      ;   Returns:
45      ;           R3       Contains value in BP
46      ;--
47
48 133436 113703 007630   GETBP:: MOVB  @#BP,R3
49 133442 042703 177400           BIC  #177400,R3
50 133446 000207           RTS   PC
51

```

Console Serial Line Support Routines

```

1
2          ;++
3          ; PURGE shifts the input buffer to eliminate space below BP, the back pointer.
4          ; Console input interrupts are blocked while the shift is occurring. R3, the
5          ; running pointer, is cleared.
6          ;
7          ; Uses:   BP      Index of first location to be kept in console input buffer
8          ;                   (Locations below BUFFER+BP] are overwritten)
9          ;                   FP      Index of first unused location in BUFFER
10         ;
11         ; Returns:
12         ;         BP      Cleared
13         ;         R3      Cleared
14         ;         FP      Indexes new first unused location in BUFFER
15         ;--
16
17 133450 106746          PURGE:: MFPS  -(SP)          isave priority
18 133452 106427 000340          MTPS  #340          idisable interrupts temporarily
19 133456 004767 177754          JSR   PC,GETBP      iset back pointer
20 133462 005000          CLR   R0           imake a pointer to the bottom of the buffer
21 133464 120337 007631          2%:  CMPB  R3,@#FP    ihas entire buffer been transferred?
22 133470 103006          BHIS  1%           iyes
23 133472 116360 007470 007470          MOVB  BUFFER(R3),BUFFER(R0) itransfer a character
24 133500 005200          INC   R0
25 133502 005203          INC   R3
26 133504 000767          BR    2%
27
28 133506 110037 007631          1%:  MOVE  R0,@#FP    ifix forward pointer,
29 133512 105037 007630          CLRB  @#BP          iback pointer,
30 133516 005003          CLR   R3           iand running pointer
31 133520 106426          MTPS  (SP)+        irestore priority
32 133522 000207          RTS   PC
33
34
35         ;++
36         ; CRLF sends a carriage return and line feed to the console (and printer).
37         ;
38         ; Returns:
39         ;         R0      Contains garbage (400 * 15 + 12)
40         ;--
41
42 133524 012700 005015          CRLF:: MOVB  #400#012+015,R0 ;<CR>, <LF>
43

```

```

1
2          ;++
3          ;   CHROUT outputs the lower byte of R0 to the console, and to the aux port
4          ;   if the printer port bit is set.  If the upper byte of R0 is non-zero,
5          ;   it is also sent to the port(s).  CHRO1 outputs only the lower byte of R0,
6          ;   and CHRO2 can be used to output a character to the aux. port only.
7          ;
8          ;   Uses:   R0       Lower byte contains first ASCII character to be output
9          ;           Upper byte contains zero or second character to be output
10         ;--
11
12         .ENABL  LSB
13 133530 004767 000004      CHROUT::JSR  PC,CHRO1      ;output lower half of R0
14 133534 105700             TSTB   R0              ;is there a second character?
15 133536 001431             BEQ    1$              ;no
16
17 133540 105737 177564      CHRO1::TSTB  @C$XCSR      ;check console transmitter status register
18 133544 100375             BPL    CHRO1          ;wait until transmitter ready bit is set
19 133546 110037 177566      MOVB   R0,@C$XBUF    ;then output a character
20 133552 000423             BR     1$              ;no, done
21 133554 132737 000004 177552  CHRO2::BITB  #004,@A$SREG  ;is aux port transmitter empty?
22 133562 001774             BEQ    CHRO2          ;no, wait until it is
23 133564 105737 177552      TSTB   @A$SREG      ;is DSR active?
24 133570 000240             NDP                    ;placeholder for above line
25 133572 106746             MFPS   -(SP)          ;save priority
26 133574 106427 000340      MTPS   #340          ;block interrupts while we
27 133600 112737 000047 177452  MOVB   #047,@A$CREG  ;enable aux port transmitter
28 133606 110037 177450      MOVB   R0,@A$XBUF    ;output a character
29 133612 113737 007711 177452  MOVB   @AUXFLG,@A$CREG ;restore previous aux port command word
30 133620 106426             MTPS   (SP)+          ;restore priority
31 133622 000300             1$: SWAB  R0
32 133624 000207             RTS    PC
33         .DSABL  LSB

```

```

1
2          .SBTTL  General I/O Support Routines
3
4          ;++
5          ;   ASCOUT outputs the character in R0 to the port(s) if it is not a control
6          ;   character.  If so, the special sequences '^A' through '^L' are sent with
7          ;   exceptions <SP>, <ESC>, and <DEL>.
8          ;
9          ;   Uses:   R0      Lower 7 bits contain character to be printed
10         ;
11         ;   Returns:
12         ;           R0      Contains garbage
13         ;--
14
15 133626 042700 177600  ASCOUT::BIC  #177600,R0      ;make R0 a 7-bit character
16 133632 120027 000177          CMPB   R0,#177      ;is it delete?
17 133636 001413          BEQ    1$           ;yes, type <DEL>
18 133640 120027 000040          CMPB   R0,#040     ;is it control?
19 133644 101331          BHI    CHROUT      ;no, type it normally
20 133646 001416          BEQ    5$           ;make spaces visible
21 133650 120027 000033          CMPB   R0,#033     ;is it escape?
22 133654 001422          BEQ    3$           ;yes, type <ESC>
23 133656 000300          SWAB   R0          ;no, type ~(letter)
24 133660 062700 040136          ADD    #400#100+136,R0 ;make it second character after '^'
25 133664 000721          BR     CHROUT      ;and type it
26
27 133666 012700 042074          1$:   MOV    #400#104+074,R0 ;type <D
28 133672 004767 177632          JSR    PC,CHROUT
29 133676 012700 046105          MOV    #400#114+105,R0 ;type EL
30 133702 000415          BR     4$
31
32 133704 012700 051474          5$:   MOV    #400#123+074,R0 ;type <S
33 133710 004767 177614          JSR    PC,CHROUT
34 133714 012700 037120          MOV    #400#076+120,R0 ;type P>
35 133720 000703          BR     CHROUT
36
37 133722 012700 042474          3$:   MOV    #400#105+074,R0 ;type <E
38 133726 004767 177576          JSR    PC,CHROUT
39 133732 012700 041523          MOV    #400#103+123,R0 ;type SC
40 133736 004767 177566          4$:   JSR    PC,CHROUT
41 133742 012700 000076          MOV    #076,R0      ;type >
42 133746 000670          BR     CHROUT
43

```

```
1
2          ;++
3          ; PRINTA prints an ASCII string pointed to by R1. The string must end with a
4          ; blank byte.
5          ;
6          ; Uses:   R1      Points to string of bytes to be printed
7          ;
8          ; Returns:
9          ;         R1      Points to byte just beyond first byte in string containing zero
10         ;--
11
12         .ENABL LSB
13 133750 004767 177564 1$: JSR PC,CHRD1
14 133754 112100 PRINTA::MOVE (R1)+,R0 ;get a character from string
15 133756 001374 RNE 1$ ;print it
16 133760 000207 RTS PC
17         .DSABL LSB
18
```



```

1
2
3           ;++
4           ; PRINT outputs a radix-50 string pointed to by R2. The string must end with
5           ; a blank word.
6           ; PRINT1 outputs a single radix-50 triplet in R1.
7           ; PRINT2 outputs a pair of radix-50 triplets pointed to by R2. No terminator
8           ; is needed.
9           ; PRINT3 outputs the most significant radix-50 character in R1 and multiplies
10          ; R1 by 40. (PRINT3 is called 3 times by PRINT1.)
11          ;
12          ; Uses:
13          ; R2 Points to RAD50 string to be printed (PRINT)
14          ; R1 Points to RAD50 pair of words to be printed (PRINT2)
15          ; R1 Contains RAD50 triplet to be printed (PRINT1)
16          ; R1 Contains RAD50 character to be printed (PRINT3)
17          ;
18          ; Returns:
19          ; R2 Points to next word after blank word in RAD50 string (PRINT)
20          ; R1 Points to word after the pair of words printed (PRINT2)
21          ; R1 Contains zero (PRINT, PRINT1, PRINT2)
22          ; R1 Contains <previous value of R1> MOD 1600, * 40, (PRINT3)
23          ;--
24          .ENABL LSB
25          1$: JSR PC,PRINT1 ;type the triplet
26          PRINT1: MOV (R2)+,R1 ;set a triplet
27          BNE 1$ ;not the end yet
28          RTS PC
29          .DSABL LSB
30          PRINT2: MOV (R2)+,R1 ;set a triplet
31          JSR PC,PRINT1 ;type it
32          MOV (R2)+,R1 ;set another
33
34          PRINT1: JSR PC,PRINT3 ;type the first character
35          PRINT4: JSR PC,PRINT3 ;and the second
36
37          PRINT3: CLR R0 ;divide by successive subtraction
38          1$: INC R0
39          SUB #50*50,R1 ;try to subtract 1600.
40          BCC 1$ ;worked, no borrow
41          DEC R0 ;fix R0
42          ADD #50*50,R1 ;and R1
43          JSR PC,R1*40 ;multiply R1 by 40
44          TST R0
45          BEQ 2$ ;0 means space
46          CNF R0,#033 ;032 means Z
47          BLO 3$ ;A through Z
48          BEQ 4$ ;033 means $
49          CMP R0,#035 ;035 is undefined, but type *
50          BEQ 5$
51          ADD #022,R0 ;0 through 9 and ','
52          6$: JMP CHROUT
53
54          2$: MOV #040,R0 ;type space
55          BR 6$
56
57          3$: ADD #100,R0 ;A through Z

```

```

58 134104 000770          BR      6$
59
60 134106 012700 000044    4$:   MOV     #044,R0      ;type $
61 134112 000765          BR      6$
62
63 134114 012700 000052    5$:   MOV     #052,R0      ;type $
64 134120 000762          BR      6$
65
66 134122 006301          RIX40:: ASL     R1          ;multiply R1 by 8
67 134124 006301          ASL     R1
68 134126 006301          ASL     R1
69 134130 010146          MOV     R1,-(SP)      ;save 8 X
70 134132 006301          ASL     R1
71 134134 006301          ASL     R1          ;32 X
72 134136 062601          ADD     (SP)+,R1     ;plus 8 X makes 40 X
73 134140 000207          RTS     PC
74

```

```

1
2          ;++
3          ; ECHO reads the line buffer as characters are typed in and echoes them
4          ; to the console (and printer, if selected). Control characters are echoed
5          ; as ^ (letter) except within comments. LINFLG is used to record comment status.
6          ; If ECHO encounters ^R, ^U, ^X, or <DELETE>, it responds or modifies the line
7          ; buffer accordingly. ECHO begins echoing characters at the running pointer
8          ; (R3) and continues until R3 = FP or a <CR> is found.
9          ;
10         ; Uses:      R3      Index to first character in BUFFER to be echoed
11         ;             FP      Index to end of data in buffer
12         ;
13         ; Returns:
14         ;             R3      Index to location in BUFFER logically just beyond last echoed
15         ;                     byte (This means R3 returns zero if ^U or ^X was the last
16         ;                     character in the buffer, etc.)
17         ;--
18
19 134142 116300 007470      ECHO:: MOVB  BUFFER(R3),R0      ;set character
20 134146 120027 000022      CMPB  R0,#022      ;is it ^R?
21 134152 001447           BEQ   1$              ;yes, retype line
22 134154 120027 000030      CMPB  R0,#030      ;is it ^X?
23 134160 001002           BNE   3$              ;no
24 134162 012700 000025      MOV   #025,R0      ;map ^X to ^U
25 134166 120027 000025      3$:  CMPB  R0,#025      ;is it ^U?
26 134172 001450           BEQ   2$              ;yes, cancel line
27 134174 120027 000015      CMPB  R0,#015      ;is it <CR>?
28 134200 001432           BEQ   17$             ;yes, stop echoing characters
29 134202 120027 000177      CMPB  R0,#177      ;is it <DELETE>?
30 134206 001453           BEQ   4$              ;yes, delete last character
31 134210 120027 000073      CMPB  R0,#073      ;is it a semicolon
32 134214 001003           BNE   5$              ;no
33 134216 152737 000020 007324  BISB  #F,COMM,@#LINFLG ;yes, show we are in a comment
34 134224 132737 000020 007324  5$:  BITB  #F,COMM,@#LINFLG ;are we in a comment?
35 134232 001006           BNE   7$              ;yes, echo characters literally
36 134234 004767 177152      JSR  PC,GETCHC
37 134240 100404           BMI   6$              ;no
38 134242 004767 177360      14$: JSR  PC,ASCOUT      ;type non-printable characters visibly
39 134246 000403           BR   8$              ;no
40
41 134250 005203           7$:  INC  R3              ;advance running pointer
42 134252 004767 177252      6$:  JSR  PC,CHROUT      ;type non-printable characters as they are
43 134256 120337 007631      8$:  CMPB  R3,@#FP      ;have we echoed everything?
44 134262 103727           BLD  ECHO           ;no, do some more
45 134264 000207           RTS  PC              ;yes, continue
46
47 134266 005203           17$: INC  R3              ;step past <CR>
48 134270 000207           RTS  PC              ;and return
49
50 134272 004767 000172      1$:  JSR  PC,10$         ;some common code
51 134276 004767 000210      JSR  PC,15$         ;remove ^R from line
52 134302 004767 177130      JSR  PC,GETBP       ;go back to the beginning
53 134306 106427 000000      MTPS #000          ;take characters again
54 134312 000761           BR   8$              ;and echo line
55
56 134314 004767 000150      2$:  JSR  PC,10$         ;some common code
57 134320 004767 177112      JSR  PC,GETBP       ;go back to the bottom of the buffer

```

```

58 134324 110337 007631          MOVB  R3,@#FP      ;indicate buffer empty
59 134330 106427 000000          MTPS  #000        ;take characters again
60 134334 000207                   RTS   PC           ;wait for new characters
61
62 134336 106427 000200          4%:  MTPS  #200        ;don't take characters while buffer is fixed
63 134342 004767 000144          JSR   PC,15%      ;remove <DELETE> from line, decrement FP
64 134346 120337 007630          CMPB  R3,@#BP     ;is there anything to delete?
65 134352 001443                   BEQ   %           ;no, continue
66 134354 005303                   DEC   R3           ;back up to character to be deleted
67 134356 032767 004000 052744 100%: BIT   #F.VT,TMPMOD ;see if video mode
68 134364 001415                   BEQ   200%        ;branch if not
69 134366 012700 000010          MOV   #10,R0      ;send backspace
70 134372 004767 177132          JSR   PC,CHROUT   ;
71 134376 012700 000040          MOV   #40,R0      ;send space
72 134402 004767 177122          JSR   PC,CHROUT   ;
73 134406 012700 000010          MOV   #10,R0      ;send backspace
74 134412 004767 177112          JSR   PC,CHROUT   ;
75 134416 000410                   BR    210%        ;skip printer '\
76 134420 012700 000134          200%: MOV   #134,R0 ;type '\
77 134424 004767 177100          JSR   PC,CHROUT   ;
78 134430 116300 007470          MOVB  BUFFER(R3),R0 ;set deleted character
79 134434 004767 177166          JSR   PC,ASCOUT   ;type it
80 134440 116300 007470          210%: MOVB  BUFFER(R3),R0 ;check if we deleted a ';'
81 134444 120027 000073          CMPB  R0,#073     ;
82 134450 001002                   BNE   13%         ;
83 134452 105037 007324          CLRB  @#LINFLG    ;yes, so we aren't in a comment anymore
84 134456 004767 000030          13%:  JSR   PC,15% ;remove deleted character from buffer
85 134462 106427 000000          9%:   MTPS  #000        ;take characters again
86 134466 000673                   BR    8%          ;echo rest of line
87
88 134470 106427 000200          10%:  MTPS  #200        ;don't take characters for a while
89 134474 004767 177126          JSR   PC,ASCOUT   ;echo the special character
90 134500 004767 177020          JSR   PC,CRLF     ;start a new line
91 134504 105037 007324          CLRB  @#LINFLG    ;initialize flags
92 134510 000207                   RTS   PC           ;
93
94 134512 110300                   15%:  MOVB  R3,R0        ;set a scratch copy of running pointer
95 134514 116060 007471 007470 16%:  MOVB  BUFFER+1(R0),BUFFER(R0) ;remove 1 character from buffer
96 134522 005200                   INC   R0           ;
97 134524 120037 007631          CMPB  R0,@#FP     ;is that everything?
98 134530 103771                   BLD   16%         ;no, do more
99 134532 105337 007631          DECB  @#FP        ;fix FP
100 134536 000207                   RTS   PC           ;

```

```

1
2                               .SBTTL Host Line Support Routines
3
4                               ;++
5                               ; HOSTIN answers aux. port receiver interrupts and buffers the incoming
6                               ; characters in HOSTBF (up to 32 characters), ^@, ^C, ^S, and ^Y are ignored
7                               ; coming from this port. When the buffer becomes half-full (16 characters),
8                               ; a ^S is sent to the host.
9                               ;--
10
11 134540 010046                HDSTIM:;MOV    R0,-(SP)        ;preserve some registers for scratch
12 134542 010146                MOV     R1,-(SP)
13 134544 113700 177550          MOVB  @#A#RBUF,R0        ;set the character, reset interrupt
14 134550 132737 000060 177552  BITB  #060,@#A#SREG    ;framing or overrun error?
15 134556 001050                BNE   3$              ;yes, beep console, and ignore character
16 134560 042700 177600          BIC   #177600,R0      ;clear all but 7 bits
17 134564 001464                BEQ   1$              ;ignore ASCII nulls
18 134566 120027 000003          CMPB  R0,#003         ;ignore ^C
19 134572 001453                BEQ   2$              ;
20 134574 120027 000031          CMPB  R0,#031        ;and ^Y
21 134600 001450                BEQ   2$              ;
22 134602 120027 000023          CMPB  R0,#023        ;and ^S so that host line traps the same chars
23 134606 001445                BEQ   2$              ;as console line, even though ignored
24 134610 113701 007672          MOVB  @#HFP,R1        ;set the host buffer input pointer
25 134614 042701 177400          BIC   #177400,R1     ;prevent sign extend
26 134620 120127 000040          CMPB  R1,#40         ;is buffer full?
27 134624 103036                BHIS  2$              ;yes, beep the console, ignore error
28 134626 110061 007632          MOVB  R0,HOSTBF(R1)  ;no, store character
29 134632 105237 007672          INCB  @#HFP          ;advance the pointer
30 134636 120127 000020          CMPB  R1,#20         ;is buffer half full?
31 134642 103435                BLD   1$              ;no, return
32 134644 132737 000002 007673  BITB  #F.STOP,@#HSTFLG;have we already sent ^S to host?
33 134652 001031                BNE   1$              ;yes, return
34 134654 106466 000006          MTPS  6(SP)          ;restore priority in case another char comes in
35 134660 012700 000023          MOV   #023,R0        ;send ^S to host
36 134664 004767 176664          JSR  PC,CHR02
37 134670 152737 000002 007673  BISB  #F.STOP,@#HSTFLG;show we have done so
38 134676 000417                BR   1$
39
40 134700 113701 007711          3$:  MOVB  @#AUXFLG,R1    ;set saved copy of A#CREG
41 134704 052701 000020          BIS   #20,R1         ;add the error reset bit
42 134710 110137 177452          MOVB  R1,@#A#CREG    ;reset errors
43 134714 113737 007711 177452  MOVB  @#AUXFLG,@#A#CREG ;restore to normal
44 134722 012700 000007          2$:  MOV   #007,R0      ;beep the console on error
45 134726 106466 000006          MTPS  6(SP)          ;restore priority in case another char comes in
46 134732 004767 176602          JSR  PC,CHR01
47 134736 012601          1$:  MOV   (SP)+,R1      ;restore registers
48 134740 012600          MOV   (SP)+,R0
49 134742 000002          RTI
50

```

```

1
2          ;++
3          ; HGETCH sets a character from the host input buffer and shifts the buffer down
4          ; one character.  If the buffer is reduced from 16 or more characters to just
5          ; four, a ^Q is sent to the host.  Do not call this routine if no characters
6          ; are available in HOSTBF (i.e., HFP = 0).
7          ;
8          ; Uses:   HFP   Index to next available location in HOSTBF
9          ;         F.STOP Status flag =1 if a ^S has been sent to the host
10         ;
11         ; Returns:
12         ;         R1   Lower byte contains the first character in HOSTBF, upper byte
13         ;         contains sign extension of bit 7
14         ;         HFP   Decrement to index new next available location in HOSTBF
15         ;         F.STOP Cleared if ^Q was sent
16         ;         R0   Contains garbage
17         ;--
18
19 134744 005001          HGETCH::CLR   R1
20 134746 113746 007632          MOVB   @#HOSTBF,-(SP) ;save the first character in buffer
21 134752 005201          1$:   INC    R1
22 134754 120137 007672          CNPB   R1,@#HFP      ;are we too close for comfort?
23 134760 103004          BHIS   3$      ;yes, disable interrupts before doing this one
24 134762 116161 007632 007631          MOVB   HOSTBF(R1),HOSTBF-1(R1) ;shift down one character
25 134770 000770          BR     1$      ;do entire buffer up to one less than HFP
26
27 134772 106746          3$:   MFPS   -(SP)      ;save priority
28 134774 106427 000340          MTPS   #340      ;disable interrupts to prevent screwups
29 135000 120137 007672          4$:   CNPB   R1,@#HFP      ;are we really done?
30 135004 103005          BHIS   5$      ;yes, all characters transferred
31 135006 116161 007632 007631          MOVB   HOSTBF(R1),HOSTBF-1(R1) ;shift down one character
32 135014 005201          INC    R1
33 135016 000770          BR     4$
34
35 135020 105337 007672          5$:   DECB   @#HFP      ;adjust buffer input pointer
36 135024 005000          CLR    R0
37 135026 123727 007672 000004          CNPB   @#HFP,#4      ;have we reduced to just four characters?
38 135034 101006          BHI    2$      ;no
39 135036 132737 000002 007673          BITB   #F.STOP,@#HSTFLG ;and was ^S sent?
40 135044 001402          BEQ   2$      ;no
41 135046 012700 000021          MOV    #021,R0      ;put ^Q in R0
42 135052 106426          2$:   MTPS   (SP)+      ;restore priority
43 135054 112601          MOVB   (SP)+,R1      ;recover character
44 135056 005700          TST   R0            ;did we decide to send ^Q?
45 135060 001405          BEQ   6$      ;no
46 135062 004767 176466          JSR   PC,CHRO2      ;yes, send ^Q to host
47 135066 142737 000002 007673          BICB   #F.STOP,@#HSTFLG ;show we have done so
48
49 135074 000207          6$:   RTS    PC

```

```

1
2          .SBTTL I/O Initialization and LED Driver
3
4          ;++
5          ; IOINIT asserts the RESET L line on the board, thereby clearing the
6          ; configuration of the peripheral chips. It then initializes the peripheral
7          ; chips to an idle state. SETAX2 configures the aux. port to operate with
8          ; a 16X clock, so that its baud rate will be equal to that of the console port.
9          ; The RS-232 control output signals are left deasserted, and the aux. port
10         ; receiver and transmitter are left disabled. SETAX1 is like SETAX2, except
11         ; that the clock divisor of the aux. port is set to 16 if R0 = 0 upon entry,
12         ; or 64 if R0 = 1. A 64X clock means that the aux. port baud rate will be
13         ; one fourth of the console port baud rate.
14         ;--
15
16 135076 000005          IOINIT::RESET          ;disable interrupts
17 135100 112737 000212 177446          MOVB    #212,@#P#CREG    ;configure 8255 with port B as input
18 135106 112737 000010 177444          MOVB    #010,@#P#PORC    ;disable LED display
19 135114 105037 177560          CLR    @#C#RCSR        ;disable DLART receiver
20 135120 105037 177564          CLR    @#C#XCSR        ;disable DLART transmitter
21 135124 005000          SETAX2: CLR    R0          ;indicate 16X clock for aux port
22 135126 112737 000200 177452 SETAX1: MOVB    #200,@#A#CREG    ;send a harmless mode word to make sure next
23                                     ;word is interpreted as a control word so that
24 135134 112737 000100 177452          MOVB    #100,@#A#CREG    ;this control word puts us back to mode load
25 135142 152737 000002 007464          R1SB    #2,@#MFB251      ;make sure baud rates will match
26 135150 063700 007464          AND    @#MFB251,R0      ;add in mode byte incase user has changed anything
27 135154 110037 177452          MOVB    R0,@#A#CREG     ;now initialize modes
28 135160 112737 000020 177452          MOVB    #020,@#A#CREG    ;reset error flag, disable 8251
29 135166 000207          RTS    PC
30
31
32         ;++
33         ; SETAUX reconfigures the aux. port to operate with a 16X clock if bit 7 of
34         ; CONFLG = 0, or with a 64X clock if bit 7 of CONFLG = 1. The aux. port
35         ; receiver and transmitter are left disabled, but the RS-232 control signals
36         ; request-to-send and data-terminal-ready are asserted. A copy of the command
37         ; byte sent to the aux. port is saved in AUXFLG for future reference, since
38         ; A#CREG is a write-only location.
39         ;--
40
41 135170 005000          SETAUX::CLR    R0
42 135172 105737 007710          TSTB    @#CONFLG        ;is 64X clock currently selected?
43 135176 100001          BPL    1$              ;no, 16X clock
44 135200 005200          INC    R0
45 135202 004767 177720          1$: JSR    PC,SETAX1     ;set aux port baud rate, disable
46 135206 112737 000042 177452          MOVB    #042,@#A#CREG    ;command 8251 to assert RTS, DTR, and disable
47 135214 112737 000042 007711          MOVB    #042,@#AUXFLG    ;save a copy since A#CREG is write only
48 135222 000207          RTS    PC
49

```

```

1
2           ;++
3           ; IOVECT initializes the interrupt vectors at locations 60, 64, 70, 74, ...,
4           ; 130, and 134 to point to an RTI instruction with priority 7. It then attaches
5           ; interrupt service routines for console break, invalid instruction, breakpoint,
6           ; and LED display refresh. The power fail vector is also attached to an RTI
7           ; instruction, with priority 6. The LED display hardware and both serial ports
8           ; are disabled.
9           ;--
10
11 IOVECT::
12 135224 012700 000060      MOV     #060,R0      ;initialize I/O vectors and disable interrupts
13 135230 012701 000014      MOV     #12,R1      ;15 coded priority vectors (2 not used)
14 135234 012720 135676      16: MOV     ##NULLI,(R0)+ ;set address element to null service routine
15 135240 012720 000340      MOV     #340,(R0)+ ;set priority level to 7
16 135244 077105             SOR     R1,#        ;
17 135246 012720 137114      MOV     #BREAK,(R0)+ ;connect console break routine
18 135252 012720 000340      MOV     #340,(R0)+ ;priority 7 for console break
19 135256 012700 000010      MOV     #010,R0
20 135262 012720 137334      MOV     #ITRAP,(R0)+ ;connect invalid instruction trap
21 135266 012720 000340      MOV     #340,(R0)+
22 135272 012720 137174      MOV     #BRKPT,(R0)+ ;connect breakpoint trap
23 135276 012720 000340      MOV     #340,(R0)+
24 135302 012737 135344 000104  MOV     #DISPLA,@#LEDIV ;connect LED display routine
25 135310 005037 007676      CLR     @#KEYFIV     ;disconnect keypress service vector
26                               ;now disable interrupt hardware for ports
27 135314 112737 000010 177444  MOVB    #010,@#P#PDR   ;disable LED display
28 135322 142737 000100 177560  BICB    #100,@#C#RCSR  ;disable console receiver
29 135330 142737 000100 177564  BICB    #100,@#C#XCSR  ;disable console transmitter
30 135336 105037 177452      CLRB    @#A#CREG      ;disable auxiliary receiver and transmitter
31 135342 000207             RTS     PC
32

```



```

1
2          .ENABL LSB
3          ;++
4          ; DISPLA is the LED refresh interrupt service routine. It multiplexes the
5          ; contents of SEGBUF to the LEDs through parallel port A, and checks if a
6          ; keypad key has been pressed. If a key has been pressed and the value in
7          ; location KEYPIV is not zero, then control will be transferred to the routine
8          ; at [KEYPIV] for interrupt driven keypad input. The keypress routine at
9          ; [KEYPIV] must restore R2, R1, and R0 from the stack (in that order) before
10         ; executing an RTI. Note that if the contents of SPCLBF are nonzero, a flashing
11         ; display of ' xx ' will appear in the LEDs instead of the contents of SEGBUF.
12         ;      Error'
13         ; DISPLA also counts the number of times it has been called (which is determined
14         ; by the hardware to be 800 times per second, when display refresh is enabled)
15         ; in the location TIME. The lower 10 bits of TIME count 0 through 799., and the
16         ; upper six bits count 0 through 59, seconds.
17         ;--
18
19 135344 005237 007730      DISPLA::INC    @#TIME      ;keep a count of 800ths of seconds in TIME
20 135350 010046              MOV     R0,-(SP)    ;preserve registers
21 135352 010146              MOV     R1,-(SP)
22 135354 010246              MOV     R2,-(SP)
23 135356 013700 007730      MOV     @#TIME,R0    ;prepare to check for overflow of 800ths
24 135362 005100              COM     R0
25 135364 032700 001440      BIT     #001440,R0
26 135370 001016              BNE     1$          ;no overflow
27 135372 042737 001777 007730  BIC     #001777,@#TIME ;overflow, so clear 800ths
28 135400 062737 002000 007730  ADD     #002000,@#TIME ;and add 1 second
29 135406 013700 007730      MOV     @#TIME,R0    ;check if overflow past 60 seconds
30 135412 005100              COM     R0
31 135414 032700 170000      BIT     #170000,R0
32 135420 001002              BNE     1$          ;no overflow of seconds
33 135422 005037 007730      CLR     @#TIME      ;yes, so clear the whole thing
34
35 135426 013700 007730      1$:    MOV     @#TIME,R0
36 135432 042700 177770      BIC     #177770,R0    ;strip R0 to make it a multiplex address
37 135436 006300              ASL     R0            ;make it a word index
38 135440 016001 007712      MOV     SEGBUF(R0),R1 ;set segment data for this digit pair
39 135444 005737 007726      TST     @#SPCLBF     ;is there a special error display?
40 135450 001422              BEQ     3$          ;no
41 135452 005001              CLR     R1
42 135454 032737 001000 007730  BIT     #001000,@#TIME ;true for approx. 1/3 of each second
43 135462 001015              BNE     3$          ;flashing error message, no less
44 135464 016001 135756      MOV     ERRSEG(R0),R1 ;set segment data for 'Error'
45 135470 132700 000004      BITB   #004,R0       ;is this digit 2 or 3?
46 135474 001410              BEQ     3$          ;no, so no overlay on upper display
47 135476 013702 007726      MOV     @#SPCLBF,R2  ;set overlay characters
48 135502 132700 000002      BITB   #002,R0       ;is this digit 3?
49 135506 001001              BNE     4$          ;digit 3 character is in upper byte
50 135510 000302              SWAB   R2            ;swap for digit 2
51 135512 105002              4$:    CLRB   R2          ;save only upper display character
52 135514 060201              ADD     R2,R1        ;combine upper and lower display characters
53 135516 112737 000010 177444  3$:    MOVB   #010,@#P$PORC ;turn off display, enable upper display latch
54 135524 000301              SWAB   R1            ;set upper display data in lower byte
55 135526 110137 177440      MOVB   R1,@#P$PORA  ;output upper display data
56 135532 000301              SWAB   R1            ;back to lower display data
57 135534 112737 000007 177444  MOVB   #007,@#P$PORC ;display still off, but upper latch closed

```

```

58 135542 110137 177440      MOVB  R1,@#P$PORA      ;output lower display data
59 135546 010001            MOV   R0,R1
60 135550 006201            ASR   R1                ;multiplex digit address (byte index)
61 135552 110137 177444      MOVB  R1,@#P$PORC      ;turn on display
62 135556 000240            NOP
63 135560 113702 177544      MOVB  @#P$PORD,R2      ;keypad return lines in bits 4 to 7
64 135564 042702 177417      BIC   #177417,R2      ;clean up keypad data
65 135570 060002            ADD   R0,R2            ;combine index and return data
66 135572 013701 007704      MOV   @#DEBNCE,R1      ;set a scratch copy of debounce state register
67 135576 001413            BEQ   5$                ;not debouncing a key, take a new one
68 135600 074201            XOR   R2,R1
69 135602 001420            BEQ   6$                ;total match, take it as a keypress
70 135604 032701 000016      BIT   #000016,R1      ;compare multiplex address field
71 135610 001027            BNE   2$                ;not the same, so ignore it
72 135612 032701 000360      BIT   #000360,R1      ;compare return line field
73 135616 001424            BEQ   2$                ;same key held, so ignore it
74 135620 005037 007704      CLR   @#DEBNCE        ;no longer same key pressed, so reset state
75 135624 000421            BR    2$
76
77 135626 004767 000046      5$: JSR   PC,7$            ;convert positional bit to number
78 135632 005701            TST   R1                ;is this a valid key depression?
79 135634 100415            BMI   2$                ;no, so ignore it
80 135636 010237 007704      MOV   R2,@#DEBNCE      ;yes, take it and wait for confirmation
81 135642 000412            BR    2$
82
83 135644 052737 100000 007704 6$: BIS   #100000,@#DEBNCE ;make sure we get only one keypress per key
84 135652 005737 007676      TST   @#KEYPIV         ;do we have an active vector?
85 135656 001404            BEQ   2$                ;no, discard keypress
86 135660 004767 000014      JSR   PC,7$            ;set a useful key number in R1
87 135664 000177 052006      JMP   @#KEYPIV         ;pass it to attached service routine
88
89 135670 012602            2$: MOV   (SP)+,R2        ;service routine at @#KEYPIV should also restore
90 135672 012601            MOV   (SP)+,R1        ;the registers in this manner
91 135674 012600            MOV   (SP)+,R0
92 135676 000002            #NULLI::RTI
93
94 135700 010200            7$: MOV   R2,R0          ;make a scratch copy
95 135702 000300            SWAB  R0                ;move key data to upper byte
96 135704 042700 007777      BIC   #007777,R0      ;keep only return line data
97 135710 012701 000004      MOV   #4,R1            ;shift out 4 bits max
98 135714 006300            8$: ASL   R0                ;shift out a bit, look for a one
99 135716 103013            BCC   9$                ;didn't find it
100 135720 001013           BNE   10$               ;more than one true bit--invalid data
101 135722 005301            DEC   R1                ;R1 is now 3, 2, 1, or 0
102 135724 006302            ASL   R2                ;least two bits are 0
103 135726 060201            ADD   R2,R1            ;combine multiplex address and positional count
104 135730 042701 177740      BIC   #177740,R1      ;clean up key code
105 135734 006202            ASR   R2                ;restore R2
106 135736 010200            11$: MOV  R2,R0           ;restore R0
107 135740 042700 177761      BIC   #177761,R0      ;restore R0
108 135744 000207            RTS   PC
109
110 135746 077116           9$: SDB  R1,8$           ;keep searching
111 135750 012701 177777      10$: MOV  #177777,R1    ;indicate invalid data
112 135754 000770           BR    11$              ;restore and exit
113
114 .DSABL LSE

```

```

115 135756 000120 000134 000120 ERRSEG: .WORD 120,134,120 ;r, o, r
116 135764 000120 000171 000000 .WORD 120,171,000 ;r, E, space
  
```

```

1
2          .SRTTL LED Display Support Routines
3
4          ;++
5          ; DSPDIG takes a hex number in the lower 4 bits of R0 and displays the
6          ; corresponding segment pattern on the LED digit indexed by R1. R1 = 0
7          ; to index the rightmost lower LED digit, = 1 to index the rightmost upper
8          ; LED digit, ..., = 13 to index the leftmost upper LED digit. Returns R1
9          ; incremented by 2 to index the next digit to the left on the same display
10         ; row.
11         ;
12         ; Uses:   R0      Hex value to be displayed
13         ;         R1      Index to LED digit on which to display it
14         ;
15         ; Returns:
16         ;         R0      Previous value with bits 15, through 4 cleared
17         ;         R1      Previous value + 2
18         ;--
19
20 135772 042700 177760          DSPDIG::BIC #177760,R0 ;range-limit R0
21 135776 020127 000013          CMP R1,#000013 ;maximum value for digit index
22 136002 101004          BHI 1$ ;out-of-range, so ignore call
23 136004 116061 136016 007712  MOVB HEXSEG(R0),SEGBUF(R1) ;display digit
24 136012 005721          TST (R1)+ ;index next digit of same display level
25 136014 000207          1$: RTS PC
26
27 136016 077 006 133 HEXSEG::,BYTE 077,006,133,117 ;0, 1, 2, 3
28 136021 117
29 136022 146 155 175 .,BYTE 146,155,175,007 ;4, 5, 6, 7
30 136025 007
31 136026 177 157 167 .,BYTE 177,157,167,174 ;8, 9, A, b
32 136031 174
33 136032 071 136 171 .,BYTE 071,136,171,161 ;C, d, E, F
34 136035 161
35
36 .EVEN
  
```

```

1
2
3           ;++
4           ; DSPNUM and DSPLNM take full words in R2 and write octal numbers to the upper
5           ; and lower LED displays, respectively.
6           ;
7           ; Uses:   R2      Octal value to be displayed
8           ;
9           ; Returns:
10          ;         R0      Contains garbage
11          ;         R1      Contains garbage
12          ;--
13 136036 012701 000001  DSPNUM::MOV  #1,R1      ;index upper display
14 136042 000402          BR          DSPNUM
15
16 136044 012701 000000  DSPLNM::MOV  #0,R1      ;index lower display
17
18 136050 010246          DSPNUM: MOV  R2,-(SP)    ;preserve R2
19 136052 010346          MOV  R3,-(SP)    ;set a scratch register
20 136054 012703 000006          MOV  #6,R3      ;count 6 digits
21 136060 010200          1%:  MOV  R2,R0      ;move octal digit to R0
22 136062 042700 177770          BIC  #177770,R0  ;make sure it's octal
23 136066 004767 177700          JSR  PC,DSPDIG   ;display digit
24 136072 000241          CLC
25 136074 006002          ROR  R2      ;move to next octal digit
26 136076 006202          ASR  R2
27 136100 006202          ASR  R2
28 136102 077312          SOB  R3,1%     ;do five digits
29 136104 012603          MOV  (SP)+,R3
30 136106 012602          MOV  (SP)+,R2
31 136110 000207          RTS  PC
32
33
34

```

```

1          ;++
2          ; DSPNAM is a routine to display names of registers, etc. on upper display.
3          ; R2 points to the message to be displayed. The message should be a string
4          ; of 6 bytes corresponding to the segment patterns for each LED digit, starting
5          ; with the rightmost digit.
6          ;
7          ; Uses:      R2      Pointer to string of segment patterns
8          ;
9          ; Returns:
10         ;           R1      Contains garbage
11         ;           R0      Contains garbage
12         ;--
13
14 136112 012701 000001      DSPNAM::MOV      #1,R1          ;index upper display
15 136116 010246           MOV      R2,-(SP)      ;preserve message index
16 136120 006302           ASL      R2          ;multiply it by 6
17 136122 010200           MOV      R2,R0
18 136124 006302           ASL      R2
19 136126 060002           ADD      R0,R2
20 136130 012700 000006     MOV      #6,R0          ;character count
21 136134 116261 136154 007712 1$: MOVBR  NAMTBL(R2),SEGBUF(R1) ;move segment data to LED buffer
22 136142 005721           TST      (R1)+         ;index next LED digit
23 136144 005202           INC      R2          ;index next character
24 136146 077006           SOB      R0,1$        ;do six characters
25 136150 012602           MOV      (SP)+,R2     ;restore R2
26 136152 000207           RTS      PC
27
28 136154 000 000 077 NAMTBL::.BYTE 0,0,077,120,0,0 ;' r0 '
   136157 120 000 000
29 136162 000 000 006 .BYTE 0,0,006,120,0,0 ;' r1 '
   136165 120 000 000
30 136170 000 000 133 .BYTE 0,0,133,120,0,0 ;' r2 '
   136173 120 000 000
31 136176 000 000 117 .BYTE 0,0,117,120,0,0 ;' r3 '
   136201 120 000 000
32 136204 000 000 146 .BYTE 0,0,146,120,0,0 ;' r4 '
   136207 120 000 000
33 136212 000 000 155 .BYTE 0,0,155,120,0,0 ;' r5 '
   136215 120 000 000
34 136220 000 000 163 .BYTE 0,0,163,155,0,0 ;' SF '
   136223 155 000 000
35 136226 000 000 071 .BYTE 0,0,071,163,0,0 ;' PC '
   136231 163 000 000
36 136234 000 000 155 .BYTE 0,0,155,163,0,0 ;' FS '
   136237 163 000 000
37 136242 120 136 167 .BYTE 120,136,167
38 136245 170 124 163 .BYTE 170,124,163 ;'FntAdr'
39 136250 000 120 136 .BYTE 000,120,136
40 136253 136 167 000 .BYTE 136,167,000 ;' Addr '
41 136256 000 075 171 .BYTE 000,075,171
42 136261 120 000 000 .BYTE 120,000,000 ;' rEG '
43 136264 006 170 124 .BYTE 006,170,124
44 136267 163 120 174 .BYTE 163,120,174 ;'brPnt1'
45 136272 133 170 124 .BYTE 133,170,124
46 136275 163 120 174 .BYTE 163,120,174 ;'brPnt2'
47 136300 117 170 124 .BYTE 117,170,124
48 136303 163 120 174 .BYTE 163,120,174 ;'brPnt3'

```

49	136306	146	170	124	.BYTE	146,170,124	
50	136311	163	120	174	.BYTE	163,120,174	;'brFnt4'
51	136314	000	130	124	.BYTE	000,130,124	
52	136317	034	161	000	.BYTE	034,161,000	;' Func '
53	136322	060	134	155	.BYTE	060,134,155	
54	136325	124	134	071	.BYTE	124,134,071	;'ConSol'
55	136330	170	120	134	.BYTE	170,120,134	
56	136333	163	200	167	.BYTE	163,200,167	;'A,Port'
57					.EVEN		

```

1          ;++
2          ;   BLNKLO and BLNKHI are routines to erase the segment data in the lower and
3          ;   upper LED buffers, respectively. Returns garbage in R0 and R1.
4          ;--
5
6 136336 012701 000000   BLNKLO::MOV   #0,R1       ;index lower display
7 136342 000402          BR       BLNKDS
8
9 136344 012701 000001   BLNKHI::MOV   #1,R1       ;index upper display
10
11 136350 012700 000006   BLNKDS: MOV   #6,R0       ;count 6 digits
12 136354 105061 007712 1$:   CLRB   SEGBUF(R1)     ;blank a digit
13 136360 005721          TST   (R1)+             ;index next digit of same display level
14 136362 077004          SOB   R0,1$           ;do six digits
15 136364 000207          RTS   PC
16
17
18          ;++
19          ;   DSPDEC takes a number in R1 and an index to either second-from-the-left LED
20          ;   digit in R2 and displays the number as a decimal value on the correspondins
21          ;   LED digits. R2 = 10 to index the lower row of LEDs, = 11 for the upper row.
22          ;   The leftmost LED digits are not affected. To display a two's complement
23          ;   decimal value, call this routine from another routine, which places a minus
24          ;   sign in the leftmost LED digit and passes the absolute value of the number
25          ;   to be displayed to this routine.
26          ;
27          ;   Uses:   R1   Decimal value to be displayed
28          ;           R2   Index to LED row on which to display it
29          ;
30          ;   Returns:
31          ;           R0, R1, Contain garbage
32          ;           R2, R3
33          ;--
34
35 136366 004767 015662   DSPDEC::JSR   PC,BINAS1   ;set 10000's digit
36 136372 010146          MOV   R1,-(SP)          ;save remainder
37 136374 010201          MOV   R2,R1            ;index left LED digit
38 136376 004767 177370   JSR   PC,DSPDIG         ;display decimal digit
39 136402 005742          TST   -(R2)            ;index next digit
40 136404 012601          MOV   (SP)+,R1         ;
41 136406 012703 000004   MOV   #4,R3            ;make a counter
42 136412 004767 015576   1$:   JSR   PC,BINASC     ;strip a digit from R1
43 136416 010146          MOV   R1,-(SP)          ;switch registers
44 136420 010201          MOV   R2,R1            ;
45 136422 004767 177344   JSR   PC,DSPDIG         ;display decimal digit
46 136426 005742          TST   -(R2)            ;index next digit to the right
47 136430 012601          MOV   (SP)+,R1         ;set back decimal number
48 136432 077311          SOB   R3,1$           ;do six digits
49 136434 000207          RTS   PC
50

```

```

1
2
3          ;++
4          ; KEYGET turns on the LEDs and waits for a keypress. The key number of the
5          ; key that was pressed is returned in R1. The keys are numbered as follows:
6          ; 00 -- [CLR]      12 -- [ 6 ]
7          ; 01 -- [ 1 ]      13 -- [ 9 ]
8          ; 02 -- [ 4 ]      14 -- [ADV]
9          ; 03 -- [ 7 ]      15 -- [BAC]
10         ; 04 -- [ 0 ]      16 -- [REG]
11         ; 05 -- [ 2 ]      17 -- [ADR]
12         ; 06 -- [ 5 ]      20 -- [SST]
13         ; 07 -- [ 8 ]      21 -- [GO ]
14         ; 10 -- [EXA]      22 -- [BPT]
15         ; 11 -- [ 3 ]      23 -- [FNC]
16         ;
17         ; Uses:
18         ;
19         ; Returns:
20         ; R1      Contains number of key that was pressed
21         ; KEYPIV  Cleared
22         ; SPCLBF  Cleared
23         ; PS      Priority reduced to 5
24         ;--
25 136436 012737 136470 007676 KEYGET::MOV  #KEYGE1,@#KEYPIV;link input routine to keypress vector
26 136444 112737 000007 177444      MOVB  #007,@#PDRC ;turn on LED displays
27 136452 106701                MFPB  R1
28 136454 120127 000277          CNPB  R1,#277 ;priority 5 is maximum for LEDs to operate
29 136460 101402                BLDS  1$
30 136462 106427 000240          MTPS  #240 ;set priority 5
31 136466 000777                1$: BR  1$ ;wait for keypress
32
33 136470 005037 007676          KEYGE1: CLR @#KEYPIV ;disconnect keypress vector
34 136474 005037 007726          CLR  @#SPCLBF ;turn off flashing error, if any
35 136500 012602                MOV  (SP)+,R2 ;restore R2
36 136502 005726                TST  (SP)+ ;discard old R1
37 136504 012600                MOV  (SP)+,R0 ;restore R0
38 136506 005726                TST  (SP)+ ;discard return address
39 136510 106426                MTPS (SP)+ ;but replace PS byte
40 136512 000207                RTS  PC ;return to caller of KEYGET
41

```



```

1
2      ;++
3      ; KEYBCD translates the key number in R1 into a BCD value for the keys 0 to 9,
4      ; 12 for the CLR key, 013 for the EXA key, and -1 for all other keys.
5      ;
6      ; Uses:      R1      Key number to be translated
7      ;
8      ; Returns:
9      ;           R1      Translated value
10     ;--
11
12     136514 116101 136522      KEYBCD::MOVB  BCDTBL(R1),R1 ;translate R1 into BCD code
13     136520 000207          RTS      PC
14
15     136522 012          BCDTBL: .BYTE 012          ;[CLR]
16     136523 001          .BYTE 001          ;[ 1 ]
17     136524 004          .BYTE 004          ;[ 4 ]
18     136525 007          .BYTE 007          ;[ 7 ]
19     136526 000          .BYTE 000          ;[ 0 ]
20     136527 002          .BYTE 002          ;[ 2 ]
21     136530 005          .BYTE 005          ;[ 5 ]
22     136531 010          .BYTE 010          ;[ 8 ]
23     136532 013          .BYTE 013          ;[EXA]
24     136533 003          .BYTE 003          ;[ 3 ]
25     136534 006          .BYTE 006          ;[ 6 ]
26     136535 011          .BYTE 011          ;[ 9 ]
27     136536 377          .BYTE 377          ;[ADV]
28     136537 377          .BYTE 377          ;[BAC]
29     136540 377          .BYTE 377          ;[REG]
30     136541 377          .BYTE 377          ;[ADD]
31     136542 377          .BYTE 377          ;[SST]
32     136543 377          .BYTE 377          ;[GO ]
33     136544 377          .BYTE 377          ;[BPT]
34     136545 377          .BYTE 377          ;[FNC]
35     .EVEN
36

```

```

1
2          ;++
3          ;   NUMGET displays the current value of R2 in the lower row of LEDs, leaving the
4          ;   upper row intact. The keypad is enabled to allow the user to modify the value
5          ;   in R2 with the 0 - 7 and CLR keys. The value in R2 is returned, whether
6          ;   altered or not, when the user presses EXA.
7          ;
8          ;   Uses:   R2      Initial value
9          ;
10         ;   Returns:
11         ;           R2      New value (if changed)
12         ;           F.CHAN  Flag in FLAGS1 is set if R2 was changed, cleared otherwise
13         ;           R1      Contains garbage
14         ;           R0      Contains garbage
15         ;--
16
17 136546 042737 000002 007706 NUMGET::BIC  #F.CHAN,@#FLAGS1;show R2 hasn't changed yet
18 136554 004767 177264          1%:   JSR  PC,DSPLNM      ;display value in R2 on lower LEDs
19 136560 004767 177652          JSR  PC,KEYGET      ;wait for a keypress
20 136564 004767 177724          JSR  PC,KEYBCD      ;convert key number into BCD
21 136570 005701          TST  R1              ;was it a legal key?
22 136572 100770          BMI  1%              ;no, set another
23 136574 020127 000012          CMP  R1,#012        ;was it numeric?
24 136600 101026          BHI  2%              ;no, it was EXA/ENT, so return value in R2
25 136602 001420          BEQ  3%              ;no, it was clear
26 136604 020127 000010          CMP  R1,#010        ;was it 8 or 9 (not valid octal digits)?
27 136610 103361          BHIS 1%              ;yes, so ignore it
28 136612 032737 000002 007706  BIT  #F.CHAN,@#FLAGS1;has R2 been changed yet?
29 136620 001004          BNE  4%              ;yes, just scroll in key
30 136622 005002          CLR  R2              ;else clear old value first
31 136624 052737 000002 007706  BIS  #F.CHAN,@#FLAGS1;indicate it has been changed
32 136632 006302          4%:   ASL  R2              ;make space for new digit
33 136634 006302          ASL  R2
34 136636 006302          ASL  R2
35 136640 060102          ADD  R1,R2          ;scroll in new digit
36 136642 000744          BR   1%              ;accept additional digits
37
38 136644 005002          3%:   CLR  R2              ;clear number for CLR key
39 136646 052737 000002 007706  BIS  #F.CHAN,@#FLAGS1;show R2 changed
40 136654 000737          BR   1%
41
42 136656 000207          2%:   RTS  PC              ;return value

```

```

1
2               .SBTTL Error Condition and Startup Entry Code
3
4               ;;;
5               ; $START is the powerup initialization routine, which is executed after the
6               ; diagnostics are completed.
7               ;--
8
9 136660 005000      $START::CLR   R0           iclear RAM from 0 to 27777
10 136662 012701 014000      MOV     #14000,R1
11 136666 005020      ;::      CLR     (R0)+
12 136670 077102      SDB     R1,1$
13 136672 010537 007726      MOV     R5,@#SPCLBF      ;put power-up error code on LEDs
14 136676 012737 007400 007746      MOV     #SCRPAD,@#SP      ;set up user stack pointer
15 136704 112737 000050 007710      MOV     #050,@#CNFLG      ;set default baud rates
16 136712 112767 000116 050544      MOV     #116,#B251      ;set default value for B251 mode byte
17 136720 012737 135676 007702      MOV     ##NULLI,@#USERVS;initialize user's interrupt vector
18 136726 012737 135676 000024      MOV     ##NULLI,@#24      ;if points at null
19 136734 012737 000340 000026      MOV     #340,@#26      ;
20 136742 012737 007320 007462      MOV     #TBLTOP,@#TBLBOT;initialize empty symbol table
21 136750 000402      BR      $HALT+4      ;don't clear error display
22
23               ;;;
24               ; $HALT is the routine invoked by the HALT button, the HALT L line on the
25               ; board, and the HALT instruction. It saves the processor registers and the
26               ; contents of USERIV and KEYPIV if a user program was running. It also
27               ; performs a bus reset and reinitialization of peripheral chips before
28               ; invoking the keypad monitor.
29               ;--
30
31 136752 005037 007726      $HALT::CLR   @#SPCLBF      iclear error code on LEDs, if any
32 136756 004767 000022      JSR     PC,UCSAVE      isave context and fix user SP
33 136762 004767 176110      JSR     PC,IOINIT      ;assert RESET line and reinitialize ports
34 136766 042737 177377 007706      BIC     #177377,@#FLAGS;reset mode, set keypad command entry
35                                   ;leave F.BRKA so breakpoints will be removed
36 136774 004767 176224      JSR     PC,IOVECT      imake sure interrupt vectors aren't damaged
37 137000 000167 000564      JMP     $MONIT        ;enter monitor
38

```

```

1
2
3      ;++
4      ; UCSAVE saves the processor registers and the contents of USERIV and KEYPIV
5      ; if a user program was running. It also initializes the monitor stack.
6      ;--
7 137004 012637 007460 UCSAVE::MOV (SP)+,@#STACK-2 ;move return address to monitor stack
8 137010 032737 000100 007706 BIT #F,USER,@#FLAGS ;running user code?
9 137016 001433 BED 1# ;no
10 137020 012637 007750 MOV (SP)+,@#PC ;yes, save user context, starting with PC
11 137024 112637 007752 MOV# (SP)+,@#PS ;PS next
12 137030 042737 177400 007752 BIC #177400,@#PS ;prevent sign extension
13 137036 010637 007746 MOV SP,@#SP ;fix and save stack pointer
14 137042 010037 007732 MOV R0,@#R0 ;save R0,
15 137046 010137 007734 MOV R1,@#R1 ;R1,
16 137052 010237 007736 MOV R2,@#R2 ;R2,
17 137056 010337 007740 MOV R3,@#R3 ;R3,
18 137062 010437 007742 MOV R4,@#R4 ;R4, and
19 137066 010537 007744 MOV R5,@#R5 ;R5
20 137072 013737 007676 007700 MOV @#KEYPIV,@#KEYPVS ;save user keypress vector
21 137100 042737 000100 007706 BIC #F,USER,@#FLAGS ;show that user context has been saved
22 137106 012706 007460 1#; MOV #STACK-2,SP ;so to monitor stack
23 137112 000207 RTS PC ;and return to calling routine in monitor
24
25
26      ;++
27      ; BREAK is the console break interrupt service routine. It is similar to HALT
28      ; except that it does not invoke the keypad monitor if the console monitor was
29      ; running previously. The host mode is terminated by BREAK.
30      ;--
31
32 137114 005737 177562 BREAK::TST @#C#RBUF ;acknowledge interrupt
33 137120 004767 177660 JSR PC,UCSAVE ;save user context
34 137124 004767 175746 JSR PC,IOINIT ;assert RESET line and reinitialize ports
35 137130 042737 035377 007706 BIC #035377,@#FLAGS ;initialize monitor modes
36 137136 012737 133000 000060 MOV #LINEIN,@#CONIN ;hook up console input service routine
37 137144 142737 000007 007710 BICB #007,@#CONFLG ;clear console line flags (but not speed)
38 137152 113700 007710 MOV# @#CONFLG,R0 ;set current baud rate
39 137156 052700 000002 BIS #000002,R0 ;enable prog. baud rate, disable xmit interrupt
40 137162 010037 177564 MOV R0,@#C#XCSR ;transfer control word to console port
41 137166 012700 000001 MOV #1,R0 ;indicate console break service
42 137172 000466 BR FERROR ;so to fatal error handler to reenter monitor
43

```

```

1
2
3      ; BRKPT is the breakpoint service routine.  If invoked from a user program, the
4      ; processor context is saved.  If invoked in response to a monitor breakpoint
5      ; rather than a single step or user breakpoint, the PC value saved from the
6      ; previous context is adjusted to point to the location which contained the
7      ; monitor breakpoint.
8      ;--
9
10     137174  106427  000340      BRKPT:: MTPS  #340      ;disable interrupts in case vector is damaged
11     137200  004767  177600      JSR    PC,UCSAVE  ;save user context
12     137204  032737  001000  007706      BIT    #F,BRKG,#FLAGS ;are we single stepping through a break
13                                     ;point
14     137212  001405      BEQ    2#        ;no, so check if single step or new breakpoint
15     137214  042737  001000  007706      BIC    #F,BRKG,#FLAGS ;reset condition
16     137222  000167  002466      JMP    $G0      ;install breakpoints and resume execution
17
18     137226  032737  000200  007706  2# :  BIT    #F,SST,#FLAGS1 ;are we single stepping?
19     137234  001155      BNE    $MONIT   ;yes, so not a breakpoint--enter monitor
20     137236  032737  000400  007706      BIT    #F,BRKA,#FLAGS ;are breakpoints installed?
21     137244  001430      BEQ    5#        ;no--user program contains BRK, so signal error
22     137246  013701  007750      MOV    @#PC,R1  ;get user PC
23     137252  005741      TST    -(R1)    ;back up to BRK instruction
24     137254  012700  007760      MOV    #BRKFIL,R0 ;index breakpoint file
25     137260  012702  000004      MOV    #4,R2    ;four entries in file
26     137264  022001      4# :  CMP    (R0)+,R1 ;did we in fact encounter a breakpoint?
27     137266  001403      BEQ    3#        ;yes, so adjust PC, no error message
28     137270  005720      TST    (R0)+    ;advance to next entry
29     137272  077204      SOB    R2,4#
30     137274  000414      BR    5#        ;no match--program contains BRK,..signal error
31
32     137276  052737  020000  007706  3# :  BIS    #F,TBAS,#FLAGS ;show we need to print break point message
33     137304  010137  007750      MOV    R1,@#PC  ;fix PC
34     137310  052737  001000  007706      BIS    #F,BRKG,#FLAGS ;show we need to single step
35     137316  032737  002000  007706      BIT    #F,KEYF,#FLAGS ;no message if in keypad mode
36     137324  001521      BEQ    $MONIT   ;so just enter monitor
37     137326  012700  000002  5# :  MOV    #2,R0    ;indicate breakpoint service
38     137332  000406      BR    FERROR   ;go to fatal error handler to reenter monitor
39
40
41     ;--
42     ; ITRAP is the illegal instruction trap routine.  If a user program was running,
43     ; the processor context is saved before invoking the monitor.
44     ;--
45
46     137334  106427  000340      ITRAP:: MTPS  #340      ;disable interrupts in case vector is damaged
47     137340  004767  177440      JSR    PC,UCSAVE  ;save user context
48     137344  012700  000003      MOV    #3,R0    ;indicate invalid instruction trap
49

```

Error Condition and Startup Entry Code

```

1
2
3
4
5
6
7
8 137350 010046          FERROR::MOV    R0,-(SP)      ;save error number
9 137352 004767 175646          JSR    PC,IOVECT          ;make sure interrupt vectors aren't damaged
10 137356 012600          MOV    (SP),R0           ;set error number
11 137360 006300          ASL    R0                ;turn it into an index
12 137362 032737 002000 007706  BIT    %F.KEYP,%#FLAGS;are we in keypad mode?
13 137370 001411          BEQ    1$               ;yes, so no console message
14 137372 016002 137422          MOV    FMESAG-2(R0),R2   ;start address of selected message
15 137376 004767 174122          JSR    PC,CRLF           ;send <CR> <LF>
16 137402 004767 174360          JSR    PC,PRINT          ;print radix-50 message
17 137406 005037 007630          CLR    @#BP             ;make sure nothing left in buffer
18 137412 000466          2$: BR    %MONIT
19
20 137414 016037 137542 007726  1$: MOV    FERCHR(R0),@#SPCLBF ;put two character error code on LEDs
21 137422 000462          BR    %MONIT
22
23 137424 137434          FMESAG::WORD  FMESG1      ;pointers to error messages
24 137426 137450          .WORD  FMESG2
25 137430 137472          .WORD  FMESG3
26 137432 137522          .WORD  FMESG4
27
28 137434 012446 074444 017502  FMESG1: .RAD50 /CONSOLE BREAK/
   137442 070511 042300
29 137446 000000          .WORD  0
30 137450 007525 003770 063141  FMESG2: .RAD50 /BREAK POINT ENCOUNTERED/
   137456 055240 020563 060426
   137464 076732 017740
31 137470 000000          .WORD  0
32 137472 035206 004051 014411  FMESG3: .RAD50 /INVALID INSTRUCTION ENCOUNTERED/
   137500 055214 071713 077167
   137506 053605 054007 102604
   137514 021025 014400
33 137520 000000          .WORD  0
34 137522 035206 004051 014423  FMESG4: .RAD50 /INVALID STACK POINTER/
   137530 076453 042320 057466
   137536 076732
35 137540 000000          .WORD  0
36
37 137542 120 076          FERCHR::BYTE  120,076      ;r,u (Ur Error)
38 137544 174 071          .BYTE  174,071          ;b,c (Cb Error)
39 137546 163 174          .BYTE  163,174          ;p,b (bP Error)
40 137550 124 006          .BYTE  124,006          ;n,i (In Error)
41 137552 163 155          .BYTE  163,155          ;p,s (SP Error)
42 137554 006 174          .BYTE  006,174          ;i,b (bI Error)
43
44

```

```

1
2      ;++
3      ; MONIT1 is the monitor entry point to be used for normal exit from user
4      ; programs. The processor context is saved, except for the PC, which retains
5      ; its previous value so that the program can be run again. $MONIT is the
6      ; entry point used by monitor routines after the processor context has been
7      ; saved.
8      ;--
9
10     137556 106746      MONIT1::MFPS      -(SP)      ;push PS, PC like from a trap
11     137560 013746 007750      MOV      @##PC,-(SP) ;but keep previous user PC value
12     137564 004767 177214      JSR      PC,UCSAVE   ;save user context
13     137570 032737 000400 007706 $MONIT::BIT      #F,BRKA,@#FLAGS ;breakpoints currently installed?
14     137576 001426      BEQ      2$          ;no
15     137600 012700 000014      MOV      #014,R0    ;index last breakpoint address
16     137604 016001 007760      3$:     MOV      BRKFIL(R0),R1 ;set breakpoint address
17     137610 001404      BEQ      4$          ;this breakpoint is not defined
18     137612 005720      TST      (R0)+      ;point to previous contents
19     137614 016011 007760      MOV      BRKFIL(R0),(R1) ;restore contents of breakpoint
20     137620 005740      TST      -(R0)      ;go back to address
21     137622 024040      4$:     CMP      -(R0),-(R0) ;backup to next address
22     137624 005700      TST      R0
23     137626 100366      BPL      3$          ;do four entries
24     137630 013704 007750      MOV      @##PC,R4    ;point to next location
25     137634 032737 020000 007706      BIT      #F,TRAS,@#FLAGS ;see if we should print message
26     137642 001404      BEQ      2$          ;branch if not
27     137644 004767 015400      JSR      PC,TYPADR   ;type address
28     137650 004767 015314      JSR      PC,TYPL01   ;disassemble next instruction
29     137654 042737 030400 007706 2$:     BIC      #F,BRKA+F,APPL+F,TRAS,@#FLAGS ;indicate breakpoints removed
30                                     ;and enable break on ^C
31     137662 032737 002000 007706      BIT      #F,KEYP,@#FLAGS ;keypad mode?
32     137670 001402      BEQ      1$          ;yes, go to keypad monitor
33     137672 000167 003026      JMP      CMONIT      ;and go to console monitor
34
35     137676 000167 000276      1$:     JMP      KMONIT
36
37         000076      .REPT <140000-.>
38         .BYTE 0
39         .ENDR
    
```

```

1
2          .SBTTL  Entry Points
3
4          ;++
5          ;  MONITOR ENTRY POINTS:
6          ;--
7
8 140000 000167 015330  START::JMP  DIAGNO      ;do diagnostics, then initialize monitor
9
10 140004 000167 176742  RSTART::JMP  $HALT      ;HALT instructions and halt line vector
11                          ;save context and restart monitor
12
13 140010 000167 177542  MONITR::JMP  MONIT1     ;standard monitor entry (no error condition)
14                          ;from user program, save context and remove
15                          ;breakpoints
16
17          ;++
18          ;  KEYPAD/LED UTILITY ENTRY POINTS:
19          ;--
20
21 140014 000167 175752  DGDISP::JMP  BSPDIG     ;Display number in R0 bits 3 to 0 as a hex
22                          ;digit at LED position indexed by R1.
23                          ;R1=12,10,06,04,02,00 -- lower display digits
24                          ;R1=13,11,07,05,03,01 -- upper display digits
25                          ;R0 bits 15 to 4 are cleared, 2 is added to R1
26
27 140020 000167 176012  UPDISP::JMP  DSPUMM     ;Display number in R2 as a 6 digit octal value
28                          ;in the upper row of LEDs. Returns 15 in R1
29                          ;and garbage in R0.
30
31 140024 000167 176014  LODISP::JMP  DSPLNM     ;Display number in R2 as a 6 digit octal value
32                          ;in the lower row of LEDs. Returns 14 in R1
33                          ;and garbage in R0.
34
35 140030 000167 176332  DECDSP::JMP  BSPDEC     ;Display number in R1 as a 5 digit decimal
36                          ;value on the row of LEDs indexed by R2.
37                          ;R2 = 010 for the lower row, 011 for the upper
38                          ;Returns zero in R1 and R3, and garbage in R0
39                          ;and R2.
40
41 140034 000167 176304  UPBLNK::JMP  BLNKHI     ;Blank the upper row of LEDs. Returns zero in
42                          ;R0 and garbage in R1.
43
44 140040 000167 176272  LOBLNK::JMP  BLNKLO     ;Blank the lower row of LEDs. Returns zero in
45                          ;R0 and garbage in R1.
46
47 140044 000167 176366  GETKEY::JMP  KEYGET     ;Turn on LEDs and wait for a keypress. Return
48                          ;key number in R1. Modifies KEYFIV.
49
50 140050 000167 176440  BCDKEY::JMP  KEYBCD     ;Translate key number in R1 to a BCD value for
51                          ;keys 0 to 9, 012 for the CLR key, 013 for the
52                          ;EX/ENTR key, and -1 for all other keys.
53
54 140054 000167 176466  GETNUM::JMP  NUMGET     ;Display number in R2 as an octal value on the
55                          ;lower row of LEDs. Activate keypad and allow
56                          ;user to modify the value in R2 with the keys
57                          ;0 through 7 and CLR. Return the value in R2,

```


58
59

changed or not, when user presses EXA. Uses
R2, R1, and R0.

Entry Points

```

1
2
3           ;++
4           ;  CONSOLE LINE UTILITY ENTRY POINTS:
5           ;--
6 140060 000167 000104           JMP   FIXBP       ;Prompt for a line from the console. Return
7                               ;the line in BUFFER with R3 cleared.
8
9 140064 000167 173346           JMP   GETBP       ;Transfer the back pointer (BP) to R3.
10
11 140070 000167 173222          JMP   GETCH       ;Retrieve character indexed by R3 from BUFFER.
12                               ;Returns character in R0, R3 incremented by 1.
13                               ;Returns N set if character is a separator,
14                               ;C set if it is lower case.
15
16 140074 000167 173306          JMP   GETLCH      ;Retrieve character indexed by <R3 - 1> from
17                               ;BUFFER. Returns same as GETCH.
18
19 140100 000167 173306          JMP   GETCHC      ;Retrieve character indexed by R3 from BUFFER
20                               ;without folding lower case. Returns character
21                               ;in R0, and R3 incremented by 1.
22
23 140104 000167 173316          JMP   GETNXT      ;Retrieve next character which is not a
24                               ;separator from BUFFER starting at location
25                               ;indexed by R3. Returns character in R0 with
26                               ;lower case folded, R3 indexes next character
27                               ;in BUFFER.
28
29 140110 000167 173334          JMP   PURGE       ;Eliminate BUFFER contents below location
30                               ;indexed by BP.
31
32 140114 000167 174022          JMP   ECHO        ;Echo contents of BUFFER to the console
33                               ;starting with location indexed by R3. Returns
34                               ;R3 indexing BUFFER location just beyond last
35                               ;echoed character.
36
37 140120 000167 173400          JMP   CRLF        ;Send a carriage return and line feed to the
38                               ;console (and printer).
39
40 140124 000167 173400          JMP   CHROUT      ;Send a (pair of) character(s) to the console
41                               ;(and printer). Character to be sent should
42                               ;be in lower byte of R0. If upper byte of R0
43                               ;is non-zero, it will be sent as a second
44                               ;character.
45
46 140130 000167 173404          JMP   CHRO1       ;Send a single character from the lower byte
47                               ;of R0 to the console (and printer). Returns
48                               ;R0 with its byte swapped.
49
50 140134 000167 173414          JMP   CHRO2       ;Send a single character from the lower byte
51                               ;of R0 to the printer (aux. port) only.
52                               ;Returns R0 with its bytes swapped.
53
54 140140 000167 173462          JMP   ASCOUT      ;Output an ASCII character in the lower 7 bits
55                               ;of R0 as a visible (string of) character(s)
56                               ;on the console. Control characters are output
57                               ;as ^<letter>, with exceptions <SP>,<ESC>,<DEL>

```

Entry Points

```
58
59 140144 000167 173604      JMP      PRINTA      ;Output a string of bytes to the console (and
60                               ;printer). R1 points to start of string, which
61                               ;must end with a blank byte. Returns R1
62                               ;pointing to byte just after blank byte.
63
64                               ;++
65                               ; PARSING AND OUTPUT UTILITY ENTRY POINTS:
66                               ;--
67
68 140150 000167 004346      JMP      GETEXP
69
70 140154 000167 004332      JMP      GENEXP
71
72 140160 000167 014560      JMP      TYPOCT
73
74 140164 000167 014470      JMP      TYPDEC
75
76 140170 005037 007630      FIXBP: CLR      @BP
77 140174 000167 173040      JMP      GETLIN
78
```

```

1
2          .SBTTL Keypad Monitor
3
4 140200 042737 000077 007706 KMONIT::BIC    #000077,@#FLAGS1;clear previous mode
5 140206 042737 000001 007756          BIC    #1,@#ADDR      ;make current address and watchpoint even
6 140214 042737 000001 007754          BIC    #1,@#WATCH    ;in case we came from console monitor
7 140222 052737 000010 007706          BIS    #F,REG,@#FLAGS1 ;set register select mode
8 140230 012704 000007                    MOV    #007,R4        ;index PC
9 140234 010402                    MOV    R4,R2          ;set up to display " PC " on upper LEDs
10 140236 004767 175650                   JSR    PC,DSPNAM      ;display it
11 140242 013705 007750                   MOV    @#PC,R5        ;set PC contents
12 140246 010502                    MOV    R5,R2          ;set up to display on lower LEDs
13 140250 004767 175570                   JSR    PC,DSPLNM      ;display it
14 140254 032737 000200 007706          BIT    #F,SST,@#FLAGS1 ;were we single stepping?
15 140262 001411                    BEQ    KLOOP          ;no
16 140264 042737 000200 007706          BIC    #F,SST,@#FLAGS1 ;clear condition
17 140272 013702 007754                   MOV    @#WATCH,R2    ;is there a watchpoint?
18 140276 001403                    BEQ    KLOOP          ;no
19 140300 011202                    MOV    @R2,R2         ;set watchpoint contents
20 140302 004767 175530                   JSR    PC,DSFUNM      ;display watchpoint contents in upper display
21
22 140306 012737 140330 007676 KLOOP::MOV    #KEYCMD,@#KEYPIV;link command input routine to keypress vector
23 140314 112737 000007 177444          MOVB   #007,@#P$PORC  ;turn on LED display
24 140322 106427 000240                   MTFPS #240           ;enable LED display interrupt
25 140326 000777                    1$: BR    1$         ;wait for keypress; key code returns in R1
26
27
28 140330 005037 007676          KEYCMD::CLR   @#KEYPIV    ;disconnect keypress vector
29 140334 012706 007462          MOV    #STACK,SP       ;pop saved registers and return vector
30 140340 005037 007726          CLR    @#SPCLBF        ;turn off flashing error, if any
31 140344 112737 000010 177444 KEYCM1::MOVB  #010,@#P$PORC ;turn off LED display for faster execution
32 140352 006301                    ASL    R1               ;make a word index of R1
33 140354 000171 140360          JMP    @DSPTBL(R1)     ;dispatch to appropriate routine
34
35 140360 141110          DSPTBL::WORD $CLR      ;[CLR]
36 140362 140430          .WORD $NUMER        ;[ 1 ]
37 140364 140430          .WORD $NUMER        ;[ 4 ]
38 140366 140430          .WORD $NUMER        ;[ 7 ]
39 140370 140430          .WORD $NUMER        ;[ 0 ]
40 140372 140430          .WORD $NUMER        ;[ 2 ]
41 140374 140430          .WORD $NUMER        ;[ 5 ]
42 140376 140430          .WORD $NUMER        ;[ 8 ]
43 140400 141122          .WORD $EXAM         ;[EXA]
44 140402 140430          .WORD $NUMER        ;[ 3 ]
45 140404 140430          .WORD $NUMER        ;[ 6 ]
46 140406 140430          .WORD $NUMER        ;[ 9 ]
47 140410 141232          .WORD $ADV          ;[ADV]
48 140412 141372          .WORD $BACK         ;[BAC]
49 140414 141610          .WORD $REG          ;[REG]
50 140416 141550          .WORD $ADDRK        ;[ADD]
51 140420 141706          .WORD $SST          ;[SST]
52 140422 141714          .WORD $GO           ;[GO ]
53 140424 141462          .WORD $BRKP         ;[BPT]
54 140426 141650          .WORD $FUNC         ;[FNC]

```

```

1
2                               .SBTTL Key Command Routines
3
4 140430 006201                $NUMER::ASR  R1          ;restore key number
5 140432 004767 176056         JSR  PC,KEYBCD    ;set BCD value for key code
6 140436 032737 000040 007706  BIT  #F.FUNC,#FLAGS1 ;is digit a function selection?
7 140444 001403                BEQ  4$          ;no
8 140446 020127 000004         CMP  R1,#004    ;is digit higher than 4?
9 140452 101315                BHI  KLOOP     ;yes, ignore digit
10 140454 032701 000010        4$: BIT  #000010,R1 ;is it 8 or 9?
11 140460 001404                BEQ  1$          ;no, a legal octal digit
12 140462 032737 000005 007706  BIT  #F.DATA+F.ADDR,#FLAGS1 ;no 8 or 9 as data or address input
13 140470 001306                BNE  KLOOP     ;ignore them if pressed
14 140472 032737 000002 007706 1$: BIT  #F.CHAN,#FLAGS1 ;has a digit key already been pressed?
15 140500 001001                BNE  3$          ;yes, just scroll digits in
16 140502 005005                CLR  R5        ;else initialize digit accumulator
17 140504 052737 000002 007706 3$: BIS  #F.CHAN,#FLAGS1 ;indicate that a digit has been pressed
18 140512 032737 000005 007706  BIT  #F.DATA+F.ADDR,#FLAGS1 ;address and data entry scroll digits
19 140520 001004                BNE  2$          ;
20 140522 042737 000002 007706  RIC  #F.CHAN,#FLAGS1 ;others have single digit entry
21 140530 006305                ASL  R5        ;prepare for fourth bit (might be 8 or 9)
22 140532 006305                2$: ASL  R5        ;scroll digit into R5 (digit accumulator)
23 140534 006305                ASL  R5
24 140536 006305                ASL  R5
25 140540 006105                ADD  R1,R5
26
27                               ;Now do execution associated with digit input
28 140542 032737 000005 007706 DIGEXE::BIT #F.DATA+F.ADDR,#FLAGS1 ;Just display new data or address
29 140550 001511                BEQ  1$          ;for others, do special displays instead
30 140552 010502                MOV  R5,R2     ;set new value
31 140554 004767 175264         JSR  PC,DSPLNN ;and display on lower LEDs
32
33 140560 032737 000001 007706  BIT  #F.DATA,#FLAGS1 ;for addresses, store in $ADDR
34 140566 001007                BNE  2$          ;
35 140570 010537 007756         MOV  R5,#$ADDR
36 140574 042737 000001 007756  RIC  #1,#$ADDR  ;make sure address is even
37 140602 000167 000362         JMP  KLOOP1    ;take another key
38
39 140606 032737 000004 007706 2$: BIT  #F.ADDR,#FLAGS1 ;but data can go many places-are we poking RAM?
40 140614 001407                BEQ  3$          ;no
41 140616 013700 007756         MOV  ##$ADDR,R0 ;yes, set a copy of current address
42 140622 004767 001330         JSR  PC,KADRCH ;check that it is not in protected RAM
43 140626 010510                MOV  R5,@R0   ;place data at current address
44 140630 000167 000334         JMP  KLOOP1    ;take another key
45
46 140634 032737 000010 007706 3$: BIT  #F.REG,#FLAGS1 ;changing registers or watchpoint contents?
47 140642 001420                BEQ  4$          ;no
48 140644 020427 000010        CMP  R4,#000010 ;does R4 index a register (0 thru 7 and PS)?
49 140650 101006                BHI  5$          ;no (R4 should be 000011)
50 140652 006304                ASL  R4        ;make it a word index
51 140654 010564 007732        MOV  R5,$R0(R4) ;and place data in user register
52 140660 006204                ASR  R4        ;restore R4
53 140662 000167 000302         JMP  KLOOP1    ;take another key
54
55 140666 013700 007754        5$: MOV  ##WATCH,R0 ;assume R4 indexes watchpoint contents
56 140672 004767 001260         JSR  PC,KADRCH ;check that it is not in protected RAM
57 140676 010510                MOV  R5,@R0   ;deposit data into watchpoint location

```

Key Command Routines

```

58 140700 000167 000264          JMP      KLOOP1          ;take another key
59
60 140704 032737 000020 007706 4%: BIT      #F,BRKS,@#FLAGS;changins breakpoint or watchpoint address?
61 140712 001526          BEQ      KLOOP1          ;no, ignore (strange! that was a safety check)
62 140714 020427 000014          CMP      R4,#000014     ;R4 indexes a breakpoint?
63 140720 103416          BLD      6$             ;no, it's not
64 140722 020427 000017          CMP      R4,#000017
65 140726 101013          BHI      6$             ;no, it's not
66 140730 006304          ASL      R4             ;make an index for word pairs
67 140732 006304          ASL      R4
68 140734 010564 007700          MOV      R5,BRKFIL-060(R4) ;place data in breakpoint address file
69 140740 042764 000001 007700          BIC      #1,BRKFIL-060(R4) ;make breakpoint address even
70 140746 006204          ASR      R4
71 140750 006204          ASR      R4
72 140752 000167 000212          JMP      KLOOP1          ;take another key
73 140756 010537 007754          MOV      R5,@#WATCH     ;assume R4 indexes watchpoint address
74 140762 042737 000001 007754          BIC      #1,@#WATCH     ;make watchpoint address even
75 140770 000167 000174          JMP      KLOOP1          ;take another key
76
77 140774 032737 000040 007706 1%: BIT      #F,FUNC,@#FLAGS;are we entering a function code?
78 141002 001407          BEQ      7$             ;no, must be entering a register number
79 141004 010100          MOV      R1,R0          ;set new disit
80 141006 012701 000004          MOV      #4,R1          ;index a lower display disit
81 141012 004767 174754          JSR      PC,DSPDIG      ;display new disit
82 141016 000167 000146          JMP      KLOOP1          ;take another key
83
84 141022 032737 000010 007706 7%: BIT      #F,REG,@#FLAGS;ibe sure we are selectins a register
85 141030 001457          BEQ      KLOOP1          ;else ignore key
86 141032 010504          MOV      R5,R4          ;save selected register number
87 141034 020427 000011          CMP      R4,#000011     ;are watchpoint contents selected?
88 141040 001414          BEQ      8$             ;yes, do special display
89 141042 010402          MOV      R4,R2          ;display register's name on upper LEDs
90 141044 004767 175042          JSR      PC,DSPNAM
91 141050 006304          ASL      R4             ;make a word index
92 141052 016405 007732          MOV      $R0(R4),R5     ;set register's contents
93 141056 006204          ASR      R4
94 141060 010502          MOV      R5,R2          ;display on lower LEDs
95 141062 004767 174756          JSR      PC,DSPLNM
96 141066 000167 000076          JMP      KLOOP1          ;take another key
97
98 141072 013702 007754          MOV      @#WATCH,R2     ;set watchpoint address
99 141076 004767 174734          JSR      PC,DSPUNM      ;show it on upper display
100 141102 017705 046646          MOV      @#WATCH,R5     ;set watchpoint contents
101 141106 000764          BR       9$
102
103
104 141110 005005          .CLR:: CLR      R5          ;clear disit accumulator
105 141112 042737 000002 007706          BIC      #F,CHAN,@#FLAGS;in case of single disit mode
106 141120 000610          BR       DIGEXE
107
108          .ENABL  LSB
109
110 141122 032737 000004 007706 $EXAM:: BIT      #F,ADDR,@#FLAGS;examining RAM?
111 141130 001421          BEQ      1$             ;no
112 141132 013702 007756          MOV      @##ADDR,R2     ;set current address
113 141136 004767 174674          JSR      PC,DSPUNM      ;show it on upper display
114 141142 017705 046610          MOV      @#ADDR,R5     ;set contents

```

Key Command Routines

```

115 141146 010502          GNEXAM:MOV  R5,R2      ;general examine entry point
116 141150 004767 174670          JSR   PC,DSPLNH  ;show it on lower display
117 141154 052737 000001 007706    BIS   #F.DATA,@#FLAGS1;iso to data entry mode
118 141162 042737 000002 007706    BIC   #F.CHAN,@#FLAGS1;indicate no change yet
119 141170 000167 177112          KLOOP1: JMP  KLOOP      ;take another key
120
121 141174 032737 000010 007706 1#:  BIT   #F.REG,@#FLAGS1 ;examining a register?
122 141202 001403          BEQ   2$           ;no
123 141204 005704          TST   R4           ;R4 initialized to -1 by $REG
124 141206 100357          BPL   GNEXAM      ;contents put in R5 by DIGEXE
125 141210 100767          BNE   KLOOP1      ;ignore EXAM; make user push a digit first
126
127 141212 032737 000040 007706 2#:  BIT   #F.FUNC,@#FLAGS1;executions special function?
128 141220 001763          BEQ   KLOOP1      ;ignore all other occurrences of EXAM
129 141222 005705          TST   R5           ;R5 initialized to -1 by $FUNC
130 141224 100761          BNE   KLOOP1      ;ignore EXAM; make user push a digit first
131 141226 000167 000774          JMP   DDFUNC      ;execute special function
132          .DSABL  LSB
133
134
135          .ENABL  LSB
136 141232 032737 000004 007706 $ADV:  BIT   #F.ADDR,@#FLAGS1;stepping through addresses?
137 141240 001405          BEQ   1$           ;no
138 141242 005237 007756          INC   @##ADDR      ;iso to next word
139 141246 005237 007756          INC   @##ADDR
140 141252 000723          BR   $EXAM
141
142 141254 032737 000010 007706 1#:  BIT   #F.REG,@#FLAGS1 ;stepping through registers?
143 141262 001425          BEQ   2$           ;no
144 141264 005204          INC   R4           ;register index
145 141266 020427 000011          CNP   R4,#000011  ;don't step past watchpoint contents
146 141272 103411          BLO   3$           ;no
147 141274 012704 000011          MOV   #000011,R4   ;select watchpoint contents
148 141300 013702 007754          MOV   @##WATCH,R2  ;set watchpoint address
149 141304 004767 174526          JSR   PC,DSPUNH    ;show it on upper display
150 141310 017705 046440          MOV   @##WATCH,R5  ;set watchpoint contents
151 141314 000714          BR   GNEXAM
152
153 141316 010402          3#:  MOV   R4,R2       ;display register's name on upper LEDs
154 141320 004767 174566          JSR   PC,DSPNAM
155 141324 006304          ASL   R4           ;make a word index
156 141326 016405 007732          MOV   $R0(R4),R5   ;set register contents
157 141332 006204          ASR   R4
158 141334 000704          BR   GNEXAM
159
160 141336 032737 000020 007706 2#:  BIT   #F.BRKS,@#FLAGS1;stepping through breakpoints?
161 141344 001711          BEQ   KLOOP1      ;no, ignore advance
162 141346 005204          INC   R4           ;increment breakpoint index
163 141350 020427 000012          CMP   R4,#000012  ;do special increment pattern
164 141354 001001          BNE   4$           ;no
165 141356 005724          TST   (R4)+
166 141360 020427 000017          4#:  CMP   R4,#000017
167 141364 101440          BLOS  BRKP1       ;make breakpoint display
168 141366 005304          DEC   R4
169 141370 000436          BR   BRKP1
170
171

```

Key Command Routines

```

172 141372 032737 000004 007706 $BACK:: BIT    $F,ADDR,##FLAGS1;steppins through addresses?
173 141400 001405          BEQ    6$          ;no
174 141402 005337 007756          DEC    ##$ADDR          ;iso to next word
175 141406 005337 007756          DEC    ##$ADDR
176 141412 000643          BR    $EXAM
177
178 141414 032737 000010 007706 6$: BIT    $F,REG,##FLAGS1 ;steppins through registers?
179 141422 001404          BEQ    7$          ;no
180 141424 005304          DEC    R4
181 141426 100333          BPL    3$          ;prevent underflow
182 141430 005004          CLR    R4
183 141432 000731          BR    3$          ;continue in $ADV
184
185 141434 032737 000020 007706 7$: BIT    $F,BRKS,##FLAGS1;steppins through breakpoints?
186 141442 001452          BEQ    KLOOP1      ;no, ignore backup
187 141444 005304          DEC    R4          ;decrement breakpoint index
188 141446 020427 000014          CMP    R4,#000014 ;do special decrement pattern
189 141452 103005          BHIS  BRKP1       ;make breakpoint display
190 141454 012704 000011          MOV    #000011,R4 ;index watchpoint
191 141460 000402          BR    BRKP1
192          .DSABL  LSB
193
194
195 141462 012704 000014          $BRKP:: MOV    #000014,R4 ;index breakpoint 0
196 141466 010402          BRKP1: MOV    R4,R2 ;show name of breakpoint on upper display
197 141470 004767 174416          JSR    PC,DSPNAM
198 141474 020427 000014          CMP    R4,#000014 ;indexins watchpoint or a breakpoint?
199 141500 103420          BLO    1$          ;the watchpoint
200 141502 006304          ASL    R4          ;make an index for word pairs
201 141504 006304          ASL    R4
202 141506 016405 007700          MOV    BRKFIL-060(R4),R5 ;set breakpoint address
203 141512 006204          ASR    R4
204 141514 006204          ASR    R4
205 141516 010502          2$: MOV    R5,R2 ;display it on lower LEDs
206 141520 004767 174320          JSR    PC,DSPLNH
207 141524 042737 000077 007706          BIC    #000077,##FLAGS1;clear previous mode
208 141532 052737 000021 007706          BIS    $F,BRKS+$F,DATA,##FLAGS1 ;set breakpoint and data mode
209 141540 000441          BR    KLOOP2
210
211 141542 013705 007754          1$: MOV    ##$WATCH,R5 ;set watchpoint address
212 141546 000763          BR    2$
213
214
215 141550 012702 000012          $ADDRK::MOV    #012,R2 ;prompt with ' Addr ' on upper display
216 141554 004767 174332          JSR    PC,DSPNAM
217 141560 013705 007756          MOV    ##$ADDR,R5 ;set previous address
218 141564 010502          MOV    R5,R2 ;show it on lower display
219 141566 004767 174252          JSR    PC,DSPLNH
220 141572 042737 000077 007706          BIC    #000077,##FLAGS1;clear previous mode
221 141600 052737 000004 007706          BIS    $F,ADDR,##FLAGS1;set current address entry mode
222 141606 000416          BR    KLOOP2
223
224
225 141610 012702 000013          $REG:: MOV    #013,R2 ;prompt with ' REG ' on upper display
226 141614 004767 174272          JSR    PC,DSPNAM
227 141620 012704 177777          MOV    #-1,R4 ;initialize register index to illegal value
228 141624 004767 174506          JSR    PC,BLNKLO ;and blank lower display

```



```

229 141630 042737 000077 007706      BIC  #000077,@#FLAGS1;clear previous mode
230 141636 052737 000010 007706      BIS  #F.REG,@#FLAGS1;and set register select mode
231 141644 000167 176436              KLOOP2: JMP  KLOOP
232
233
234 141650 012702 000020              $FUNC:: MOV  #020,R2      ;prompt with ' Func ' on upper display
235 141654 004767 174232              JSR  PC,DSPNAM
236 141660 012705 177777              MOV  #-1,R5           ;initialize accumulator to illegal value
237 141664 004767 174446              JSR  PC,BLANKLD      ;and blank lower display
238 141670 042737 000077 007706      BIC  #000077,@#FLAGS1;clear previous mode
239 141676 052737 000040 007706      BIS  #F.FUNC,@#FLAGS1;and set function select mode
240 141704 000757                    BR   KLOOP2
241
242
243 141706 052737 000200 007706  $SST:: BIS  #F.SST,@#FLAGS1 ;show we are single stepping
244
245 141714 023727 007746 007400  $GO::  CMP  @##SP,#SCRPAD  ;is user SP in his RAM?
246 141722 101004                    $GO1:: BHI  1%          ;no, show error message
247 141724 023727 007746 000004      CMP  @##SP,#4         ;and is there room for 2 pushes?
248 141732 103007                    BHI5   2%          ;yes, SP is acceptable
249 141734 042737 000200 007706  1%:   BIC  #F.SST,@#FLAGS1 ;else abort
250 141742 012700 000004              MOV  #4,R0           ;SP Error
251 141746 000167 175376              JMP  FERRQR         ;go to fatal error handler to reenter monitor
252
253 141752 013702 007736              2%:   MOV  @##R2,R2        ;restore user context,
254 141756 013703 007740              MOV  @##R3,R3
255 141762 013704 007742              MOV  @##R4,R4
256 141766 013705 007744              MOV  @##R5,R5
257 141772 013737 007700 007676      MOV  @#KEYPVS,@#KEYPIV ;and user keypress vector
258 142000 013706 007746              MOV  @##SP,SP        ;we're on user's stack now
259 142004 013746 007752              MOV  @##PS,-(SP)     ;push PS, PC like from a trap
260 142010 042716 000020              BIC  #020,(SP)       ;clear T-bit
261 142014 032737 000200 007706      BIT  #F.SST,@#FLAGS1 ;are we single stepping?
262 142022 001422                    BEQ  3%            ;no
263 142024 052716 000020              4%:   BIS  #020,(SP)     ;yes, set T-bit
264 142030 012737 137174 000014  9%:   MOV  #BRKPT,@#014   ;make sure BPT trap is still connected
265 142036 012737 000340 000016      MOV  #340,@#016
266 142044 013700 007732              MOV  @##R0,R0        ;finish restoring context
267 142050 013701 007734              MOV  @##R1,R1
268 142054 013746 007750              MOV  @##PC,-(SP)     ;push PC
269 142060 052737 000100 007706      BIS  #F.USER,@#FLAGS1;show we are running user code
270 142066 000006                    RTT                    ;and simulate return from trap
271
272 142070                                3%:
273 142070 032737 001000 007706  5%:   BIT  #F.BRK,@#FLAGS1;single stepping over a break point
274 142076 001352                    BNE  4%            ;if yes then don't install
275
276 142100 012700 007760              MOV  #BRKFIL,R0     ;start of break point table
277 142104 012701 000004              MOV  #4,R1          ;four entrys
278 142110 013020                    100%: MOV  @(R0)+,(R0)+    ;suck up data from break point locations
279 142112 077102                    SOB  R1,100%        ;
280 142114 012700 007760              MOV  #BRKFIL,R0     ;start of break point table
281 142120 012701 000004              MOV  #4,R1          ;same as last time
282 142124 005710                    200%: TST  (R0)          ;first see if there was really one set
283 142126 001405                    BEQ  300%          ;branch if not
284 142130 012730 000003              MOV  #3,@(R0)+     ;set break point
285 142134 005720                    TST  (R0)+         ;skip to next

```

Key Command Routines

```

286 142136 077106          400%: SOB R1,200$      ifour entrys
287 142140 000402          BR 500$
288 142142 032020          300%: BIT (R0)+,(R0)+  ifskip this entry
289 142144 000774          BR 400$      ifso back for next
290 142146 052737 000400 007706 500%: BIS $F,BRKA,@#FLAGS$ ifindicate breakpoints installed
291 142154 000725          BR 9$       ifand so
292
293
294
295 142156 032737 040000 007706 KADRCH: BIT $F,PROT,@#FLAGS$ ifkeypad scratch protection on?
296 142164 001017          BNE 1$      ifno, deposit anywhere
297 142166 020027 020000          CMP R0,#20000 ifis address above upper instance of scratchpad?
298 142172 103014          BHS 1$      ifyes, definitely doesn't address protected RAM
299 142174 020027 007400          CMP R0,#SCRPAD ifis it below lower instance of scratchpad?
300 142200 103411          BLO 1$      ifyes
301 142202 020027 010000          CMP R0,#10000 ifis it within lower instance of scratchpad?
302 142206 103403          BLO 2$      ifyes, address is illegal
303 142210 020027 017400          CMP R0,#SCRPAD+10000 ifis it between instances of scratchpad RAM?
304 142214 103403          BLO 1$      ifyes, not within them, so address is OK
305 142216 005000          2%: CLR R0   ifindex user RAM error message
306 142220 000167 175124          JMP FERROR  ifso to fatal error handler to reenter monitor
307
308 142224 000207          1%: RTS PC   ifaddress is acceptable
309

```

```

1
2                               .SBTTL Special Keypad Monitor Functions
3
4 142226 006305                DOFUNC::ASL  R5           ;R5 was 0, 1, 2, 3, or 4
5 142230 000175 142234        JMP      @FNCTBL(R5)    ;dispatch to selected function
6
7 142234 142246                FNCTBL::WORD CANCEL      ;0 means cancel breakpoints
8 142236 142270                .WORD   GO$LED      ;1 means turn on LEDs and go at user PC
9 142240 142572                .WORD   CKON1     ;2 means cold start console monitor
10 142242 142320               .WORD   SPEEDS    ;3 means select baud rates and console type
11 142244 142562               .WORD   NPROT     ;4 means disable keypad monitor scratch protec.
12
13 142246 012700 000020        CANCEL::MOV  #020,R0    ;8 words in breakpoint file
14 142252 105060 007757        1$:  CLRB   BRKFL-1(R0) ;clear address/contents pairs
15 142256 077003                SOB     R0,1#
16 142260 005037 007754        CLR     @##WATCH     ;cancel watchpoint also
17 142264 000167 175710        FNCDON: JMP  KMDWIT    ;go to register select mode
18
19 142270 112737 000007 177444 GO$LED::MOVB  #007,@#P$PORC ;turn on LED displays
20 142276 000167 177412        JMP     $GO           ;and so
21
22 142302 000454                SPDTBL::WORD 300.
23 142304 001130                .WORD   600.
24 142306 002260                .WORD   1200.
25 142310 004540                .WORD   2400.
26 142312 011300                .WORD   4800.
27 142314 022600                .WORD   9600.
28 142316 045400                .WORD   19200.
29
30 142320 012702 000021        SPEEDS::MOV  #021,R2    ;index "ConSol"
31 142324 004767 173562        JSR     PC,DSPNAM     ;show prompt on upper display
32 142330 113705 007710        MOVB   @#CNFLG,R5    ;set baud rate bits
33 142334 042705 177707        BIC    #177707,R5    ;mask out other flags
34 142340 006205                5$:  ASR     R5         ;shift to word index position
35 142342 006205                ASR     R5
36 142344 016501 142302        MOV     SPDTBL(R5),R1 ;set corresponding baud rate
37 142350 012702 000010        MOV     #010,R2      ;index lower display
38 142354 004767 174006        JSR     PC,DSPDEC     ;display decimal number
39 142360 004767 174052        3$:  JSR     PC,KEYGET  ;wait for a key
40 142364 020127 000010        CMP     R1,#010      ;was it "Enter"?
41 142370 001436                BEQ     1$           ;yes, go to aux port speed
42 142372 020127 000016        CMP     R1,#016      ;was is a function key?
43 142376 103067                BHIS   10$          ;yes, exit
44 142400 020127 000014        CMP     R1,#014      ;was it advance?
45 142404 001410                BEQ     2$           ;yes, modify baud rate
46 142406 020127 000015        CMP     R1,#015      ;was it backup?
47 142412 001362                BNE     3$           ;no, wait for another key
48
49 142414 005305                DEC     R5           ;decrease baud rate
50 142416 005305                DEC     R5
51 142420 100007                BPL     4$           ;show new rate
52 142422 005005                CLR     R5           ;prevent underflow
53 142424 000405                BR      4$
54
55 142426 005725                2$:  TST     (R5)+     ;increase baud rate
56 142430 020527 000016        CMP     R5,#016      ;overflow?
57 142434 103401                BLD     4$           ;no

```

```

58 142436 005745          TST    -(R5)
59 142440 006305          4%:   ASL    R5          ;store index in baud rate bits
60 142442 006305          ASL    R5
61 142444 006305          ASL    R5
62 142446 106137 007710   RDLB  @#CONFLG        ;preserve high order bit (F.A64X)
63 142452 110537 007710   MOVB  R5,@#CONFLG
64 142456 106037 007710   RORB  @#CONFLG
65 142462 006205          ASR    R5
66 142464 000725          BR     5%
67
68 142466 012702 000022   1%:   MOV    #022,R2        ;index 'A.Port'
69 142472 004767 173414   JSR   PC,DSPNAM        ;show prompt
70 142476 016501 142302   8%:   MOV    SPDTBL(R5),R1   ;set console baud rate
71 142502 105737 007710   TSTB  @#CONFLG        ;is 64 X set for aux port?
72 142506 100002          RPL    9%             ;no, same speeds
73 142510 006201          ASR    R1             ;else console rate / 4
74 142512 006201          ASR    R1
75 142514 012702 000010   9%:   MOV    #010,R2        ;index lower display
76 142520 004767 173642   JSR   PC,DSPDEC        ;display baud rate
77 142524 004767 173706   JSR   PC,KEYGET        ;wait for a key
78 142530 020127 000010   CMP   R1,#010         ;is it 'Enter'?
79 142534 001653          BEQ   FNCDON          ;yes, finished
80 142536 020127 000016   CMP   R1,#016         ;was it a function button?
81 142542 103005          BHIS  10%            ;yes, exit
82 142544 012700 000200   MOV   #F.A64X,R0      ;no, toggle aux port baud rate
83 142550 074037 007710   XOR   R0,@#CONFLG    ;CONFLG is an even byte
84 142554 000750          BR     8%            ;wait for enter key
85
86 142556 000167 175562   10%:  JMP   KEYCM1          ;execute function key
87
88 142562 052737 040000 007706 NPROT:: BIS  #F.PROT,@#FLAGS1 ;turn off protection
89 142570 000635          BR     FNCDON        ;go to register select mode
    
```

```

1
2
3           .SBTTL Console Monitor
4           ;++
5           ; REGISTER USAGE:
6           ;           R5      last expression value.
7           ;           R4      current term value and scratch current address
8           ;           R3      running pointer for input buffer
9           ;           R2      pointer and general passing to routines
10          ;           R1      computational scratch
11          ;           R0      passing to bottom level routines and scratch
12          ;
13          ; Cold entry point:
14          ;--
15
16 142572 012737 006500 007746 CMON1::MOV    $$UR,$$SP      ;initialize user SP for console monitor
17 142600 012737 040100 007330      MOV    $400*100+100,$$TMPMOD ;set default modes
18 142606 012737 000004 007325      MOV    $F,MULL,$$LASLIM;initialize last line flags
19 142614 005037 007630      CLR    $$BP          ;initialize console buffer pointers
20 142620 105037 007672      CLR    $$HFP        ;clear host buffer pointer also
21 142624 142737 000007 007710      BIC    $007,$$CONFLG ;clear console line flags (but not speed)
22 142632 105037 007673      CLR    $$HSTFLG     ;clear host flags
23 142636 012767 133000 035214      MOV    $LINEIN,CONIN ;hook up console input service routine
24 142644 113700 007710      MOVB  $$CONFLG,R0   ;set current baud rate
25 142650 052700 000002      BIS    $000002,R0   ;enable prod, baud rate, disable xmit interrupt
26 142654 010037 177564      MOV    R0,$$C$XCSR ;transfer control word to console port
27 142660 004767 172304      JSR   PC,SETAUX    ;set aux port baud rate and disable
28 142664 052737 002000 007706      BIS    $$KEYP,$$FLAGS1;disable keypad monitor
29 142672 106427 000000      MTPS  $000         ;enable interrupts
30 142676 012737 000100 177560      MOV    $000100,$$C$RCSR;enable console receiver interrupt
31 142704 004767 170614      JSR   PC,CRLF     ;start a new line
32 142710 004767 170610      JSR   PC,CRLF
33 142714 012702 143052      MOV    $GRET,R2    ;point to greeting message
34 142720 004767 171042      JSR   PC,PRINT    ;type message
35
36          ;++
37          ; Warm entry point:
38          ;--
39
39 142724 012767 133000 035126 CMONIT::MOV    $LINEIN,CONIN ;hook up console input service routine
40 142732 106427 000000      MTPS  $000         ;enable interrupts
41 142736 012737 000100 177560      MOV    $000100,$$C$RCSR;enable console receiver interrupt
42 142744 032737 004000 007706      BIT    $F,HOST,$$FLAGS1;is host mode enabled?
43 142752 001412      BEQ   1$          ;no, clear host flags to terminate functions
44 142754 012737 134540 000120      MOV    $HOSTIN,$$AUXIN ;hook up host input service interrupt
45 142762 112737 000046 007711      MOVB  $046,$$AUXFLG ;save a copy of new A$CREG value
46 142770 112737 000046 177452      MOVB  $046,$$A$CREG ;reenable aux port receiver interrupt
47 142776 000402      BR    2$
48
49 143000 105037 007673      1$: CLR    $$HSTFLG     ;stop host activity
50 143004 032737 000200 007706 2$: BIT    $F,SST,$$FLAGS1 ;were we single stepping?
51 143012 001501      BEQ   CMON3       ;no, scrap line
52 143014 042737 000200 007706      BIC    $F,SST,$$FLAGS1 ;clear flag from $G01 routine
53 143022 004767 006702      JSR   PC,TYPREG   ;yes, display registers
54 143026 013704 007750      MOV    $$PC,R4    ;point to next location
55 143032 004767 012212      JSR   PC,TYPADR   ;type address
56 143036 004767 012126      JSR   PC,TYPLOI  ;disassemble next instruction
57 143042 005737 007466      TST   $$REPEAT   ;were we repeating?

```

58 143046 001463	BEQ	CMON3	ino
59 143050 000476	BR	CMON4	yes, keep command line

```

1
2 143052 076725 000207 055207 GRET: ,RAD50 /TEM CONSOLE MONITOR V1.0 /
   143060 045710 051646 035557
   143066 070226 143076 000000
3 143074 000000 ,WORD 0
4 143076 001640 004540 WHAT: ,RAD50 / WHAT /
5 143102 000000 ,WORD 0
6 143104 007716 023132 001156 OVERFL: ,RAD50 /BUFFER OVERFLOW/
   143112 021026 046557
7 143116 000000 ,WORD 0
8
9 143120 004767 170400 PRNERR::JSR PC,CRLF ;type <CR> <LF>
10 143124 012700 056007 MOV #400#134#007,R0 ;type <BELL> \
11 143130 004767 170374 JSR PC,CHROUT
12 143134 004767 170276 JSR PC,GETBP ;set running pointer to back pointer
13 143140 004767 170776 JSR PC,ECHO ;echo buffer to console to indicate error
14 143144 012700 000134 MOV #134,R0 ;type \ <CR>
15 143150 004767 170354 JSR PC,CHROUT
16 143154 004767 170344 JSR PC,CRLF
17 143160 000167 170602 JMP PRINT ;print error message
18
19 ;++
20 ; Command error entry point:
21 ;--
22
23 143164 012706 007462 CHON2::MOV #STACK,SP ;we came here from unspecified depth
24 143170 012702 143076 MOV #WHAT,R2 ;point to error message
25 143174 004767 177720 JSR PC,PRNERR ;print erroneous command line and error message
26 143200 012700 000077 MOV #077,R0 ;type ? <CR>
27 143204 004767 170320 JSR PC,CHROUT
28 143210 000402 BR CHON3
29
30 ;++
31 ; General error entry; scrap rest of command line, terminate host input:
32 ;--
33
34 143212 004767 177702 ERROR::JSR PC,PRNERR ;print error message
35 143216 142737 000200 007673 CHON3::BICB #F,LOAD,@#HSTFLG;stop loading from host
36 143224 012706 007462 CHON31:MOV #STACK,SP ;we don't know at which level error occurred
37 143230 005037 007630 CLR @#BP ;clear both back pointer and forward pointer
38 143234 005037 007466 CLR @#REPEAT ;don't keep repeating bad command line
39 143240 152737 000004 007324 BICB #F,NULL,@#LINFLG;act like line was not empty
40
41 ;++
42 ; Fresh command line and repeat entry point:
43 ;--
44
44 143246 000337 007324 CHON4::SWAB @#LINFLG ;save old line parsing flass
45 143252 105037 007324 CLRB @#LINFLG ;reset line parsing flass for new line
46 143256 004767 170166 JSR PC,PURGE ;purge bottom of buffer, clear running pointer
47 143262 005737 007466 TST @#REPEAT ;repeat line or prompt for a new one?
48 143266 001403 BEQ 1$ ;prompt for a new one
49 143270 005337 007466 DEC @#REPEAT ;count repetitions
50 143274 000543 BR CHON5 ;and re-execute line
51
52 143276 113737 007331 007330 1$:MOVB @#PERMOD,@#TMPMOD ;restore permanent modes
53 143304 004767 170214 JSR PC,CRLF ;start a new line
54 143310 012701 076725 MOV #<24#50+05>#50+15,R1 ;RAD50 triplet 'TEM'

```

55	143314	004767	170464		JSR	PC,PRINT1	i;type a single RAD50 triplet
56	143320	012700	020076		MOV	#400#040#076,R0	i;type > <SPACE>
57	143324	004767	170200		JSR	PC,CHROUT	
58	143330	123703	007631	2%:	CMPB	@#FP,R3	i;see if a character has been typed
59	143334	001415			BEQ	3%	i;no, so check host buffer
60	143336	004767	170600	8%:	JSR	PC,ECHO	i;echo character to console
61	143342	126327	007467	000015	CMPB	BUFFER-1(R3),#015	i;is last character a <CR>?
62	143350	001367			BNE	2%	i;no, wait for one
63	143352	105037	007324		CLRB	@#LINFLG	i;reset line flag from ECHO
64	143356	005003			CLR	R3	i;set running pointer to beginning of line
65	143360	142737	000001	007673	BICB	#F,BYTB,@#HSTFLG	i;clear host line byte bucket (~0) flag
66	143366	000506			BR	CMONS	i;execute line
67							
68	143370	105737	007672	3%:	TSTB	@#HFP	i;see if we have a character from the host
69	143374	001755			BEQ	2%	i;no, so back and check console
70	143376	105737	007673		TSTB	@#HSTFLG	i;are we loading or just listening?
71	143402	100444			BMI	4%	i;loading
72	143404	132737	000001	007673	RITB	#F,BYTB,@#HSTFLG	i;are we ignoring host messages?
73	143412	001403			BEQ	9%	i;no
74	143414	004767	171324		JSR	PC,HGETCH	i;yes, get character from host
75	143420	000743			BR	2%	i;and throw it away
76							
77	143422	004767	170076	9%:	JSR	PC,CRLF	i;start a new line to show message from host
78	143426	012701	032153		MOV	#<10*50+17>#50+23,R1	i;RAD50 triplet 'HOS'
79	143432	004767	170346		JSR	PC,PRINT1	i;type a single RAD50 triplet
80	143436	012700	037124		MOV	#400#076+124,R0	i;'T'
81	143442	004767	170062		JSR	PC,CHROUT	i;complete the header
82	143446	105737	007672	5%:	TSTB	@#HFP	i;see if we have a character
83	143452	001411			BEQ	7%	i;no, check console for keypress
84	143454	004767	171264		JSR	PC,HGETCH	i;set character from host buffer
85	143460	132737	000001	007673	BITB	#F,BYTB,@#HSTFLG	i;are we ignoring host messages?
86	143466	001003			BNE	7%	i;yes, throw character away
87	143470	010100			MOV	R1,R0	
88	143472	004767	170032		JSR	PC,CHROUT	i;else echo it to console
89	143476	123703	007631	7%:	CMPB	@#FP,R3	i;see if console typed a character
90	143502	001761			BEQ	5%	i;if not, continue host message
91	143504	005003			CLR	R3	i;prepare to re-echo console input
92	143506	105037	007324		CLRB	@#LINFLG	
93	143512	000671			BR	1%	i;redisplay prompt and echo buffer again
94							
95	143514	004767	171224	4%:	JSR	PC,HGETCH	i;set character from host buffer
96	143520	106427	000340		MTPS	#340	i;block console interrupt for a while
97	143524	113702	007631		MOV	@#FP,R2	i;set console buffer input pointer
98	143530	042702	177400		BIC	#177400,R2	i;prevent sign extend
99	143534	001003			BNE	10%	i;we aren't at beginning of line
100	143536	120127	000012		CMPB	R1,#012	i;throw away line feeds at beginning of line
101	143542	001407			BEQ	11%	
102	143544	120227	000140	10%:	CMPB	R2,#140	i;is buffer full?
103	143550	103007			BHS	6%	i;yes, abort load process
104	143552	110162	007470		MOV	R1,BUFFER(R2)	i;transfer character to console input buffer
105	143556	105237	007631		INCB	@#FP	i;advance buffer input pointer
106	143562	106427	000000	11%:	MTPS	#000	i;restore interrupts
107	143566	000660			BR	2%	i;echo buffer to console
108							
109	143570	106427	000000	6%:	MTPS	#000	i;restore interrupts
110	143574	012702	143104		MOV	#OVERFL,R2	i;point to error message
111	143600	000167	177406		JMP	ERROR	i;and print it


```

112
113          ;++
114          ;   Next command entry point:
115          ;--
116
117 143604 004767 167616      CMONS:: JSR   PC,GETNXT      ;set first meaningful character
118 143610 000402              BR       CMON6
119
120 143612 004767 167570      CONTIN::JSR  PC,GETLCH      ;recover last acquired character
121
122 143616 005737 007466      CMON6:: TST   @@REPEAT      ;are we repeating?
123 143622 001004              BNE     1$                ;yes, so keep last command
124 143624 110337 007630      MOVW   R3,@#BP           ;else throw it away
125 143630 105337 007630      DECB   @#BP
126 143634 000337 007326      1$:   SWAB   @#CURCOM      ;save old command flags
127 143640 105037 007326      CLRB   @#CURCOM         ;and initialize them for next command
128 143644 000402              BR       CMON7
129          ;++
130          ;   Parse top level:
131          ;--
132
133          .ENABL  LSB
134 143646 004767 167554      COMLUP: JSR   PC,GETNXT      ;set next meaningful character
135 143652 120027 000015      CMON7: CMPB   R0,#015       ;is it <CR>?
136 143656 001544              BEQ     DOCR              ;yes, so to <CR> handler
137
138 143660 152737 000004 007324 1$:   BISB   #F,NULL,@#LINFLG;show that line is not empty
139 143666 132737 000020 007324      BITB   #F.COMM,@#LINFLG;are we in a comment?
140 143674 001364              BNE     COMLUP           ;yes, keep looking for <CR>
141 143676 120027 000054      CMPB   R0,#054           ;is it comma?
142 143702 001761              BEQ     COMLUP           ;ignore commas at top level
143 143704 120027 000033      CMPB   R0,#033           ;is it <ESC>?
144 143710 001005              BNE     2$                ;no
145 143712 000402              BR       ERROR2
146
147 143714 000167 000354      4$:   JMP     DDEXPR        ;go to general expression handler
148 143720 000167 177240      ERROR2: JMP   CMON2
149
150 143724 120027 000042      2$:   CMPB   R0,#042         ;is character control, space, or '!'?
151 143730 103773              BLO     ERROR2           ;yes, invalid command
152 143732 120027 000137      CMPB   R0,#137           ;is it above underscore?
153 143736 101370              BHI     ERROR2           ;yes, invalid command
154 143740 001006              BNE     5$                ;not an underscore
155 143742 010537 007336      MOVW   R5,@#SAVEXP       ;execute underscore
156 143746 152737 000010 007324 1$:   BISB   #F.SAVX,@#LINFLG;show pending underscore
157 143754 000713              BR       CMON5           ;set next command
158
159 143756 120027 000136      5$:   CMPB   R0,#136         ;is it caret?
160 143762 001010              BNE     6$                ;no
161 143764 005337 007756      DEC     @##ADDR          ;execute caret
162 143770 105737 007330      TSTB   @#THPMOD         ;are we in byte mode?
163 143774 100703              BMI     CMON5           ;yes, back up a byte
164 143776 005337 007756      DEC     @##ADDR          ;else back up a whole word
165 144002 000700              BR       CMON5           ;set next command
166
167 144004 120027 000134      6$:   CMPB   R0,#134         ;is it reverse slant?
168 144010 001017              BNE     7$                ;no

```

169	144012	106437	007327			MTPS	@#LASCOM	i	did the last expression have undefined symbols	
170	144016	001010				BNE	8#	i	yes, refuse setting of the current address	
171	144020	010537	007756			MOV	R5,@##ADDR	i	execute reverse slant	
172	144024	010504				MOV	R5,R4	i	set new address	
173	144026	004767	011216			JSR	PC,TYPADR	i	type R4 as number or label	
174	144032	004767	011042			JSR	PC,TYPLOC	i	type location addressed by R4	
175	144036	000662				BR	CMONS	i	set next command	
176										
177	144040	012702	151372			8#:	MOV	#UNDEFX,R2	i	error, undefined expression
178	144044	000167	177142				JMP	ERROR		
179										
180	144050	120027	000133			7#:	CMPB	R0,#133	i	is it left bracket?
181	144054	101321					BHI	ERROR2	i	no, invalid command
182	144056	120027	000100				CMPB	R0,#100	i	is it alphabetic?
183	144062	101314					BHI	4#	i	yes, beginning of expression or instruction
184	144064	001715					BEQ	ERROR2	i	no, at sign is invalid command
185									i	we've narrowed R0 to the 042 to 077 range
186	144066	116001	155774				MOVB	PARAMS-040(R0),R1	i	set parsing code
187	144072	042701	177770				BIC	#177770,R1	i	mask out other information
188	144076	006301					ASL	R1	i	make it a word index
189	144100	000171	144104				JMP	@PARDSP(R1)	i	and dispatch accordingly
190							.DSABL	LSB		
191										
192	144104	143164				PARDSP:	.WORD	CMON2	i	0 means invalid command
193	144106	144274					.WORD	DDEXPR	i	1 means set expression or instruction
194	144110	151426					.WORD	DOLABL	i	2 means define label
195	144112	144124					.WORD	DOCOMH	i	3 means start comment
196	144114	151164					.WORD	DOSYMB	i	4 means define symbol
197	144116	144134					.WORD	DONVAL	i	5 means type numerically
198	144120	143164					.WORD	CMON2	i	6 means valid as operator within expressions
199	144122	144274					.WORD	DDEXPR	i	7 means both 1 and 6 are true
200										
201										
202	144124	152737	000020	007324		DOCOMH:	BISB	#F.COMM,@#LINFLG	i	indicate we're in a comment
203	144132	000624					BR	CMONS	i	and continue
204										
205										
206	144134	004767	167364			DONVAL:	JSR	PC,CRLF	i	type last expression numerically
207	144140	016746	043164				MOV	TMPMOD,-(SP)	i	save flags
208	144144	042767	000200	043156			BIC	#F.BYTM,TMPMOD	i	make word mode
209	144152	010501					MOV	R5,R1		
210	144154	004767	010716				JSR	PC,TYPMH1		
211	144160	012667	043144				MOV	(SP)+,TMPMOD	i	restore flags
212	144164	000167	177414				JMP	CMONS		
213										
214										
215	144170	105737	007673			DOCR::	TSTB	@#HSTFLG	i	are we loading from host?
216	144174	100426					BMI	3#	i	yes, suppress <CR> command feature
217	144176	106437	007324				MTPS	@#LINFLG	i	set command line flags
218	144202	001423					BEQ	3#	i	line is not empty, so do nothing
219	144204	106437	007325				MTPS	@#LASLIN		
220	144210	103427					BCS	5#	i	last line caused single step, so step again
221	144212	013704	007756				MOV	@##ADDR,R4	i	set current address
222	144216	132737	000004	007325			BITB	#F.NULL,@#LASLIN	i	did last line dump locations?
223	144224	001004					BNE	6#	i	no, so don't advance
224	144226	013704	007334				MOV	@#ADVADR,R4	i	advance past last dump
225	144232	010437	007756				MOV	R4,@##ADDR		

```
226 144236 016705 033535      6%:  MOV    -1,R5
227 144242 004767 010734          JSR    PC,TYPLIN      ;type a line of locations
228 144246 010437 007334          MOV    R4,@#ADVADR    ;save next address for subsequent dump
229
230 144252 005737 007466      3%:  TST    @#REPEAT      ;are we repeating?
231 144256 001002          BNE    4$             ;yes, so keep command line
232 144260 110337 007630          MOVW  R3,@#BP        ;no, discard executed commands
233 144264 000167 176756      4%:  JMP    CHGN4
234
235 144270 000167 003122      5%:  JMP    STEP          ;execute single step
```

```

1
2           .SBTTL Parsing Routines
3           ;++
4           ; DOEXPR parses an expression, instruction, directive, or verb, and handles
5           ; it accordingly.  If appropriate, expressions are deposited at the current
6           ; location.  Note that the old value of R2 (which may be a pointer to a symbol
7           ; from the last invocation of DOEXPR) is saved before parsing an expression.
8           ; This is to accomodate the .ESC directive, whose parsing will change the value
9           ; of R2.  The old value is needed for sequences such as <symbol> .ESC N .
10          ; The old value of EXPFLG is also saved, for the same reason.
11          ;--
12
13          .ENABL LSB
14 144274 004767 000212 DOEXPR::JSR PC,GENEXP ;set an expression
15 144300 120027 000072   CMPB R0,#072 ;is next character '?'
16 144304 001453   BEQ 14$ ;yes, ignore expression
17 144306 120027 000075   CMPB R0,#075 ;is next character '='?
18 144312 001450   BEQ 14$ ;yes, ignore expression
19 144314 106437 007326   MTPS @#CURCOM ;has R5 been set?
20 144320 102010   BVC 2$ ;yes, not just a name
21 144322 106437 007332   MTPS @#EXPFLG ;what is name?
22 144326 100063   BPL 8$ ;predefined symbol
23 144330 102003   BVC 9$ ;defined user symbol, proceed
24 144332 005737 007706   TST @#FLAGS1 ;else see if we are in PASS1 mode
25 144336 100031   BPL 3$ ;user defined name has no value in PASS2, error
26
27 144340 010405   9$: MOV R4,R5 ;set name's value
28 144342 120027 000077   2$: CMPB R0,#077 ;is next character '??'
29 144346 001432   BEQ 14$ ;yes, do nothings
30 144350 120027 000134   CMPB R0,#134 ;is it \?
31 144354 001427   BEQ 14$ ;yes, do nothings
32 144356 120027 000137   CMPB R0,#137 ;is it _?
33 144362 001424   BEQ 14$ ;yes, do nothings
34 144364 120027 000033   CMPB R0,#033 ;is it <ESC>?
35 144370 001421   BEQ 14$ ;yes, do nothings
36 144372 106437 007324   MTPS @#LINFLG ;has underline been executed?
37 144376 100420   BMI 6$ ;yes, dump RAM locations
38 144400 004767 010120   JSR PC,ADRCM1 ;check if odd address or not in user RAM
39 144404 105737 007330   TSTB @#TMPMOD ;are we in byte mode?
40 144410 100406   BMI 4$ ;yes
41 144412 010514   MOV R5,@R4 ;deposit data word
42 144414 005237 007756   INC @##ADDR
43 144420 000403   BR 5$
44
45 144422 000167 176536   3$: JMP CMON2
46
47 144426 110514   4$: MOVB R5,@R4 ;deposit byte of data
48 144430 005237 007756   5$: INC @##ADDR
49 144434 000167 177152   14$: JMP CONTIN ;recover last character, parse next command
50
51
52 144440 142737 000010 007324 6$: RICB #F.SAVX,@#LINFLG
53 144446 013704 007336   MOV @#SAVEXP,R4 ;set startins address
54 144452 004767 010524   7$: JSR PC,TYPLIN ;dump a line of locations
55 144456 001366   11$: BNE 14$
56 144460 132737 000100 007330 10$: BITB #F.INST,@#TMPMOD ;avoid twains duplicate addresses
57 144466 001771   BEQ 7$

```

58	144470	004767	010512		JSR	PC, TYPLIN+4	
59	144474	000770			BR	11\$	
60							
61	144476	102403		8\$:	BVS	12\$	is a directive or verb
62	144500	103317			RCC	9\$	is R0, ., B1, etc.
63	144502	000167	005500		JMP	GETINS	else is an instruction
64							
65	144506	000167	002460	12\$:	JMP	DOVERB	handle directives and verbs
66					.DSABL	LSB	
67							

Parsing Routines

```

1          ;++
2          ;   GETEXP parses a whole expression and returns its value in R5.
3          ;   GENEXP parses an expression or recognizes an instruction or symbol.
4          ;--
5
6          .ENABL  LSB
7 144512 152737 000002 007326 GENEXP::BISB  #F,VALU,@#CURCOM; set flag to allow non-valued objects
8 144520 000402          BR          7%          ; such as undefined symbols, directives, etc.
9
10 144522 004767 000452          GETEXP::JSR  PC,EXINI1  ; initialize and set valued expression
11 144526 005037 007332          7%: CLR      @#EXPFLG  ; clear level counter and term flags
12 144532 152737 000004 007326          BISB  #F,UNDF,@#CURCOM; clear undefined expression flag
13 144540 005004          GETEX1: CLR   R4          ; initialize term register
14 144542 120027 000053          CMPB  R0,#053  ; is it +?
15 144546 001464          BEQ   GETOPV  ; yes, accept operator
16 144550 120027 000055          CMPB  R0,#055  ; is it -?
17 144554 001461          BEQ   GETOPV
18 144556 120027 000043          GETERM::CMPB R0,#043  ; is it #?
19 144562 001004          BNE   1%
20 144564 004767 000402          JSR   PC,EXINIT  ; set valued expression
21 144570 004767 166632          JSR   PC,GETNXT  ; set next character
22 144574 120027 000074          1%: CMPB  R0,#074  ; is it <?
23 144600 001522          BEQ   LPAREN    ; yes, do left parenthesis
24 144602 120027 000133          CMPB  R0,#133  ; is it [?
25 144606 001517          BEQ   LPAREN    ; yes, do left parenthesis
26 144610 120027 000047          CMPB  R0,#047  ; is it '?
27 144614 001003          BNE   3%
28 144616 004767 001236          JSR   PC,GETLIT  ; parse literal term
29 144622 000436          BR          GETOPV  ; force valued expression and set operator
30 144624 004767 007526          3%: JSR   PC,NUMCH  ; is it numeric?
31 144630 100403          BMI   2%
32 144632 004767 000450          JSR   PC,GETNMB  ; set number
33 144636 000430          BR          GETOPV  ; force valued expression, set operator
34 144640 004767 007436          2%: JSR   PC,RADCH  ; is it RAD50 character?
35 144644 100476          BMI   13%
36 144646 004767 000576          JSR   PC,GETNAM  ; parse named object
37 144652 106737 007332          MFPS  @#EXPFLG  ; save condition flags
38 144656 106437 007332          MTPS  @#EXPFLG  ; recover actual values
39 144662 100406          BMI   8%
40 144664 102446          BVS  ENDEXP    ; a directive or verb, try to exit
41 144666 103016          BCC  GETOPR    ; predefined symbol, set operator
42 144670 106437 007326          MTPS  @#CURCOM  ; is this a valued expression?
43 144674 102442          BVS  ENDEXP    ; no, we have an instruction, try to exit
44 144676 000412          BR          GETOPR  ; use base value of instruction as data, set op.
45
46 144700 102011          8%: BVC  GETOPR    ; proceed if user symbol if defined
47 144702 142737 000004 007326          BICB  #F,UNDF,@#CURCOM; show we have encountered an undefined symbol
48 144710 005737 007706          TST  @#FLAGS1  ; are we in PASS1?
49 144714 100032          RPL  ENDEXP    ; no, don't allow undefined symbol in expression
50 144716 000402          BR          GETOPR  ; yes, proceed with expression
51
52 144720 004767 000246          GETOPV::JSR  PC,EXINIT  ; initialize data expression
53 144724 004767 166456          GETOPR::JSR  PC,GETLCH  ; recover last character
54 144730 120027 000076          CMPB  R0,#076  ; is it >?
55 144734 001506          BEQ   RPAREN    ; yes, do right parenthesis
56 144736 120027 000135          CMPB  R0,#135  ; is it ]?
57 144742 001507          BEQ   FPAREN    ; yes, do right parenthesis fetch

```

Parsing Routines

58	144744	120027	000041	CMPB	R0,#041	iis character below ?	
59	144750	103412		BLD	6\$	iyes, not an operator	
60	144752	120027	000060	CMPB	R0,#060	iis it above ?	
61	144756	103007		BHIS	6\$	iyes, not an operator	
62	144760	116001	155774	MOVB	PARAMS-040(R0),R1	i set parsing code	
63	144764	042701	177770	BIC	#177770,R1	imask out the garbage	
64	144770	020127	000006	CMP	R1,#6	iis it an operator?	
65	144774	103053		BHIS	DODPER	iyes, do operator	
66							
67	144776	004767	000216	6\$:	JSR	PC,EXDOOP	ino more operators, but do previous one
68							
69	145002	105737	007333	ENDEXP:	TSTB	@#LEVEL	iare we at top level?
70	145006	001015			BNE	13\$	ino, error
71	145010	106437	007326		MTPS	@#CURCOM	iis expression supposed to be valued?
72	145014	102410			RVS	5\$	inot necessarily
73	145016	106437	007332		MTPS	@#EXPFLG	iwas term valued?
74	145022	100401			BMI	9\$	ino, but it was user defined
75	145024	102406			BVS	13\$	ino, error
76	145026	102003		9\$:	BVC	5\$	iwas valued, no problem
77	145030	005737	007706		TST	@#FLAGS1	iare we in PASS1?
78	145034	100101			RPL	12\$	ino, undefined symbol is not allowed
79	145036	000167	166344	5\$:	JMP	GETLCH	i set last character back in R0
80							
81	145042	000167	176116	13\$:	JMP	CMON2	ishow error
82							
83	145046	020627	007410	LPAREN:	CMP	SP:#BOTTOM+10,+22,+2.	iis there room to recurse?
84							;(that was 5 words for future JSRs, 11 for interrupts, and 1 safety)
85	145052	103773			BLD	13\$	ino, parenthesis too deep
86	145054	004767	000112		JSR	PC,EXINIT	iforce upper expression to be valued
87	145060	010546			MOV	R5,-(SP)	isave accumulator
88	145062	113746	007343		MOVB	@#OPERAT,-(SP)	iand previous operator
89	145066	105237	007333		INCB	@#LEVEL	ishow we are going deeper
90	145072	004767	000102		JSR	PC,EXINI1	iinitialize accum. and operator for next level
91	145076	004767	166324		JSR	PC,GETNXT	i get first character of nested expression
92	145102	004767	177432		JSR	PC,GETEX1	i get value of nested expression
93	145106	010504			MOV	R5,R4	inested value becomes operand
94	145110	112637	007343		MOVB	(SP)+,@#OPERAT	irecover operator
95	145114	012605			MOV	(SP)+,R5	iand accumulator
96	145116	004767	166304		JSR	PC,GETNXT	i get character after right paren
97	145122	000700			BR	GETOPR	iand look for next operator
98							
99	145124	004767	000042	DODPER:	JSR	PC,EXINIT	iforce valued expression
100	145130	004767	000064		JSR	PC,EXDOOP	iexecute previous operator
101	145134	004767	166246		JSR	PC,GETLCH	i set new operator
102	145140	110037	007343		MOVB	R0,@#OPERAT	istore it
103	145144	004767	166256		JSR	PC,GETNXT	i set next character
104	145150	000602			BR	GETERM	ilook for a term
105							
106	145152	105337	007333	RPAREN:	DECB	@#LEVEL	iare we at top level?
107	145156	100731			BMI	13\$	iyes, too many right parenthesis
108	145160	000417			BR	EXDOOP	ido last operator inside parenthesis and return
109							
110	145162	004767	177764	FPAREN:	JSR	PC,RPAREN	
111	145166	011505			MOV	@R5,R5	ido a fetch
112	145170	000207			RTS	PC	
113							
114	145172	106437	007326	EXINIT:	MTPS	@#CURCOM	ihas expression already been initialized?

115	145176	102007		BVC	4\$		ives
116	145200	112737	000053	007343	EXINI1:	MOV	#053,@#OPERAT ;set + as default operator
117	145206	005005				CLR	R5 ;clear expression accumulator
118	145210	142737	000002	007326		BICB	#F.VALU,@#CURCOM;show expression is valued
119	145216	000207		4\$:		RTS	PC
120							
121	145220	106437	007326	EXDDOP:		MTPS	@#CURCOM ;if expression is valued, do operator
122	145224	102774				BVS	4\$;not valued
123	145226	113701	007343			MOV	@#OPERAT,R1 ;set operator
124	145232	006301				ASL	R1 ;make a word index
125	145234	000171	145146			JMP	@OPRBL-102(R1) ;execute operator
126							
127	145240	012702	151372	12\$:		MOV	#UNDEFX,R2 ;point to undefined expression message
128	145244	000167	175742			JMP	ERROR ;show error
129						.DSABL	LSB
130							
131	145250	152130		OPRBL:	.WORD	LOGOR	!;
132	145252	143164			.WORD	CMON2	!;
133	145254	143164			.WORD	CMON2	!#
134	145256	143164			.WORD	CMON2	!\$
135	145260	143164			.WORD	CMON2	!%
136	145262	152134			.WORD	LOGAND	!&
137	145264	143164			.WORD	CMON2	!'
138	145266	143164			.WORD	CMON2	!(
139	145270	143164			.WORD	CMON2	!)
140	145272	152142			.WORD	MULTIP	!*
141	145274	152154			.WORD	ADDIT	!+
142	145276	143164			.WORD	CMON2	!;
143	145300	152160			.WORD	SURTR	!-
144	145302	143164			.WORD	CMON2	!.
145	145304	152164			.WORD	DIVID	!/


```

1
2                               ;++
3                               ; GETNMB parses a number and returns all condition flags lowered.
4                               ;--
5
6 145306 005303          GETNMB::DEC    R3           ;save running pointer to peek ahead
7 145310 010346          MOV           R3,-(SP)
8 145312 004767 166000   1%: JSR      PC,GETCH
9 145316 004767 007034   JSR      PC,NUMCH           ;is it 0 through 9?
10 145322 100373        BPL           1%           ;yes, wait for first non-numeric character
11 145324 012603        MOV           (SP)+,R3       ;go back to beginning of number
12 145326 005001        CLR           R1
13 145330 120027 000056   CMPB    R0,#056           ;is first non-numeric character a period?
14 145334 001427        BEQ           2%           ;yes, parse a decimal number
15 145336 004767 006740   JSR      PC,RADCH        ;is it alphabetic?
16 145342 100402        BMI           3%           ;no, parse an octal number
17 145344 000167 175614   4%: JMP      CNON2        ;illegal, label begins with a number
18
19 145350 004767 165742   3%: JSR      PC,GETCH
20 145354 004767 006776   JSR      PC,NUMCH        ;is it 0 through 7 or 8 through 9?
21 145360 100410        BMI           5%           ;no, end of number
22 145362 102770        BVS           4%           ;8 and 9 are illegal in octal numbers
23 145364 162700 000060   SUB     #060,R0          ;make it BCD
24 145370 006301        ASL           R1
25 145372 006301        ASL           R1
26 145374 006301        ASL           R1
27 145376 060001        ADD     R0,R1           ;accumulate digits
28 145400 000763        BR        3%           ;set next digit
29
30 145402 005303          5%: DEC     R3
31 145404 004767 166016   6%: JSR      PC,GETNXT     ;set character after number
32 145410 010104        MOV     R1,R4          ;store value of number
33 145412 000207        RTS     PC
34
35 145414 004767 165676   2%: JSR      PC,GETCH
36 145420 004767 006732   JSR      PC,NUMCH
37 145424 100767        BMI     6%           ;trailing decimal point
38 145426 162700 000060   SUB     #060,R0
39 145432 006301        ASL     R1
40 145434 010146        MOV     R1,-(SP)
41 145436 006301        ASL     R1
42 145440 006301        ASL     R1
43 145442 062601        ADD     (SP)+,R1
44 145444 060001        ADD     R0,R1          ;accumulate digits
45 145446 000762        BR     2%           ;set next digit

```

Parsins Routines

```

1
2          ;++
3          ; GETNAM parses a name and converts it into RAD50 at locations TBLTOP and
4          ; TBLTOP+2. If the name has a corresponding value, it is placed in R4.
5          ; R2 is an index to where the name is stored. It can index either the RAD50
6          ; table in ROM for predefined symbols, or the user symbol table. GETNAM
7          ; returns condition flags to indicate these options as follows:
8          ; V=1 means the name has no value
9          ; N=1 means the name is user defined (not predefined)
10         ; C=1 means the name is a mnemonic, directive, or verb
11         ; N=0, V=0, C=0 means that the name is R0, SP, PC, BI, UR, etc.
12         ; N=0, V=0, C=1 means that the name is an instruction
13         ; N=0, V=1, C=1 means that the name is directive or verb
14         ; N=1, V=1 means that the name is an undefined symbol
15         ; N=1, V=0 means that the name is a user symbol
16         ;--
17
18         .ENABL  LSB
19 145450 112737 000003 007341 GETNAM:MOVB #3,@#COUNT2
20 145456 005001          CLR R1
21 145460 004767 006732 1$: JSR PC,ASCRA0 ;convert ASCII in R0 to RAD50 in R1
22 145464 004767 165626      JSR PC,GETCH ;set next character
23 145470 004767 006606      JSR PC,RADCH ;is it a legal RAD50 character?
24 145474 100431          BMI 2$ ;no, end of name
25 145476 105337 007341      DECB @#COUNT2
26 145502 001366          BNE 1$
27 145504 010137 007320      MOV R1,@#TBLTOP
28 145510 112737 000003 007341 MOVB #3,@#COUNT2
29 145516 005001          CLR R1
30 145520 004767 006672 3$: JSR PC,ASCRA0
31 145524 004767 165566      JSR PC,GETCH
32 145530 004767 006546      JSR PC,RADCH
33 145534 100421          BMI 6$
34 145536 105337 007341      DECB @#COUNT2
35 145542 001366          BNE 3$
36 145544 004767 165546 4$: JSR PC,GETCH ;discard all characters after 6
37 145550 004767 006526      JSR PC,RADCH ;is character still part of name?
38 145554 100373          BPL 4$ ;yes
39 145556 000412          BR 5$
40
41 145560 004767 000264 2$: JSR PC,PADRAD ;pad R1 with spaces
42 145564 010137 007320      MOV R1,@#TBLTOP ;store first word of name
43 145570 012737 000003 007341 MOV #3,@#COUNT2
44 145576 005001          CLR R1
45 145600 004767 000244 6$: JSR PC,PADRAD
46 145604 010137 007322 5$: MOV R1,@#TBLTOP+2
47 145610 005303          DEC R3 ;set next meaningful character
48 145612 004767 165610      JSR PC,GETNXT
49 145616 013700 007320      MOV @#TBLTOP,R0 ;R0, R1 contain converted name
50
51 145622 005002          CLR R2 ;compare name to predefined ones
52 145624 020062 156142 10$: CMP R0,DSYMS(R2)
53 145630 001011          BNE 11$
54 145632 020162 156144      CMP R1,DSYMS+2(R2)
55 145636 001414          BEQ PDSVAL ;set value of predefined symbol
56 145640 020127 006200      CMP R1,#50#50#02 ;might this be a byte mode mnemonic?
57 145644 001003          BNE 11$ ;no

```

```

58 145646 005762 156144          TST    DSYMS+2(R2)    ;and is name the same without trailing 'B'?
59 145652 001406                BEQ    PDSVAL        ;yes, handle byte mode case
60 145654 022222                11%:  CMP    (R2)+,(R2)+    ;advance to next name
61 145656 020227 000770        CMP    R2,#TOPNAM    ;end of table?
62 145662 103760                BLO   10%           ;not yet
63 145664 000167 003670        21%:  JMP    GETSYM     ;no match to predefined symbols
64
65 145670 020227 000054        PDSVAL::CMP R2,#DSYMUR-DSYMS ;now get symbol's value in R4
66 145674 103401                BLO   13%           ;R0 through .
67 145676 002005                BGE   14%           ;B1 or higher
68 145700 006202                13%:  ASR    R2           ;make a word index
69 145702 016204 007732        MOV    %R0(R2),R4    ;R0 through .
70 145706 006302                ASL   R2
71 145710 000405                BR    15%
72
73 145712 020227 000100        14%:  CMP    R2,#DIRVRB-DSYMS
74 145716 103004                BHIS  7%            ;BASIC or higher
75 145720 016204 007700        MOV    BRKFIL-<DSYMUR+4-DSYMS>(R2),R4 ;B1 through B4
76 145724 000257                15%:  CCC                    ;flag a predefined symbol
77 145726 000207                RTS   PC           ;and return
78
79 145730 020227 000300        7%:   CMP    R2,#CMNEMS-DSYMS
80 145734 103003                BHIS  17%           ;CLC or higher
81 145736 000277                SCC                    ;flag a directive or verb
82 145740 000250                CLN
83 145742 000207                RTS   PC           ;and return
84
85 145744 006202                17%:  ASR    R2
86 145746 016204 155440        MOV    CBASES-<<CMNEMS-DSYMS>/2>(R2),R4 ;set value of mnemonic
87 145752 006302                ASL   R2
88 145754 020227 000340        CMP    R2,#MNEMS-DSYMS
89 145760 103003                BHIS  18%           ;HALT or higher
90 145762 012702 000064        MOV    #MNCCC-MNEMS,R2 ;index mnemonic CCC
91 145766 000421                BR    19%
92
93 145770 020127 006200        18%:  CMP    R1,#50*50*02    ;byte mode name?
94 145774 001014                BNE   20%           ;no
95 145776 020027 075131        CMP    R0,#<23*50+27>*50+01    ;SWAB mnemonic?
96 146002 001411                BEQ   20%           ;yes, not byte mode
97 146004 006202                ASR   R2
98 146006 006202                ASR   R2
99 146010 105762 155744        TSTB  PARAMS-<<MNEMS-DSYMS>/4>(R2) ;instruction legal in byte mode
100 146014 100323                BPL   21%           ;no
101 146016 006302                ASL   R2
102 146020 006302                ASL   R2
103 146022 052704 100000        BIS   #100000,R4    ;adjust opcode to byte mode
104 146026 162702 000340        20%:  SUB    #MNEMS-DSYMS,R2 ;adjust index to instruction table
105 146032 000257                19%:  CCC                    ;flag an instruction
106 146034 000261                SEC
107 146036 000207                RTS   PC
108                .DSABL  LSB
109
110 146040 012700 000040        PADRI: MOV    #040,R0
111 146044 004767 006346        JSR   PC,ASCRAID
112 146050 105337 007341        PADRAD: DECB  @#COUNT2    ;routine to pad RAD50 triplet with spaces
113 146054 001371                BNE   PADRI
114 146056 000207                RTS   PC

```

Parsins Routines

```

1
2           ;;;
3           ;   GETLIT parses ASCII literals.
4           ;--
5
6 146060 004767 165326   GETLIT::JSR   PC,GETCHC   ;set next character, don't fold case
7 146064 120027 000015           CMPB   R0,#015   ;is it <CR>?
8 146070 001002           BNE    1$
9 146072 000167 175066           JMP    CMON2    ;error, no character on line
10
11 146076 010004           1$:   MOV    R0,R4   ;store value
12 146100 000167 165322           JMP    GETNXT   ;set next character, return
13
14
15 146104 110137 007330   SETMOD: MOVB   R1,@#TMPMOD
16 146110 004767 165312           JSR    PC,GETNXT
17 146114 005303           DEC    R3
18 146116 105737 007330           TSTB   @#TMPMOD
19 146122 100403           BMI    1$
20 146124 042737 000001 007756   BIC    #1,@#ADDR   ;fix address when leaving byte mode
21 146132 120027 000015   1$:   CMPB   R0,#015   ;is mode command last command on line?
22 146136 001106           BNE    ECOND      ;no
23 146140 113737 007330 007331   MOVB   @#TMPMOD,@#PERMOD   ;yes, set permanent modes
24 146146 000502           BR     ECOND
25
26 146150 152701 000002   EABS:: BISB   #F,ABS,R1
27 146154 000753           BR     SETMOD
28
29
30 146156 012700 000020   ECAN:: MOV    #20,R0
31 146162 105060 007757   1$:   CLRB   BRKFIL-1(R0)
32 146166 077003           SOB    R0,1$
33 146170 005037 007754           CLR    @#WATCH
34 146174 000467           BR     ECOND
35
36
37 146176 142701 000100   EBYT:: BICB   #F,INST,R1
38 146202 152701 000200           BISB   #F,BYTM,R1
39 146206 000736           BR     SETMOD
40
41 146210 142701 000200   EINS:: BICB   #F,BYTM,R1
42 146214 152701 000100           BISB   #F,INST,R1
43 146220 000731           BR     SETMOD
44
45 146222 012737 007320 007462   ECLR:: MOV    #TBLTOP,@#TBLBOT;move symbol table pointer to empty position
46 146230 000451           BR     ECOND
47
48 146232 132737 000002 007327   EDEL:: BITB   #F,VALU,@#LASCOM;was last 'command' valued?
49 146240 001447           EDEL1: BEQ    EERR      ;yes, not just a name
50 146242 106437 007332           MTPS   #EXPFLG
51 146246 100044           BPL    EERR      ;can't delete predefined symbols
52 146250 102443           BVS    EERR      ;can't delete a forward reference
53 146252 162702 000006           SUB    #6,R2      ;back up to specified entry
54 146256 004767 003620           JSR    PC,TYPSYM  ;type current value before deleting
55 146262 004767 003406           JSR    PC,DELSYM  ;delete entry
56 146266 000432           BR     ECOND
57

```

```

58 146270 142701 000300      ENUM:: BICB  #F.INST+F.BYTM,R1
59 146274 000703              BR      SETMOD
60
61
62 146276 010537 007466      EREP:: MOV   R5,@@REPEAT    ;enable repeated execution
63 146302 110337 007630      MOVB  R3,@@BP              ;act like a new line starts after <ESC> N
64 146306 000167 174734      JMP   CMDNA                 ;execute rest of line repeatedly
65
66 146312 142701 000002      ESYM:: BICB  #F.ABS,R1
67 146316 000672              BR      SETMOD
68
69
70 146320 004767 165102      EEXT:: JSR   PC,GETNXT      ;set character after Z
71 146324 004767 176172      JSR   PC,GETEXP           ;parse transfer address
72 146330 000115              JMP   @R5
73
74 146332 016746 040772      EREG:: MOV   TMPMOD,-(SP)   ;for byte mode
75 146336 042767 000200 040764 BIC   #F.BYTM,TMPMOD      ;
76 146344 004767 003360      JSR   PC,TYPREG          ;type contents of registers and watchpoint
77 146350 012667 040754      MOV   (SP)+,TMPMOD       ;restore flass
78 146354 000167 175224      ECON: JMP   CMON5
79 146360 000167 174600      EERR: JMP   CMON2
80
81 146364 106437 007324      ENOV:: MTPS @#LINFLG      ;see if underline has been executed
82 146370 100373              BPL   EERR                ;it hasn't--error-- missing parameter
83 146372 010502              MOV   R5,R2              ;set final address of source memory block
84 146374 013701 007336      MOV   @@SAVEXP,R1        ;set starting address of source
85 146400 142737 000010 007324 BICB  #F.SAUX,@@LINFLG    ;show that saved expression has been used
86 146406 005737 007706      TST   @@FLAGS1          ;are we in PASS1 mode?
87 146412 100760              BMI   ECON               ;yes, don't execute move
88 146414 020201              CMP   R2,R1              ;is final address below starting address?
89 146416 103760              BLD   EERR               ;yes, show error
90 146420 105737 007330      TSTB  @@TMPMOD          ;are we in byte mode?
91 146424 100404              BMI   1$                ;yes
92 146426 042701 000001      BIC   #1,R1              ;else force starting address to even boundary
93 146432 052702 000001      BIS   #1,R2              ;and final address to odd boundary
94 146436 013704 007756      1$:  MOV   @@ADDR,R4        ;set starting address of destination block
95 146442 060204              ADD   R2,R4              ;calculate final address from length of source
96 146444 160104              SUB   R1,R4
97 146446 010400              MOV   R4,R0              ;save result for later
98 146450 105737 007330      TSTR  @@TMPMOD          ;are we in byte mode?
99 146454 100401              BMI   2$                ;yes
100 146456 005304              DEC   R4                 ;adjust result for previous BIC and BIS
101 146460 004767 006044      2$:  JSR   PC,ADRCH2       ;check that final destination address is legal
102 146464 004767 006034      JSR   PC,ADRCH1         ;set starting destination address and check it
103                               ;now see if destination addresses straddle protected RAM
104 146470 020027 006500      CMP   R0,##UR           ;is final addr below lower instance of prot RAM
105 146474 103415              BLD   3$                ;yes, destination range OK
106 146476 020427 020000      CMP   R4,#20000         ;is start addr above upper instance of prot RAM
107 146502 103012              BHS   3$                ;yes, destination range OK
108 146504 020427 010000      CMP   R4,#10000         ;does destin overlap lower instance of prot RAM
109 146510 103403              BLD   4$                ;yes, range is unacceptable
110 146512 020027 020000      CMP   R0,#20000        ;is final addr below upper instance of prot RAM
111                               ;this is a trick test -- we know R0 is not
112                               ;within the range #UR+10000--17777 from ADRCH2
113 146516 103404              BLD   3$                ;yes, range doesn't overlap upper instance
114 146520 012702 154476      4$:  MOV   #USERAM,R2       ;point to user RAM error message
    
```

Parsing Routines

115	146524	000167	174462		JMP	ERROR	i>print error message
116							
117	146530	020401		3%:	CMP	R4,R1	i>now see if destination block is above or below
118	146532	101004			BHI	5%	i>source -- above means use backward transfer
119	146534	112124		6%:	MOVB	(R1)+,(R4)+	i>else do forward transfer
120	146536	020102			CMP	R1,R2	
121	146540	101775			BLOS	6%	i>do entire block
122	146542	000704			BR	ECON	
123							
124	146544	005202		5%:	INC	R2	i>adjust pointers for autodecrement
125	146546	005200			INC	R0	
126	146550	114240		7%:	MOVB	-(R2),-(R0)	i>perform backward transfer
127	146552	020201			CMP	R2,R1	
128	146554	101375			BHI	7%	i>do entire block
129	146556	000676			BR	ECON	
130							
131	146560	106437	007324	EFND::	MTPS	@#LIMFLG	i>see if underline has been executed
132	146564	100275			BPL	EERR	i>it hasn't--error--missing parameter
133	146566	010537	007334		MOV	R5,@#ADVADR	i>save upper search range limit
134	146572	142737	000010	007324	BICB	#F,SAVX,@#LIMFLG	i>show that saved expression has been used
135	146600	004767	164622		JSR	PC,GETNXT	i>set character after F
136	146604	004767	175712		JSR	PC,GETEXP	i>set value to be searched for
137	146610	024646			CMP	-(SP),-(SP)	i>make space for two scratch location on stack
138	146612	010516			MOV	R5,(SP)	i>save value to be searched for on stack
139	146614	005066	000002		CLR	2(SP)	i>initialize complement of search mask to zero
140	146620	120027	000137		CMPB	R0,#137	i>is next character an underscore?
141	146624	001010			BNE	1%	i>no, optional mask parameter not present
142	146626	004767	164574		JSR	PC,GETNXT	i>set character after underscore
143	146632	004767	175664		JSR	PC,GETEXP	i>set search mask
144	146636	010566	000002		MOV	R5,2(SP)	i>store it on stack
145	146642	005166	000002		COM	2(SP)	i>complement mask
146	146646	013704	007336	1%:	MOV	@#SAVEXP,R4	i>set starting address of search range
147	146652	105737	007330	2%:	TSTB	@#TMPMOD	i>are we in byte mode
148	146656	100430			BMI	4%	i>yes, perform bitwise search
149	146660	042704	000001		BIC	#1,R4	i>for word search, make sure address is even
150	146664	011401			MOV	@R4,R1	i>set a word
151	146666	011600			MOV	(SP),R0	i>set search value
152	146670	074001			XOR	R0,R1	i>compare values
153	146672	046601	000002		BIC	2(SP),R1	i>ignore bits that are masked out
154	146676	001005		5%:	BNE	3%	i>no match
155	146700	004767	006344		JSR	PC,TYPADR	i>type address of matched location
156	146704	004767	006204		JSR	PC,TYPL01	i>type contents, but not as an instruction
157	146710	000405			BR	6%	i>TYPL01 already incremented R4
158	146712	005204		3%:	INC	R4	i>advance to next location
159	146714	105737	007330		TSTB	@#TMPMOD	i>are we in byte mode?
160	146720	100401			BMI	6%	i>yes, advance by one
161	146722	005204			INC	R4	i>else advance a whole word
162	146724	020437	007334	6%:	CMP	R4,@#ADVADR	i>see if entire range has been checked
163	146730	101750			BLOS	2%	i>continue searching
164	146732	022626			CMP	(SP)+,(SP)+	i>discard stacked values
165	146734	000167	174652		JMP	CONTIN	i>recover character in R0, parse next command
166							
167	146740	111401		4%:	MOVB	@R4,R1	i>set a byte
168	146742	011600			MOV	(SP),R0	i>set search value
169	146744	074001			XOR	R0,R1	i>compare values
170	146746	046601	000002		BIC	2(SP),R1	i>ignore bits that are masked out
171	146752	042701	177400		BIC	#177400,R1	i>ignore discrepancies in high byte

Parsing Routines

```

172 146756 000747          BR      5$
173
174
175 146760 004767 164540      EMOD:: JSR      PC,CRLF      ;start a new line
176 146764 013704 007706      MOV      @#FLAGS,R4      ;set a copy of some mode flags
177 146770 012702 156256      MOV      #DIRVB1,R2      ;point to PASS1 command
178 146774 032704 100000      BIT      #F.PAS1,R4      ;see if we're in PASS1 or PASS2
179 147000 004767 000132      JSR      PC,1$          ;print proper mode and point to NOPROT
180 147004 032704 004000      BIT      #F.HOST,R4      ;see if HOST or NOHOST is active
181 147010 004767 000122      JSR      PC,1$          ;print proper mode
182 147014 113704 007330      MOVB     @#THPMOD,R4      ;set a copy of the rest of the mode flags
183 147020 012702 156366      MOV      #DIRVBI,R2      ;point to INSTRU command
184 147024 132704 000100      BITB     #F.INST,R4      ;see if instruction mode is active
185 147030 001402          BEQ      4$            ;branch if not
186 147032 004767 000100      JSR      PC,1$          ;print proper mode
187 147036          4$:
188 147036 012702 156346      MOV      #DIRVBA,R2      ;point to ABSOLU command
189 147042 132704 000002      BITB     #F.ABS,R4      ;see if absolute or symbolic mode is active
190 147046 001002          BNE      6$            ;absolute mode
191 147050 012702 156412      MOV      #DIRVBS,R2      ;point to symbolic mode
192 147054 004767 000056      6$: JSR      PC,1$          ;print proper mode
193 147060 012702 156362      MOV      #DIRVBH,R2      ;point to BYTE command
194 147064 132704 000200      BITB     #F.BYTH,R4      ;see if byte mode is active
195 147070 001402          BEQ      7$            ;it isn't, print nothings
196 147072 004767 164676      JSR      PC,PRINT2      ;else print BYTE command
197 147076 012702 156376      7$: MOV      #DIRVBN-4,R2      ;point to word mode
198 147102 132704 000300      BITB     #F.INST!F.BYTH,R4 ;if not byte or instruction then word
199 147106 001002          BNE      20$           ;
200 147110 004767 000022      JSR      PC,1$
201 147114 012702 156432      20$: MOV      #DIRVBT+14,R2 ;point to vtmode
202 147120 032767 004000 040203 BIT      #F.VT,PERMOD ;see if set
203 147126 004767 000004      JSR      PC,1$          ;do show wich vt mode
204 147132 000167 177216      JMP      ECDN          ;and return
205
206 147136 106746          1$: MFPS     -(SP)        ;save Z flag
207 147140 106416          MTPS     (SP)        ;set back a copy
208 147142 001001          BNE      2$            ;print command pointed to by R2
209 147144 022222          CMP      (R2)+,(R2)+   ;else advance to next command
210 147146 004767 164622      2$: JSR      PC,PRINT2 ;print command, advance R2
211 147152 012700 004440      MOV      #400#011+040,R0 ;<space> <tab>
212 147156 004767 164346      JSR      PC,CHROUT
213 147162 106426          MTPS     (SP)+        ;recover Z flag
214 147164 001401          BEQ      3$            ;if we advanced before, don't do it again
215 147166 022222          CMP      (R2)+,(R2)+ ;skip over next command
216 147170 000207          3$: RTS      PC

```

```

1
2
3
4 147172 162702 000100 DOVERB::SUB #DIRVRB-DSYMS,R2;remove offset from index to verb
5 147176 006202 ASR R2 ;make it a word index
6 147200 020227 000042 CMP R2,#VRBDS1-VRBDSP ;is this verb an <ESC> seq. replacement
7 147204 103403 BLO 1% ;no
8 147206 005303 DEC R3 ;make it so that we can use previous routines
9 147210 013701 007330 MOV @#TMPMOD,R1 ;from the escape sequence handler
10 147214 000172 147220 1%: JMP @VRBDSP(R2) ;dispatch to proper handling routine
11
12
13 147220 VRBDSP::
14 147220 147320 .WORD HELP
15 147222 147432 .WORD GO
16 147224 147416 .WORD STEP
17 147226 150202 .WORD PASS1
18 147230 150212 .WORD PASS2
19 147232 150222 .WORD HOST
20 147234 150306 .WORD NOHOST
21 147236 150504 .WORD TALK
22 147240 150442 .WORD LOAD
23 147242 143164 .WORD CMON2 ;.START
24 147244 143216 .WORD CMON3 ;.END
25 147246 147776 .WORD $.EVEN
26 147250 147700 .WORD $.ASCII
27 147252 147466 .WORD $.BYTE
28 147254 147560 .WORD $.WORD
29 147256 147634 .WORD $.BLKB
30 147260 147654 .WORD $.BLKW
31 147262 146150 VRBDS1: .WORD EABS ;ABSOLU point to the start of escape
32 147264 146156 .WORD ECAN ;CANCEL
33 147266 136752 .WORD $HALT ;EXIT
34 147270 146176 .WORD EBYT ;BYTE
35 147272 146210 .WORD EINS ;INSTRU
36 147274 146222 .WORD ECLR ;CLEAR$
37 147276 147444 .WORD DELETE ;DELETE
38 147300 146270 .WORD ENUM ;WORD
39 147302 150012 .WORD $RPEAT ;REPEAT
40 147304 146312 .WORD ESYN ;SYMBOL
41 147306 146760 .WORD EMOD ;SHOWMO
42 147310 146332 .WORD EREG ;SHOWRE
43 147312 152040 .WORD TYPTBL ;SHOWSY
44 147314 147374 .WORD VTON ;VTON
45 147316 147406 .WORD VTOFF ;VTOFF
46
47 147320 012702 156242 HELP:: MOV #DIRVRB,R2 ;type the verb/directive table on the console
48 147324 004767 164174 2%: JSR PC,CRLF ;start a new line
49 147330 112737 000010 007341 MOVB #8,@#COUNT2 ;put 8 commands on a line
50 147336 004767 164432 1%: JSR PC,PRINT2 ;print a command
51 147342 020227 156442 CMP R2,#CMNEMS ;have we done the whole table?
52 147346 103010 BHIS 3% ;yes
53 147350 105337 007341 DECB @#COUNT2 ;have we finished a line?
54 147354 001763 BEB 2% ;yes
55 147356 012700 004440 MOV #400#011+040,R0 ;type <space> <tab>
56 147362 004767 164142 JSR PC,CHROUT
57 147366 000763 BR 1%

```



```

58
59 147370 000167 174216      3$:   JMP      CONTIN      ;recover character in R0, continue parsing line
60
61 147374 052767 004000 037726 VTGN:  BIS      #F,VT,TMPMOD      ;turn vt mode on
62 147402 000167 174176      VTJMP: JMP      CMONS
63
64 147406 042767 004000 037714 VTOFF: BIC      #F,VT,TMPMOD      ;turn vt off
65 147414 000772                      BR       VTJMP
66
67 147416 052737 000200 007706 STEP:: BIS      #F,SST,@#FLAGS1 ;show we are single stepping
68 147424 152737 000001 007324      BISR      #F,SST1,@#LINFLG
69 147432 023727 007746 006500 GD::  CMP      @##SP,##UR      ;is user SP in his RAM?
70 147440 000167 172256      JMP      $G01         ;use keypad monitor GD routine
71
72
73 147444 005203                      DELETE::INC      R3          ;counteract earlier DEC R3 in this case
74 147446 004767 175040                      JSR      PC,GENEXP      ;set symbol name to be deleted
75 147452 005303                      DEC      R3            ;now prepare to use <ESC> sequence routine
76 147454 132737 000002 007326      BITB     #F,VALU,@#CURCOM ;replaces test on LASCOM in EDEL
77 147462 000167 176552                      JMP      EDEL1         ;go to routine to delete a symbol
78
79
80 147466 013737 007756 007334 $.BYTE::MOV @##ADDR,@#ADVADR ;set a scratch copy of current address
81 147474 004767 175022      1$:   JSR      PC,GETEXP      ;set a value to be deposited
82 147500 113746 007330      MOVB     @#TMPMOD,-(SP) ;fool ADRCHK into thinking we are in byte mode
83 147504 052737 000200 007330      BIS      #F,BYTM,@#TMPMOD
84 147512 013704 007334      MOV      @#ADVADR,R4      ;set running copy of current address
85 147516 004767 005006      JSR      PC,ADRCH2       ;check that it's not in protected RAM
86 147522 112637 007330      MOVB     (SP)+,@#TMPMOD  ;restore proper mode bits
87 147526 110514      MOVB     R5,R4           ;deposit a byte
88 147530 005237 007334      INC      @#ADVADR        ;advance running address
89 147534 120027 000054      CMPEB   R0,#054         ;is next character a comma?
90 147540 001003      BNE      2$             ;no
91 147542 004767 163660      JSR      PC,GETNXT      ;set character after comma
92 147546 000752      BR       1$             ;deposit another byte
93
94 147550 013737 007334 007756 2$:   MOV      @#ADVADR,@##ADDR ;update current address
95 147556 000571      BR       VCON
96
97
98 147560 013737 007756 007334 $.WORD::MOV @##ADDR,@#ADVADR ;set a scratch copy of current address
99 147566 004767 174730      1$:   JSR      PC,GETEXP      ;set a value to be deposited
100 147572 013704 007334      MOV      @#ADVADR,R4      ;set running address
101 147576 004767 004726      JSR      PC,ADRCH2       ;check that it's not in protected RAM
102 147602 010524      MOV      R5,(R4)+        ;deposit a word
103 147604 010437 007334      MOV      R4,@#ADVADR      ;advance running copy of current address
104 147610 120027 000054      CMPEB   R0,#054         ;is next character a comma?
105 147614 001003      BNE      2$             ;no
106 147616 004767 163604      JSR      PC,GETNXT      ;set character after comma
107 147622 000761      BR       1$             ;deposit another word
108
109 147624 013737 007334 007756 2$:   MOV      @#ADVADR,@##ADDR ;update current address
110 147632 000543      BR       VCON
111
112
113 147634 004767 174662      $.BLKB::JSR PC,GETEXP      ;set argument
114 147640 106437 007326      MTPS    @#CURCOM        ;check if it contained undefined symbols

```

115	147644	001146		BNE	VERR1		iit did, refuse .BLKB directive	
116	147646	060537	007756	ADD	R5,@##ADDR		iadd it to current address	
117	147652	000533		BR	VCON			
118								
119								
120	147654	004767	174642	\$.BLKW::JSR	PC,GETEXP		i set argument	
121	147660	106437	007326	MTPS	@#CURCOM		icheck if it contained undefined symbols	
122	147664	001136		BNE	VERR1		iit did, refuse .BLKW directive	
123	147666	010504		MOV	R5,R4		imake a scratch copy	
124	147670	006304		ASL	R4		idouble it	
125	147672	060437	007756	ADD	R4,@##ADDR		iadd it to current address	
126	147676	000521		BR	VCON			
127								
128								
129	147700	110037	007345	\$.ASCII::MOV	R0,@#DELIM		isave delimiter	
130	147704	013737	007756	007334	MOV	@##ADDR,@#ADVADR	i set a scratch copy of current address	
131	147712	004767	163474	1\$:	JSR	PC,GETCHC	i set next character, don't fold to upper case	
132	147716	120037	007345	CMPB	R0,@#DELIM		i have we reached delimiter yet?	
133	147722	001417		BEQ	2\$		iyes, done	
134	147724	113746	007330	MOV	@#TMPMOD,-(SP)		ifool ADDRCHK into thinking we are in byte mode	
135	147730	052737	000200	007330	BIS	#F,BYTH,@#TMPMOD		
136	147736	013704	007334	MOV	@#ADVADR,R4		i set running copy of current address	
137	147742	004767	004562	JSR	PC,ADRCHK2		icheck that it's not in protected RAM	
138	147746	112637	007330	MOV	(SP)+,@#TMPMOD		irestore proper mode bits	
139	147752	110014		MOV	R0,@R4		ideposit a byte	
140	147754	005237	007334	INC	@#ADVADR		iadvance running address	
141	147760	000754		BR	1\$			
142								
143	147762	013737	007334	007756	2\$:	MOV	@#ADVADR,@##ADDR	iupdate current address
144	147770	004767	163432	JSR	PC,GETNXT		i set character after delimiter	
145	147774	000462		BR	VCON			
146								
147								
148	147776	005237	007756	\$.EVEN::INC	@##ADDR			
149	150002	042737	000001	007756	BIC	#1,@##ADDR		
150	150010	000454		BR	VCON			
151								
152								
153	150012	005203		\$.REPEAT::INC	R3		icounteract earlier DEC R3 in this case	
154	150014	004767	174502	JSR	PC,GETEXP		i set repeat count	
155	150020	106437	007326	MTPS	@#CURCOM		icheck if it contained undefined symbols	
156	150024	001056		BNE	VERR1		iit did, refuse REPEAT command	
157	150026	005705		TST	R5		i see if a repeat of zero was given	
158	150030	001450		BEQ	RERR		irefuse zero	
159	150032	120027	000054	CMPB	R0,#054		icheck that a comma follows	
160	150036	001043		BNE	VERR		ino comma, error, missing parameter	
161	150040	000167	176232	JMP	EREP		iso to repeat command line routine	
162								
163								
164								
165	150044	005203		.ENABL	LSB			
166	150046	004767	000006	MOVE::	INC	R3	icounteract earlier DEC R3 in this case	
167	150052	005303		JSR	PC,1\$		i set address pair	
168	150054	000167	176304	DEC	R3		iprepare to use <ESC> sequence routine	
169				JMP	ENOV		iso to routine to move a memory block	
170	150060	004767	174436	1\$:	JSR	PC,GETEXP	i set first address	
171	150064	120027	000137	CMPB	R0,#137		iis it followed by an underline?	

Directive and Verb Handling Routines

```

172 150070 001026          RNE  VERR      ;no, missing parameter
173 150072 010537 007336  MOV  R5,@$SAVEP ;save the first address
174 150076 152737 000010 007324  BISB  $F,SAUX,@$LINFLG;show we have saved something in SAVEP
175 150104 004767 163316  JSR  PC,GETNXT  ;set character after underline
176 150110 000167 174406  JMP  GETEXP     ;set second address
177
178
179 150114 005203          FIND:: INC  R3      ;counteract earlier DEC R3 in this case
180 150116 004767 177736  JSR  PC,1$     ;set address pair
181 150122 120027 000054  CMPB R0,#054   ;is it followed by a comma?
182 150126 001007          BNE  VERR      ;no, missing parameter
183 150130 000167 176424  JMP  EFND      ;go to routine to find a value (search memory)
184      .DSABL  LSB
185
186
187 150134 052737 040000 007706  NOPROT::BIS  $F,PROT,@$FLAGS;turn off monitor scratch protection
188                          ;fall into VCON
189 150142 000167 173450  VCON:  JMP  CMDN6  ;continue parsing
190
191 150146 000167 173012  VERR:  JMP  CMDN2  ;general error
192
193 150152 012702 151412  RERR:  MOV  $REPDF0,R2 ;repeat of zero error
194 150156 000167 173030  JMP  ERROR
195
196 150162 012702 151372  VERR1: MOV  $UNDEFX,R2  ;undefined expression error
197 150166 000167 173020  JMP  ERROR
198
199 150172 042737 040000 007706  PROTEC::BIC  $F,PROT,@$FLAGS;turn on monitor scratch protection
200 150200 000760          BR   VCON
201
202 150202 052737 100000 007706  PASS1::BIS  $F,PAS1,@$FLAGS;show we are in PASS1 mode
203 150210 000754          BR   VCON
204
205 150212 042737 100000 007706  PASS2::BIC  $F,PAS1,@$FLAGS;show we are in PASS2 mode
206 150220 000750          BR   VCON
207
208
209 150222 032737 004000 007706  HOST:: BIT  $F,HOST,@$FLAGS;are we already in host mode?
210 150230 001346          BNE  VERR      ;yes, error
211 150232 105037 007672  CLRB  @$HFP    ;initialize host input buffer
212 150236 105037 007673  CLRB  @$HSTFLG ;hand host flags
213 150242 004767 164656  JSR  PC,SETAX2 ;match aux port baud rate to console port
214 150246 052737 004000 007706  BIS  $F,HOST,@$FLAGS;set host mode
215 150254 105737 177550  TSTB  @$ARBUFF ;discard any garbage in the port
216 150260 012737 134540 000120  MOV  $HOSTIN,@$AUXIN ;hook up host input service routine
217 150266 112737 000046 007711  MOVB  #046,@$AUXFLG ;save a new copy of A$CREG value
218 150274 112737 000046 177452  MOVB  #046,@$A$CREG ;enable aux port receiver interrupt
219 150302 000167 173304  JMP  CONTIN   ;recover character in R0
220
221
222 150306 105737 007673  NCHOST::TSTB @$HSTFLG ;are we in host load mode?
223 150312 100715          BMI  VERR      ;yes, refuse NCHOST command
224 150314 105037 007673  CLRB  @$HSTFLG ;clear host flags
225 150320 112737 000042 007711  MOVB  #042,@$AUXFLG ;save a copy of new A$CREG value
226 150326 112737 000042 177452  MOVB  #042,@$A$CREG ;disable aux port receiver interrupt
227 150334 042737 004000 007706  BIC  $F,HOST,@$FLAGS;show we are not in host mode
228 150342 000677          BR   VCON

```

Directive and Verb Handling Routines

```

229
230
231 150344 056 123 124 $.START: .ASCII /.START/ ;strings to match for enablins of load mode
    150347 101 122 124
232 150352 015 012 OPTION: .BYTE 015,012 ;<CR>, <LF>
233 150354 105 040 050 .ASCIZ /E (EXIT), R (RESUME), B (SEND BREAK), ^A (SEND ^A) ??
    150357 105 130 111
    150362 124 051 054
    150365 040 122 040
    150370 050 122 105
    150373 123 125 115
    150376 105 051 054
    150401 040 102 040
    150404 050 123 105
    150407 116 104 040
    150412 102 122 105
    150415 101 113 051
    150420 054 040 136
    150423 101 040 050
    150426 123 105 116
    150431 104 040 136
    150434 101 051 040
    150437 077 000
234 150441 000 .BYTE 0 ;because the linker won't zero fill
235
236 .ENABL LSR
237 150442 BLOAD::
238 150442 032737 004000 007706 LOAD:: BIT #F.HOST,@#FLAGS1;is host mode enabled?
239 150450 001410 BEQ LERR ;no, refuse LOAD command
240 150452 105737 007673 TSTB @#HSTFLG ;are we in host load mode?
241 150456 100405 BMI LERR ;yes, refuse LOAD command
242 150460 152737 000100 007673 BISR #F.LDST,@#HSTFLG;show we want to start loading upon .START
243 150466 005002 CLR R2 ;initialize index to match .START strings
244 150470 000417 BR TALK1
245
246 150472 032737 020000 007706 LERR: BIT #F.TBAS,@#FLAGS1;are we running Tiny Basic?
247 150500 001622 BEQ VERR ;no
248 150502 000207 12$: RTS PC ;yes, return to BASIC
249
250 150504 032737 004000 007706 TALK:: BIT #F.HOST,@#FLAGS1;is host mode enabled?
251 150512 001615 BEQ VERR ;no, refuse LOAD command
252 150514 105737 007673 TSTB @#HSTFLG ;are we in host load mode?
253 150520 100612 BMI VERR ;yes, refuse TALK command
254 150522 142737 000100 007673 BICB #F.LDST,@#HSTFLG;show we want to ignore .START
255
256 150530 104427 000300 TALK1: MTPS #300 ;disable interrupts except console break
257 150534 005737 177562 TST @#C#RBUF ;clear any garbage in the ports
258 150540 105737 177550 TSTR @#A#RBUF
259 150544 004767 162754 JSR PC;CRLF ;start a new line
260 150550 112737 000011 177444 MOVB #011,@#P#PORC ;turn on LED during virtual terminal operation
261
262 150556 004767 000364 6$: JSR PC;CLRAPE ;clear aux port error
263 150562 105737 177560 1$: TSTB @#C#RCSR ;user type a character?
264 150566 100017 BPL 2$ ;no
265 150570 113701 177552 MOVB @#A#SREG,R1 ;check host port status
266 150574 000240 NOP ;*** place holder for the above line ***
267 150576 006201 ASR R1

```

268	150600	103012		BCC	2%		;CTS not asserted or transmitter not empty
269	150602	013700	177562	MOV	@#C#RBUF,R0		;get character from console
270	150606	100407		BMI	2%		;ignore error characters
271	150610	042700	177600	BIC	#177600,R0		;make sure no parity bit set
272	150614	120027	000001	CMPB	R0,#1		;is it ^A ?
273	150620	001450		BEQ	3%		;yes, exit talk mode
274	150622	110037	177450	9%: MOV	R0,@#A#XBUF		;no, output character to host
275							
276	150626	113701	177552	2%: MOV	@#A#SREG,R1		;check host port status
277	150632	132701	000002	BITB	#002,R1		;do we have a character?
278	150636	001751		BEQ	1%		;no
279	150640	113700	177550	MOV	@#A#RBUF,R0		;get character from host
280	150644	042700	177400	BIC	#177400,R0		;prevent sign extend
281	150650	132701	000170	BITR	#170,R1		;is it break or an error?
282	150654	001340		BNE	6%		;yes, ignore character
283	150656	132737	000100 007673	BITB	#F.LDST,@#HSTFLG		;are we looking for .START?
284	150664	001420		BEQ	5%		;no
285	150666	010001		MOV	R0,R1		;make a scratch copy of character
286	150670	005702		TST	R2		;don't fold the period
287	150672	001402		BEQ	14%		
288	150674	042701	000040	BIC	#040,R1		;fold lower case to upper case
289	150700	120162	150344	14%: CMPB	R1,#.STRT(R2)		;does character match?
290	150704	001007		BNE	4%		;no, so back to the '.' in .START and try again
291	150706	005202		INC	R2		
292	150710	020227	000006	CMP	R2,#6		;have we matched entire strings?
293	150714	103404		BLO	5%		;no, continue with next character
294	150716	110037	177566	MOV	R0,@#C#XBUF		;send the character without delay!
295	150722	000470		BR	7%		;a match, start loading
296							
297	150724	005002		4%: CLR	R2		
298	150726	105737	177564	5%: TSTB	@#C#XCSR		;is console port ready for a character
299	150732	100313		BPL	1%		;no, drop character
300	150734	110037	177566	MOV	R0,@#C#XBUF		;yes, echo character to console
301	150740	000710		BR	1%		
302							
303	150742	012701	150352	3%: MOV	#OPTIDN,R1		;point to message
304	150746	004767	163002	JSR	PC,PRINTA		;print ASCII string on console
305	150752	105737	177560	8%: TSTR	@#C#RCSR		;wait for a character
306	150756	100375		BPL	8%		
307	150760	013700	177562	MOV	@#C#RBUF,R0		;get character from console
308	150764	100772		BMI	8%		;character caused an error, try again
309	150766	042700	177600	BIC	#177600,R0		;make sure no parity bit set
310	150772	110037	177566	MOV	R0,@#C#XBUF		;echo character to console
311	150776	120027	000001	CMPB	R0,#1		;is it ^A?
312	151002	001707		BEQ	9%		;yes, send it to host
313	151004	042700	000040	BIC	#040,R0		;fold lower case to upper
314	151010	120027	000102	CMPB	R0,#102		;is it B?
315	151014	001012		BNE	10%		;no
316	151016	012701	100000	MOV	#100000,R1		
317	151022	112737	000057 177452	MOV	#057,@#A#CREG		;send break
318	151030	077101		11%: SOB	R1,.		;wait a while
319	151032	112737	000047 177452	MOV	#047,@#A#CREG		
320	151040	000650		BR	1%		;resume two-way communication
321							
322	151042	120027	000105	10%: CMPB	R0,#105		;is it E?
323	151046	001245		BNE	1%		;no, resume communication
324	151050	112737	000010 177444	MOV	#010,@#P#PORC		;turn off LED indicator

Directive and Verb Handling Routines

```

325 151056 112737 000046 177452      MOVB  #046,@#A#CREG  ;disable aux port transmitter interrupt
326 151064 106427 000000              MTPS  #000           ;reenable interrupts
327 151070 032737 020000 007706      BIT   #F.TBAS,@#FLAGS;are we running Tiny Basic?
328 151076 001201              13#: BNE  12#         ;yes, return to BASIC
329 151100 000167 172112              JMP  CMON3          ;return to console monitor
330
331 151104 152737 000200 007673 7#:  BISB  #F.LOAD,@#HSTFLG;set host load mode
332 151112 112737 000010 177444      MOVB  #010,@#P#PORC  ;turn off LED indicator
333 151120 112737 000046 177452      MOVB  #046,@#A#CREG  ;disable aux port transmitter interrupt
334 151126 106427 000000              MTPS  #000           ;reenable interrupts A.S.A.P.
335 151132 032737 020000 007706      BIT   #F.TBAS,@#FLAGS;are we running Tiny Basic?
336 151140 001356              BNE  13#         ;yes, return to BASIC
337 151142 000167 172056              JMP  CMON31         ;return to console monitor
338
339 151146 112737 000067 177452 CLRAPE: MOVB #067,@#A#CREG ;clear aux port error flag
340 151154 112737 000047 177452      MOVB  #047,@#A#CREG  ;asser DTR, RTS, enable aux port xmit and rcv
341 151162 000207              RTS   PC
342              .DSABL  LSB

```

```

1
2                .SBTTL Symbol and Label Handlins Routines
3
4                .ENABL LSB
5 151164 132737 000002 007327 D0SYMB:BITB  $F,VALU,@#LASC0M;check that last 'command' was not valued
6 151172 001423                BEQ      3$      ;was evaluated, error
7 151174 010237 007334                MOV      R2,@#ADVADR ;store index here
8 151200 004767 162222                JSR      PC,GETNXT   ;set next meaningful character after =
9 151204 106437 007332                MTPS    @#EXPLG     ;see what kind of symbol we are definins
10 151210 100434                BMI     1$         ;user defined name
11 151212 103413                BCS     3$         ;was a mnemonic, directive, or verb--error
12 151214 004767 173302                JSR      PC,GETEXP   ;set value to be equated
13 151220 106437 007326                MTPS    @#CURCOM    ;check if it contained undefined symbols
14 151224 001010                BNE     13$        ;it did, refuse symbol definition
15 151226 013702 007334                MOV      @#ADVADR,R2 ;set back index
16 151232 020227 000054                CMP      R2,#DSYMUR-DSYMS;what is name?
17 151236 103407                BLO     4$         ;R0 through .
18 151240 101012                BHI     5$         ;B1 through B4
19 151242 000167 171716                3$:    JMP      CMON2 ;else show error
20
21 151246 012702 151372                13$:   MOV      #UNDEFX,R2 ;undefined expression error
22 151252 000167 171734                JMP      ERROR
23
24
25 151256 006202                4$:    ASR      R2      ;make a word index
26 151260 010562 007732                MOV      R5,#R0(R2) ;set appropriate register, watchpoint, or ','
27 151264 000404                BR      6$
28
29
30 151266 042705 000001                5$:    BIC      #1,R5     ;breakpoints have even addresses
31 151272 010562 007700                MOV      R5,BRKFIL-<DSYMUR+4-DSYMS>(R2) ;set appropriate breakpoint
32 151276 000167 172314                6$:    JMP      CMON6
33
34 151302 102021                14$:   BVC      2$         ;symbol already defined
35 151304 013746 007320                MOV      @#TBLTOP,-(SP) ;save name of symbol we are definins
36 151310 013746 007322                MOV      @#TBLTOP+2,-(SP)
37 151314 004767 173202                JSR      PC,GETEXP   ;set value to be set
38 151320 106437 007326                MTPS    @#CURCOM    ;check if it contained undefined symbols
39 151324 001006                BNE     12$        ;it did, ignore symbol definition
40 151326 010504                MOV      R5,R4      ;set value to be set
41 151330 012637 007322                MOV      (SP)+,@#TBLTOP+2;recover name
42 151334 012637 007320                MOV      (SP)+,@#TBLTOP
43 151340 000446                BR      9$
44
45 151342 022626                12$:   CMP      (SP)+,(SP)+ ;POP saved name
46 151344 000754                BR      6$         ;quietly ignore symbol definition
47
48 151346 010446                2$:    MOV      R4,-(SP)   ;save present value of symbol
49 151350 004767 173146                JSR      PC,GETEXP   ;set value to be set
50 151354 106437 007326                MTPS    @#CURCOM    ;check if it contained undefined symbols
51 151360 001332                BNE     13$        ;it did, refuse symbol definition
52 151362 010504                MOV      R5,R4      ;put value in R4
53 151364 022604                10$:   CMP      (SP)+,R4   ;see if symbol or label is being changed
54 151366 001325                BNE     3$         ;it is, which is not permitted
55 151370 000742                BR      6$         ;no change, so quietly ignore command
56
57 151372 102564 020071 054114 UNDEFX: .RAD50 /UNDEFINED EXPRESSION/

```

Symbol and Label Handling Routines

```

151400 000340 063325 074701
151406 057760
58 151410 000000 .WORD 0
59 151412 070530 017574 001136 REPDF0: .RAD50 /REPEAT OF ZERO/
151420 002025 071330
60 151424 000000 .WORD 0
61
62 151426 132737 000002 007327 DOLABL::BITR #F.VALU,@#LASCOM
63 151434 001702 BEQ 3$
64 151436 004767 161764 JSR PC,GETNXT
65 151442 106437 007332 MTPS @#EXPFLG ;see what kind of label we are defining
66 151446 100275 BPL 3$ ;predefined symbols can't be labels
67 151450 102037 BVC 8$ ;user name already defined
68 151452 013704 007756 MOV @##ADDR,R4 ;set value to be set
69
70 151456 013702 007462 9$: MOV @#TBLBOT,R2 ;set bottom of symbol table
71 151462 162702 000006 SUB #6,R2 ;go down one entry
72 151466 020227 006500 CMP R2,#$UR ;is there space for new entry?
73 151472 101011 BHI 11$ ;yes
74 151474 012702 151504 MOV #TBLFUL,R2 ;point to message
75 151500 000167 171506 JMP ERROR ;print error message
76 151504 076452 045710 024324 TBLFUL: .RAD50 /TABLE FULL/
151512 045400
77 151514 000000 .WORD 0
78
79 151516 106427 000340 11$: MTPS #340 ;make symbol table alterations atomic
80 151522 010237 007462 MOV R2,@#TBLBOT ;by blocking console break, etc.
81 151526 013722 007320 MOV @#TBLTOP,(R2)+ ;store first word of name
82 151532 013722 007322 MOV @#TBLTOP+2,(R2)+ ;and second word
83 151536 010422 MOV R4,(R2)+ ;store symbol's value
84 151540 106427 000000 MTPS #000 ;restore interrupt service
85 151544 000167 172042 JMP CONTIN ;recover character in R0; set next command
86
87 151550 010446 8$: MOV R4,-(SP) ;save present value of label
88 151552 013704 007756 MOV @##ADDR,R4 ;set current address, which is value to be set
89 151556 000702 BR 10$
90 .DSABL LSB

```


Symbol and Label Handlings Routines

```

1          ;++
2          ; GETSYM matches a new name to the symbol table.
3          ;--
4
5 151560 013702 007462 GETSYM:;MOV @#TBLBOT,R2 ;start searching at bottom of table
6 151564 020227 007320 2%: CMP R2,#TBLTOP ;any more entries?
7 151570 103403 BLO 1% ;yes
8 151572 005004 CLR R4
9 151574 000277 SCC ;ifas name not found
10 151576 000207 RTS PC
11
12 151600 022237 007320 1%: CMP (R2)+,@#TBLTOP ;does first word of name match?
13 151604 001007 BNE 3% ;no
14 151606 022237 007322 CMP (R2)+,@#TBLTOP+2;second word match?
15 151612 001005 BNE 4% ;no
16 151614 012204 MOV (R2)+,R4 ;set value
17 151616 000270 SEN ;show this is a user defined symbol
18 151620 000242 CLV ;and that is has a value
19 151622 000207 RTS PC
20
21 151624 005722 3%: TST (R2)+
22 151626 005722 4%: TST (R2)+ ;advance to next entry
23 151630 000755 BR 2%
24
25          ;++
26          ; MATSYM matches the number in R1 to symbol values, starting with the most
27          ; recently defined. Returns a pointer in R2 if matched. Returns N set if
28          ; no match.
29          ;--
30
31 151632 132737 000002 007330 MATSYM:;BITB #F.ABS,@#TMPMOD ;absolute mode?
32 151640 001013 BNE 1% ;yes, indicate no match
33 151642 013702 007462 MOV @#TBLBOT,R2 ;set bottom of table
34 151644 020227 007320 2%: CMP R2,#TBLTOP ;any more entries?
35 151652 103006 BHIS 1% ;no
36 151654 026201 000004 CMP 4(R2),R1 ;do we have a match?
37 151660 001404 BEQ 3% ;yes
38 151662 062702 000006 4%: ADD #6,R2 ;go to next name
39 151666 000767 BR 2%
40
41 151670 000270 1%: SEN
42 151672 000207 3%: RTS PC
43

```

Symbol and Label Handling Routines

```

1          ;++
2          ; DELSYM deletes the symbol table entry just below R2.
3          ;--
4
5 151674 062737 000006 007462 DELSYM::ADD    #6,@#TBLBOT
6 151702 020237 007462      2%:   CMP      R2,@#TBLBOT      idone yet?
7 151706 101001              BHI      1%           ino, move up all lower entries
8 151710 000207              RTS      PC
9
10 151712 016242 177770      1%:   MOV      -10(R2),-(R2)   imove three words for each entry
11 151716 016242 177770              MOV      -10(R2),-(R2)
12 151722 016242 177770              MOV      -10(R2),-(R2)
13 151726 000765              BR       2%
14
15          ;++
16          ; TYPREG types the contents of registers R0 through PS and the watchpoint
17          ; address and contents if the address is not zero.
18          ;--
19
20 151730 112737 000011 007340 TYPREG::MOVB  #9,@#COUNT1   icount 9 registers
21 151736 012737 156142 007334      MOV      #DSYMS,@#ADVADR iborrow ADVADR as a pointer
22 151744 004767 161554      1%:   JSR      PC,CRLF      istart a new line
23 151750 013702 007334              MOV      @#ADVADR,R2
24 151754 004767 162014              JSR      PC,PRINT2        itype name of register
25 151760 010237 007334              MOV      R2,@#ADVADR
26 151764 162702 156146              SUB      #DSYMS+4,R2      iso back to register name
27 151770 004767 173674              JSR      PC,PDSVAL        iset contents of register
28 151774 012700 020075              MOV      #400#040+075,R0 itype = <space>
29 152000 004767 161524              JSR      PC,CHRROUT
30 152004 010401              MOV      R4,R1           iset register contents
31 152006 004767 003064              JSR      PC,TYPNM1        itype numerically
32 152012 105337 007340              DECB    @#COUNT1
33 152016 001352              BNE     1%
34 152020 013704 007754              MOV      @#WATCH,R4      iset watchpoint address
35 152024 001404              BEQ     2%               imatchpoint not set
36 152026 004767 003216              JSR      PC,TYPADR        itype watchpoint address
37 152032 000167 003056              JMP     TYPL01           itype in current mode
38 152036 000207      2%:   RTS      PC
39
40          ;++
41          ; TYPTBL types the entire user symbol table.
42          ;--
43
44 152040 016746 035264      TYPTBL::MOV  THPMOD,-(SP)   imake symbols show up in word mode
45 152044 042767 000200 035256      BIC     #F.BYTN,THPMOD    ;
46 152052 013702 007462      MOV     @#TBLBOT,R2      istart at the bottom of the table
47 152056 020227 007320      7%:   CMP     R2,@#TBLTOP   idone yet?
48 152062 103003              BHIS   1%               ives
49 152064 004767 000012              JSR     PC,TYPSYM        itype the name and value of one symbol
50 152070 000772              BR     7%
51 152072 012667 035232      1%:   MOV     (SP)+,THPMOD   iput back to whatever mode it was in
52 152076 000167 171502              JMP     CHONS            iset next command
53
54 152102 004767 161416      TYPSYM::JSR  PC,CRLF      istart a new line
55 152106 004767 161662              JSR     PC,PRINT2        itype symbol name
56 152112 012700 020075              MOV     #400#040+075,R0 itype = <space>
57 152116 004767 161406              JSR     PC,CHRROUT

```

Symbol and Label Handling Routines

```

58 152122 012201          MOV    (R2)+,R1    ;set value
59 152124 000167 002746   JMP    TYPNM1      ;type it numerically
    
```

Symbol and Label Handling Routines

```

1
2          .SRTTL Arithmetic Routines
3
4 152130 050405          LOGOR:: BIS    R4,R5
5 152132 000207          RTS    PC
6
7 152134 005104          LOGAND::COM   R4
8 152136 040405          BIC    R4,R5
9 152140 000207          RTS    PC
10
11 152142 010500         MULTIP::MOV   R5,R0    ;make a copy of first factor
12 152144 005005         CLR    R5          ;clear accumulator
13 152146 060005         1%:  ADD   R0,R5    ;multiply by successive addition
14 152150 077402         SOB   R4,1%    ;count down second factor
15 152152 000207          RTS    PC
16
17 152154 060405         ADDIT::ADD   R4,R5
18 152156 000207          RTS    PC
19
20 152160 160405         SUBTR::SUB   R4,R5
21 152162 000207          RTS    PC
22
23 152164 005000         DIVID::CLR   R0          ;initialize quotient
24 152166 005704         TST   R4          ;are we dividing by zero?
25 152170 001403         BEQ   2%          ;yes, return 17777
26 152172 005200         1%:  INC   R0          ;increment quotient
27 152174 160405         SUB   R4,R5    ;try subtracting divisor from dividend
28 152176 103375         BCC   1%          ;successful, no underflow
29 152200 010005         2%:  MOV   R0,R5    ;replace accumulator with quotient
30 152202 005305         DEC   R5          ;adjust for excessive increment of R0
31 152204 000207          RTS    PC
    
```

```

1
2
3
4
5
6
7
8
9 152206 010446          GETINS::MOV   R4,--(SP)      ;save the base opcode value
10 152210 004767 002310   JSR   PC,ADRCN1          ;check that it's even and in user RAM
11 152214 006202          ASR   R2                 ;make a byte index
12 152216 006202          ASR   R2                 ;for the PARAMS table
13 152220 010437 007336   MOV   R4,@$SAVEXP        ;save opcode address here
14 152224 142737 000010 007324   BICB  #$SAVX,@$LINFLG    ;prevent interference with normal use of SAVEXP
15 152232 012624          MOV   (SP)+,(R4)+        ;store base opcode value
16 152234 010437 007334   MOV   R4,@$ADVADR        ;first operand address
17 152240 105037 007342   CLRB  @#MODE             ;initialize addressing mode scratch
18 152244 116201 156034   MOVB  PARAMS(R2),R1      ;set format code
19 152250 006201          ASR   R1                 ;make a word index
20 152252 006201          ASR   R1
21 152254 042701 177741   BIC   #177741,R1         ;mask out the garbage
22 152260 004771 152276   JSR   PC,@INSDSP(R1)     ;set appropriate operand sequence
23 152264 013737 007334 007756   MOV   @#$ADVADR,@#$ADDR  ;update current address
24 152272 000167 171314   JMP   CONTIN             ;recover next character, parse next command
25
26 152276 153172          INSDSP: .WORD  CONCOD      ;00 means decode condition codes
27 152300 143164          .WORD  CMON2             ;01 means not an instruction
28 152302 152770          .WORD  INSDON           ;02 means no parameters
29 152304 153152          .WORD  EXTRAP           ;03 means NNN for EMT and TRAP
30 152306 153024          .WORD  BRDISP           ;04 means 8-bit branch displacement
31 152310 152724          .WORD  REGDST           ;05 means R,DD for JSR and XOR
32 152312 153016          .WORD  OPRAND           ;06 means SS or DD
33 152314 153006          .WORD  SRC DST          ;07 means SS,DD
34 152316 152714          .WORD  RTSREG           ;10 means R for RTS
35 152320 153102          .WORD  SOB DSP         ;11 means R,NN for SOB
36
37
38 152322 004767 173122   GETREG::JSR  PC,GETNAM    ;parse a symbolic name
39 152326 100414          BMI   3$                ;user defined name, error
40 152330 103413          BCS   3$                ;instruction mnemonic, error
41 152332 004767 161050   JSR   PC,GETLCH          ;set next character in R0
42 152336 020227 000040   CMP   R2,#DSYMP$-DSYMP$ ;is it R0 through PC?
43 152342 103006          BHI   3$                ;no, error
44 152344 006202          ASR   R2                 ;set register number
45 152346 006202          ASR   R2
46 152350 060237 007342   ADD   R2,@#MODE          ;put it in bottom 3 bits of addressing mode
47 152354 000250          CLN
48 152356 000207          RTS   PC
49
50 152360 000270          3$:  SEN
51 152362 000207          RTS   PC
52
53
54 152364 120027 000100   FULOPR::CMPB R0,#100     ;is first character @?
55 152370 001005          BNE   5$                ;no
56 152372 062737 000010 007342   ADD   #10,@#MODE         ;yes, deferred mode
57 152400 004767 161022   JSR   PC,GETNXT          ;set character after @

```

58	152404	004767	001672	5%:	JSR	PC,RADCH	might it be a register name?	
59	152410	100407			RMI	6%	no, definitely not a register	
60	152412	010046			MOV	R0,-(SP)	save character	
61	152414	010346			MOV	R3,-(SP)	save running pointer	
62	152416	004767	177700		JSR	PC,GETREG	and try to set a register name	
63	152422	100044			BPL	7%	successful, mode 0 or 1	
64	152424	012603			MOV	(SP)+,R3	restore running pointer	
65	152426	012600			MOV	(SP)+,R0	restore character in R0	
66	152430	120027	000050	6%:	CMPB	R0,#050	is it (?)	
67	152434	001445			BEQ	4%	yes, mode 1, 2, or 3	
68	152436	120027	000055		CMPB	R0,#055	is next character -?	
69	152442	001011			BNE	9%	no definitely not autodecrement	
70	152444	010346			MOV	R3,-(SP)	save running pointer	
71	152446	004767	160754		JSR	PC,GETNXT	set character after -	
72	152452	120027	000050		CMPB	R0,#050	is it (?)	
73	152456	001430			BEQ	10%	yes, mode 4 or 5	
74	152460	012603			MOV	(SP)+,R3	restore running pointer	
75	152462	012700	000055		MOV	#055,R0	restore - in R0	
76	152466	120027	000043	9%:	CMPB	R0,#043	is it #?	
77	152472	001454			BEQ	11%	yes, do immediate and absolute	
78	152474	004767	000142		JSR	PC,GETARG	set an index or relative address	
79	152500	062737	000060	007342	ADD	#60,@#MODE	indexed addressing mode	
80	152506	120027	000050		CMPB	R0,#050	is it an index?	
81	152512	001421			BEQ	12%	yes	
82	152514	013704	007334		MOV	@#ADVADR,R4	set next operand address	
83	152520	163744	007334		SUB	@#ADVADR,-(R4)	make this operand a PC relative address	
84	152524	062737	000007	007342	ADD	#07,@#MODE	implicit PC reference	
85	152532	000207			RTS	PC		
86								
87	152534	022626		7%:	CMP	(SP)+,(SP)+	discard saved character and running pointer	
88	152536	000207			RTS	PC	return mode 0 or 1	
89								
90	152540	062737	000030	007342	10%:	ADD	#30,@#MODE	for -(, add 40 total to addressing mode
91	152546	005726			TST	(SP)+	discard saved running pointer	
92	152550	062737	000010	007342	4%:	ADD	#10,@#MODE	for (, add 10 to addressing mode
93	152556	004767	160644		12%:	JSR	PC,GETNXT	set character after (
94	152562	004767	177534		JSR	PC,GETREG	should be a register name	
95	152566	100501			BMI	2%	not a register name, error	
96	152570	120027	000051		CMPB	R0,#051	should be followed by)	
97	152574	001076			BNE	2%	no, error	
98	152576	004767	160624		JSR	PC,GETNXT	set character after)	
99	152602	120027	000053		CMPB	R0,#053	is it +?	
100	152606	001005			BNE	8%	no, done	
101	152610	062737	000010	007342	ADD	#10,@#MODE	mode 2 or 3	
102	152616	004767	160604		JSR	PC,GETNXT	set character after +	
103	152622	000207		8%:	RTS	PC		
104								
105	152624	062737	000027	007342	11%:	ADD	#27,@#MODE	immediate, absolute are modes 2 and 3 with PC
106	152632	004767	160570		JSR	PC,GETNXT	set character after #	
107	152636	000167	000000		JMP	GETARG	store immediate data or absolute address	
108								
109	152642	004767	171654		GETARG::JSR	PC,GETEXP	set an argument in R5	
110	152646	013704	007334		MOV	@#ADVADR,R4	set address of operand	
111	152652	004767	001652		JSR	PC,ADRCH2	check that we are still in user RAM	
112	152656	010514			MOV	R5,@R4	store argument at operand address	
113	152660	062737	000002	007334	ADD	#2,@#ADVADR	increment address	
114	152666	000207			RTS	PC		

```

115
116
117 152670 120027 000054      COMACH: CMPB  R0,#054      ;is first operand followed by a comma?
118 152674 001036              BNE          2$          ;no, error
119 152676 004767 000042      JSR          PC,INSDN1   ;store first operand mode
120 152702 004767 160520      JSR          PC,GETNXT   ;set character after comma
121 152706 105037 007342      CLR$        @#MODE
122 152712 000207              RTS          PC
123
124 152714 004767 177402      RTSREG::JSR  PC,GETREG   ;set a register number
125 152720 100424              BNE          2$          ;error, not a register name
126 152722 000416              BR          INSDN2      ;add it to opcode
127
128 152724 004767 177372      REGDST::JSR  PC,GETREG   ;set a register number
129 152730 100420              BNE          2$          ;error, not a register name
130 152732 004767 177732      JSR          PC,COMACH   ;check for comma between operands
131 152736 004767 177422      JSR          PC,FULOPR   ;set a full operand
132 152742 000406              BR          INSDN2
133
134 152744 113701 007342      INSDN1: MOV$  @#MODE,R1   ;set addressing mode bits
135 152750 000301              SWAB        R1          ;shift it 6 bits to the left
136 152752 006201              ASR         R1
137 152754 006201              ASR         R1
138 152756 000402              BR          1$
139
140 152760 113701 007342      INSDN2: MOV$  @#MODE,R1   ;set addressing mode bits
141 152764 060177 034346      1$:  ADD     R1,@SAVEXP   ;add them to opcode
142
143 152770 000207              INSDN: RTS   PC
144
145 152772 000167 170166      2$:  JMP     CMON2
146
147 152776 012702 153266      20$: MOV     #BRANCH,R2   ;point to branch error message
148 153002 000167 170204      JMP     ERROR
149
150 153006 004767 177352      SRCDST::JSR  PC,FULOPR   ;set first operand
151 153012 004767 177652      JSR          PC,COMACH   ;check for comma between operands
152 153016 004767 177342      DPRAND::JSR  PC,FULOPR   ;set second operand
153 153022 000756              BR          INSDN2      ;store second operand mode
154
155
156 153024 004767 171472      BRDISP::JSR  PC,GETEXP   ;set branch destination
157 153030 106437 007326      MTP$        @#CURCOM     ;did expression contain undefined symbols?
158 153034 001402              BEQ         14$         ;no, no problem
159 153036 013705 007756      MOV         @##ADDR,R5   ;yes, make instruction 'BR .' so no range error
160 153042 163705 007334      14$: SUB     @#ADVADR,R5   ;make it relative
161 153046 006205              ASR         R5          ;make it a word offset
162 153050 032705 177600      BIT         #177600,R5   ;does upper byte match bit 7?
163 153054 001405              BEQ         13$         ;yes
164 153056 005105              COM         R5
165 153060 032705 177600      BIT         #177600,R5
166 153064 001344              BNE          20$        ;no, branch range error
167 153066 005105              COM         R5
168 153070 042705 177400      13$: BIC     #177400,R5   ;make it an 8-bit number
169 153074 060577 034236      ADD         R5,@SAVEXP   ;add displacement to op-code
170 153100 000207              RTS          PC
171

```

172	153102	004767	177214	S0BDSP::JSR	PC,GETREG	i;set register
173	153106	100731		RMI	2%	ierror, not a register name
174	153110	004767	177554	JSR	PC,COMACH	i;check for comma between operands
175	153114	004767	171402	JSR	PC,GETEXP	
176	153120	106437	007326	MTPS	@#CURCOM	i;did expression contain undefined symbols?
177	153124	001402		BEG	21%	ino, no problem
178	153126	013705	007756	MOV	@#ADDR,R5	i;yes, make instruction "BR ." so no range error
179	153132	163705	007334	21%: SUB	@#ADVADR,R5	i;make it relative
180	153136	006205		ASR	R5	
181	153140	005405		NEG	R5	i;SOB instruction uses nesated displacement
182	153142	032705	177700	BIT	#177700,R5	i;6-bit displacement
183	153146	001313		BNE	20%	i;range or direction error
184	153150	000747		BR	13%	
185						
186	153152	004767	171344	ENTRAP::JSR	PC,GETEXP	i;set trap number
187	153156	032705	177400	BIT	#177400,R5	i;should be <256.
188	153162	001303		BNE	2%	i;>255., error
189	153164	000741		BR	13%	
190						
191	153166	004767	160234	19%: JSR	PC,GETNXT	i;set character after comma
192	153172	120027	000103	CONCDD::CMPB	R0,#103	i;R1 must be 0 when entering -- is it C?
193	153176	001003		BNE	16%	
194	153200	052701	000001	BIS	#1,R1	i;set C bit
195	153204	000421		BR	15%	
196	153206	120027	000126	16%: CMPB	R0,#126	i;is it V?
197	153212	001003		BNE	17%	
198	153214	052701	000002	BIS	#2,R1	i;set V bit
199	153220	000413		BR	15%	
200	153222	120027	000132	17%: CMPB	R0,#132	i;is it Z?
201	153226	001003		BNE	18%	
202	153230	052701	000004	BIS	#4,R1	i;set Z bit
203	153234	000405		BR	15%	
204	153236	120027	000116	18%: CMPB	R0,#116	i;is it N?
205	153242	001253		BNE	2%	ino, illegal character
206	153244	052701	000010	BIS	#10,R1	i;set N bit
207						
208	153250	004767	160152	15%: JSR	PC,GETNXT	i;set next character
209	153254	120027	000054	CMPB	R0,#054	i;is it a comma?
210	153260	001742		BEG	19%	i;yes, get some more
211	153262	005301		DEC	R1	i;compensate for opcode base value = 241 or 261
212	153264	000637		BR	1%	i;add condition bits to opcode
213				.DSARL	LSB	
214						
215	153266	007521	054000	001155	BRANCH: .RAD50	/BRANCH OUT OF RANGE/
	153274	076417	022622	004167		
	153302	017500				
216	153304	000000		.WORD	0	

```

1          ;++
2          ; TYPINS decodes the instruction at address R4 and types it followed on the
3          ; next line by the next address.
4          ;--
5
6 153306 011405      TYPINS::MOV    @R4,R5      ;set opcode
7 153310 010500      MOV      R5,R0
8 153312 006300      ASL      R0              ;set bits 6 through 13
9 153314 006300      ASL      R0
10 153316 000300     SWAB     R0
11 153320 042700 177700 BIC     #177700,R0
12 153324 110037 007342 MOVVB   R0,@#NODE      ;store them here for operand decode
13 153330 005705     TST     R5
14 153332 100027     BPL     5$              ;not a byte mode instruction
15 153334 042705 100000 BIC     #100000,R5     ;first see if it's a byte mode instruction
16 153340 004767 000130 JSR     PC,#1$         ;match to table, set index
17 153344 006202     ASR     R2
18 153346 105762 156034 TSTB   PARAMS(R2)
19 153352 100017     BPL     5$              ;not byte mode, do it normally
20 153354 006302     ASL     R2              ;restore R2
21 153356 012405     MOV     (R4)+,R5      ;fix R4, restore R5
22 153360 010246     MOV     R2,-(SP)     ;save index
23 153362 006302     ASL     R2              ;make index to mnemonic
24 153364 062702 156502 ADD     #MEMS,R2
25 153370 011201     MOV     @R2,R1       ;set 3 character mnemonic
26 153372 004767 160406 JSR     PC,PRINT1     ;type it
27 153376 012602     MOV     (SP)+,R2     ;restore R2
28 153400 012701 006200 MOV     #50#50#2,R1  ;type B <SP> <SP>
29 153404 004767 160374 JSR     PC,PRINT1
30 153410 000412     BR      6$
31
32 153412 012405      5$: MOV     (R4)+,R5     ;set opcode
33 153414 004767 000054 JSR     PC,#1$       ;match to table
34 153420 010246     MOV     R2,-(SP)     ;save index
35 153422 006302     ASL     R2              ;make index to mnemonic
36 153424 062702 156502 ADD     #MEMS,R2
37 153430 004767 160340 JSR     PC,PRINT2     ;type mnemonic
38 153434 012602     MOV     (SP)+,R2     ;set index back
39 153436 012700 000011 6$: MOV     #011,R0     ;type <tab>
40 153442 004767 160062 JSR     PC,CHROUT
41 153446 006202     ASR     R2              ;make byte instruction index
42 153450 116201 156034 MOVVB   PARAMS(R2),R1 ;set format code
43 153454 006201     ASR     R1
44 153456 006201     ASR     R1
45 153460 042701 177741 BIC     #177741,R1    ;mask out the garbage
46 153464 004771 153514 JSR     PC,@INTDSP(R1) ;type operands
47 153470 000167 001554 JMP     TYPADR        ;type next address and a tab
48
49 153474 012702 000212 1$: MOV     #PARAMS-BASES-2,R2 ;index last base value
50 153500 020562 155620 3$: CMP     R5,BASES(R2) ;compare opcode to base value
51 153504 103002     BHIS   4$              ;match
52 153506 005742     TST    -(R2)          ;keep looking
53 153510 000773     BR     3$
54 153512 000207     4$: RTS    PC
55
56 153514 154132     INTDSP: .WORD  TCONCD ;CVZN
57 153516 153540     .WORD  NOPCOD   ;16 bit value, not an opcode

```


Instruction Encode/Decode Routines

58	153520	153564		.WORD	INTDOW	ino operands
59	153522	153722		.WORD	TEMTRP	iEMT and TRAF
60	153524	153734		.WORD	TBRDSP	i8-bit displacement
61	153526	153574		.WORD	TRGDST	iR,DD
62	153530	153554		.WORD	TOPRND	iSS or DD
63	153532	153544		.WORD	TSRDST	iSS,DD
64	153534	153566		.WORD	TRTSRG	iR
65	153536	153676		.WORD	TSOBDS	i6-bit positive displacement
66						
67				.ENABL	LSB	
68	153540	005744		NOPCOD::TST	-(R4)	i back up to number
69	153542	000553		BR	12#	itype it as data
70						
71	153544	004767	000200	TSRDST::JSR	PC,TFULOP	itype an operand
72	153550	004767	000026	13#: JSR	PC,TCOMMA	itype comma
73	153554	004767	000102	TOPRND::JSR	PC,GETDST	iset bits 0 through 5 of opcode
74	153560	004767	000164	JSR	PC,TFULOP	itype operand
75	153564	000207		INTDOW: RTS	PC	
76						
77	153566	004767	000070	TRTSRG::JSR	PC,GETDST	
78	153572	000421		BR	TREGIS	itype register name
79						
80	153574	004767	000036	TRGDST::JSR	PC,TREGIS	itype register
81	153600	000763		BR	13#	itype comma and second operand
82						
83	153602	012700	000054	TCOMMA: MOV	#054,R0	
84	153606	000167	157716	JMP	CHROUT	
85						
86	153612	012700	000050	TPREG:: MOV	#050,R0	itype (
87	153616	004767	157706	JSR	PC,CHROUT	
88	153622	004767	000010	JSR	PC,TREGIS	itype register name
89	153626	012700	000051	MOV	#051,R0	
90	153632	000167	157672	JMP	CHROUT	
91						
92	153636	113700	007342	TREGIS::MOVB	@#MODE,R0	iset bits 0 through 2
93	153642	042700	177770	BIC	#177770,R0	
94	153646	006300		ASL	R0	iindex register name
95	153650	006300		ASL	R0	
96	153652	016001	156142	MOV	DSYMS(R0),R1	iset RAD50 triplet
97	153656	000167	160126	JMP	PRINT4	itype it
98						
99	153662	110537	007342	GETDST: MOVB	R5,@#MODE	
100	153666	142737	000300	BICB	#300,@#MODE	
101	153674	000207		11#: RTS	PC	
102						
103	153676	004767	177734	TSOBDS::JSR	PC,TREGIS	itype register name
104	153702	004767	177674	JSR	PC,TCOMMA	itype comma
105	153706	010501		MOV	R5,R1	iset lower 6 bits
106	153710	042701	177700	BIC	#177700,R1	
107	153714	006301		ASL	R1	imake it a byte displacement
108	153716	005401		NEG	R1	
109	153720	000407		BR	6#	
110						
111	153722	010501		TEMTRP::MOV	R5,R1	iset lower 8 bits without sign extend
112	153724	042701	177400	BIC	#177400,R1	
113	153730	005744		TST	-(R4)	iTYPLD2 will advance R4
114	153732	000404		BR	7#	

115					
116	153734	110501	TBRDSP::MOV	R5,R1	i: set lower 8 bits with sign extend
117	153736	006301		ASL R1	i: make it a byte displacement
118	153740	060401	6%:	ADD R4,R1	i: add in the "PC"
119	153742	005744		TST -(R4)	i: TYPL02 will advance R4
120	153744	000167	7%:	JMP TYPL02	i: use this value
121					
122	153750	113702	TFULOP::MOV	@MODE,R2	i: addressing mode scratch
123	153754	032702		BIT #10,R2	i: deferred?
124	153760	001404		BEQ 1%	i: no
125	153762	012700		MOV #100,R0	i: type 2
126	153766	004767		JSR PC,CHROUT	
127	153772	120227	1%:	CMPB R2,#20	i: mode 0 or 1?
128	153776	103717		BLD TREGIS	i: yes, just type register name
129	154000	120227		CMPB R2,#60	i: mode 6 or 7?
130	154004	103034		BHIS 2%	i: yes, indexed
131	154006	120227		CMPB R2,#27	i: immediate?
132	154012	001423		BEQ 3%	i: yes
133	154014	120227		CMPB R2,#37	i: absolute?
134	154020	001420		BEQ 3%	i: yes
135	154022	120227		CMPB R2,#40	i: autodecrement?
136	154026	103404		BLD 4%	i: no
137	154030	012700		MOV #055,R0	i: type -
138	154034	004767		JSR PC,CHROUT	
139	154040	004767	4%:	JSR PC,TPREG	i: type register name in parenthesis
140	154044	120227		CMPB R2,#40	i: autoincrement?
141	154050	103245		BHIS INTIDON	i: no, finished
142	154052	012700		MOV #053,R0	i: type +
143	154056	000167		JMP CHROUT	
144					
145	154062	012700	3%:	MOV #043,R0	i: type #
146	154066	004767		JSR PC,CHROUT	
147	154072	000167	12%:	JMP TYPL01	i: type next location
148					
149	154076	120227	2%:	CMPB R2,#67	i: relative?
150	154102	001406		BEQ 5%	i: yes
151	154104	120227		CMPB R2,#77	i: relative deferred?
152	154110	001403		BEQ 5%	i: yes
153	154112	004767		JSR PC,TYPL01	i: type next location
154	154116	000635		BR TPREG	i: type register name in parenthesis
155					
156	154120	012401	5%:	MOV (R4)+,R1	i: set next location
157	154122	060401		ADD R4,R1	i: add in the "PC"
158	154124	005744		TST -(R4)	i: TYPL02 will increment R4, don't do it now
159	154126	000167		JMP TYPL02	i: use the value
160					
161	154132	010501	TCONCD::MOV	R5,R1	i: set opcode
162	154134	006201		ASR R1	i: C bit set?
163	154136	103004		BCC 8%	i: no
164	154140	012700		MOV #103,R0	i: type C
165	154144	004767		JSR PC,CHROUT	
166	154150	006201	8%:	ASR R1	i: V bit set?
167	154152	103004		BCC 9%	
168	154154	012700		MOV #126,R0	
169	154160	004767		JSR PC,CHROUT	
170	154164	006201	9%:	ASR R1	i: Z bit set?
171	154166	103004		BCC 10%	

172	154170	012700	000132		MOV	#132,R0	
173	154174	004767	157330		JSR	PC,CHROUT	
174	154200	006201		10%:	ASR	R1	iN bit set?
175	154202	103234			BCC	11\$	
176	154204	012700	000116		MOV	#116,R0	
177	154210	000167	157314		JMP	CHROUT	
178					.DSABL	LSB	

```

1
2           .SBTTL Conversion and Typeout Routines
3       ;++
4       ;   BINASC strips the thousands digit from the value in R1 and multiplies R1
5       ;   by 10. Use BINAS1 to set the 10000's digit.
6       ;--
7
8 154214 005000   BINASC::CLR   R0           ;divide by successive subtraction
9 154216 005200   1$:   INC     R0
10 154220 162701 001750   SUB     #1000.,R1      ;try to subtract 1000.
11 154224 103374           BCC     1$           ;worked, no borrow
12 154226 005300           DEC     R0           ;fix R0
13 154230 062701 001750   ADD     #1000.,R1      ;and R1
14 154234 006301           ASL     R1           ;multiply R1 by 10
15 154236 010146           MOV     R1,-(SP)
16 154240 006301           ASL     R1
17 154242 006301           ASL     R1           ;times 8
18 154244 062601           ADD     (SP)+,R1      ;plus 2 makes 10 X
19 154246 062700 000060   ADD     #060,R0      ;add 060 to make it an ASCII digit
20 154252 000207           RTS     PC
21
22
23 154254 005000   BINAS1::CLR   R0           ;divide by successive subtraction
24 154256 005200   1$:   INC     R0
25 154260 162701 023420   SUB     #10000.,R1    ;try to subtract 10000.
26 154264 103374           BCC     1$           ;worked, no borrow
27 154266 005300           DEC     R0           ;fix R0
28 154270 062701 023420   ADD     #10000.,R1    ;and R1
29 154274 062700 000060   ADD     #060,R0      ;make an ASCII digit
30 154300 000207           RTS     PC
31
32       ;++
33       ;   RADCH checks if the character in R0 is a legal RAD50 character. Returns
34       ;   N=1 if not.
35       ;--
36
37 154302 120027 000133   RADCH::CMPB  R0,#133    ;higher than Z?
38 154306 103017           BHIS   2$
39 154310 120027 000101   CMPB  R0,#101         ;A or higher?
40 154314 103016           BHIS   1$
41 154316 120027 000072   CMPB  R0,#072         ;: or higher?
42 154322 103011           BHIS   2$
43 154324 120027 000060   CMPB  R0,#060         ;0 or higher?
44 154330 103010           BHIS   1$
45 154332 120027 000056   CMPB  R0,#056         ;period?
46 154336 001405           BEQ   1$
47 154340 120027 000044   CMPB  R0,#044         ;dollar sign?
48 154344 001402           BEQ   1$
49 154346 000270   2$:   SEN
50 154350 000207           RTS     PC
51 154352 000250   1$:   CLN
52 154354 000207           RTS     PC
53

```

```

1          ;++
2          ; NUMCH checks if the character in R0 is numeric and returns class as follows:
3          ; N=0, V=0 character is 0 through 7
4          ; N=0, V=1 character is 8 or 9
5          ; N=1 any other character
6          ;--
7
8 154356 120027 000060 NUMCH::CMPB R0,#060 ;is it 0 or higher?
9 154362 103413          BLO 1$ ;no, non-numeric
10 154364 120027 000072 CMPB R0,#072 ;is it higher than 9?
11 154370 103010          BHIS 1$ ;yes, non-numeric
12 154372 120027 000070 CMPB R0,#070 ;is it higher than 7?
13 154376 103002          BHIS 2$ ;yes, 8 or 9
14 154400 000257          CCC ;flag 0 through 7
15 154402 000207          RTS PC
16 154404 000257          2$: CCC
17 154406 000262          SEV ;flag 8 or 9
18 154410 000207          RTS PC
19 154412 000270          1$: SEN
20 154414 000207          RTS PC
21
22          ;++
23          ; ASCRAD multiplies R1 by 40 and adds the RAD50 equivalent of R0.
24          ;--
25
26 154416 004767 157500 ASCRAD::JSR PC,R1X40 ;multiply R1 by 40
27 154422 120027 000040 CMPB R0,#040
28 154426 001416          BEQ 1$ ;spaces converts to 0
29 154430 120027 000056 CMPB R0,#056
30 154434 001414          BEQ 4$ ;period is 34
31 154436 120027 000044 CMPB R0,#044
32 154442 001412          BEQ 5$ ;dollar sign is 33
33 154444 060001          ADD R0,R1
34 154446 162701 000100 SUB #100,R1
35 154452 120027 000101 CMPB R0,#101
36 154456 103002          BHIS 1$ ;alphabets are 1 through 32
37 154460 062701 000056 ADD #056,R1 ;numerics are 36 through 47
38 154464 000207          1$: RTS PC
39
40 154466 005201          4$: INC R1
41 154470 062701 000033 5$: ADD #033,R1
42 154474 000207          RTS PC
43
44
45 154476 054754 000566 001533 USERAM: .RAD50 /NOT IN USER RAM/
154504 021020 070265
46 154510 000000          .WORD 0
47
48 154512 057144 000054 015725 ODDADR: .RAD50 /ODD ADDRESS/
154520 074670
49 154522 000000          .WORD 0

```

```

1
2          .ENABL LSB
3          ;++
4          ; ADRCHK checks that the number in R4 is even if current mode is word size.
5          ; ADRCH1 moves the current address into R4 and checks that it is in user RAM.
6          ;--
7
8 154524 013704 007756      ADRCH1::MOV    @$$ADDR,R4
9 154530 032737 040000 007706 ADRCH2::BIT    #F.PROT,@$FLAGS!;monitor scratch protection disabled?
10 154536 001024                BNE    ADRCHK      ;yes, proceed to odd/even check
11 154540 020427 020000        CMP    R4,#20000    ;above upper instance of monitor scratch?
12 154544 103021                BHIS   ADRCHK      ;yes
13 154546 020427 006500        CMP    R4,$$UR     ;below lower instance?
14 154552 103416                BLO   ADRCHK      ;yes
15 154554 020427 010000        CMP    R4,#10000   ;within lower instance of monitor scratch?
16 154560 103407                BLO   2$          ;yes, user not allowed to deposit there
17 154562 010446                MOV    R4,-(SP)
18 154564 162704 010000        SUB    #10000,R4   ;translate R4 from 1xxxx range to 0xxxx
19 154570 020427 006500        CMP    R4,$$UR     ;
20 154574 012604                MOV    (SP)+,R4    ;restore R4
21 154576 103404                BLO   ADRCHK      ;it's between instances of scratch
22 154600 012702 154476        2$:  MOV    $USERAM,R2 ;point to message
23 154604 000167 166402        1$:  JMP    ERROR    ;type error message
24
25 154610 105737 007330        ADRCHK::TSTB   @$$TMPMOD
26 154614 100406                BMI    3$
27 154616 032704 000001        BIT    #1,R4
28 154622 001403                BEQ    3$
29 154624 012702 154512        MOV    $ODDADR,R2 ;point to message
30 154630 000765                BR     1$
31
32 154632 000207                3$:  RTS    PC
33          .DSABL LSB
34
35          ;++
36          ; TYPASC types R1 as an ASCII byte or word.
37          ;--
38
39 154634 105737 007330        TYPASC::TSTB   @$$TMPMOD
40 154640 100404                BMI    1$
41 154642 000301                SWAB   R1          ;in word mode, type high order character first
42 154644 004767 000002        JSR    PC,1$
43 154650 000301                SWAB   R1
44 154652 010100                1$:  MOV    R1,R0    ;type low order character in R1
45 154654 000167 156746        JMP    ASCOUT

```

```

1          ;++
2          ; TYPDEC types the value in R1 as a signed, 5-digit decimal number followed
3          ; by a period. TYPDEC1 can be used to type unsigned numbers.
4          ;--
5
6 154660 012700 000040      TYPDEC::MOV    #040,R0
7 154664 005701              TST      R1
8 154666 100003              BPL      1$
9 154670 005401              NEG      R1
10 154672 012700 000055      MOV      #055,R0
11 154676 004767 156626      1$: JSR      PC,CHROUT
12 154702 112737 000005 007341  MOVVB   #5,@#COUNT2
13 154710 004767 177340      JSR      PC,BINASC1
14 154714 000402              BR      3$
15 154716 004767 177272      5$: JSR      PC,BINASC
16 154722 004767 156602      3$: JSR      PC,CHROUT
17 154726 105337 007341      DECB   @#COUNT2
18 154732 001371              BNE     5$
19 154734 012700 000056      MOV      #056,R0          ;type a period
20 154740 000167 156564      JMP      CHROUT
21
22          ;++
23          ; TYPDEC1 types R1 as an octal word.
24          ;--
25
26 154744 005000              TYPDEC1::CLR    R0
27 154746 032767 000200 032354  BIT     #F.BYTM,TMPMOD ;is it in byte mode
28 154754 001407              BEQ     1$              ;branch if yes
29 154756 112767 000003 032355  MOVVB   #3,COUNT2      ;if byte mode only three digits
30 154764 000301              SWAB   R1              ;only low byte out
31 154766 006301              ASL     R1
32 154770 006100              ROL     R0
33 154772 000411              BR      3$
34 154774 112767 000006 032337  1$: MOVVB   #6,COUNT2
35 155002 000405              BR      3$
36 155004 005000              2$: CLR     R0
37 155006 006301              ASL     R1
38 155010 006100              ROL     R0
39 155012 006301              4$: ASL     R1
40 155014 006100              ROL     R0
41 155016 006301              3$: ASL     R1
42 155020 006100              ROL     R0
43 155022 062700 000060      ADD     #060,R0
44 155026 004767 156476      JSR      PC,CHROUT
45 155032 105337 007341      DECB   @#COUNT2
46 155036 001362              BNE     2$
47 155040 000207              RTS     PC
48
49

```

```

1          ;++
2          ;  TYPSPC types a tab, except in instruction mode (type nothings) .
3          ;--
4
5 155042 132737 000100 007330 TYPSPC::BITB  #F,INST,@#TMPMOD
6 155050 001004          BNE 1#
7 155052 012700 004440          MOV  #400#011+040,R0 ;space, tab
8 155056 000167 156446          2#: JMP  CHROUT
9 155062 000207          1#:  RTS  PC
10
11         ;++
12         ;  TYPNUM types the value in R1 as a symbol or number.
13         ;  TYPNM1 types the value in R1 as a number only.
14         ;--
15
16 155064 004767 174542          TYPNUM::JSR  PC,MATSYM      ;try to match a symbol
17 155070 100402          BMI  TYPNM1      ;unsuccessful, type numerically
18 155072 000167 156676          JMP  PRINT2      ;else type symbol's name
19 155076 000722          TYPNM1::BR  TYPOCT
20
21         ;++
22         ;  TYPLOC types the location addressed by R4 in the current mode. R4 is advanced
23         ;  to the next location. TYPLO1 can be used within instruction typeout.
24         ;--
25
26 155100 004767 177504          TYPLOC::JSR  PC,ADRCHK
27 155104 132737 000100 007330          BITB  #F,INST,@#TMPMOD
28 155112 001026          BNE  TYPLO1
29 155114 011401          TYPLO1::MOV  @R4,R1      ;set current location as a word
30 155116 105737 007330          TSTB  @#TMPMOD      ;are we in byte mode?
31 155122 100003          BPL  TYPLO2      ;no
32 155124 111401          MOVB  @R4,R1      ;else set location as a byte
33 155126 042701 177400          BIC  #177400,R1      ;prevent sign extension
34 155132 004767 174474          TYPLO2::JSR  PC,MATSYM      ;try to match a symbol
35 155136 100403          BMI  2#      ;unsuccessful
36 155140 012700 133774          3#:  MOV  #PRINT2,R0
37 155144 000402          BR  5#
38 155146 012700 155076          2#:  MOV  #TYPNM1,R0
39 155152 004710          5#:  JSR  PC,(R0)
40 155154 005204          INC  R4
41 155156 105737 007330          TSTB  @#TMPMOD
42 155162 100401          BMI  6#
43 155164 005204          INC  R4
44 155166 000207          6#:  RTS  PC
45
46 155170 010546          TYPLOI::MOV  R5,-(SP)      ;save R5 because TYPINS uses it
47 155172 004767 176110          JSR  PC,TYPINS      ;type an instruction
48 155176 012605          MOV  (SP)+,R5      ;restore R5
49 155200 000207          RTS  PC
50

```



```

1          ;++
2          ; TYPLIN types the next 8 locations or four locations if in binary mode.
3          ;--
4
5 155202 004767 000042          TYPLIN::JSR   PC,TYPADR   ;type address in R4 as label or octal number
6 155206 112737 000010 007340      MOVB   #8.,@#COUNT1
7 155214 004767 177660          1$:   JSR   PC,TYPLDC   ;type location in current mode; advance R4
8 155220 004767 177616          JSR   PC,TYSPSP
9 155224 005704          TST   R4           ;for 177776 to 0 case
10 155226 001406          BEQ   3$
11 155230 020405          CMP   R4,R5        ;done yet?
12 155232 101003          BHI   2$           ;yes
13 155234 005337 007340          DEC   @#COUNT1
14 155240 001365          BNE   1$
15 155242 000207          2$:   RTS   PC
16 155244 000244          3$:   CLZ
17 155246 000207          RTS   PC
18          ;++
19          ; TYPADR types the value in R4 as a label or octal number, followed by : <SP>
20          ;--
21
22 155250 016746 032054          TYPADR::MOV   TMPMOD,-(SP) ;for byte mode
23 155254 042767 000200 032046      BIC   #F,BYTH,TMPMOD ;
24 155262 004767 156236          JSR   PC,CRLF
25 155266 010401          MOV   R4,R1
26 155270 132737 000002 007330      BITB  #F,ABS,@#TMPMOD
27 155276 001006          RNE   1$
28 155300 004767 174326          JSR   PC,HATSYN
29 155304 100403          BHI   1$
30 155306 004767 156462          JSR   PC,PRINT2
31 155312 000402          BR   2$
32
33 155314 004767 177424          1$:   JSR   PC,TYPOCT
34 155320 012700 020072          2$:   MOV   #400#040+072,R0
35 155324 012667 032000          MOV   (SP)+,TMPMOD ;restor mode
36 155330 000167 156174          JMP   CHROUT

```

```

1
2                .SBTTL Power-up Diagnostics
3
4                .ENABL LSB
5
6 155334 005203          DIAGNO::INC    R3          ;use r3 as disit to display number
7 155336 012737 000377 177440          MOV     $377,$P$PORA ;light all segments
8 155344 012737 000011 177444          MOV     $11,$P$PORC  ;turn on led
9 155352 012705 003133                    MOV     $400*006+133,R5 ;initialize error code '12'
10 155356 012701 007776                    MOV     $7776,R1      ;fill RAM with pattern, saving location for JSR
11 155362 010102                    MOV     R1,R2
12 155364 005000                    CLR     R0
13 155366 010041          1$: MOV     R0,-(R1)
14 155370 005200                    INC     R0
15 155372 005201                    INC     R1
16 155374 077104                    SOB     R1,1$
17 155376 012706 010000                    MOV     $10000,SP     ;set SP to top of RAM
18 155402 012700 005777                    MOV     $5777,R0      ;checksum complement for RAM pattern
19 155406 004767 000146                    JSR     PC,CHKSUM     ;checksum words from R1 (=0) to R2 (=7776)
20 155412 042703 177770                    BIC     $177770,R3    ;mask of all but disit select
21 155416 110337 177444                    MOV     R3,$P$PORC   ;turn off led and turn on disit
22 155422 012701 133000                    MOV     $133000,R1   ;index ROMs
23 155426 012702 160000                    MOV     $END,R2      ;set upper checksum limit to top of ROMs
24 155432 005000                    CLR     R0
25 155434 004767 000120                    JSR     PC,CHKSUM     ;perform checksum of ROM
26 155440 005705                    TST     R5
27 155442 001035                    BNE     2$
28 155444 012705 000117                    MOV     $400*000+117,R5 ;initialize error code '3'
29
30 155450 012700 177560                    MOV     $177560,R0    ;point to console
31 155454 105760 000002          SLUTST: TSTR  2(R0)    ;clear out and recieved garbage
32 155460 152760 000004 000004          BISR     $4,4(R0)    ;set mainenance bit
33 155466 112701 000060                    MOV     $60,R1
34 155472 105760 000004          SLU.1: TSTR  4(R0)    ;ready to xmit
35 155476 100375                    BPL     SLU.1
36 155500 110160 000006                    MOV     R1,6(R0)
37 155504 105710          SLU.2: TSTB  (R0)    ;received a char
38 155506 100376                    BPL     SLU.2
39 155510 126001 000002                    CMPR   2(R0),R1
40 155514 001010                    BNE     2$
41 155516 105201                    INCB   R1
42 155520 120127 000072                    CMPB   R1,$72
43 155524 001362                    BNE     SLU.1
44 155526 142760 000004 000004          BICB   $4,4(R0)
45 155534 105005                    CLRB   R5
46 155536 005705          2$: TST     R5
47 155540 001005                    BNE     4$
48 155542 132737 000002 177564          BITB   $2,$177564
49 155550 001401                    BEQ     4$
50 155552 000670                    BR     DIAGNO
51 155554 000167 161100          4$: JMP     $START    ;initialize scratchpad RAM and start monitor
52
53                .ENABL LSB
54
55 155560 062100          CHKSUM::ADD  (R1)+,R0    ;accumulate checksum
56 155562 020102                    CMP     R1,R2
57 155564 103775                    BLD    CHKSUM

```

58	155566	000305		SWAB	R5	
59	155570	005700		TST	R0	itest result
60	155572	001001		BNE	1\$	ichecksum failed
61	155574	105005		CLRB	R5	ifelse clear error
62	155576	000207	1\$:	RTS	PC	

```

1
2          .SBTTL Special Data Structures
3
4          ;++
5          ; INSTRUCTION ENCODE/DECODE and COMMAND DECODE TABLES:
6          ; The five parameters listed for each opcode are:
7          ;     1. Base value of the opcode
8          ;     2. String name of the opcode
9          ;     3. Parameter organization:
10         ;         00 - decode condition code bits (N,Z,V,C)
11         ;         01 - mnnnn (16 bit number)
12         ;         02 - no parameters
13         ;         03 - nnn (8 bit number)
14         ;         04 - xxx (8 bit displacement)
15         ;         05 - r,dd
16         ;         06 - dd or ss
17         ;         07 - ss,dd
18         ;         10 - r
19         ;         11 - r,nn (6 bit displacement)
20         ;     4. Equals 1 if instruction is legal in byte mode
21         ;
22         ; The fifth parameter is a code used by the top-level parser to decode ASCII
23         ; characters in the 040 to 077 range. The values of this parameter are:
24         ;     0 - Illegal character at top level
25         ;     1 - Interpret as the first character of an expression or instruction
26         ;     2 - :, or label definition
27         ;     3 - ;, or comment
28         ;     4 - =, or symbol definition
29         ;     5 - ?, or type as a number
30         ;     6 - Valid as an operator within expressions
31         ;     7 - Both 1 and 6 apply
32         ;--
33
34         .MACRO INSTRS
35         CODE 00000,<HALT >,2,0,0 ;space
36         CODE 00001,<WAIT >,2,0,6 ;!
37         CODE 00002,<RTI >,2,0,0 ;'
38         CODE 00003,<BPT >,2,0,1 ;#
39         CODE 00004,<IOT >,2,0,1 ;$
40         CODE 00005,<RESET >,2,0,0 ;%
41         CODE 00006,<RTT >,2,0,6 ;&
42         CODE 00007,< >,1,0,1 ;'
43         CODE 00100,<JMP >,6,0,0 ;(
44         CODE 00200,<RTS >,10,0,0 ;)
45         CODE 00210,< >,1,0,6 ;#
46         CODE 00240,<NOP >,2,0,7 ;+
47         CODE 00241,<CL >,0,0,0 ;,
48         CODE 00257,<CCC >,2,0,7 ;-
49         CODE 00260,< >,1,0,1 ;.
50         CODE 00261,<SE >,0,0,6 ;/
51         CODE 00277,<SCC >,2,0,1 ;0
52         CODE 00300,<SWAB >,6,0,1 ;1
53         CODE 00400,<BR >,4,0,1 ;2
54         CODE 001000,<BNE >,4,0,1 ;3
55         CODE 001400,<BEQ >,4,0,1 ;4
56         CODE 002000,<BGE >,4,0,1 ;5
57         CODE 002400,<BLT >,4,0,1 ;6
    
```

58		CODE	003000,<BGT	>:4:0:1	?7
59		CODE	003400,<BLE	>:4:0:1	?8
60		CODE	004000,<JSR	>:5:0:1	?9
61		CODE	005000,<CLR	>:6:1:2	?:
62		CODE	005100,<CDM	>:6:1:3	?;
63		CODE	005200,<INC	>:6:1:1	?<
64		CODE	005300,<DEC	>:6:1:4	?=
65		CODE	005400,<NEG	>:6:1:0	?>
66		CODE	005500,<ADC	>:6:1:5	??
67		CODE	005600,<SBC	>:6:1:0	
68		CODE	005700,<TST	>:6:1:0	
69		CODE	006000,<RRR	>:6:1:0	
70		CODE	006100,<ROL	>:6:1:0	
71		CODE	006200,<ASR	>:6:1:0	
72		CODE	006300,<ASL	>:6:1:0	
73		CODE	006400,<	>:1:0:0	
74		CODE	006700,<SXT	>:6:0:0	
75		CODE	007000,<	>:1:0:0	
76		CODE	010000,<MOV	>:7:1:0	
77		CODE	020000,<CMP	>:7:1:0	
78		CODE	030000,<BIT	>:7:1:0	
79		CODE	040000,<BIC	>:7:1:0	
80		CODE	050000,<BIS	>:7:1:0	
81		CODE	060000,<ADD	>:7:0:0	
82		CODE	070000,<	>:1:0:0	
83		CODE	074000,<XOR	>:5:0:0	
84		CODE	075000,<	>:1:0:0	
85		CODE	077000,<SOB	>:11:0:0	
86		CODE	100000,<BPL	>:4:0:0	
87		CODE	100400,<BHI	>:4:0:0	
88		CODE	101000,<BHI	>:4:0:0	
89		CODE	101400,<BLDS	>:4:0:0	
90		CODE	102000,<BVC	>:4:0:0	
91		CODE	102400,<BVS	>:4:0:0	
92		CODE	103000,<BCC	>:4:0:0	
93		CODE	103000,<BHIS	>:4:0:0	
94		CODE	103400,<BCS	>:4:0:0	
95		CODE	103400,<BLO	>:4:0:0	
96		CODE	104000,<EMT	>:3:0:0	
97		CODE	104400,<TRAP	>:3:0:0	
98		CODE	105000,<	>:1:0:0	
99		CODE	106400,<NTPS	>:6:0:0	
100		CODE	106500,<	>:1:0:0	
101		CODE	106700,<MFPS	>:6:0:0	
102		CODE	107000,<	>:1:0:0	
103		CODE	160000,<SUB	>:7:0:0	
104		CODE	170000,<	>:1:0:0	
105			.ENDM		
106					
107	155600	000241	CBASES::,WORD	000241	?CLC
108	155602	000242	,WORD	000242	?CLV
109	155604	000244	,WORD	000244	?CLZ
110	155606	000250	,WORD	000250	?CLN
111	155610	000261	,WORD	000261	?SEC
112	155612	000262	,WORD	000262	?SEV
113	155614	000264	,WORD	000264	?SEZ
114	155616	000270	,WORD	000270	?SEN

```

115
116             .MACRO CODE   BASVAL,MNEM,FORMAT,BYTHOD,MISC
117             .WORD   BASVAL
118             .ENDM
119 155620      BASES::INSTRS
120
121             .MACRO CODE   BASVAL,MNEM,FORMAT,BYTHOD,MISC
122             .BYTE   <BYTHOD*200>+<FORMAT*10>+MISC
123             .ENDM
124 156034      PARAMS::INSTRS
125
126 156142 072460 000000      DSYMS:: .RAD50 /R0 /           ;Predefined symbols must not be 3 characters in
127 156146 072530 000000      .RAD50 /R1 /           ;length, or else a 'byte mode' version of each
128 156152 072600 000000      .RAD50 /R2 /           ;name will also match, i.e. XYZB will match to
129 156156 072650 000000      .RAD50 /R3 /           ;XYZ. Only byte mode mnemonics should be 3
130 156162 072720 000000      .RAD50 /R4 /           ;characters long.
131 156166 072770 000000      .RAD50 /R5 /
132 156172 074500 000000      .RAD50 /SF /
133 156176 062170 000000      .RAD50 /PC /
134 156202 063370 000000      DSYMFS: .RAD50 /PS /
135 156206 111100 000000      .RAD50 /WP /
136 156212 127400 000000      .RAD50 /, /
137 156216      DSYMUR:
138 156216 000000 000000      .RAD50 / /
139 156222 010530 000000      .RAD50 /R1 /
140 156226 010600 000000      .RAD50 /R2 /
141 156232 010650 000000      .RAD50 /R3 /
142 156236 010720 000000      .RAD50 /R4 /
143
144 156242 031324 062000      DIRVRR:: .RAD50 /HELP /
145 156246 027030 000000      .RAD50 /GO /
146 156252 074745 062000      .RAD50 /STEP /
147 156256 062073 075630      DIRVRI: .RAD50 /PASS1 /
148 156262 062073 075700      .RAD50 /PASS2 /
149 156266 032153 076400      .RAD50 /HOST /
150 156272 054740 060314      .RAD50 /NDHOST/
151 156276 076464 042300      .RAD50 /TALK /
152 156302 046531 014400      .RAD50 /LOAD /
153 156306 131014 004444      .RAD50 /,START/
154 156312 127726 014400      .RAD50 /,END /
155 156316 127736 020560      .RAD50 /,EVEN /
156 156322 127473 012061      .RAD50 /,ASCII/
157 156326 127551 076710      .RAD50 /,BYTE /
158 156332 131247 070440      .RAD50 /,WORD /
159 156336 127534 042420      .RAD50 /,BLKR /
160 156342 127534 044130      .RAD50 /,BLKW /
161 156346 003243 057665      DIRVBA: .RAD50 /ABSOLU/
162 156352 011366 011624      DIRVBR: .RAD50 /CANCEL/
163 156356 021411 076400      DIRVBD: .RAD50 /EXIT /
164 156362 010174 017500      DIRVRB: .RAD50 /BYTE /
165 156366 035203 077745      DIRVBI: .RAD50 /INSTRU/
166 156372 012245 004420      .RAD50 /CLEAR /
167 156376 014724 021145      .RAD50 /DELETE/
168 156402 111052 014400      DIRVRM: .RAD50 /WORD /
169 156406      DIRVRQ:
170 156406 070530 017574      DIRVRF: .RAD50 /REPEAT/
171 156412 075265 007344      DIRVRS: .RAD50 /SYMBOL/
    
```

```

172 156416 074017 110727      DIRVPT: .RAD50 /SHOWMO/
173 156422 074017 111225      .RAD50 /SHOWRE/
174 156426 074017 111321      .RAD50 /SHOWEY/
175 156432 106257 053600      .RAD50 /UTON /
176 156436 106257 023160      .RAD50 /VTOFF /
177
178 156442 012243 000000      CMNEMS: .RAD50 /CLC /
179 156446 012266 000000      .RAD50 /CLV /
180 156452 012272 000000      .RAD50 /CLZ /
181 156456 012256 000000      .RAD50 /CLN /
182 156462 073613 000000      .RAD50 /SEC /
183 156466 073636 000000      .RAD50 /SEV /
184 156472 073642 000000      .RAD50 /SEZ /
185 156476 073626 000000      .RAD50 /SEN /
186
187                          .MACRO CODE      BASVAL,MNEM,FORMAT,BYTHOD,MISC
188                          .RAD50 'MNEM'
189                          .ENDM
190 156502                    MNEMS: INSTRS
191          156566            MNCCC ==      MNEMS+064
192          000770            TOPNAM ==      . - DSYMS          ;length of preceding RAD50 table
    
```

```

1
2          ;SBTTL Spare Locations, Checksum, and ROM I.D.
3      ;++
4      ; The locations defined by the following directive are free and must be
5      ; zero for the power-up diagnostics to work properly. When adding code
6      ; in this space or changing any of the previous code, adjust the checksum
7      ; value at CHECKS = 157774 so that the ROMs as a whole will have a zero
8      ; word checksum.
9      ;--
10
11          000642            .REPT <157774->
12                          .BYTE 0
13                          .ENBR
14      ;++
15      ; The following word adjusts the ROM checksum to zero.
16      ;--
17
18 157774 052744            CHECKS: .WORD 52744          ;12-MAY-83 R. Arnett
19
20      ;++
21      ; The last byte in each of the ROMs is used to identify the ROMs.
22      ;--
23
24 157776 010              LD.ID: .BYTE 010          ;Version 1.0, low byte
25 157777 010              HI.ID: .BYTE 010          ;Version 1.0, high byte
26
27 160000 140000            END: .END START
    
```

Symbol table

ADDIT	152154 G	CONOUT=	000064 G	EDEL	146232 G	F.LOAD=	000200	IDINIT	135076 G
ADRCHK	154610 G	CONTIN	143612 G	EDEL1	146240	F.MNEN=	000001	IOVECT	135224 G
ADRCH1	154524 G	COUNT1=	007340 G	EERR	146360	F.NULL=	000004	ITRAP	137334 G
ADRCH2	154530 G	COUNT2=	007341 G	EEXT	146320 G	F.PAS1=	100000	KADRCH	142156
ADVADR=	007334 G	CRLF	133524 G	EFND	146560 G	F.PROT=	040000	KEYBCD	136514 G
ASCOUT	133626 G	CURCOM=	007326 G	EINS	146210 G	F.REG =	000010	KEYCMD	140330 G
ASCRAD	154416 G	C%RBUF=	177562 G	EMOD	146760 G	F.SAUX=	000010	KEYCM1	140344 G
AUXFLG=	007711 G	C%RCSR=	177560 G	EMOV	146364 G	F.SST =	000200	KEYGET	136436 G
AUXIN =	000120 G	C%XBUF=	177566 G	ENTRAP	153152 G	F.SST1=	000001	KEYGE1	136470
AUXOUT=	000124 G	C%XCSR=	177564 G	END	160000 G	F.STOP=	000002	KEYPIU=	007676 G
A%CREG=	177452 G	DEBNCE=	007704 G	ENEXF	145002	F.TRAS=	020000	KEYFVS=	007700 G
A%RBUF=	177550 G	DECDSP	140030 G	ENUM	146270 G	F.UNDF=	000004	KLOOP	140306 G
A%SREG=	177552 G	DELETE	147444 G	EREG	146332 G	F.USER=	000100	KLOOP1	141170
A%XBUF=	177450 G	DELIM =	007345 G	EREP	146276 G	F.USRD=	000010	KLOOP2	141644
BASES	155620 G	DELSYM	151674 G	ERRDR	143212 G	F.VALU=	000002	KMONIT	140200 G
BCDKEY	140050 G	DGDISP	140014 G	ERRDR2	143720	F.VT =	004000	LASCOM=	007327 G
BCDTBL	136522	DIAGNO	155334 G	ERRSEG	135756	GENEXP	144512 G	LASLIN=	007325 G
BINASC	154214 G	DIGEXE	140542 G	ESYM	146312 G	GETARG	152642 G	LEDIV =	000104 G
BINAS1	154254 G	DIRVRA	156346	EXDDOP	145220	GETBP	133436 G	LERR	150472
BLNKDS	136350	DIRVRB	156352	EXINIT	145172	GETCH	133316 G	LEVEL =	007333 G
BLNKH1	136344 G	DIRVBD	156356	EXINI1	145200	GETCHC	133412 G	LINEIN	133000 G
BLNKH0	136336 G	DIRVRH	156362	EXPFLG=	007332 G	GETDST	153662	LINFLG=	007324 G
BLOAD	150442 G	DIRVBI	156366	FERCHR	137542 G	GETERM	144556 G	LOAD	150442 G
BOTTOM=	007346 G	DIRVBN	156402	FERRDR	137350 G	GETEXP	144522 G	LOBLNK	140040 G
BP =	007630 G	DIRVBD	156406	FIND	150114 G	GETEX1	144540	LODISP	140024 G
BRANCH	153266	DIRVBP	156406	FIXBP	140170	GETINS	152206 G	LOGAND	152134 G
BRDISP	153024 G	DIRVBS	156412	FLAGS1=	007706 G	GETKEY	140044 G	LOGDR	152130 G
BREAK	137114 G	DIRVBT	156416	FMESAG	137424 G	GETLCH	133406 G	LO.ID	157776 G
BRKFIL=	007760 G	DIRVBI	156256	FMESG1	137434	GETLIN	133240 G	LPAREN	145046
BRKPT	137174 G	DIRVRB	156242 G	FMESG2	137450	GETLIT	146060 G	MATSYM	151632 G
BRKP1	141466	DISPLA	135344 G	FMESG3	137472	GETNAM	145450 G	MFB251=	007464 G
BUFFER=	007470 G	DIVID	152164 G	FMESG4	137522	GETNMB	145306 G	MNCCC =	156566 G
CANCEL	142246 G	DOCOMM	144124	FNCDDN	142264	GETNUM	140054 G	MNEMS	156502 G
CBASES	155600 G	DOCR	144170 G	FNCTBL	142234 G	GETNXT	133426 G	MODE =	007342 G
CHECKS	157774 G	DDEXPR	144274 G	FP =	007631 G	GETDPR	144724 G	KDNITR	140010 G
CHKSUM	155560 G	DFUNC	142226 G	FPAREN	145162	GETOPV	144720 G	MONIT1	137556 G
CHROUT	133530 G	DOLABL	151426 G	FULOPR	152364 G	GETREG	152322 G	MOVE	150044 G
CHRO1	133540 G	DOWNVAL	144134	F.ABS =	000002	GETSYM	151560 G	MULTIP	152142 G
CHRO2	133554 G	DOOPER	145124	F.ADDR=	000004	GNEXAM	141146 G	NANTBL	136154 G
CLRAPE	151146	DOSYMB	151164 G	F.APPL=	010000	GO	147432 G	NKWDST	150306 G
CNNEMS	156442 G	DOVERB	147172 G	F.A64X=	000200	GOLED	142270 G	NOPCDD	153540 G
CMONIT	142724 G	DSPDEC	136366 G	F.BRKA=	000400	GRET	143052	NOPROT	150134 G
CMON1	142572 G	DSPDIG	135772 G	F.BRKG=	001000	HELP	147320 G	NPROT	142562 G
CMON2	143164 G	DSPLNM	136044 G	F.BRKS=	000020	HEXSEG	133016 G	NUMCH	154356 G
CMON3	143216 G	DSPNAM	136112 G	F.BYTB=	000001	HFP =	007672 G	NUMGET	136546 G
CMON31	143224	DSPNUM	136050	F.BYTM=	000200	HGETCH	134744 G	DDADR	154512
CMON4	143246 G	DSPTBL	140360 G	F.CHAN=	000002	HI.ID	157777 G	OPERAT=	007343 G
CMONS	143604 G	DSPUNH	136036 G	F.COMM=	000020	HOST	150222 G	OPRAND	153016 G
CMON6	143616 G	DSYMPS	156202	F.CPBO=	000010	HOSTRF=	007632 G	OPRTRL	145250
CMON7	143652	DSYMS	156142 G	F.CPB1=	000020	HOSTIN	134540 G	OPTION	150352
CNTLC =	007344 G	DSYMR	156216	F.CPB2=	000040	HSTFLG=	007673 G	OVERFL	143104
COMACH	152670	EARS	146150 G	F.DATA=	000001	INSDN1	152744	PADRAD	146050
COMLUP	143646	EBYT	146176 G	F.FUNC=	000040	INSDN2	152760	PADR1	146040
CONBRK=	000140 G	ECAN	146156 G	F.HOST=	004000	INSDDN	152770	PARAMS	156034 G
CONCOD	153172 G	ECHO	134142 G	F.INST=	000100	INSDSP	152276	FARDSP	144104
CONFLG=	007710 G	ECLR	146222 G	F.KEYF=	002000	INTDDN	153564	PASS1	150202 G
CONIN =	000060 G	ECON	146354	F.LDST=	000100	INTDSP	153514	PASS2	150212 G

Symbol table

PDSVAL 145670 G	SAVEXP= 007336 G	TCONCD 154132 G	TYPSPM 152102 G	\$G01 141722 G
PERMOD= 007331 G	SCRPAD= 007400 G	TEXTRP 153722 G	TYPTBL 152040 G	\$HALT 136752 G
PFVEC = 000024 G	SCRPM1= 006500 G	TFULDP 153750 G	UCSAVE 137004 G	\$MONIT 137570 G
PRINT 133766 G	SEGBUF= 007712 G	TIME = 007730 G	UNDEFX 151372 G	\$NULLI 135676 G
PRINTA 133754 G	SETAUX 135170 G	TMPMOD= 007330 G	UPBLNK 140034 G	\$NUMER 140430 G
PRINT1 134004 G	SETAX1 135126 G	TOPNAM= 000770 G	UPDISP 140020 G	\$PC = 007750 G
PRINT2 133774 G	SETAX2 135124 G	TOPRND 153554 G	USERAM 154476 G	\$PS = 007752 G
PRINT3 134014 G	SETHOD 146104 G	TPREG 153612 G	USERIV= 000100 G	\$REG 141610 G
PRINT4 134010 G	SLUTST 155454 G	TREGIS 153636 G	USERVS= 007702 G	\$REPEAT 150012 G
PRNERR 143120 G	SLU.1 155472 G	TRGDST 153574 G	VCON 150142 G	\$R0 = 007732 G
PROTEC 150172 G	SLU.2 155504 G	TRTSRG 153566 G	VERR 150146 G	\$R1 = 007734 G
PURGE 133450 G	SOBDSF 153102 G	TSOBDS 153676 G	VERR1 150162 G	\$R2 = 007736 G
PWRFVS: 007674 G	SPCLBF= 007726 G	TSRDST 153544 G	VRBDSF 147220 G	\$R3 = 007740 G
P\$CREG= 177446 G	SPDTBL 142302 G	TYPADR 155250 G	VRBDS1 147262 G	\$R4 = 007742 G
P\$PDRA= 177440 G	SPEEDS 142320 G	TYPASC 154634 G	VTJMP 147402 G	\$R5 = 007744 G
P\$PORC= 177444 G	SRCDS1 153006 G	TYPDEC 154660 G	VTOFF 147406 G	\$SF = 007746 G
P\$PORD= 177544 G	STACK = 007462 G	TYPINS 153306 G	VTON 147374 G	\$SST 141706 G
P\$PORI= 177542 G	START 140000 G	TYPLIN 155202 G	WHAT 143076 G	\$START 136660 G
P\$PORO= 177442 G	STEP 147416 G	TYPLOC 155100 G	\$ADDR = 007756 G	\$UR = 006500 G
RADCH 154302 G	SUBTR 152160 G	TYPLOI 155170 G	\$ADDRK 141550 G	\$WATCH= 007754 G
REGDST 152724 G	TALK 150504 G	TYPLO1 155114 G	\$ADV 141232 G	\$.ASCI 147760 G
REPEAT= 007466 G	TALK1 150530 G	TYPLO2 155132 G	\$BACK 141372 G	\$.BLKF 147634 G
REFOFO 151412 G	TBLBOT= 007462 G	TYPNM1 155076 G	\$BRKP 141462 G	\$.BLKW 147654 G
RERR 150152 G	TBLFUL 151504 G	TYPNUM 155064 G	\$CLR 141110 G	\$.BYTE 147466 G
RPAREN 145152 G	TBLTOP= 007320 G	TYPOCT 154744 G	\$EXAM 141122 G	\$.EVEN 147776 G
RSTART 140004 G	TBRDSP 153734 G	TYPROG 151730 G	\$FUNC 141650 G	\$.STRT 150344 G
RTSREG 152714 G	TCDMA 153602 G	TYPSPC 155042 G	\$G0 141714 G	\$.WORD 147560 G
R1X40 134122 G				

. ABS. 160000 000 (RW,I,GBL,ABS,DVR)
 000000 001 (RW,I,LCL,REL,CON)

Errors detected: 0

*** Assembler statistics

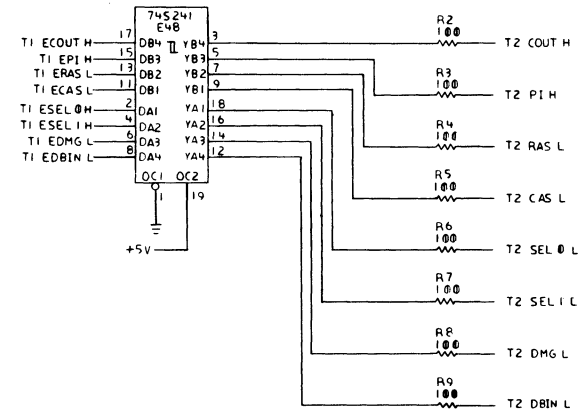
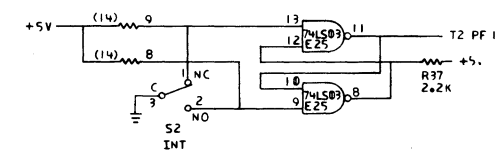
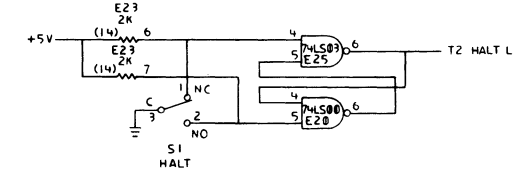
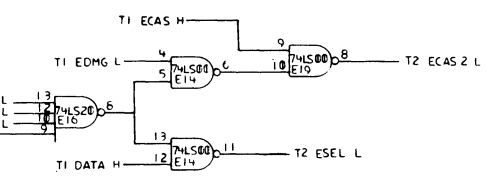
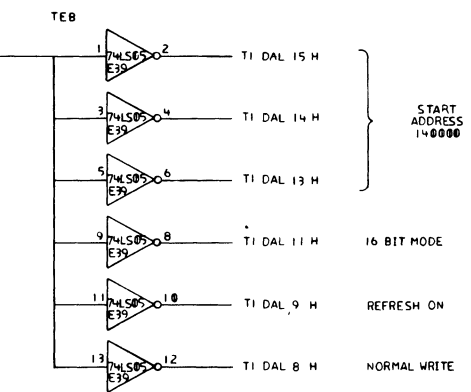
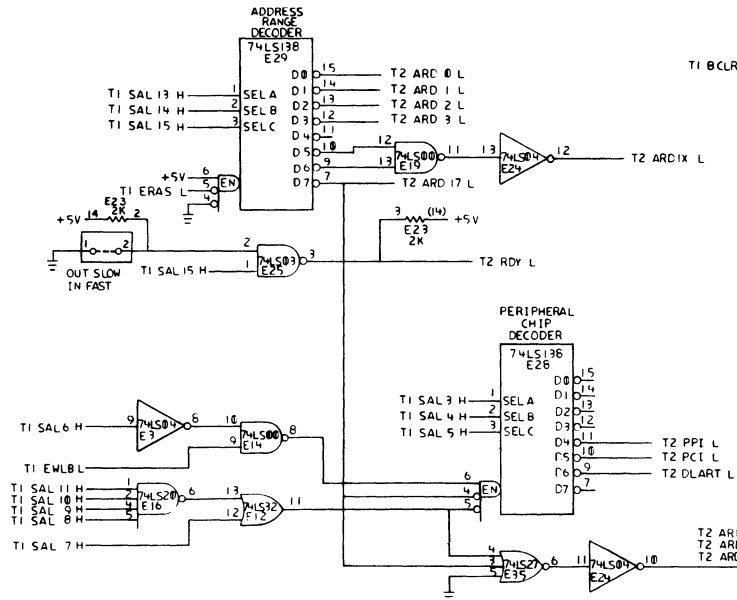
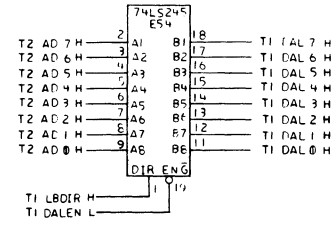
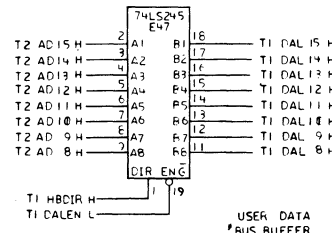
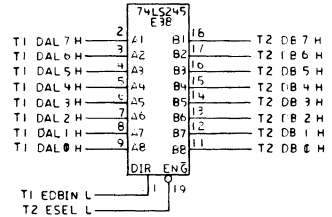
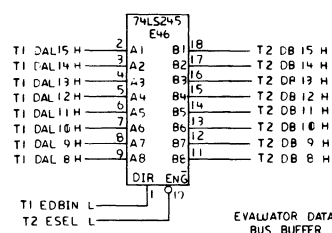
Work file reads: 0
 Work file writes: 0
 Size of work file: 10760 Words (43 Pages)
 Size of core pool: 13312 Words (52 Pages)
 Operating system: RT-11

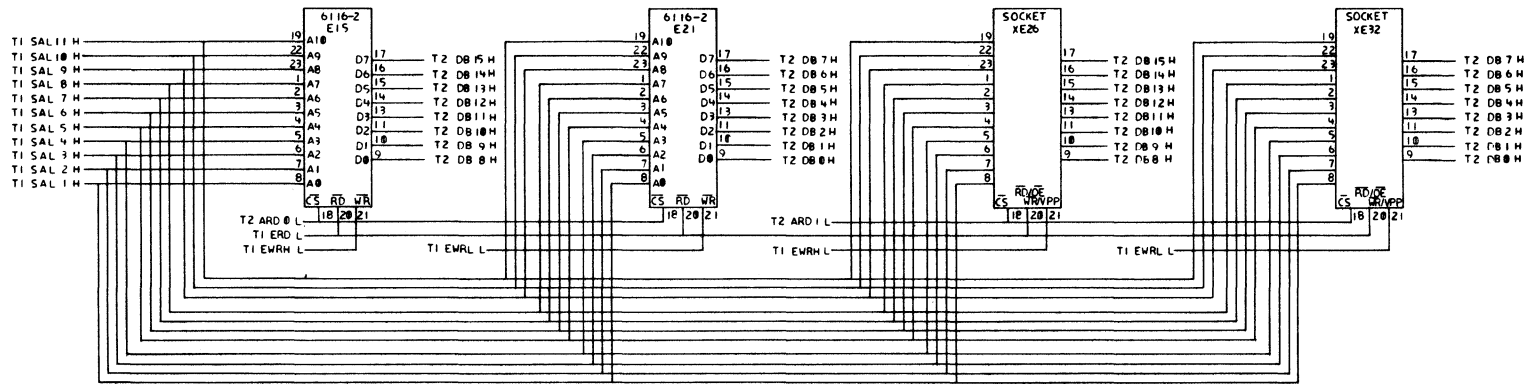
Elapsed time: 00:02:30.51
 DK:MON2,DK:MON2/C=DK:MON2

APPENDIX B SCHEMATIC DRAWINGS

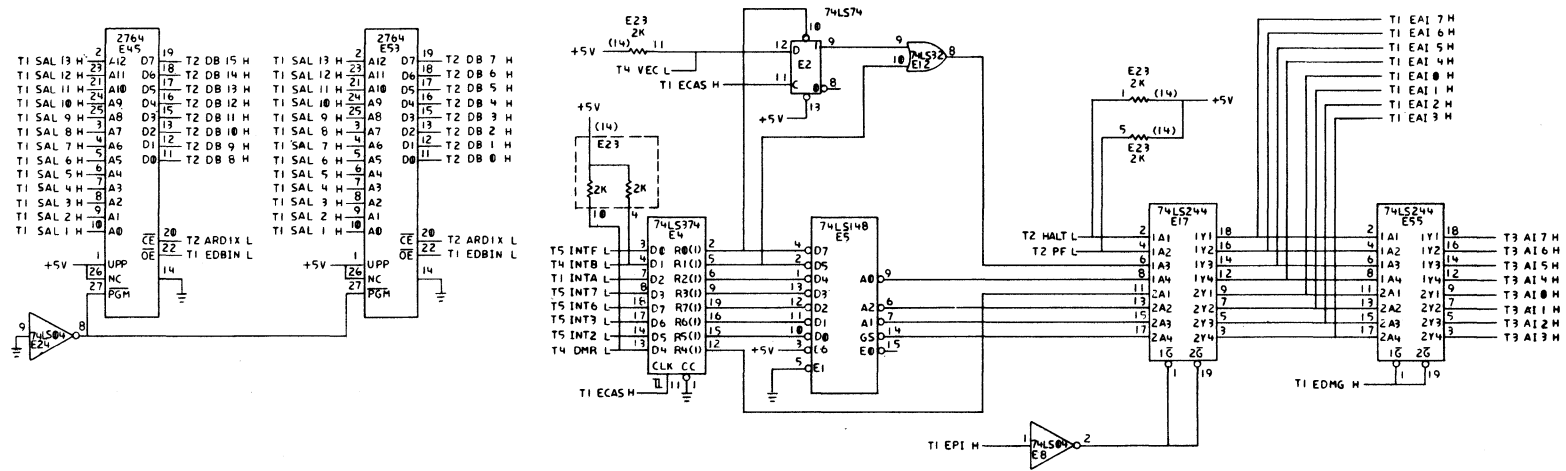


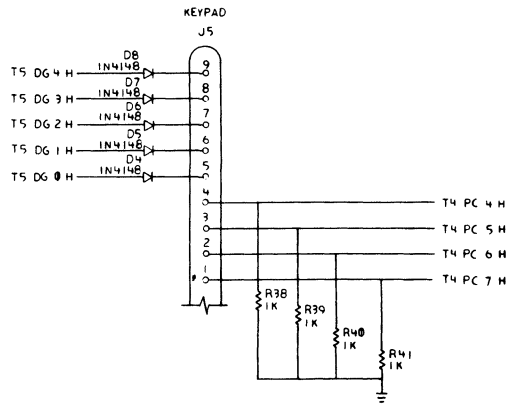
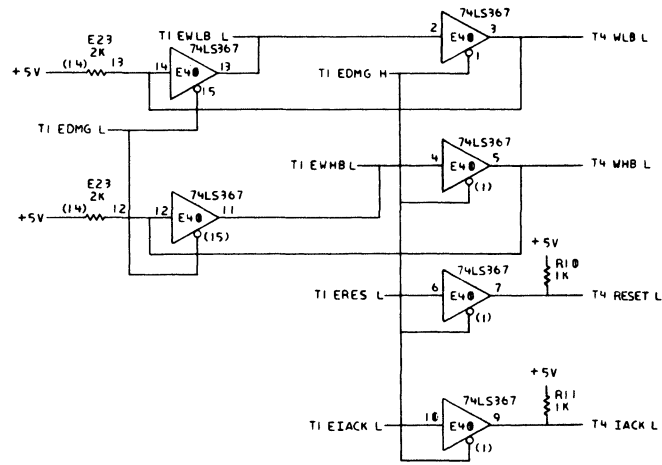
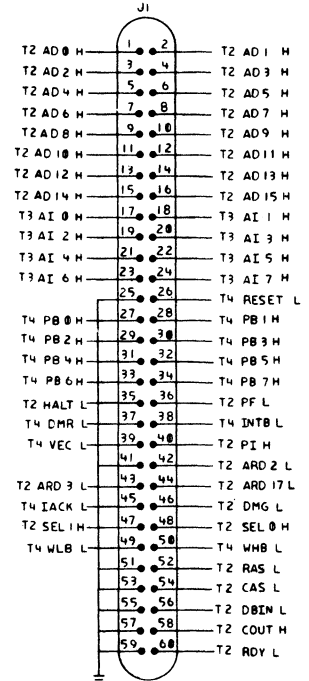
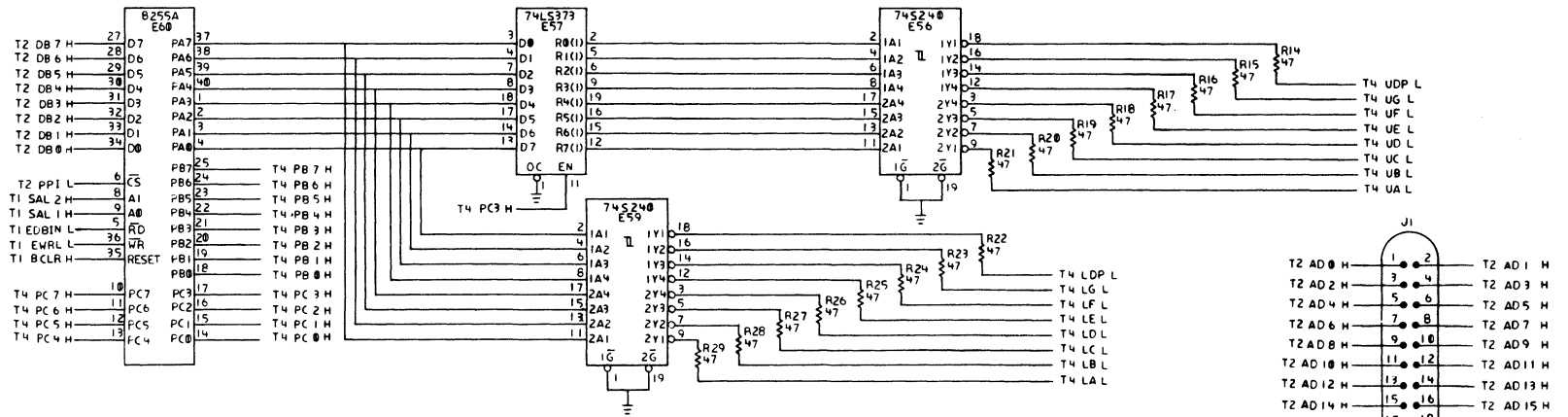
Figure B-1 DCT11-EM Component Placement





B-4



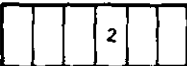

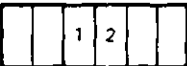

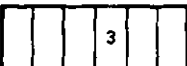





APPENDIX C ERROR MESSAGES

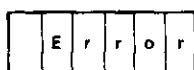
C.1 DIAGNOSTIC ERROR MESSAGES

The following flashing error messages are displayed if the DCT11-EM fails its diagnostic tests. These are fundamental errors and must be corrected before you attempt to do anything else.

UPPER DISPLAY	LOWER DISPLAY	MEANING
		RAM checksum test failed. Try replacing RAM chips.
		ROM checksum test failed. Try replacing ROM chips.
		RAM and ROM checksum tests failed. Board may have major problems.
		DLART feedback test failed. Board may have major problems.

C.2 KEYPAD ERROR MESSAGES

If the DCT11-EM detects an error while you are using the keypad, it displays a flashing error message on the upper and lower rows of LEDs. The upper row of LEDs displays a two-character error code and the lower row of LEDs displays:



The entire error message disappears and the DCT11-EM is returned to register mode when you press the HALT switch or any key on the keypad.

The keypad error messages are as follows.



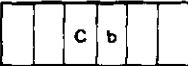

UPPER DISPLAY	LOWER DISPLAY	MEANING
		You tried to execute a program with a BPT instruction in it.
		A break was detected on the console terminal port while the keypad was active.

Table C-1 Console Messages

Message	Meaning
BRANCH OUT OF RANGE	You input a branch instruction which was too far from the branch destination. Recall that branch instructions have a range of -128 to 127 (decimal) words.
BREAK POINT ENCOUNTERED	A breakpoint (B1, B2, B3, or B4) or a BPT instruction was encountered while your program was executing.
BUFFER OVERFLOW	The console monitor was accepting input from a host and a line exceeding 96 characters was received. As with all errors generated during host loading, this error stops the DCT11-EM from loading further input from the host.
CONSOLE BREAK	The BREAK key on the console keyboard was pressed. If you are executing a program, execution stops and the DCT11-EM internal register values at the time of the break are saved. If you are in HOST, TALK, or LOAD modes, CONSOLE BREAK causes a return to NOHOST mode.
INVALID INSTRUCTION ENCOUNTERED	An invalid op code was encountered while your program was running.
INVALID STACK POINTER	The stack pointer did not have a value between 4 and 6500 (inclusive) when you started your program or when you single stepped.
NOT IN USER RAM	You attempted to alter a location that was in the protected monitor scratchpad area (locations 6500 through 7777, inclusive).
ODD ADDRESS	You attempted to access a full word in RAM on an odd address boundary. Use the BYTE mode to examine or alter locations at odd addresses.
REPEAT OF ZERO	You input a REPEAT command that had a count of zero.
TABLE FULL	You attempted to define more than 66 symbols.
UNDEFINED EXPRESSION	You attempted to use an undefined parameter, operand, or other data object when a defined expression was required. For example, you referenced an undefined symbol while in PASS2 mode.
WHAT ?	You attempted to input a line that had a syntax error in it.

INDEX

A

- ABSOLU command, 4-7, 4-29
- Address decoding, 6-6
- Address pointer, 4-6
- Address space, 5-13
 - standard RAM space, 5-15
 - expansion RAM space, 5-15
 - monitor space, 5-15
 - I/O space, 5-15
 - for external hardware, 5-16
 - reserved, 5-16
- Altering memory
 - with console, 4-10
 - with keypad, 3-10, 3-32
- Altering registers
 - with console, 4-5
 - with keypad, 3-4, 3-32
- .ASCII command, 4-21, 4-29

B

- .BLKB command, 4-21, 4-29
- .BLKW command, 4-21, 4-29
- Breakpoints canceling
 - with console, 4-20
 - canceling with keypad, 3-27, 3-32
 - displaying with console, 4-20
 - displaying with keypad, 3-20, 3-32
 - setting with console, 4-19
 - setting with keypad, 3-20, 3-32
- Buffers, 6-5
- .BYTE command, 4-21, 4-30
- BYTE command, 4-7, 4-30

C

- Cancel breakpoints, 3-27, 3-32
- CANCEL command, 4-16, 4-20, 4-30
- CLEAR command, 4-14, 4-30
- Connectors, 6-8
 - 60-pin (J1), 6-8 through 6-9
 - auxiliary serial (J3), 6-10
 - console serial (J4), 6-11
 - power supply (J2), 6-10
- Console
 - altering memory with, 4-10
 - altering registers with, 4-5
 - command summary, 4-29

- connecting a, 4-1
- control commands, 4-3
- default modes, 4-2, 4-21
- entering programs with, 4-12
- examining registers with, 4-5
- examining memory with, 4-7
- host loading with, 4-22
- input rules, 4-2
- monitor, 1-1, 3-28, 5-1, 5-15
- output formatting modes, 4-6
- single stepping with, 4-17
- transferring control to and from, 4-2
- using breakpoints with, 4-19
- using watchpoint with, 4-16
- Console commands
 - ABSOLU command, 4-7, 4-29
 - .ASCII command, 4-21, 4-29
 - .BLKB command, 4-21, 4-29
 - .BLKW command, 4-21, 4-29
 - .BYTE command, 4-21, 4-30
 - BYTE command, 4-7, 4-30
 - CANCEL command, 4-16, 4-30
 - CLEAR command, 4-14, 4-30
 - DELETE command, 4-14, 4-30
 - .END command, 4-26, 4-30
 - .EVEN command, 4-21, 4-30
 - EXIT command, 4-2, 4-30
 - GO command, 4-16, 4-30
 - HELP command, 4-21, 4-30
 - HOST command, 4-23, 4-30
 - INSTRU command, 4-7, 4-30
 - LOAD command, 4-24, 4-30
 - NOHOST command, 4-23, 4-30
 - PASS1 command, 4-25, 4-30
 - PASS2 command, 4-25, 4-30
 - REPEAT command, 4-21, 4-31
 - SHOWMO command, 4-21, 4-31
 - SHOWRE command, 4-5, 4-31
 - SHOWSY command, 4-14, 4-31
 - .START command, 4-25, 4-31
 - STEP command, 4-17, 4-31
 - SYMBOL command, 4-7, 4-31
 - TALK command, 4-23, 4-38
 - VTOFF command, 4-3, 4-31
 - VTON command, 4-3, 4-31
 - .WORD command, 4-21, 4-31
- CTRL/A, 4-3, 4-24, 4-25, 4-32
- CTRL/C, 4-3, 4-32

CTRL/O, 4-3, 4-24, 4-32
CTRL/Q, 4-3, 4-32
CTRL/R, 4-3, 4-32
CTRL/S, 4-3, 4-32
CTRL/U, 4-3, 4-32
CTRL/X, 4-3, 4-32
CTRL/Y, 4-3, 4-32

D

Default operating modes, 2-3, 4-2, 4-21
DCT11-AA microprocessor, 1-1, 6-3
DELETE command, 4-14, 4-30
Directives, 4-20
Debugging, see Breakpoints,
Single stepping, Watchpoints

E

.END command, 4-36, 4-30
EPROM, 6-4
.EVEN command, 4-21, 4-30
EXIT command, 4-2, 4-30
Error Messages
 console, C-2
 diagnostic, 2-4, C-1
 keypad, C-1
Examining memory
 with console, 4-7
 with keypad, 3-7
Examining registers
 with console, 4-6
 with keypad, 3-2
Expressions, 4-3

G

GO command, 4-16, 4-30
Go with LEDs, 3-27, 3-32

H

Host Loading
 configuration, 4-22
 example of process, 4-28
 preparing a loadable program, 4-25
 the HOST command, 4-23
 the TALK command, 4-23
HELP command, 4-21, 4-30
HALT switch, 3-24, 6-7
HOST command, 4-25, 4-30

I

INT switch, 3-25, 6-7
INSTRU command, 4-7, 4-30
Instruction format, 4-11
Interrupts, 5-5, 6-6

J

Jumpering, 2-4, 6-4

K

Keypad
 altering registers with, 3-4
 altering memory locations with, 3-10
 command summary, 3-32
 entering programs with, 3-12, 3-16
 examining registers with, 3-2
 examining memory locations with, 3-7
 monitor, 1-1, 2-3, 5-1, 5-15
 number system, 3-2
 single stepping with, 3-17
 special functions, 3-26
 cancel breakpoints, 3-27, 3-32
 go with LEDs, 3-27, 3-32
 release protection, 3-30, 3-32
 set baud rates, 3-28, 3-32
 start console, 3-34, 3-32
 using breakpoints with, 3-19
 using watchpoint with, 3-22

L

LOAD command, 4-24, 4-30

M

MACRO-11, 4-20
Memory expansion module, 6-11
Monitor, 1-1, 2-3, 3-8, 3-28, 3-30, 4-2, 5-1,
5-15, 6-4
Monitor subroutines,
 console, 5-8 through 5-13
 ASCOUT, 5-9
 CHRO2, 5-9
 CHROUT, 5-9
 CRLF, 5-9
 GETCH, 5-9
 GETCHC, 5-9
 GETLIN, 5-8
 GETNXT, 5-9

PRINTA, 5-9
TYPDEC, 5-9
TYPOCT, 5-9
using, 5-10
keypad/LED, 5-2 through 5-8
BCDKEY, 5-3
DECDSP, 5-3
DGDISP, 5-2
GETKEY, 5-3
GETNUM, 5-4
LOBLNK, 5-3
LODISP, 5-3
UPBLNK, 5-3
UPDISP, 5-3
using, 5-4

N

NOHOST command, 4-23, 4-30

P

PASS1 command, 4-25, 4-30
PASS2 command, 4-25, 4-30
Peripheral chips, 6-4
address decoding, 6-6
auxiliary serial line (8251A), 6-5
DLART console serial line, 6-5
parallel port (8255A), 6-4
reserved locations for, 5-15, 6-4
Processor status word, 3-2
Power supply
cable, 2-1
connector, 2-2, 6-8, 6-10
requirements, 2-1

R

RAM, 6-5
Release protection, 3-30, 3-32
REPEAT command, 4-21, 4-31

S

Set baud rates, 3-28, 3-32
SHOWMO command, 4-21, 4-31
SHOWRE command, 4-5, 4-31
SHOWSY command, 4-14, 4-31
Single stepping
with console, 4-17
with keypad, 3-17
Stack pointer, 2-3, 3-2, 5-2
.START command, 4-25, 4-31
Start console, 3-28, 3-32

STEP command, 4-17, 4-31
Subroutines, see Monitor subroutines
SYMBOL command, 4-7, 4-31

Symbols
defining, 4-14
deleting, 4-14
displaying, 4-14
symbol name rules, 4-13

Symbol table
deleting, 4-14
displaying, 4-14

System specifications, 1-1

T

TALK command, 4-23, 4-31
Timing, 6-7
Traps, 6-6

U

User LED, 3-25, 5-5

V

VTOFF command, 4-3, 4-31
VTON command, 4-3, 4-31

W

Watchpoints
canceling with console, 4-16
canceling with keypad, 3-27
displaying with console, 4-16
displaying with keypad, 3-20, 3-22
setting with console, 4-16
setting with keypad, 3-22
.WORD command, 4-21, 4-31



AMP 208004-2

AMP 208004-0



88888888
88888888

PC	PS	WP	ADR	FNC
7	8	9	ADR	FNC
PROTEC	SP	REG	CPT	
4	5	6	REG	CPT
GOLED	CONSOL	BAUD	BAC	GD
1	2	3	BAC	GD
CANCEL	ENTER	CLR	0	EXA
CLR	0	EXA	ADV	SST