

# **Tektronix®**

**8540**  
INTEGRATION UNIT  
**SYSTEM USERS**  
**MANUAL**  
OS/40 VERSION 1



**PLEASE CHECK FOR CHANGE INFORMATION  
AT THE REAR OF THIS MANUAL.**

**8540**  
INTEGRATION UNIT  
**SYSTEM USERS  
MANUAL**  
OS/40 VERSION 1

Tektronix, Inc.  
P.O. Box 500  
Beaverton, Oregon 97077

070-3939-00  
Product Group 61

Serial Number \_\_\_\_\_

First Printing NOV 1981  
Revised OCT 1983

## LIMITED RIGHTS LEGEND

Software License No. \_\_\_\_\_

Contractor: Tektronix, Inc.

Explanation of Limited Rights Data Identification Method

Used: Entire document subject to limited rights.


Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

## RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Copyright © 1981 Tektronix, Inc. All rights reserved. Contents of this publication may not be reproduced in any form without the written permission of Tektronix, Inc.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and  are registered trademarks of Tektronix, Inc. TELEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Printed in U.S.A. Specification and price change privileges are reserved.

## ABOUT THIS MANUAL

This manual is your guide to using the 8540 Integration Unit and its operating system, OS/40.

This manual is **not** about programming techniques, specific microprocessors, or specific applications. Instead, it presents the information that will enable you to use the 8540 for your own tasks in hardware-software integration.

In this manual, you'll find detailed information on all standard 8540 features, including emulation and intersystem communication. This manual also provides introductory information on several hardware products you can use with your 8540. Those products are described in detail in other user manuals.

For new users, a Learning Guide introduces you to the 8540 and its system options, and includes sample demonstration runs. There's also an Index to help you find your way around the manual, and a Glossary that describes new terminology and terms that are used in a new way.

For users of DOS/50 (the operating system of the TEKTRONIX 8550 Microcomputer Development Lab), the Technical Notes section contains a list of differences between DOS/50 and OS/40.

# CONTENTS

	Page
<b>SECTION 1 LEARNING GUIDE</b>	
Introduction .....	1-1
Specification, Installation, Configuration, and Verification .....	1-1
Overview of the 8540 System .....	1-2
Getting Started .....	1-8
Starting Up the 8540 .....	1-8
Turning Off the 8540 .....	1-10
How to Enter OS/40 Commands .....	1-10
Establishing Communication with the Host .....	1-11
More About OS/40 Commands .....	1-13
8540/8560 Configurations .....	1-15
Emulator Demonstration Run .....	1-17
For Continued Learning .....	1-40
Overview of 8540 User Manuals .....	1-41
 <b>SECTION 2 OPERATING PROCEDURES</b>	
Communicating with an 8560 .....	2-1
Communicating with a Host Computer .....	2-3
Communicating with an 8550 .....	2-5
Downloading from an 8002A .....	2-7
Program Execution .....	2-8
Program Debugging .....	2-10
Debugging the Program in the Prototype .....	2-13
Using the Trigger Trace Analyzer .....	2-14
Memory Manipulation .....	2-19
System I/O .....	2-20
 <b>SECTION 3 COMMAND DICTIONARY</b>	
Command Index .....	3-1
Introduction .....	3-2
Other System Commands .....	3-2
Command Syntax .....	3-2
Special Keys .....	3-6
Command Memory Areas .....	3-7
Commands .....	3-8
 <b>SECTION 4 INTERSYSTEM COMMUNICATION</b>	
Introduction .....	4-1
Terminology .....	4-1
8540/8560 TERM Interface .....	4-1
COM Interface .....	4-2
8540/8550 Interface .....	4-4
8540/8002A Interface .....	4-6
8540/General Host Interface .....	4-7

<b>SECTION 5 EMULATION</b>	<b>Page</b>
Introduction .....	5-1
The Emulator .....	5-1
Emulation Modes .....	5-1
Selecting the Target Processor .....	5-3
Basic Emulation Tasks .....	5-3
Considerations for Modes 1 and 2 .....	5-6
Symbolic Debug .....	5-7
The Memory Allocation Controller .....	5-9
The Trigger Trace Analyzer .....	5-13

**SECTION 6 SERVICE CALLS**

Introduction .....	6-1
Overview .....	6-1
Limitations of File Handling SVCs .....	6-4
I/O Channels .....	6-4
SVC Applications .....	6-5
SVC Demonstration .....	6-6
SVC Functions .....	6-10

**SECTION 7 EMULATOR SPECIFICS**

**SECTION 8 PROM PROGRAMMER SPECIFICS**

**SECTION 9 TECHNICAL NOTES**

Note 1. Practical Limits of Emulation .....	9-1
Note 2. Comparison of OS/40 Version 1 with DOS/50 Version 2 .....	9-1
Note 3. Comparison of the TTA and the RTPA .....	9-2

**SECTION 10 ERROR MESSAGES**

**SECTION 11 TABLES**

Conversion Tables .....	11-1
Tektronix Hexadecimal Format .....	11-4
Motorola Load Module Format .....	11-8
Intel Load Module Format .....	11-10

**SECTION 12 GLOSSARY**

**SECTION 13 INDEX**

---

## Section 1 LEARNING GUIDE

	Page
<b>Introduction</b> .....	1-1
<b>Specification, Installation, Configuration, and Verification</b> .....	1-1
<b>Overview of the 8540 System</b> .....	1-2
Uses of the 8540 .....	1-2
8540 Parts and Functions .....	1-4
The Host Computer .....	1-7
<b>Getting Started</b> .....	1-8
Starting Up the 8540 .....	1-8
Turning Off the 8540 .....	1-10
How to Enter OS/40 Commands .....	1-10
Establishing Communication with the Host .....	1-11
More about OS/40 Commands .....	1-13
<b>8540/8560 Configurations</b> .....	1-15
<b>Emulator Demonstration Run</b> .....	1-17
Introduction .....	1-17
Examine the Demonstration Program .....	1-19
Assemble and Load the Demonstration Program .....	1-22
Case 1: Assemble on the 8560; Download to the 8540 .....	1-22
Case 2: Download from Your Host to the 8540 .....	1-28
Case 3: Patch the Program into Memory .....	1-30
Run the Demonstration Program .....	1-32
Monitor Program Execution .....	1-35
Summary of Emulator Demonstration Run .....	1-39
<b>For Continued Learning</b> .....	1-40
<b>Overview of 8540 User Manuals</b> .....	1-41

---

## TABLES

<b>Table No.</b>		<b>Page</b>
1-1	Jack Assignments and Device Names for 8540 Peripherals .....	1-8
1-2	COM Interface Checklist .....	1-9
1-3	Basic 8560 Editing Commands .....	1-25

## ILLUSTRATIONS

<b>Fig. No.</b>		<b>Page</b>
1-1	Role of 8540 in product design .....	1-3
1-2	8540 logical subsections .....	1-4
1-3	Demonstration program .....	1-16
1-4	Demonstration program: Extended Tekhex format .....	1-18
1-5	Demonstration program flowchart .....	1-21
1-6	Host computer commands for preparing demonstration program .....	1-28



# Section 1

## LEARNING GUIDE

### INTRODUCTION

This Learning Guide gives an overview of the features and functions of the 8540 Integration Unit and its operating system, OS/40. It also presents a demonstration that gives you hands-on experience with the 8540. This Learning Guide is divided into the following topics:

- **Overview of the 8540 System.** Explains the role of the 8540 in the development of microprocessor-based products and describes the parts of a complete 8540 system.
- **Getting Started.** Provides instructions for starting up your 8540 and establishing communication with a host computer.
- **8540/8560 Configurations.** Explains three ways to connect an 8540, an 8560, and a system terminal.
- **Emulator Demonstration Run.** Shows you how to assemble, load, execute, and monitor a simple program.
- **For Continued Learning.** Helps you decide where to go next in this manual to accomplish your own tasks.
- **Overview of 8540 User Manuals.** Describes the types of user manuals that support your 8540.

#### NOTE

*Throughout this manual, the term "microprocessor" refers to both microprocessors and microcomputers.*

### SPECIFICATION, INSTALLATION, CONFIGURATION, AND VERIFICATION

For information on how to install and verify your 8540 hardware, and for product specifications, refer to your 8540 Installation Guide.

## OVERVIEW OF THE 8540 SYSTEM

### Uses of the 8540

Three principal stages in the development of a microprocessor-based product are:

1. **hardware development:** design and construction of a hardware prototype of the product
2. **software development:** design and creation of the program(s) that will execute in the microprocessor that controls the product
3. **hardware/software integration:** monitoring the software as it executes in the prototype hardware, and modifying the software or hardware to correct problems.

The 8540 Integration Unit is an important tool in stages 2 and 3, software development and hardware/software integration.

**Software Development.** While your prototype hardware is under development, the 8540 can help you debug the programs that will execute in the prototype. After you have written your program and compiled or assembled it on a separate computer (referred to as the **host computer**), you can download the program to the 8540 to be executed. As your program executes in the 8540, an **emulator** (also called an **emulator processor**) performs the functions of the prototype microprocessor. You can use the memory and I/O facilities of the 8540 to simulate the signals your prototype microprocessor will have to deal with.

**Hardware/Software Integration.** Once your prototype hardware is built, you can test its operation by temporarily replacing the prototype microprocessor with a **prototype control probe** from the emulator. Under this arrangement, the 8540 acts as the prototype's central processing unit, and you can monitor your program as it interacts with the prototype hardware. By doing so, you can discover errors that may remain in the software or hardware.

Your 8540 can support a variety of microprocessors. To emulate a particular microprocessor, you select the appropriate emulator hardware and support software. This design concept allows you to use a single piece of equipment for the design support of many types of microprocessor-based products.

Figure 1-1 shows a general procedure for using the 8540 Integration Unit to develop a microprocessor-based product.

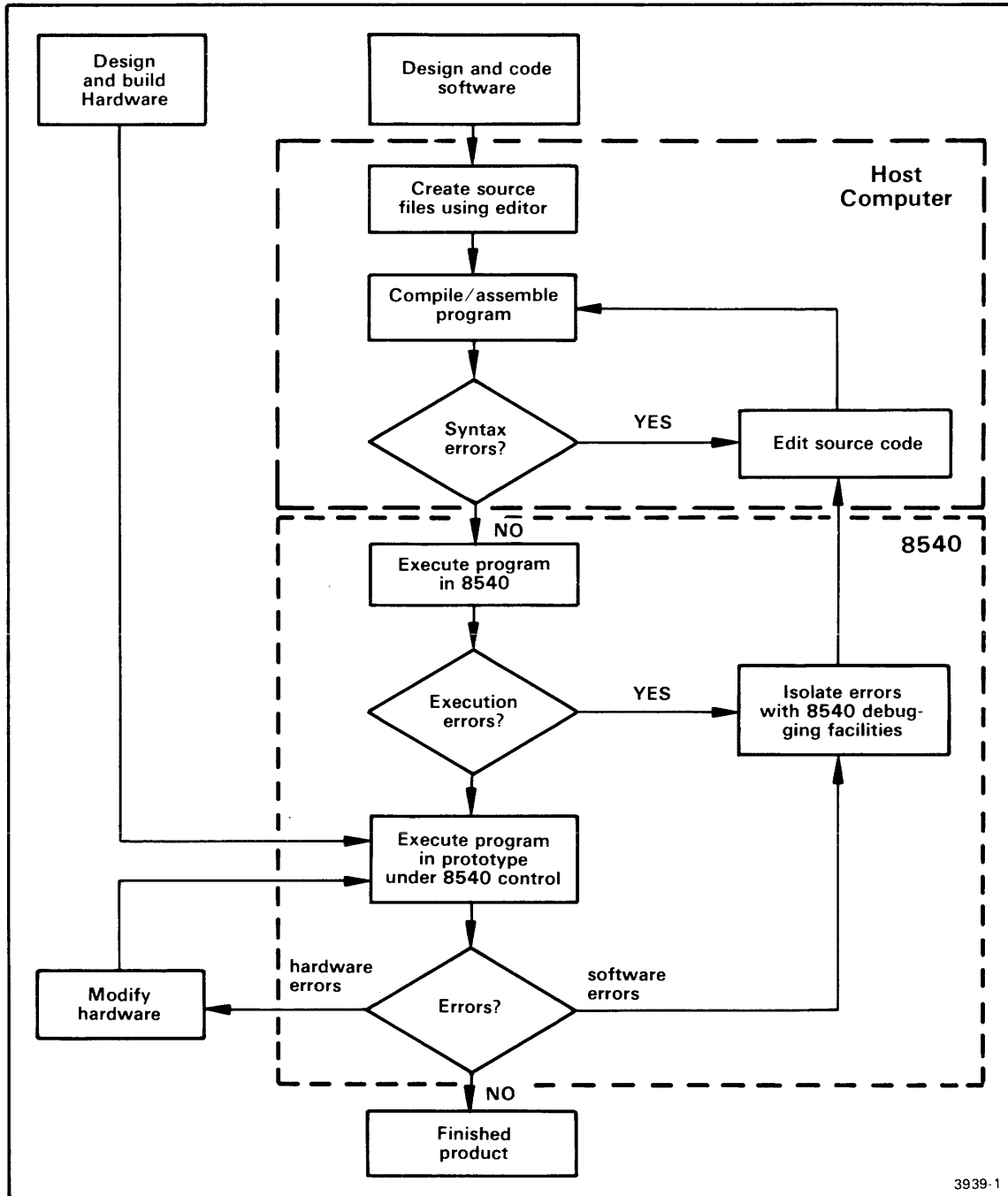


Fig. 1-1. Role of 8540 in product design.

You can use the 8540 to accomplish the tasks shown in the lower box. The host computer performs the tasks shown in the upper box.

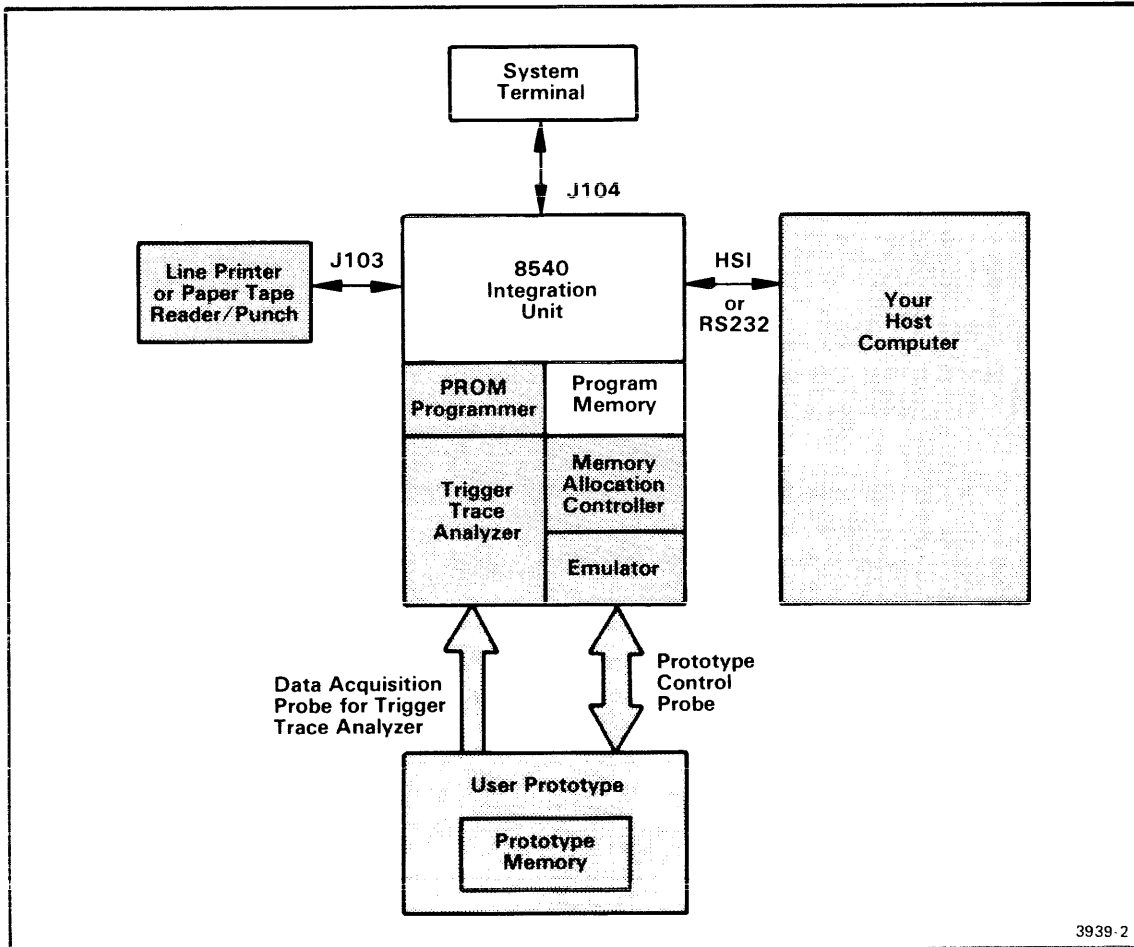


Fig. 1-2. 8540 logical subsections.

This functional diagram shows the parts of a complete 8540 system. Shaded areas indicate equipment that is not part of the minimum 8540 configuration.

### 8540 Parts and Functions

Figure 1-2 shows the components of a complete 8540 system.

#### Minimum 8540 System

A minimum 8540 system consists of an 8540 Integration Unit and a system terminal.

**8540 Integration Unit.** The 8540 mainframe houses the following components of the 8540 system:

1. **OS/40**, the ROM-based operating system of the 8540. OS/40 supervises all functions of the 8540, which include:
  - a. general input and output
  - b. program execution, monitoring, and debugging
  - c. PROM programming
  - d. communication with the host computer.
2. **Program memory** resides in the 8540. Your 8540 may be configured with 32K, 64K, or 128K of program memory. As you develop your software and integrate it with your hardware, you can use program memory as a substitute for the memory that will reside in your prototype hardware.
3. **Optional hardware** devices (described later under the heading "System Options") are attached to or installed in the 8540 mainframe.

**System Terminal.** The system terminal is a CRT or other RS-232-C-compatible I/O device through which you communicate with the 8540. Unless you specify otherwise, OS/40 accepts commands from the system terminal keyboard and displays output on the screen or printer of the system terminal. To specify the system terminal in an OS/40 command line, use the device name CONI (CONsole Input) or CONO (CONsole Output).

### System Options

**COM Interface Package.** The COM Interface package enables you to establish communication with a host computer using the OS/40 COM command. This option consists of an RS-232-C-compatible communication cable and a ROM (containing the COM command software) that inserts into the 8540's System ROM board. The 8540 does not need this option in order to communicate with an 8560 Multi-User Software Development Unit.

**Emulator.** An emulator consists of one to three circuit boards that reside in the 8540 mainframe. The emulator generally contains a microprocessor of the same type as the one being emulated, and also contains control circuitry that allows you to start, stop, and monitor program execution, using OS/40 commands. Your 8540 can contain up to two single-board emulators or one multiple-board emulator at a time.

**Prototype Control Probe.** The prototype control probe connects the prototype hardware to the emulator and contains additional control circuitry.

Once your prototype and emulator are connected, you can begin transferring responsibility for timing, I/O, and memory functions from the 8540 to the prototype hardware. To indicate which of these functions are handled by the 8540 and which are handled by the prototype, you use the OS/40 EM command to specify the **emulation mode**. Mode 0 (system mode) uses only 8540 facilities, and is the only mode you can use until your prototype is connected. Mode 1 (partial emulation mode) uses a mixture of 8540 and prototype facilities. Mode 2 (full emulation mode) uses all of the prototype's facilities. In all three modes you control program execution through the 8540. Refer to the Emulation section of this manual for a more detailed explanation of the three emulation modes.

**Trigger-Trace Analyzer (TTA).** The TTA is a debugging tool that can capture and display bus information, control execution of your program, signal external devices, and aid in performance analysis. The TTA consists of two circuit boards that reside in the 8540 mainframe and data acquisition hardware that is installed in the 8540 rear panel. If you are familiar with the TEKTRONIX Real-Time Prototype Analyzer (RTPA) and would like to begin using the TTA, see the Technical Notes section of this manual for a comparison of the two devices. For more detailed explanations of the TTA, refer to the Emulation section of this manual and to the TTA Users Manual, which is provided with your TTA hardware.

**Memory Allocation Controller (MAC) Option.** The MAC option is a circuit board that resides in the 8540 mainframe. It is designed for use with emulators for the Z8001, Z8002, 68000, and other microprocessors that can access memory outside the range of 8540 program memory. Using OS/40 commands that control the MAC option, you can map address blocks used by your program into program memory. For more information on the MAC option, refer to the Emulation section of this manual.

**PROM Programmer.** The PROM Programmer consists of a circuit board that resides in the 8540 mainframe and a separate assembly that inserts into the 8540 front panel. You may use the PROM Programmer to read data from a PROM into program or prototype memory, to burn a PROM with data from program or prototype memory, or to compare the PROM's contents with the contents of memory.

Tektronix offers a separate **characteristic module** for each family of PROM chips supported. To use a particular family of PROMs, insert the appropriate characteristic module into the front panel assembly. Refer to the PROM Programmer Specifics section of this manual for general information about the 8540's PROM Programmer and for specific information about your characteristic module.

**WARNING**

*Because of potential shock hazards, do not attempt to operate the PROM Programmer before reading the introductory information in the PROM Programmer Specifics section of this manual.*

**Line Printer.** The line printer is a hardcopy output device that attaches to jack J103 on the 8540 rear panel. You can route almost any display produced by OS/40 to the line printer by specifying LPT as the output device.

**Paper Tape Reader/Punch.** You can attach a paper tape reader/punch to jack J103 on the 8540 rear panel. The paper tape reader/punch may be used as an input device (device name PPTR) or as an output device (PPTP).

## The Host Computer

A program to be executed on the 8540 must be prepared on a separate computer, called the **host**. Programming facilities that are usually provided by the host include file management, text editing, compiling, assembling, and communication with the 8540.

The Operating Procedures section of this manual describes several procedures for intersystem communication. The Intersystem Communication section of this manual treats the topic in greater detail.

## The 8560 as Host

The TEKTRONIX 8560 Multi-User Software Development Unit is a time-sharing computer that is specially designed to serve as host to up to eight 8540s simultaneously. TNIX, the operating system of the 8560, is derived from Bell Laboratories' UNIX™ operating system. TNIX incorporates the many software development tools of UNIX™, plus additional features for handling tasks that are unique to the development of microprocessor-based software. With your 8540 and 8560 operating together under an arrangement called **TERM mode**, you have access to OS/40 and TNIX at the same time. TERM mode is discussed in more detail later in this Learning Guide and in the 8560 System Users Manual. (UNIX is a registered trademark of Bell Laboratories, Incorporated, Murray Hill, New Jersey.)

## Other Hosts

If your host computer is not an 8560, you will probably want to have the optional COM Interface package installed in your 8540. This option enables you to use the OS/40 COM command to set up communication between the 8540 and your host computer. The parameters of the COM command configure your 8540 so that it can use the same RS-232-C-compatible communication protocol as a terminal attached to your host. Once communication is established, special directives instruct the COM command to transfer data (such as your program) to or from the host.

Your host computer must convert your program to Tektronix Hexadecimal Format (Tekhex) before the COM command can download your program to the 8540. An example of Tekhex is given in the Emulator Demonstration Run later in this Learning Guide. Tekhex and the COM command are described in detail in the Intersystem Communication section.

### NOTE

*In order for an OS/40 command (other than COM) or a user program on the 8540 to access files on the host, the 8540 and the host must be operating in an 8560-style TERM interface. In this manual, it is assumed that the 8560 is the only computer that can host such a TERM interface with the 8540. However, the host side of the 8540/8560 communication protocol may be implemented on computers other than the 8560; such computers could also host a TERM interface with the 8540.*

## GETTING STARTED

This subsection provides the basic information you need in order to begin using your 8540, including how to turn the system on and off, how to enter commands, and how to establish communication with a host computer.

It is assumed that your 8540 has been unpacked, installed, and checked out, and that your system terminal and other peripherals have been configured to communicate with the 8540. If any part of your system is not ready for use, refer to your 8540 Installation Guide for instructions.

Throughout this section, it is assumed that the system terminal you use to control the 8540 is connected (via a cable) to the 8540. The next subsection, "8540/8560 Configurations," discusses three different ways to connect an 8540, an 8560, and a system terminal.

### Starting Up the 8540

#### Make Sure Your 8540 Is Plugged In

To make sure that your system is properly plugged in, verify that each of the following steps has been performed:

- Plug the system terminal into a power socket and connect it to the jack labeled **TERMINAL** on the 8540 rear panel.
- Plug any other peripherals into power sockets and connect them to the 8540 rear panel. Table 1-1 gives a complete list of jack assignments.
- If you want to operate your 8540 in **TERM** mode with an 8560, connect them with a line capable of supporting HSI (RS-422) protocol. Plug the female end of the line into the HSI jack on the 8540 rear panel, and plug the male end into an HSI I/O jack on the 8560 rear panel. (Check with your 8560 system engineer to verify that the HSI I/O jack is configured for HSI protocol.)
- If you want to operate your 8540 with a host other than the 8560, connect a terminal line from your host to one of the **REMOTE** jacks (J101 or J102) on the 8540 rear panel. Use the space in Table 1-2 to record the jack number and **REMOTE** port switch settings that are appropriate for your host.
- Plug the 8540 into a standard wall socket.

**Table 1-1**  
**Jack Assignments and Device Names for 8540 Peripherals**

Peripheral Device	Jack	Device Names
system terminal	J104	CONI (input), CONO (output)
line printer	J103	LPT
paper tape reader/punch	J103	PPTR (reader), PPTP (punch)
external computer	J101	REMI (input), REMO (output)
external computer	J102	REMI (input), REMO (output)



**Table 1-2  
COM Interface Checklist<sup>a</sup>**

Item	For your host, use:
REMOTE jack number: J101 (DTE) or J102 (DCE)	
MODE SELECT switch setting	
REMOTE BAUD switch setting	
COM command parameters to establish communication	
COM command parameters for downloads	
COM command parameters for uploads	
Log on to host computer	
Prepare program in Tekhex format.	
Download program to 8540.	
Upload program from 8540.	
Log off.	

<sup>a</sup> Fill in the information that is appropriate for your host computer. This table is not applicable if your host is an 8560.

### Start Up the 8540 and Its Peripherals

If this is the first time the system is being turned on, refer to the detailed power-up and system verification procedure in the 8540 Installation Guide.

Turn on the 8540, system terminal, and peripherals in any order. The power switch for the 8540 is on the front panel. The 8540 performs an automatic self-test on power-up or restart. This test is described in the 8540 Installation Guide.

Within a few seconds, the message

```
8540 B00T Vx.x
```

should appear on your system terminal and the SELF TEST light on the 8540 front panel should go out. Next, OS/40 checks its EEPROMs for a command string called STARTUP. OS/40 automatically executes the commands in STARTUP whenever you power up or restart your system. The STARTUP string is discussed in more detail later in this section.

After the commands in the STARTUP string have been executed, you should see the message  
OS/40 Vx.x (xxxx-xx) xx/xx/xx Copyright (C) 1981 Tektronix, Inc.  
followed by the OS/40 prompt character (>). The numbers in parentheses (xxxx-xx) can be used  
by your Tektronix service representative to determine whether your system is up to date.

If your system fails to respond as expected, check the following points:

- If you get no response within five seconds of starting up the 8540, toggle the RESTART switch on the front panel. Startup should proceed normally.
- The "Copyright" message is displayed only if your 8540 is in LOCAL mode. Your STARTUP string may contain a command that takes the 8540 out of LOCAL mode and establishes communication with your host computer. Press the RETURN key once or twice; your host computer may respond with a login prompt.

If your 8540 and your host computer are not on speaking terms yet, you need to be sure that your 8540 starts up in LOCAL mode. You can prevent execution of the STARTUP string by setting switch position 1 (DIP switch number S1100) on the 8540's System Controller board to 1 (open). Refer to your 8540 Installation Guide for instructions on accessing this switch.

If your system has startup problems that cannot be remedied by either of these solutions, refer to the verification procedures in your 8540 Installation Guide or contact your Tektronix service representative.

## Turning Off the 8540

You may turn off the 8540 and its peripherals in any order. (But don't do it now; you're just getting started.)

## How to Enter OS/40 Commands

This Learning Guide uses the following conventions in presenting information that is entered or displayed on the system terminal:

- Prompt—The OS/40 prompt character > is shown at the beginning of each command line to remind you to wait for the prompt before entering the command.
- Carriage return—In this Learning Guide, the symbol <CR> is shown at the end of each command line to remind you to end each command with a carriage return. On most keyboards, you enter a carriage return by pressing the RETURN key.
- Underlined—Characters to be entered by you are underlined. Responses by OS/40 are not underlined.
- Numbers—Addresses are in hexadecimal notation unless otherwise indicated. The suffix letters H (hexadecimal) and T (decimal) may be used for clarity when both hexadecimal and decimal numbers are used in the same discussion. For example, 10H = 16T.
- CTRL-x—Several **control characters** have special meaning to OS/40. Each control character is entered by pressing the CTRL key and another key simultaneously. To suspend a display as it appears on the system terminal, for example, you enter CTRL-S by holding down the CTRL key and pressing the S key. To resume the display (CTRL-Q), hold down the CTRL key and press Q. To interrupt the command or program that OS/40 is executing, enter CTRL-C.

### Correcting Mistakes in a Command Line

If you notice a mistake in your command line before you enter a carriage return, you have two ways of correcting the line: delete the entire line and start again, or correct the characters one-by-one.

- To delete the entire line, type CTRL-U. You may then reenter the line.
- To delete characters one-by-one, press the BACKSPACE or RUBOUT key. Either key will backspace the cursor and erase the deleted character.

### Command Example

Let's assume that your 8540 is started up and waiting for you to enter a command. (You should see the ">" prompt on your system terminal.) Enter the following command to calculate the sum of the hexadecimal numbers 44 and 55:

```
> calc 44+55 <CR>  
99H
```

OS/40 responds with the answer in hexadecimal. Now let's try something more exciting.

### Establishing Communication with the Host

The ">" prompt indicates that the 8540 is in **LOCAL mode**, acting as a stand-alone computer. The 8540 starts up in LOCAL mode and stays in that mode until it receives a "CONFIG TERM" or "COM" command that sets up communication with the host. Once you have downloaded a program from the host, you may return the 8540 to LOCAL mode to execute and debug your program. This discussion explains how to establish communication with an 8560 (TERM interface) or with some other host computer (COM interface), and how to return the 8540 to LOCAL mode.

#### 8560 TERM Interface

The sequence of commands you enter to establish communication with the 8560 depends on how your 8540, 8560, and system terminal are configured. The three most likely configurations are discussed in the next subsection, "8540/8560 Configurations." For now, let's assume that your system terminal is connected to your 8540 and that the line that runs to the 8560 is plugged into the HSI jack on the 8540 (as described earlier under "Starting Up the 8540").

Enter the following command to establish communication with the 8560:

```
> config term <CR>
```

Now the 8540 is in **TERM mode**: every character you type is sent directly to the 8560. Press the RETURN key once or twice; the 8560 responds with its "login:" prompt. Type in your 8560 user name and password, as described in the Learning Guide of your 8560 System Users Manual.

Now you're logged in to TNIX, the operating system of the 8560. The TNIX prompt is "\$". Try typing a TNIX command:

```
$ date <CR>
```

TNIX responds with the current date and time.

Try typing an OS/40 command:

```
$ calc 44+55 <CR>  
99H
```

TNIX recognizes that CALC is an OS/40 command, so it sends the command back to the 8540 for processing by OS/40. OS/40 sends its response back to the 8560, and the 8560 passes the response to you.

Notice that TNIX expects you to enter commands in lowercase. If you type "CALC 44+55", TNIX responds "CALC: not found."

In the Emulator Demonstration Run later in this Learning Guide, you'll see some practical applications of the 8540/8560 TERM interface. For now, log out from the 8560 and return the 8540 to LOCAL mode:

```
S config local; logout <CR>
```

The CONFIG command tells the 8540 to stop sending commands to the 8560, and the LOGOUT command tells the 8560 to stop accepting commands. Notice that these two commands cannot be entered in a different order or on different lines if they are to have the desired effect.

### CAUTION

*When your 8540 is in TERM mode, do not toggle the RESTART switch on the 8540 front panel. If you need to interrupt system operation, type CTRL-C.*

*If you restart the 8540 while it is in TERM mode, the 8540 and 8560 may not be able to resume communication if any OS/40 commands are still active on the 8560. If your system seems to hang after you restart the 8540, you or your 8560 superuser must kill the OS/40 commands from a different terminal. (Use the TNIX command "ps -ax" to list all active processes and use "kill -9" to kill those processes associated with your terminal.)*

## COM Interface

Table 1-2 (earlier in this section) has space for recording the parameters of the COM command you enter to establish communication with your host computer. Here are some general rules that may apply to your host:

- The default COM parameters are satisfactory for some computers. Try entering the COM command without any parameters.
- For most computers, the COM parameters E, L, and M are sufficient to specify your host's communication protocol.
- The COM parameters P, T, C, F, and HS affect data transfer protocols, and can be ignored until you are ready to test your host computer's upload and download software.

- For more information on the COM command and how to configure your 8540 communication hardware, refer to the Intersystem Communication section of this manual.

When you enter the COM command with the appropriate parameters, the 8540 responds "COM Vx.x." Now any characters you type are sent directly to the host. Try logging on to your host and entering a few commands. The host should respond just as it would if the terminal were attached directly to the host.

Try entering an OS/40 command:

```
calc 44+55 <CR>
```

Unless your host has its own CALC command, you will probably receive an error message. When the COM command is executing on the 8540, OS/40 cannot accept other commands.

The null character (ASCII code 00) has a special meaning to the COM command. On most terminals, the null character is entered as CTRL-@ (hold down the CTRL key and press the keys that produce the "@" character). When you enter the null character, COM intercepts the characters you type next and interprets them as a directive to transfer data, display status information, or terminate communication.

For now, log out of your host, and then terminate communication by entering the null character and pressing the ESC key on your terminal:

```
(Log out.)  
(null)(esc)
```

The 8540 should be in LOCAL mode once again.

## More about OS/40 Commands

### Type-Ahead

OS/40 has a feature called **type-ahead** that allows you to enter additional commands even before the current command has finished executing. When OS/40 finishes a command, it issues the > prompt and goes on to the next command entered, if any. If you are in the process of entering a command when the prompt appears, just keep typing; the prompt has no effect on the line being entered, other than to disrupt it visually.

Command lines that have been entered but have not yet begun execution are stored in the **type-ahead buffer**. This buffer also holds the command line currently being typed and has a capacity of 128 characters, including carriage returns. To display the contents of the type-ahead buffer, enter CTRL-R. To delete all text in the type-ahead buffer, type CTRL-U or CTRL-C.

For the sake of clarity in this manual, type-ahead will not be demonstrated; it will be assumed that you will wait for the > prompt before entering your next command.

### Multiple Commands

You may enter more than one command on a line if you separate the commands with semicolons. Be sure that the command line does not exceed 80 characters.

## Strings

To save typing, you can assign names to frequently used strings of characters. For example, suppose the command line that establishes communication with your host computer is:

```
> COM P=4F4B EOL=0DOA T=02 M=6 <CR>
```

The line

```
> HELLO='COM P=4F4B EOL=0DOA T=02 M=6' <CR>
```

defines a string named HELLO. Once this string is defined, every time you enter the command

```
> $HELLO <CR>
```

the string 'COM P=4F4B EOL=0DOA T=02 M=6' is substituted, and the 8540 establishes communication with your host. (Notice that the dollar sign is omitted when you define the string, but present when you use the string.)

You can use the OS/40 PERMSTR command to save strings in the 8540's EEPROMs, so that they are available for future use. Strings that are not saved in EEPROM are lost when you restart the 8540 or turn it off. For example, to save the HELLO string, type

```
> PERMSTR HELLO <CR>
```

You can also use the PERMSTR command to delete or display strings from EEPROM. If a string already exists in EEPROM, you must delete the old version of the string before you can save a new version.

If you have a sequence of commands to be executed every time you start up the 8540, store them in the STARTUP string. For example, if you want to select the Z80A emulator and establish communication with an 8560 every time you start up the 8540, type:

```
> STARTUP='CONFIG TERM; SEL Z80' <CR>
> PERMSTR -D STARTUP <CR>
> PERMSTR STARTUP <CR>
```

The first PERMSTR command deletes the old EEPROM version of STARTUP, if any. The second PERMSTR command saves the new version.

If your STARTUP string contains a CONFIG TERM command, that command must be the first command in the string.

The name of a string can be 1 to 8 letters or digits; the first character must be a letter. In a string name, a lowercase letter does not match the corresponding uppercase letter.

**TERM Mode Considerations.** On the 8560, strings are defined and used according to the same general rules as described here for the 8540. When your 8540 is operating in TERM most with an 8560, a string definition (stringname=string) or string substitution (\$stringname) always refers to an 8560 string unless you explicitly specify that the string belongs to the 8540. To define an 8540 string, precede the string definition with the word "8540" (8540 stringname=string). To use an 8540 string, precede the dollar sign with a backslash (\\$stringname).

For most TERM mode applications, it is easiest to use only 8560 strings. You may use 8560 strings in 8540 commands. However, the 8560 does not recognize strings on the 8540, and the OS/40 PERMSTR command does not recognize 8560 strings.

### Uppercase and Lowercase

OS/40 commands can be entered in uppercase, lowercase, or a mixture of the two. Uppercase and lowercase letters are interchangeable except in the name of a string, an emulator (for example, Z80), or a standard 8540 device (CONI, CONO, LPT, PPTR, PPTP, REMI, and REMO must be capitalized).

Command examples in this manual may be shown in either uppercase or lowercase. Except as noted, any parameter shown in lowercase can also be entered in uppercase.

#### NOTE

*If your 8540 is operating in TERM mode with an 8560, all OS/40 command names must be entered in lowercase.*

## 8540/8560 CONFIGURATIONS

This subsection describes the three common ways to connect an 8540, an 8560, and a system terminal, and explains how to establish communication between the 8540 and the 8560 in each configuration.

### A. Terminal-8540-HSI-8560

In this configuration, the 8540 has its own system terminal, and so can operate in either LOCAL mode or TERM mode.

**Cable Connections.** The terminal connects to the TERMINAL jack (J104) on the 8540 rear panel. The 8540 and 8560 communicate via a line that runs from the HSI jack on the 8540 to the HSI I/O jack on the 8560. The HSI I/O jack must be configured for HSI (RS-422) protocol.

**Establishing Communication.** Start up the 8540. Enter the OS/40 command **config term** to enter TERM mode. Log in to the 8560.

### B. Terminal-8540-RS232-8560

You may use this configuration in place of configuration A if your 8540 and 8560 are at separate sites and so must communicate via a modem.

**Cable Connections.** The terminal connects to the TERMINAL jack (J104) on the 8540 rear panel. The 8540 and 8560 communicate via a link that runs from the DTE jack (J101) on the 8540 (possibly through a modem) to an HSI I/O jack on the 8560. The HSI I/O jack must be configured for RS-232-C protocol. The MODE SELECT switch on the 8540 rear panel should be set to DTE1.

**Establishing Communication.** Start up the 8540. Enter the OS/40 command

```
> config term i=r t=7 <CR>
```

to enter TERM mode. Log in to the 8560.

If you cannot establish communication, your 8560 I/O port may be configured incorrectly. Log in to the 8560 on a different terminal and enter the following command:

```
$ stty IU >/dev/tty <CR>
```

where **n** is the number of the HSI I/O jack to which the 8540 is attached. Then restart the 8540 and try the **config** command again.

### C. Terminal-8560-HSI-8540

In this configuration, the 8540 can be operated from any terminal that is attached to the 8560. Since the 8540 cannot operate in LOCAL mode in this configuration, its STARTUP string must contain a **config term** command. To establish this STARTUP string, perform the following steps:

1. Attach a terminal to the 8540 (as in configurations A and B) and start up the 8540 in LOCAL mode.
2. Enter the following lines to create the STARTUP string and store it in EEPROM:

```
> STARTUP='config term' <CR>
> PERMSTR -D STARTUP <CR>
> PERMSTR STARTUP <CR>
```

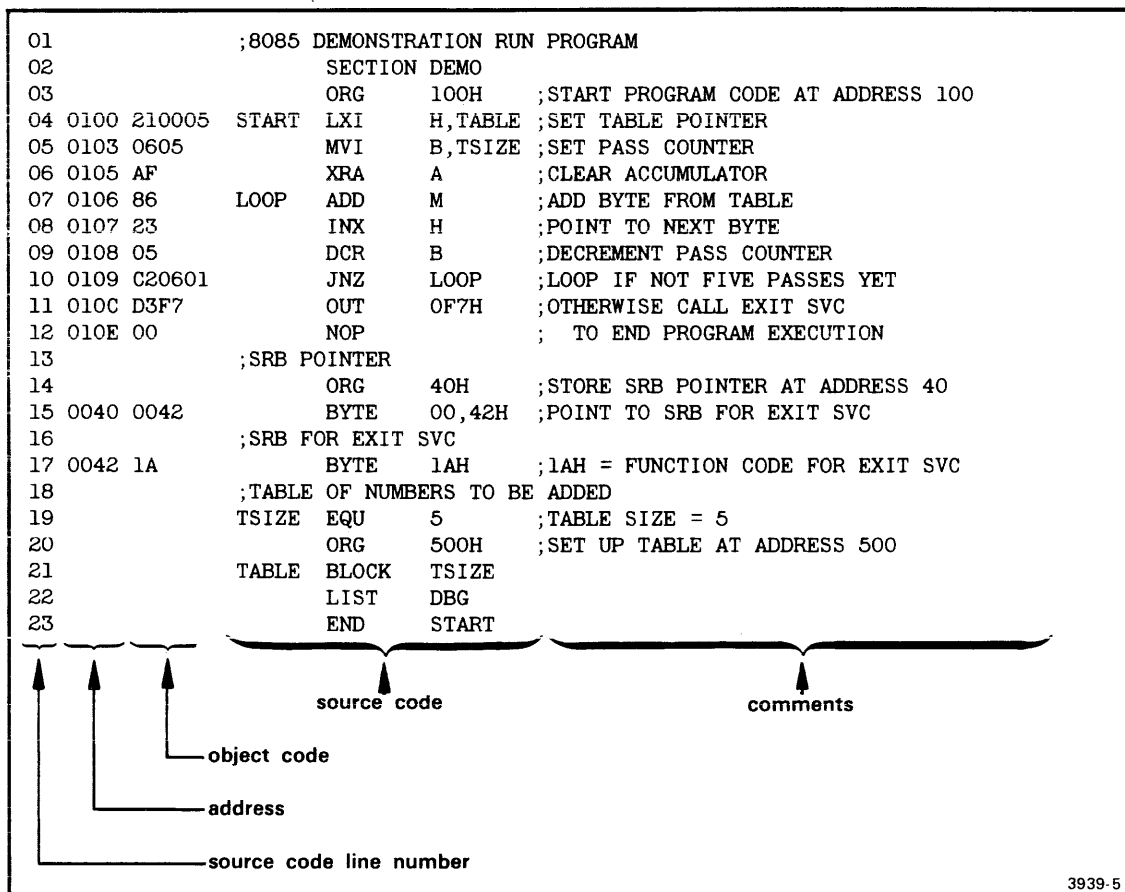


Fig. 1-3. Demonstration program.



**Cable Connections.** The terminal connects to an HSI I/O jack on the 8560. This HSI I/O jack must be configured for RS-232-C protocol. The 8540 and 8560 communicate via a line that runs from the HSI jack on the 8540 to another HSI I/O jack on the 8560. This HSI I/O jack must be configured for HSI (RS-422) protocol.

**Establishing Communication.** Log in to the 8560. To indicate which 8540 you want to use, enter the line `IU=n;export IU`, where `n` is the number of the HSI I/O jack to which the 8540 is attached. Start up the 8540. The STARTUP string puts the 8540 into TERM mode automatically.

## EMULATOR DEMONSTRATION RUN

### Introduction

This demonstration run shows you how to load, execute, and monitor a simple 8085A assembly language program on your 8540. If you have an emulator other than the 8085A, refer to the appropriate Emulator Specifics supplement in this manual for a demonstration that is parallel to this one.

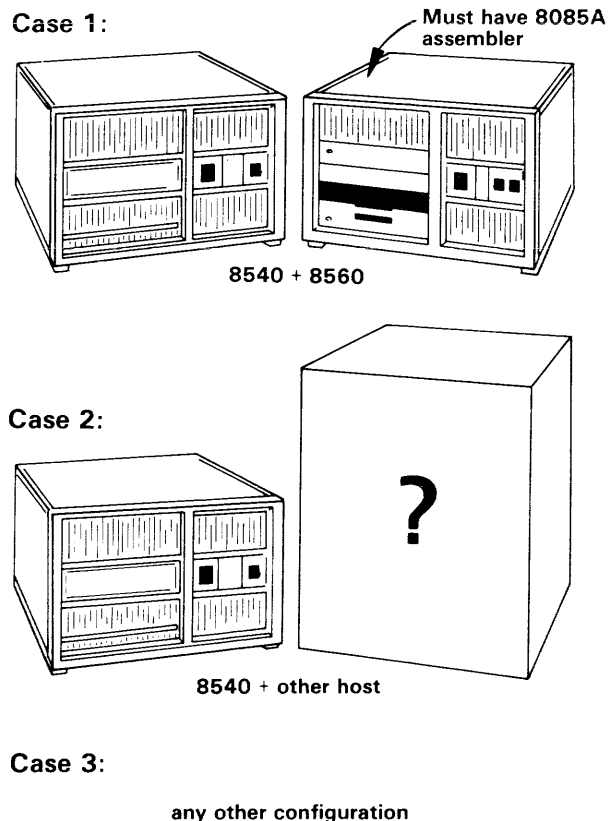
In order to perform this demonstration, your 8085A emulator board and emulator control software ROM must be installed in your 8540.

Figure 1-3 shows the source and object code for the demonstration program.

If you have an 8560, and your 8560 has an 8080A/8085A assembler installed, you can create and assemble the program on the 8560 and then download it to the 8540. This demonstration shows how.

If you have an 8540 that is connected to a host computer other than an 8560, we can't give you a specific list of commands for creating and assembling the program on your host (since we don't know what host you're using). However, Fig. 1-4 gives the object code for the program in Extended Tekhex format. You can create the Tekhex file using your host's assembler or text editor, then download the file to the 8540 via the 8540's optional COM interface.

If neither of these cases applies to you, you can patch the program into memory using the P command. This demonstration shows how.



3964-5

Once the program is loaded or patched into memory, you can execute the program on your emulator.

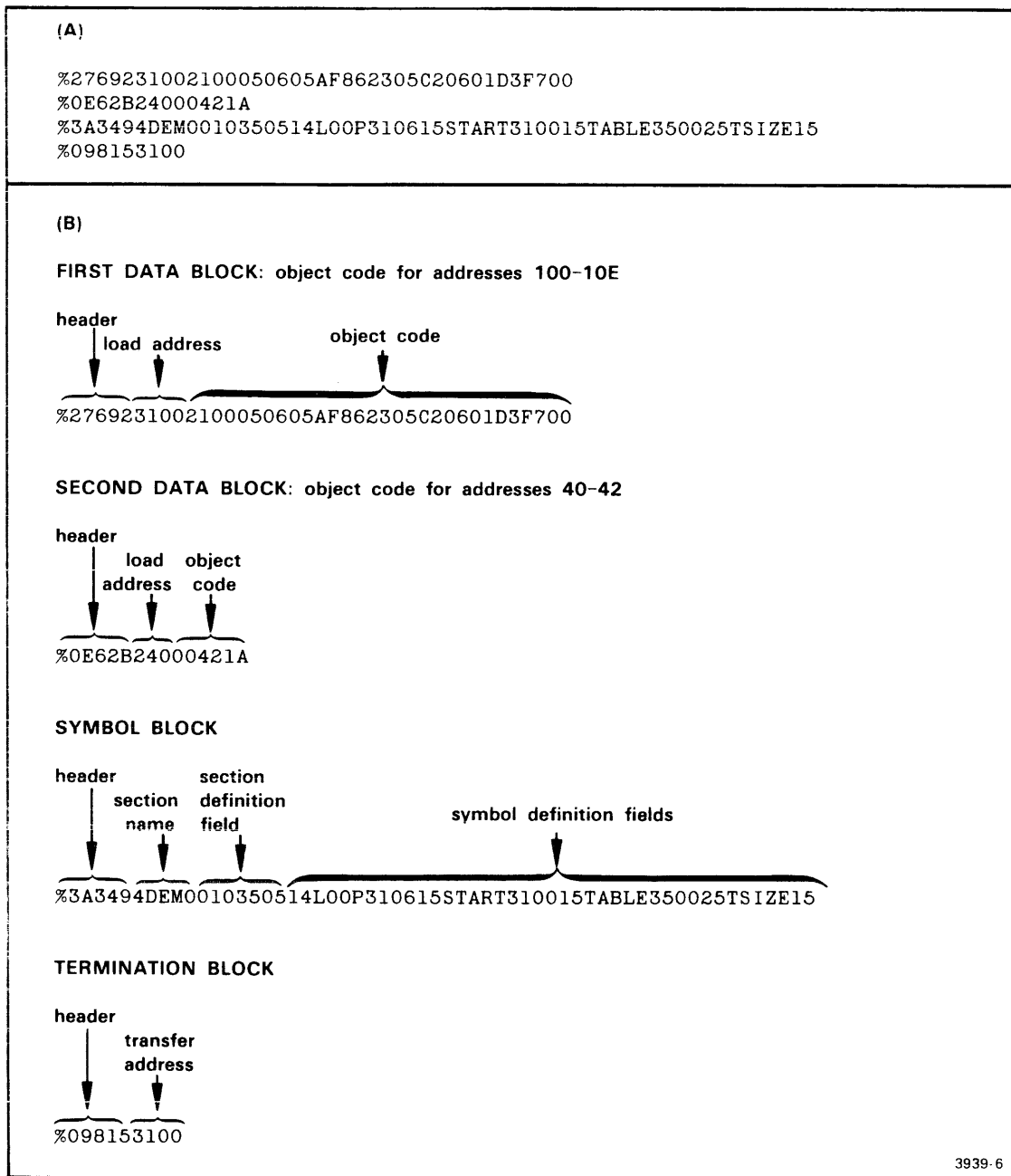


Fig. 1-4. Demonstration program: Extended Tekhex format.

Figure 1-4A shows an Extended Tekhex load module that contains the object code and program symbols for the demonstration program. Figure 1-4B gives the meanings of the different fields in the message blocks. If you have a host computer other than an 8560, you can create this load module and download it to your 8540.

## Examine the Demonstration Program

### Explanation of Demonstration Run Program Source Code

The demonstration run program adds five numbers from a table stored in locations 500–504 in program memory and leaves the sum in the accumulator. You will place values in the table later in this demonstration.

The source file contains two kinds of statements: Tektronix assembler directives and 8085A assembly language instructions. (Text following a semicolon (;) in a source line is treated as a comment by Tektronix assemblers.) The 8085A assembly language instructions represent the operations performed by the program; these instructions are discussed under the heading “How the Demonstration Run Program Works,” later in this subsection. The assembler directives provide additional information needed to create the correct object code. The following paragraphs outline the functions of the assembler directives.

**SECTION DEMO.** Every program consists of one or more **sections**. This directive declares a program section called DEMO. All object code for the demonstration program will reside in this section. (When you get into more practical programming applications, you may divide your program into different sections to reduce memory consumption or to make it easier to organize your program into RAM and ROM areas. Your Assembler Users Manual discusses the concept of sections in detail.)

**ORG 100H.** This directive tells the assembler where in memory to locate the object code for the next instruction. In this case, the object code for the 8085A instruction LXI H, TABLE will be stored at memory location 100.

**ORG 40H.** This directive specifies that the information for the Exit service call is to be stored at address 40. A service call (SVC) is a request for OS/40 to perform a special service for an executing program. An Exit SVC ends program execution and returns control to the operating system.

An SVC always has at least three parts:

- an I/O instruction that initiates the SVC
- a service request block (SRB) that contains the parameters of the SVC to be performed
- an SRB pointer that tells where in memory the SRB is located.

SVCs are explained in the Service Calls section of this manual.

In this program, the instruction OUT 0F7H directs OS/40 to perform the SVC whose parameters are pointed to by the address in locations 40 and 41.

**BYTE 00,42H.** This directive specifies that the SRB pointer (the address of the SRB for the Exit SVC) is 0042.

**BYTE 1AH.** This SRB contains only one parameter: the SVC function code (1A = Exit). No other parameters are needed.

**TSIZE EQU 5.** This directive assigns the value 5 to the symbol TSIZE. Other statements in the program use the symbol TSIZE when referring to the size of the table of numbers to be added.

**ORG 500H.** This directive specifies that the table will start at address 500.

**BLOCK TSIZE.** This directive allocates 5 (TSIZE) bytes to the table. The contents of the table are undefined; you will put values into the table later in this demonstration. The symbol TABLE, which represents the address of the table (500), is used by the LXI instruction.

**LIST DBG.** The LIST directive controls various assembler options. LIST DBG specifies that the symbols in your source code (START, LOOP, TSIZE, and TABLE) will be placed in the object file. Later in this demonstration, these symbols will be used in OS/40 commands and displays.

**END START.** The END directive signals the end of the source code and specifies that START is the **transfer address**: the address of the first instruction to be executed when you start the program with the G (Go) command. Since START is the label of the LXI instruction, that instruction will be executed first.

### How the Demonstration Program Works

The steps of the program are illustrated in the flowchart in Fig. 1-5.

**Set Table Pointer.** The first instruction in the program, LXI H, TABLE, loads the address of the table (500) into the H-L register pair. As a result, the H-L register pair points to the first element of the table. The label START represents the address of this instruction. START is used by the END directive to specify that the LXI instruction is the first to be executed.

**Set Pass Counter.** Register B is used as the pass counter. The MVI B, TSIZE instruction moves the value 5 into register B. This step sets the number of passes to 5. Each time a number is taken from the table and added into the accumulator, register B is decremented.

**Clear Accumulator.** The XRA A instruction sets the accumulator to zero. We want the accumulator to be cleared when we start adding numbers from the table.

**Add Byte from Table.** The ADD M instruction adds the data addressed by the H-L register pair into the accumulator. The label LOOP represents the address of this instruction; this label is used by the JNZ instruction.

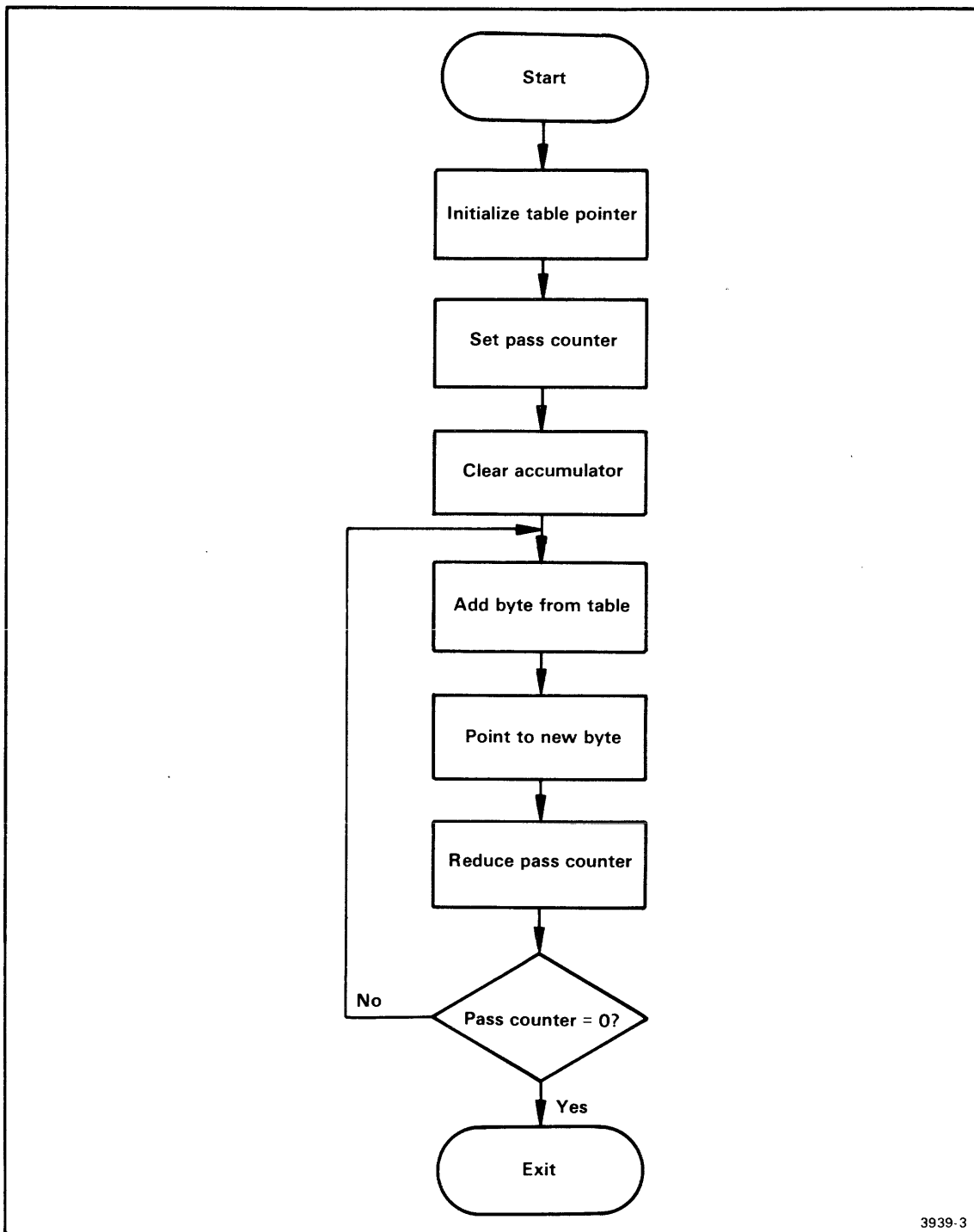


Fig. 1-5. Demonstration program flowchart.

This flowchart presents the algorithm for the program used in this demonstration run. The program adds the elements of a table in memory and leaves the sum in the accumulator. The demonstration run shows how to download the program into memory, execute the program, and monitor its execution.

**Point to Next Byte.** The INX H instruction increments the address in the H-L register pair; the H-L register pair then points to the next byte in the table. For example, the H-L register pair is initialized to contain the address 500. After the INX H instruction is first executed, the H-L register pair will contain 501, the address of the second element in the table.

**Decrement Pass Counter.** The DCR B instruction decrements register B, the pass counter. In this program, because the DCR B instruction follows the ADD M instruction, the pass counter is decremented each time a number is added to the accumulator.

**Loop If Not Five Passes Yet.** The JNZ instruction effectively checks the contents of register B and jumps to the LOOP label if the register does not contain zero. If register B contains zero, the program proceeds to the OUT 0F7H instruction.

**Exit.** The OUT 0F7H instruction followed by the NOP is a call to the Exit SVC. This SVC invokes the operating system to handle termination of the program. A NOP always follows an SVC invocation to allow the system time to execute the SVC.

## Assemble and Load the Demonstration Program

Now it's time to create the program so you can run it on your emulator. One of the following discussions describes the set of steps that is appropriate for your hardware configuration:

- For 8560 users: **Case 1: Assemble on the 8560; Download to the 8540**
- For users with host computers other than the 8560: **Case 2: Download from Your Host to the 8540**
- For other hardware configurations: **Case 3: Patch the Program into Memory.**

Go ahead and work through the discussion that's appropriate for you. Once you've put the program into program memory, turn to the heading "Run the Demonstration Program," later in this section.

### Case 1: Assemble on the 8560; Download to the 8540

This discussion shows you how to create the demonstration program source code and assemble it on the 8560, then download it to 8540 program memory. If your 8560 does not have an 8080A/8085A assembler, you cannot complete this part of the demonstration, so skip ahead to the heading "Case 3: Patch the Program into Memory" for instructions.

#### Start Up and Log In

Start up your 8540, make sure it's in TERM mode, and log in to the 8560 operating system, as described earlier in this Learning Guide.

Since you're logged in to TNIX, your system prompt is "\$". (Later in the demonstration, we'll show the system prompt as ">", in deference to people using 8540s in LOCAL mode.) Every command you enter is processed by TNIX. If you enter an OS/40 command, TNIX passes it to the 8540.

Enter the following line to select the 8080A/8085A assembler on the 8560:

```
$ uP=8085; export uP <CR>
```

Now select the 8085A emulator on the 8540:

```
$ sel 8085 <CR>
```

The **sel** command automatically sets the emulation mode to 0.

### Create the Demonstration Program

Enter the following TNIX command lines to create an empty directory called **demo** and make **demo** the working directory. You'll create your source file and related files in this **demo** directory.

```
$ mkdir demo <CR>
```

```
$ cd demo <CR>
```

Now use the TNIX editor, **ed**, to create the demonstration program source file. The following command line invokes the editor and specifies that you want to create a file called **asm**:

```
$ ed asm <CR>
```

```
?asm
```

The editor responds "?asm" to remind you that **asm** does not already exist. Notice that the editor does **not** give a prompt to let you know it's ready for input.

**Enter the Text.** Now enter the editor command **a** (add text) and type in the program. Use the **BACKSPACE** key to erase typing mistakes.

```

a <CR>
      column  column  column
      8      16      24
      ↓      ↓      ↓
;8085 DEMONSTRATION RUN PROGRAM <CR>
SECTION DEMO <CR>
ORG 100H ;START PROGRAM CODE AT ADDRESS 100 <CR>
START LXI H, TABLE ;SET TABLE POINTER <CR>
MVI B, TSIZE ;SET PASS COUNTER <CR>
XRA A ;CLEAR ACCUMULATOR <CR>
LOOP ADD M ;ADD BYTE FROM TABLE <CR>
INX H ;POINT TO NEXT BYTE <CR>
DCR B ;DECREMENT PASS COUNTER <CR>
JNZ LOOP ;LOOP IF NOT FIVE PASSES YET <CR>
OUT OF7H ;OTHERWISE CALL EXIT SVC <CR>
NOP ; TO END PROGRAM EXECUTION <CR>
;SRB POINTER <CR>
ORG 40H ;STORE SRB POINTER AT ADDRESS 40 <CR>
BYTE 00, 42H ;POINT TO SRB FOR EXIT SVC <CR>
;SRB FOR EXIT SVC <CR>
BYTE 1AH ;1AH = FUNCTION CODE FOR EXIT SVC <CR>
;TABLE OF NUMBERS TO BE ADDED <CR>
TSIZE EQU 5 ;TABLE SIZE = 5 <CR>
ORG 500H ;SET UP TABLE AT ADDRESS 500 <CR>
TABLE BLOCK TSIZE <CR>
LIST DBG <CR>
END START <CR>
. <CR>
    
```

At the end of your text, enter a period on a line by itself. The editor will go back to accepting commands.

**Check for Errors.** Enter the following editor command to display the text you have entered. Check for typing mistakes.

```

1,$p <CR>
↑↑↑
|
| print command: displays the lines
| in the designated range
|
| designates last line in file
|
| designates first line in file
    
```

If you made any mistakes, go ahead and fix them. In case you're not familiar with **ed**, Table 1-3 lists the commands you need in order to add, delete, or replace any line. For more information on **ed**, refer to your 8560 System Users Manual.



**Table 1-3**  
**Basic 8560 Editing Commands**

Command	Function
mm,nnp <CR>	Displays lines mm through nn
nn <CR>	Makes line nn the current line
d <CR>	Deletes the current line
a <CR> <line(s) of text> . <CR>	Adds text after the current line
c <CR> <line(s) of text> . <CR>	Replaces the current line with the text you type in

Once your text is correct, enter the w command to write the text to the source file, asm:

```
w <CR>
896
```

The editor responds with the number of characters it wrote to the file.

Finally, enter the q command to quit the editor and return to TNIX:

```
q <CR>
$ ← TNIX prompt
```

**Assemble the Source Code.** The TNIX asm (assemble) command translates assembly language (source code) into binary machine language (object code). The asm command also creates an assembler listing which can be used to correlate the object code with the source code. Enter the following command line to assemble the source code in the file asm and create the listing and object files asml and obj:

```
$ asm obj asml asm <CR>
```

```
Tektronix ASM 8080/8085
Vxx.xx-xx (8560)
*****Pass 2
```

```
23 Lines Read
23 Lines Processed
0 Errors
```

Enter the following command to print the assembler listing on the 8560's line printer:

```
$ lpr asml <CR>
```

Check page 1 of your listing. Did the assembler issue any error messages? There should be none. If your source code contains errors, take the following steps:

1. Refer to your Assembler Users Manual to find out what the error messages mean.
2. Enter the command **ed asm** to get back into the editor and fix the mistakes in your source code. Exit the editor with the **w** and **q** commands, as before.
3. Enter the command **asm obj asm! asm** to re-assemble your source code.

### Link the Object Code

The linker creates an executable load file from one or more object files. Enter the following command to create a load file called **load** from your object file, **obj**. Be sure to capitalize all parameters as shown.

```
$ link -d -o obj -o load <CR>
```

The **-d** option causes the linker to pass the program symbols from the object file to the load file, for use in programming debugging.

The files generated by the **asm** and **link** commands should now be in your working directory, **demo**. Enter the following command to list the files in your working directory:

```
$ ls <CR>
asm
asm1
load
obj
```

Notice that there are now four files listed in your directory. **obj** and **asm1** were created by the assembler, and **load** was created by the linker.

### Download the Program to the 8540

Now it's time to download the object code produced by the 8560's linker into 8540 program memory.

**Zero Out Memory.** Before you download any code, use the **OS/40 F (Fill)** command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no effect on how the program is loaded.) Enter the following command line to fill memory from address 40 through address 11F with zeros:

```
$ f 40 11f 00 <CR>
```

**Check That Memory Was Filled with Zeros.** Check the contents of memory with the OS/40 D (Dump) command. The D command's display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40-11F with the following command line:

```
$ d 40 11f <CR>
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

**Download the Object Code.** Enter the following command line to download the object code from the 8560 file `load` to 8540 program memory:

```
$ lo <load> <CR>
```

  
 load file

**Download the Program Symbols.** Recall that the source code for the demonstration program contains the directive `LIST DBG`. Because of this directive, the object file contains a list of the symbols that appear in the source code, and the values associated with those symbols. Because you included the `-d` option in the link command line, those symbols were passed to the load file. Use the OS/40 `SYMLO` command to download those symbols into the symbol table in 8540 system memory.

```
$ symlo -s <load> <CR>
```

The `-S` option means that both addresses and scalars are downloaded. If you omit the `-S`, only addresses are downloaded. (A scalar is a number that is not an address—for example, `TSIZE`, the length of the table.)

Later in this demonstration, whenever you use a symbol in an OS/40 command line, OS/40 refers to the symbol table to find the value that the symbol stands for.

You've assembled and linked the demonstration program and downloaded it into memory. Now skip ahead to the heading "Run the Demonstration Program."

## Case 2: Download from Your Host to the 8540

This discussion gives some general instructions for downloading the demonstration program from an unspecified host computer to 8540 program memory. If your 8540 is not equipped with the optional COM Interface Package, you cannot complete this part of the demonstration, so skip ahead to the heading "Case 3: Patch the Program into Memory" for instructions.

Since we don't know what host computer you're using, we can only provide a general outline for creating the demonstration program and downloading it to the 8540. Once you have determined the command sequence that is appropriate for your host, record this information in the space provided in 1-6.

<p><b>Create the Extended Tekhex Load Module</b></p>  <p><b>Prepare the 8540</b></p> <p>(Start up the 8540.)</p> <p>&gt; <u>SEL 8085</u> &lt;CR&gt;</p> <p>&gt; <u>F 40 11F 00</u> &lt;CR&gt;</p> <p>&gt; <u>D 40 11F</u> &lt;CR&gt;</p> <p><b>Establish Communication</b></p>  <p><b>Download the Load Module</b></p>  <p><b>Terminate Communication</b></p>  <p style="text-align: right;">3939-7</p>
---

Fig. 1-6. Host computer commands for preparing demonstration program.

### Create the Extended Tekhex Load Module

In order for the object code to be downloaded to the 8540, it must be in Extended Tekhex format, as shown in Fig. 1-4, earlier in this demonstration. You can create the load module in one of two ways:

1. Use your host computer's text editor, and key the load module in by hand.
2. Use your host computer's 8085A assembler:
  - a. Translate the demonstration program into the language of your host's 8085A assembler.
  - b. Create and assemble the source file.
  - c. Link the object code, if necessary.
  - d. Translate the object code produced by the assembler or linker into Extended Tekhex format. The Intersystem Communication section of this manual provides a general algorithm for conversion to Extended Tekhex format.

### Prepare the 8540

Start up your 8540 and enter the following command to select the 8085A emulator:

```
> SEL 8085 <CR>
```

The SEL command automatically sets the emulation mode to 0.

**Zero Out Memory.** Before you download any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no effect on how the program is loaded.) Enter the following command line to fill memory from address 40 through address 11F with zeros:

```
> F 40 11F 00 <CR>
```

**Check That Memory Was Filled with Zeros.** Check the contents of memory with the OS/40 D (Dump) command. The D command's display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40–11F with the following command line:

```
> D 40 11F <CR>
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

### Download the Load Module to the 8540

Be sure that your 8540 and your host computer are connected via an RS-232-C-compatible communications link. Then perform the following steps to download the Tekhex load module to 8540 program memory. (Refer to the Intersystem Communication section of this manual to determine the commands and parameters that are appropriate for your host computer.)

- a. Enter the 8540 COM command to establish communication. (The parameters of the COM command are host-specific.) Log on to your host and execute any necessary host initialization commands.
- b. Enter the command line that downloads the Tekhex load module to the 8540. This command line consists of a host computer command that performs the download, followed by a null character (CTRL-@ on most terminals) and a carriage return. COM places the object code in 8540 program memory, and puts the program symbols into the symbol table in 8540 system memory.
- c. Log off from your host, and then terminate COM command execution by entering the null character, then pressing the ESC key.

Once you've downloaded the program to the 8540, skip ahead to the heading "Run the Demonstration Program."

### Case 3: Patch the Program into Memory

This discussion shows you how to patch the demonstration program into 8540 program memory using the P command, and then add the program symbols into the symbol table using the ADDS command.

Ordinarily, you would download the object code and symbols from a binary or hexadecimal load file on a host computer, as illustrated for Cases 1 and 2. The procedure presented here is **not** normally used for preparing a program for execution. Use this procedure only if you have no standard means for preparing the program, but would still like to try out your emulator.

### Start Up the 8540

Start up your 8540 and enter the following command to select the 8085A emulator:

```
> SEL 8085 <CR>
```

The SEL command automatically sets the emulation mode to 0.

### Zero Out Memory

Before you patch in any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. Enter the following command line to fill memory from address 40 through address 11F with zeros:

```
> F 40 11F 00 <CR>
```

**Check That Memory Was Filled with Zeros.** Check the contents of memory with the OS/40 D (Dump) command. The D command's display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40-11F with the following command line:

```
> D 40 11F <CR>
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

### Patch the Object Code into Memory

The OS/40 P (Patch) command stores a sequence of bytes into memory, replacing the previous memory contents. Enter the following command to store the object code for the first three instructions in the program (LXI, MVI, and XRA) starting at location 100:

```
> P 100 210005 0605 AF <CR>
```

Now patch in the next four instructions (ADD, INX, DCR, and JNZ)...

```
> P 106 86 23 05 C20601 <CR>
```

... and now the last two instructions (OUT and NOP):

```
> P 10C D3F7 00 <CR>
```

Finally, patch in the Exit SVC information at address 40:

```
> P 40 00421A <CR>
```

You'll check the contents of memory later in this demonstration.

### Put Symbols into the Symbol Table

Later in this demonstration, you will use symbols from the demonstration program (START, LOOP, TSIZE, and TABLE) when communicating with OS/40. Whenever you use a symbol in a command line, OS/40 consults a symbol table in 8540 system memory to find the values that the symbol stands for. Enter the following command line to add the program symbols to the symbol table, along with their values:

```
> ADDS START=100 LOOP=106 -S TSIZE=5 TABLE=500 <CR>
```

The -S parameter indicates that TSIZE is a scalar, not an address.

The ADDS command cannot provide all the symbol-related information that is provided by the SYMLO command (as in Case 1) or the COM command (as in Case 2). Because this information is missing, some of the symbolic displays you produce later in this demonstration will not match the displays shown in this manual. For more information on the ADDS command, refer to the Command Dictionary of this manual.

You've patched the demonstration program into program memory and placed the program symbols in the symbol table. Now it's time to run the program.

### Run the Demonstration Program

From now until the end of the demonstration, the commands you are to enter are shown in lowercase. If you are not logged in to an 8560, you may enter commands in either lowercase or uppercase. If you **are** using an 8560, you **must** enter the name of every command in lowercase (and your system prompt is "\$", not ">").

Now that you've loaded the program into memory, you need to:

1. verify that the program was loaded correctly
2. put values into the table in memory, for the program to add.

**Check Memory Contents Again.** Before you loaded the program, you filled memory locations 40-11F with zeros. Look at the same memory area again with the following command line:

```
> d 40 11f <CR>
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000040 00 42 1A 00 00 00 00 00 00 00 00 00 00 00 00 00 .B.....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 21 00 05 06 05 AF 86 23 05 C2 06 01 D3 F7 00 00 !.....#.....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```



The object code is loaded in two different blocks:

- The 8085A machine instructions are loaded at address 100 (specified by the first ORG directive in the source code).
- The information for the Exit SVC is loaded at address 40 (specified by the second ORG directive).

The contents of the table at address 500 are still undefined, but you'll put some values into the table in just a few minutes.

**Turn On Symbolic Display.** Enter the following command to tell OS/40 to modify its displays by replacing hexadecimal numbers with symbols from your program, where appropriate:

```
> synd on <CR>
```

**Disassemble the Object Code.** The DI (Disassemble) command displays memory contents both in hexadecimal notation and in assembly language mnemonics. You can use the DI command to verify that the object code in memory corresponds to your source code. Enter the following command to disassemble the area of memory occupied by the executable part of your program:

```
> di 100 10e <CR>
```

LOC	INST	MNEM	OPER
SECTION (DEMO)			
START	210005	LXI	H,0500
+000103	0605	MVI	B,05
+000105	AF	XRA	A
LOOP	86	ADD	M
+000107	23	INX	H
+000108	05	DCR	B
+000109	C20601	JNZ	0106
+00010C	D3F7	OUT	F7
+00010E	00	NOP	

Compare the DI display with the assembler listing you generated earlier, or refer back to Fig. 1-3.

The line "SECTION (DEMO)" in the DI display indicates that the object code being disassembled comes from the program section called DEMO. In fact, the entire memory area used by your program (location 0 through the end of the table—location 504) belongs to section DEMO. This section was declared by the SECTION directive in the source code. (If you used the ADDS command to create your symbols, as in Case 3, the section name shown in the DI display is NO.SECTION.)

The LOC (location) column of the DI display contains information that enables you to correlate the display with your assembler listing. The symbols START and LOOP in the DI display correspond to the labels START and LOOP in the source code. For those lines of the display where the location does not correspond to a label in the symbol table, DI substitutes the address of the instruction **relative to the beginning of the section**, as shown in the address field of your assembler listing. If you don't load the pertinent symbols and related information into the symbol table (using a command such as SYMLO), the DI command supplies absolute (actual) addresses in the LOC column. (Since section DEMO begins at address 0, the relative address, or **offset**, is the same as the absolute address in this display. This offset feature is much more useful for sections that **don't** start at address 0.)

Now you've seen that OS/40 can use the symbol table to translate numbers into symbols to make a display easier to read. OS/40 can also translate a symbol in a command line into an address. For example, since OS/40 knows that the symbol START is equivalent to the address 100, you could have entered the DI command in any of the following ways:

```
di 100 10E
di START 10E
di start start+0e
di 100 START+0E
```

Notice that a symbol can be entered in either lowercase or uppercase.

The feature that enables OS/40 to correlate symbols from your program with the numbers they represent is termed **symbolic debug**.

**Put Values into the Table in Memory.** The demonstration program sums five numbers from a table in memory. Use the P (Patch) command to store the numbers 1, 2, 3, 4, and 5 into the table. Do you remember what the address of the table is? It doesn't matter, as long as you remember that the symbol TABLE represents that address.

```
> p table 0102030405 <CR>
```

**Check the Contents of the Table.** Use the D command to display the contents of the table. (When you don't specify an upper boundary for the area to be dumped, the D command dumps 16 bytes.)

```
> d table <CR>
```

```
000500 01 02 03 04 05 CD 4E 04 3A C7 16 FE 00 CA 35 05 .....N:.....5.
```

Notice that bytes 500-504 (the table) contain the values you patched in. Bytes 505-50F contain random data left over from previous system operations.

The following command dumps only the contents of the table:

```
> d table table+tsize-1 <CR>
```

```
000500 01 02 03 04 05 .....
```

### Start Program Execution

Enter the G (Go) command to start program execution at location 100, the transfer address specified by the END directive in the source code. (If you followed "Case 3: Patch the Program into Memory," you must enter "g start" instead.)

```
> g <CR>
```

LOC	INST	MNEM	OPER	SP	F	A	B	C	D	E	H	L	IM	SOD
SECTION (DEMO)														
+00010F	00	NOP		0000	54	0F	00	00	00	00	05	05	00	0
+00010F	<BREAK	>												

The program executes, and when the Exit SVC occurs, the program breaks (stops), and the contents of the emulator registers are displayed. The accumulator contains the sum of the numbers in the memory table: 1+2+3+4+5=0F.

### Monitor Program Execution

You have assembled, loaded, and executed the demonstration program. The rest of this demonstration shows you some commands for monitoring program execution. You can watch the changes in the emulator's registers and observe the effect of each instruction as the program proceeds.

**Trace All Instructions.** The TRA (TRAcE) command lets you observe the changes in the 8085A registers as the program proceeds. When you enter a TRA command and then start execution with the G command, display lines are sent to the system terminal. As each instruction executes, the display line shows the instruction (as in the DIsassemble display) and the contents of the registers after that instruction has executed. Enter the following command to trace all of the program's instructions:

```
> tra all <CR>
```

Enter the command G START (or G 100) to resume program execution back at the beginning of the program:

```
> g start <CR>
```

As the program executes, the following trace is displayed. Remember that you can type CTRL-S to suspend the display and CTRL-Q to resume the display.

```

LOC      INST  MNEM  OPER      SP      F  A  B  C  D  E  H  L  IM  SOD
SECTION (DEMO)
START 210005 LXI   H,0500    0000  54  0F 00 00 00 00 05 00 00 0
+000103 0605 MVI   B,05     0000  54  0F 05 00 00 00 05 00 00 0
+000105 AF    XRA   A         0000  44  00 05 00 00 00 05 00 00 0
LOOP  86    ADD   M         0000  00  01 05 00 00 00 05 00 00 0
+000107 23    INX   H         0000  00  01 05 00 00 00 05 01 00 0
+000108 05    DCR   B         0000  10  01 04 00 00 00 05 01 00 0
+000109 C20601 JNZ   0106     0000  10  01 04 00 00 00 05 01 00 0
LOOP  86    ADD   M         0000  04  03 04 00 00 00 05 01 00 0
+000107 23    INX   H         0000  04  03 04 00 00 00 05 02 00 0
+000108 05    DCR   B         0000  14  03 03 00 00 00 05 02 00 0
+000109 C20601 JNZ   0106     0000  14  03 03 00 00 00 05 02 00 0
LOOP  86    ADD   M         0000  04  06 03 00 00 00 05 02 00 0
+000107 23    INX   H         0000  04  06 03 00 00 00 05 03 00 0
+000108 05    DCR   B         0000  10  06 02 00 00 00 05 03 00 0
+000109 C20601 JNZ   0106     0000  10  06 02 00 00 00 05 03 00 0
LOOP  86    ADD   M         0000  04  0A 02 00 00 00 05 03 00 0
+000107 23    INX   H         0000  04  0A 02 00 00 00 05 04 00 0
+000108 05    DCR   B         0000  10  0A 01 00 00 00 05 04 00 0
+000109 C20601 JNZ   0106     0000  10  0A 01 00 00 00 05 04 00 0
LOOP  86    ADD   M         0000  04  0F 01 00 00 00 05 04 00 0
+000107 23    INX   H         0000  04  0F 01 00 00 00 05 05 00 0
+000108 05    DCR   B         0000  54  0F 00 00 00 00 05 05 00 0

LOC      INST  MNEM  OPER      SP      F  A  B  C  D  E  H  L  IM  SOD
SECTION (DEMO)
+000109 C20601 JNZ   0106     0000  54  0F 00 00 00 00 05 05 00 0
+00010C D3F7  OUT   F7         0000  54  0F 00 00 00 00 05 05 00 0
+00010C <BREAK      TRACE>

```

After the accumulator is cleared, it begins to store the sum of the numbers being added. The ADD M instruction adds a number from the table into the accumulator. At the end of the program, the accumulator contains the sum of the numbers you put into the table.

Register B, the pass counter, is set to contain 5 (TSIZE) at the beginning of the program. It decreases by one (because of the DCR B instruction) each time a number is added into the accumulator. The program ends after register B reaches zero.

The H-L register pair, set to contain 500 (TABLE) at the start of the program, increases by one (because of the INX H instruction) each time a number is added to the accumulator. At the end of the program, the register pair has been incremented five times and contains 505.

**Trace to the Line Printer.** By adding the parameter >LPT to a command, you can direct that command's output to the line printer instead of to the system terminal. First, verify that your line printer is properly connected and powered up. Then enter the following command to execute the program with trace output directed to the line printer:

**NOTE**

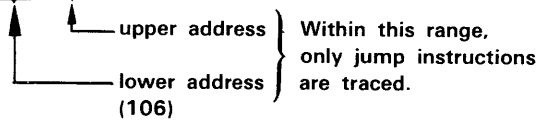
If you are operating in TERM mode with an 8560, use one of the following commands in place of the command shown:

- `g start /lp1r` sends the display to the 8560's line printer.
- `g start \>LPT` sends the display to the 8540's line printer.

> g start >LPT <CR>

**Trace Jump Instructions Only.** Another way to monitor the program's execution is to look only at the jump instructions. By tracing the jump instructions, you can still observe the changes in the registers, but you save time and space by not tracing the instructions within the loop. Enter the following command to trace only the jump instructions when the loop is being executed:

> tra jmp loop 109 <CR>



**Check the Status of the Trace.** The TRA command without any parameters displays the trace conditions that are currently set. Because you can have up to three trace selections in effect at the same time, it can be useful to be able to see which selections are active. Check your trace status with the following command line:

```
> tra <CR>
TRACE ALL,000000,00FFFF
TRACE JMP,LOOP,000109
```

As you've specified, TRA ALL is in effect for addresses 0-105, TRA JMP is in effect for addresses 106-109, and TRA ALL is again in effect for addresses 10A-FFFF.

Again, start your program with the G command. The following trace is displayed:

> g start <CR>

LOC	INST	MNEM	OPER	SP	F	A	B	C	D	E	H	L	IM	SOD
SECTION (DEMO)														
START	210005	LXI	H,0500	0000	54	0F	00	00	00	00	05	00	00	0
+000103	0605	MVI	B,05	0000	54	0F	05	00	00	00	05	00	00	0
+000105	AF	XRA	A	0000	44	00	05	00	00	00	05	00	00	0
+000109	C20601	JNZ	0106	0000	10	01	04	00	00	00	05	01	00	0
+000109	C20601	JNZ	0106	0000	14	03	03	00	00	00	05	02	00	0
+000109	C20601	JNZ	0106	0000	10	06	02	00	00	00	05	03	00	0
+000109	C20601	JNZ	0106	0000	10	0A	01	00	00	00	05	04	00	0
+00010C	D3F7	OUT	F7	0000	54	0F	00	00	00	00	05	05	00	0
+00010C	<BREAK		TRACE>											

As with the TRA ALL display, observe that register B (the pass counter) is decremented; the H-L register pair (the table pointer) is incremented; and the accumulator stores the sum of the numbers from the table. With the TRA JMP selection in effect, the instructions within the loop are not displayed.

**Set a Breakpoint.** Now that you've seen how the program adds the numbers together, here's a new task: to add only the third and fourth numbers from the table. To perform this task, you want the pass counter to contain 2, and the table pointer to contain 502 (the address of the third number in the table). You can accomplish these changes without altering the object code in memory. First, stop program execution after the pass counter and the table pointer have been set. Next, while the program is stopped, enter new values for the pass counter and table pointer. When execution resumes, the program will treat the new values as if they were the original programmed values.

Enter the following command line to trace all of the instructions as the program executes:

```
> tra all <CR>
```

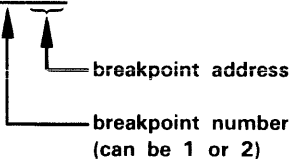
Check the status of the trace with the following command line:

```
> tra <CR>
TRACE    ALL,000000,00FFFF
```

The trace selections you set earlier are made obsolete by the TRA ALL command just entered.

Now you set a breakpoint so that the program stops after the table pointer and pass counter have been set. The following command will cause the program to stop after it executes the MVI instruction at address 103:

```
> bk 1 103 <CR>
```



Use the G command to start program execution:

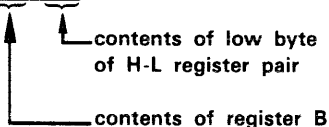
```
> g start <CR>
```

LOC	INST	MNEM	OPER	SP	F	A	B	C	D	E	H	L	IM	SOD
SECTION (DEMO)														
START	210005	LXI	H,0500	0000	54	0F	00	00	00	00	05	00	00	0
+000103	0605	MVI	B,05	0000	54	0F	05	00	00	00	05	00	00	0
+000103	<BREAK		TRACE, BKPT1>											

The TRA ALL command enabled display of all instructions up to and including the instruction at the breakpoint.

**Set New Values in Pass Counter and Table Pointer; Check Results.** Now that you've reached the breakpoint, you can change the contents of the registers while execution is stopped. The break display shows that register B (the pass counter) contains 5, and the H-L register pair (the table pointer) contains the address 500. Use the S (Set) command to set the number of passes to two and set the table pointer to 502.

```
> s B=2 L=2 <CR>
```



The S command does not produce a display, but you can use the DS (Display Status) command to check the values in the registers you changed. DS displays the contents of each emulator register and status flag. Check the result of the previous S command with the following command line:

```
> ds <CR>
PC=0105 SP=0000 F=54 A=0F B=02 C=00 D=00 E=00 H=05 L=02
SOD=0 SID=0 I7=0 I6=0 I5=0 IE=0 M7=1 M6=1 M5=1
```

The DS display shows that the pass counter and table pointer now contain the new values.

**Resume Program Execution.** If you enter the G command with no parameters, program execution starts where it left off. Resume program execution after the breakpoint with the following command:

```
> g <CR>
```

LOC	INST	MNEM	OPER	SP	F	A	B	C	D	E	H	L	IM	SOD
SECTION (DEMO)														
+000105	AF	XRA	A	0000	44	00	02	00	00	00	05	02	00	0
LOOP	86	ADD	M	0000	04	03	02	00	00	00	05	02	00	0
+000107	23	INX	H	0000	04	03	02	00	00	00	05	03	00	0
+000108	05	DCR	B	0000	10	03	01	00	00	00	05	03	00	0
+000109	C20601	JNZ	0106	0000	10	03	01	00	00	00	05	03	00	0
LOOP	86	ADD	M	0000	00	07	01	00	00	00	05	03	00	0
+000107	23	INX	H	0000	00	07	01	00	00	00	05	04	00	0
+000108	05	DCR	B	0000	54	07	00	00	00	00	05	04	00	0
+000109	C20601	JNZ	0106	0000	54	07	00	00	00	00	05	04	00	0
+00010C	D3F7	OUT	F7	0000	54	07	00	00	00	00	05	04	00	0
+00010C	<BREAK		TRACE>											

Notice that the program performed two passes through the loop, and that the program added the third and fourth numbers in the table: 3+4=7.

## Summary of Emulator Demonstration Run

You have assembled, loaded, executed, and monitored the demonstration run program. Review the commands you used:

- SEL—selects the 8085A assembler and emulator
- ASM (TNIX command)—creates object code from an assembly language program
- LINK (TNIX command)—links object code into a load module
- F—fills an area of memory with a specified value
- D—displays memory contents in ASCII and hexadecimal format
- LO or COM—downloads object code into memory
- SYMLO or COM—downloads program symbols for use in symbolic debug
- DI—translates memory contents into assembly language mnemonics
- P—patches a string of bytes into memory
- G—begins or resumes program execution
- TRA—selects instructions to be traced during program execution

- BK—sets a breakpoint
- S—modifies emulator registers
- DS—displays emulator registers

### Delete the Demonstration Run Files

Now that you've finished the demonstration run, you can delete the files you created on the host. If your files are on the 8560, you can use the following procedure to delete them.

Enter the following command to remove all the files in the working directory:

```
$ rm * <CR>
```

Now move from the **demo** directory back into the parent directory and remove the **demo** directory itself:

```
$ cd .. <CR>  
$ rmdir demo <CR>
```

To log out from your 8560 and return your 8540 to LOCAL mode, type:

```
$ config local; logout <CR>
```

You can turn off the 8540 and its peripherals in any order.

## FOR CONTINUED LEARNING

This Learning Guide introduced the basic concepts needed to use your 8540. It gave you an overview of the 8540 and related products, and showed you how to prepare, download, execute, and monitor a program. To obtain more detailed explanations of various 8540 operations, refer to the following sections:

**Section 2, Operating Procedures.** Describes a series of tasks and lists the commands needed to perform these tasks.

**Section 3, Command Dictionary.** Provides a description and examples of each OS/40 command. The Command Dictionary is arranged alphabetically by command name. It is preceded by a classified list of commands to help you choose a command by its function if you don't remember its name.

**Section 4, Intersystem Communication.** Gives the technical information needed to set up communication between the 8540 and another computer system.

**Section 5, Emulation.** Summarizes the emulation features of the 8540. The emulator demonstration run in this Learning Guide demonstrated program execution in emulation mode 0. Section 5 discusses modes 1 and 2 as well, and gives detailed explanations of symbolic debug, the Memory Allocation Controller (MAC) option, and the Trigger-Trace Analyzer (TTA).



**Section 6, Service Calls.** Explains what service calls are and how you use them in your program to access I/O devices on the 8540 and files on the 8560.

**Section 7, Emulator Specifics.** Provides a place to insert your Emulator Specifics supplements. Each supplement gives reference information that is specific to a particular emulator, and also contains a demonstration run and installation instructions for that emulator.

**Section 8, PROM Programmer Specifics.** Describes the general features and operation of the PROM Programmer. Add your PROM Programmer Specifics supplements to this section. Each supplement provides reference information that is specific to a particular PROM Programmer characteristic module.

**Section 9, Technical Notes.** Provides miscellaneous technical information. Technical Note 1 discusses the practical limits of emulation. Technical Note 2 summarizes the differences between OS/40 Version 1 and DOS/50 Version 2, the operating system of the 8550 Microcomputer Development Lab. Technical Note 3 summarizes the differences between the RTPA and the TTA.

**Section 10, Error Messages.** Explains each OS/40 error message, giving a description of the problem and possible solutions.

**Section 11, Tables.** Summarizes reference information in tabular form.

**Section 12, Glossary.** Defines special terms used in this manual.

**Section 13, Index.** Gives you a place to start when you don't know where else to look.

## OVERVIEW OF 8540 USER MANUALS

This subsection describes the types of user manuals you will receive as you add options to your 8540 system, and explains how each manual relates to the product it documents.

### Core Manuals and Specifics Supplements

Many software products produced by Tektronix have a **microprocessor-independent** part and a **microprocessor-specific** part. The microprocessor-independent part is common to every product of the same type, and minimizes your effort in using your 8540 with a new microprocessor. The microprocessor-specific part allows you to exploit the features of the microprocessor you have chosen.

For example, on the 8560, all B Series assemblers have the same standard directives, error messages, and special features, but each assembler supports the instruction set, addressing modes, and other features of a different microprocessor.

To document software that has both microprocessor-independent and microprocessor-specific features, Tektronix provides **core manuals** and **specifics supplements**.

### Core Manuals

A core manual is a looseleaf book that documents the microprocessor-independent features of a product. A core manual has at least one empty section into which specifics supplements are inserted.

Examples of core manuals include:

- the 8540 System Users Manual (this manual), which describes the standard features of OS/40 and introduces you to many system options. This manual has an Emulator Specifics section and a PROM Programmer Specifics section.
- the 8500 Modular MDL Series Assembler Core Users Manual for B Series Assemblers, which tells you how to use the B Series Linker and Library Generator and describes the microprocessor-independent features of the B Series Assembler. This Assembler Users Manual has an Assembler Specifics section. (It also contains a Host Specifics section, since B Series assemblers are available on the 8550 as well as the 8560.)

### Specifics Supplements

A specifics supplement is a relatively short document that is designed to be inserted into the Specifics section of a core manual. A specifics supplement explains the microprocessor-specific features of a software product. It usually tells how to install the software into the operating system and provides a microprocessor-specific demonstration run.

Examples of specifics supplements include:

- Emulator Specifics and PROM Programmer Specifics supplements to this manual
- Assembler Specifics supplements to the Assembler Users Manual

### Stand-Alone Manuals

Some products are either fully microprocessor-independent or fully microprocessor-specific. Such a product is documented in a single complete manual.

An example of a stand-alone manual is the Trigger-Trace Analyzer Users Manual, which describes the TTA in detail and explains its various applications.

### Reference Booklets and Cards

Most software products are also accompanied by a reference booklet or reference card that summarizes the most commonly used information from the user manual.

## Section 2

# OPERATING PROCEDURES

	Page		Page
<b>Introduction</b> .....	2-1	<b>Using the Trigger Trace Analyzer</b> .....	2-14
<b>Communicating With an 8560</b> .....	2-1	Breaking on a Designated Instruction .....	2-14
Establishing Communication With the 8560 .....	2-1	Breaking on a Specified Pass Through a Loop .....	2-15
Downloading a Program from the 8560		Breaking When Execution Proceeds Outside	
to 8540 Program Memory .....	2-2	a Designated Range .....	2-15
Uploading a Program from 8540		Breaking When the Program Writes in	
Program Memory to the 8560 .....	2-2	a Designated Memory Area .....	2-15
Downloading Program Symbols from the 8560 .....	2-2	Saving an Execution Trace Record Without	
Terminating Communication with an 8560 .....	2-2	Interrupting the Program .....	2-16
<b>Communicating with a Host Computer</b> .....	2-3	Recording Instructions Before and After	
Establishing Communication with the		a Designated Instruction .....	2-16
Host Computer .....	2-3	Recording Instructions Executed After	
Downloading a Program from the Host Computer		a Designated Instruction .....	2-17
to 8540 Program Memory .....	2-4	Determining the Execution Time of a	
Uploading a Program from 8540 Program		Program Segment .....	2-17
Memory to the Host Computer .....	2-5	Counting the Occurrences of an Event .....	2-17
Terminating Communication with the Host .....	2-5	Measuring the Interval Between Probe Events .....	2-18
<b>Communicating with an 8550</b> .....	2-5	Breaking on a Probe Event .....	2-18
Establishing Communication with an 8550 .....	2-5	<b>Memory Manipulation</b> .....	2-19
Downloading a Program from the 8550 to		Displaying the Contents of Memory .....	2-19
8540 Program Memory .....	2-6	Changing the Contents of a Section of Memory .....	2-19
Uploading a Program from 8540 Program		Initializing a Block of Memory .....	2-20
Memory to the 8550 .....	2-7	Breaking When the Program Writes in	
Terminating Communication With an 8550 .....	2-7	a Designated Memory Block .....	2-20
<b>Downloading from an 8002A</b> .....	2-7	Disassembling the Contents of Memory .....	2-20
<b>Program Execution</b> .....	2-8	<b>System I/O</b> .....	2-20
Selecting the Target Processor .....	2-8	Displaying the Current Channel Assignments .....	2-20
Selecting the Emulation Mode .....	2-8	Assigning a Channel .....	2-21
Executing the Program .....	2-8	Closing a Channel .....	2-21
Displaying the Emulator Registers .....	2-9		
Setting the Emulator Registers .....	2-9		
Creating and Defining a Symbol .....	2-9		
Creating a Virtual Memory Map .....	2-9		
Logging Commands to the Line Printer .....	2-10		
<b>Program Debugging</b> .....	2-10		
Turning Symbolic Output On and Off .....	2-10		
Tracing All of the Program .....	2-10		
Tracing Part of the Program .....	2-11		
Tracing Program Branches .....	2-11		
Stepping Through the Program .....	2-12		
Setting a Breakpoint .....	2-12		
Clearing a Breakpoint .....	2-13		
Displaying Breakpoints .....	2-13		
Executing a Segment of Code Repeatedly .....	2-13		
<b>Debugging the Program in the Prototype</b> .....	2-13		
Selecting SVCs .....	2-13		
Selecting Prototype I/O .....	2-14		
Displaying Memory Map Assignments .....	2-14		

## ILLUSTRATIONS

Fig. No.		
2-1	Hardware configuration for 8540/8560 interface .....	2-1
2-2	Establish communication with the host computer .....	2-4
2-3	Virtual memory map assignments .....	2-10

## Section 2

# OPERATING PROCEDURES

Section 1, the Learning Guide, presented a general overview of the 8540's operating system and a simple demonstration program. This section presents some common procedures using OS/40 commands with your 8540 Integration Unit. Each procedure identifies parameters that you supply when performing the procedure.

The procedures in this section are given in the following format:

**Description:** A summary of the action(s) performed by the procedure.

**Procedure:** The information entered or displayed at the system terminal.

The following conventions are used in the procedure description:

Underlined: A character sequence entered by the user.

No underline: A character sequence displayed by OS/40.

**Bold**: An exact character sequence; if these characters are underlined, enter them exactly as shown.

Not bold: A parameter that you supply when performing the procedure.

**Parameters:** A description of the values to be supplied by you.

**Comments:** The operating limits and options for this procedure.

**Examples:** One or more demonstrations of correct entry format.

**See also:** Cross-references to related procedures.

## COMMUNICATING WITH AN 8560

### Establishing Communication with the 8560

**Description:** This procedure establishes a TERM interface with an 8560. With your 8540 and 8560 operating in TERM mode, you

have access to all resources of both systems simultaneously.

The 8540 is connected to the 8560 by an HSI cable. The system terminal is connected to the 8540 by an RS-232-C cable. Figure 2-1 shows how the various units are connected together.

**Procedure:** (Connect the cables as shown in Fig. 2-1.) (Start up the 8540.)

>config term <cr>

<cr>

**Login:**

(Log in to the 8560.)

**Comments:** Refer to your 8560 System Users Manual for more information concerning this procedure.

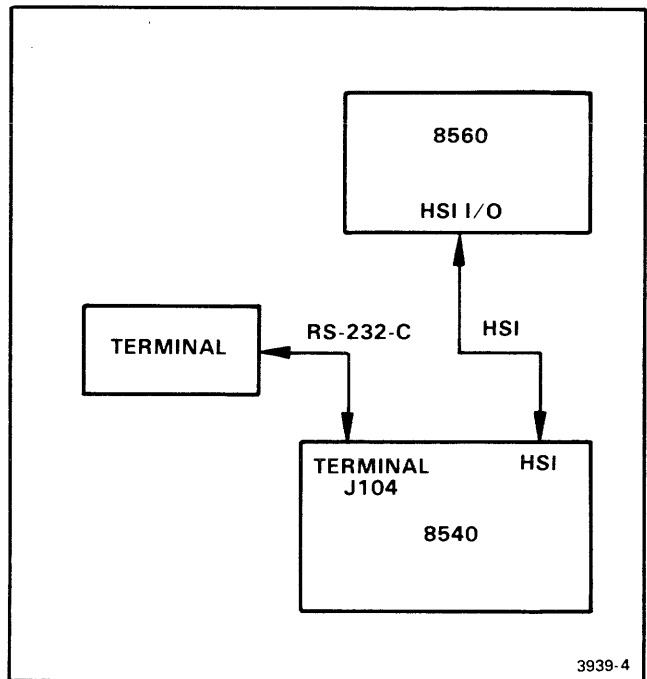


Fig. 2-1. Hardware configuration for 8540/8560 interface.

To communicate with an 8560, connect the cables as shown in this figure.

### Downloading a Program From the 8560 to 8540 Program Memory

Description: This procedure downloads a file from the 8560 to 8540 program memory.

Procedure: \$ lo <8560file

Parameters: **8560file**—The name of the 8560 file to load into 8540 program memory.

Comments: To load an 8560 file into prototype memory, first select emulation mode 2, then follow the above procedure.

#### NOTE

*The file must be in Tektronix A Series or B Series load module format, as produced by a Tektronix assembler or linker.*

See also: 

- Uploading a program from 8540 program memory to the 8560

### Uploading a Program from 8540 Program Memory to the 8560

Description: This procedure saves a program from 8540 program memory onto a file on the 8560.

Procedure: \$ sav >8560file loadd hiadd transadd

Parameters: **8560file**—The 8560 file that is to receive the program from 8540 program memory.

**loadd**—The lower limit of the data block to be saved.

**hiadd**—The upper limit of the data block to be saved.

**transadd**—The transfer address of the program.

Comments: The file is created in Tektronix A Series or B Series load module format, as produced by a Tektronix assembler or linker.

See also: 

- Downloading a program from the 8560 to 8540 program memory

### Downloading Program Symbols from the 8560

Description: This procedure loads the symbols from an 8560 load file into the symbol table in 8540 system memory.

Procedure: \$ symlo <8560file -a -s

Parameters: **8560file**—The name of the 8560 load file containing the symbol definitions you want to load.

#### NOTE

*The file must be in A Series or B Series load module format, as produced by a Tektronix linker.*

Comments: The **-a** option causes the command to append the symbols to those already in the table. If you omit the **-a**, the existing symbols are overwritten. If you omit the **-s**, scalars are not loaded.

See also: 

- Creating and defining a symbol

### Terminating Communication with an 8560

Description: This procedure terminates communication between the 8540 and the 8560.

Procedure: \$ config local; logout

Comments: This command line breaks the communication link between the 8540 and the 8560. Both the 8540 and the 8560 are returned to normal operations. The CONFIG command tells the 8540 to stop passing commands to the 8560. The LOGOUT command tells TNIX (the 8560 operating system) to stop accepting commands.

See also: 

- Establishing communication with the 8560

## COMMUNICATING WITH A HOST COMPUTER

### Establishing Communication with the Host Computer

**Description:** This procedure prepares the 8540 to operate as an "intelligent" terminal to a host computer. Six types of data transfer are available while communicating with a host computer. The two most commonly used data transfers are:

1. A **formatted download**, which transfers an object file in Tekhex format from the host computer to 8540 program memory.
2. A **formatted upload**, which transfers object code in Tekhex format from 8540 program memory to a file on the host computer.

Before establishing communication, you must first have attached your communication link (a cable or modem) to the proper jack (usually J101) on the 8540 rear panel.

#### NOTE

*The following procedures are deliberately simplified because of the differences in host computer requirements. Refer to the Intersystem Communication section of this manual, and to your system*

*engineer, for the host-computer-specific commands and COM parameters that you will use in these procedures.*

**Procedure:** (Establish data transmission between the host computer and the 8540. See Fig. 2-2.)  
> **com setup**  
(Initialize your terminal session, as required by the host computer.)

**Parameters:** **setup**—A sequence of parameters that informs the 8540 of the host computer-dependent features. Refer to the Intersystem Communication section of this manual to determine this sequence.

**Comments:** After you have determined the specific procedure for establishing communication with your host computer, record this information in the space provided in Fig. 2-2.

To terminate communication, enter the null character (ASCII 00) and press the ESC key on your terminal. On the TEKTRONIX 4024/4025 and CT8500 terminals, the null character is generated by entering CTRL-@ (pressing the @ key while holding down the CTRL key).

Other terminals may generate the null character differently. Refer to your terminal users manual for specific information.

<p>Prepare the 8540:</p>	<p>(Record here the steps to be taken when preparing your 8540 for communication with the host computer. See the Intersystem Communication section of this manual for more information.)</p>
<p>Enter the COM command:</p>	<p>(Record here the appropriate COM command parameters, as designated in the Intersystem Communication section of this manual.)</p>
<p>Host computer commands:</p>	<p>(Record here the host computer commands to initialize your terminal session.)</p>

3939-8

Fig. 2-2. Establishing communication with the host computer.

After you have determined the specific procedure for establishing communication with your host computer, record this information in the space provided in Fig. 2-2, for ease in future reference. Refer to the Intersystem Communication section of this manual for further information.

### Downloading a Program From the Host Computer to 8540 Program Memory

**Description:** This procedure transfers a load file in Tekhex format from the host computer to 8540 program memory (a formatted download).

A program must be available on the host computer to send Tekhex blocks to the 8540 and interpret the 8540's responses. Refer to the Intersystem Communication section of this manual for the requirements of this program.

Procedure: (Establish communication with the host computer.)  
command(null)

(This command transfers a Tekhex formatted file from the host computer to the 8540. The object code is loaded into program memory at the locations specified in the Tekhex data blocks. When transfer is complete, **DNLOAD:** is displayed and **OS/40** is ready to accept another upload or download command.)

Parameters: **command**—A host computer command which invokes a program that downloads a Tekhex load module to the 8540.

(**null**)—A character generated on most terminals by entering CTRL-@.

See also: 

- Establishing communication with the host computer

### Uploading a Program from 8540 Program Memory to the Host Computer

Description: This procedure transfers the contents of 8540 program memory to the host computer in Standard or Extended Tekhex format (a formatted upload). You can use the **F** parameter of the **COM** command to distinguish between Standard Tekhex and Extended Tekhex. For more information about the **COM** command, refer to the Intersystem Communication section of this manual.

A program must be available on the host computer to receive Tekhex blocks from the 8540. Refer to the Intersystem Communication section for the requirements of this program.

Procedure: (Establish communication with the host computer.)

(Load into 8540 program memory the program that you want to upload.)

command(null)loadd hiadd transadd

(When the transfer is completed, **UPLOAD:** is displayed on the terminal. At this point, **OS/40** is ready to accept another upload or download command.)

Parameters: **command**—A host computer command which invokes a program that uploads a

Tekhex load module from 8540 program memory.

(**null**)—A character generated on most terminals by entering CTRL-@.

**loadd**—The lower boundary of the block of memory to be transferred.

**hiadd**—The upper boundary of the block of memory to be transferred.

**transadd**—The transfer address.

See also: 

- Establishing communication with the host computer

### Terminating Communication with the Host

Description: This procedure terminates the communication link between the 8540 and the host computer, returning the system to normal operation.

Procedure: (Log out from the host computer.)  
(Enter a null character and press the ESC key.)  
(null)(ESC)

Comments: Enter the null character by entering CTRL-@ (typing @ while holding down the CTRL key).

See also: 

- Establishing communication with the host computer

## COMMUNICATING WITH AN 8550

### Establishing Communication with an 8550

Description: This procedure prepares the 8540 for communication with an 8550 used as the host computer. Six types of data transfer are available while communicating with the 8550; the two most common are:

1. A **formatted download**, which transfers a file in Tekhex format from the 8550 to 8540 program memory.
2. A **formatted upload**, which transfers the contents of 8540 program memory in Tekhex format to a file on the 8550.



Procedure: (Connect the RS-232-C cable, as described in the "Comments" part of this procedure.)

(Enter the following command on the 8540 terminal:)

```
> com p=7E7E
COM Vx.x
```

(Enter the following command on the 8550:)

```
> host
HOST Vx.x
```

(The 8550 is ready to be used as the host computer. Subsequent displays and commands are at the 8540 system terminal. DOS/50 prompts you for a command which informs the 8550 of the type of transfer.)

Enter U or D; A, B, or H; and 8550 filespec:

Comments: The 8550 must be connected to the 8540 with an RS-232-C cable. The female end of the cable connects to the J101 jack on the 8540 mainframe. The male end connects to the DCE jack (J102) on the 8301. The MODE SELECT switches on both systems should be in the CNTL (L) position. To terminate communication enter (null)(esc) on the 8540. Then enter (CTRL-C) followed by the command a -a on the 8550.

**NOTE**

*When the HOST command successfully completes a data transfer, the error code FF (end-of-file) is returned to the 8550 system terminal. This does not signify an error condition; it is merely an informational message.*

*Certain limitations exist in the data transfer rate. See the Intersystem Communication section of this manual for information concerning the baud rate.*

*The HOST command waits about 30 seconds after each file transfer before issuing the next prompt.*

- See also:
- Downloading a program from the 8550 to 8540 program memory
  - Uploading a program from 8540 program memory to the 8550

**Downloading a Program from the 8550 to 8540 Program Memory**

Description: This procedure transfers an executable object file in Tekhex format from the

8550 to 8540 program memory (a formatted download). All commands are entered on the 8540 system terminal. For more information about Tekhex format, refer to the Tables section of this manual.

Procedure: (Establish communication between the 8550 and the 8540.)

Enter U or D; A, B, or H; and 8550 filespec:

DHfilespec

Enter transfer sequence, beginning with null character:

(null)

Parameters: **filespec**—The 8550 file containing the object program that you want to download to 8540 program memory.

(null)—A character generated on most terminals by entering CTRL-@.

Comments: The memory location and transfer address are specified by the contents of the object file.

Example: (Establish communication between the 8550 and the 8540.)

Enter U or D; A, B, or H; and 8550 filespec:

DH/VOL/MYVOL/PROG1/LOAD

Enter transfer sequence, beginning with null character:

(null)

\*0000 1E (First block sent to addresses 0000-001D.)

\*001E 1E (Second block sent to addresses 001E-003B.)

⋮

\*0200 0000 (Termination block—transfer address-0200.)

DNLOAD:

The Tekhex formatted file, /VOL/MYVOL/PROG1/LOAD is downloaded to 8540 program memory. The memory locations and transfer address are specified by the contents of the file.

- See also:
- Establishing communication with an 8550
  - Uploading a program from 8540 program memory to the 8550

## Uploading a Program from 8540 Program Memory to the 8550

**Description:** This procedure transfers the contents of program memory in the 8540 to an 8550 Tekhex formatted file (a formatted upload). All commands are entered on the 8540 system terminal. For more information about Tekhex format, refer to the Tables section of this manual.

**Procedure:** (Establish communication between the 8550 and the 8540.)  
Enter U or D; A, B, or H; and 8550 filespec:  
UHfilespec  
Enter transfer sequence, beginning with null character:  
(null)loadd hiadd transadd

**Parameters:** **filespec**—The 8550 file that is to receive the data from the 8540.

**(null)**—A character generated on most terminals by entering CTRL-@.

**loadd**—The lower boundary of the block of memory to be transferred.

**hiadd**—The upper boundary of the block of memory to be transferred.

**transadd**—The transfer address.

**Example:** (Establish communication between the 8540 and the 8550.)

Enter U or D; A, B, or H; and 8550 filespec:

UH/VOL/MYVOL/PROG1/LOAD

Enter transfer sequence, beginning with null character:

(null)OF000 OF3FF OF200

BLOCK SENT AT: F000

BLOCK SENT AT: F01E

.

.

BLOCK SENT AT: F3FC

BLOCK SENT AT: F200

(Termination block.)

UPLOAD:

In this example, bytes F000–F3FF of program memory are uploaded to the 8550 file /VOL/MYVOL/PROG1/LOAD in Tekhex format. F200 is the transfer address.

See also:

- Establishing communication with an 8550
- Downloading a program from the 8550 to 8540 program memory

## Terminating Communication With an 8550

**Description:** This procedure terminates the communication link between the 8540 and the 8550.

**Procedure:** (Enter (null)(ESC) on the 8540 terminal by typing CTRL-@ and pressing the ESC key. Then enter the following commands on the 8550 terminal.)

(CTRL-C)  
>>a -a

**Comments:** These commands terminate the communication programs that are running and return the systems to normal operation.

See also:

- Establishing communication with an 8550

## DOWNLOADING FROM AN 8002A

### Downloading from an 8002A

**Description:** This procedure transfers an executable object file in Tekhex format from the 8002A to 8540 program memory.

**Procedure:** (Establish communication between the 8540 and the 8002A.)

(Connect the RS-232-C cable, as described in the "Comments" part of this procedure.)

(Enter the following command on the 8540 system terminal:)

> COM P=3F E=L L=I

(Enter the following command on the 8002A system terminal:)

> SEND

(Enter the following command on the 8540 system terminal:)

filename(null)

(When the download is complete, the 8540 terminal displays the message DNLOAD:

(To terminate communication, enter (null) (ESC) on the 8540 terminal by typing CTRL-@ and pressing the ESC key.)

Parameters: **filename**—The name of the Tekhex file on the 8002A.

**(null)**—A character generated on most terminals by entering CTRL-@.

Comments: The 8540 is connected to the 8002A via an RS-232-C cable. The cable connects to jack J102 (DTE) on the 8540 rear panel and jack J101 on the 8002A rear panel. The MODE SELECT switch on the 8540 should be set to CNTL (L), and the REMOTE BAUD switch to 2400. On the 8002A System Communication board, verify that the J1 jumper is in the lower position, and that the PORT1 baud rate is set to 2400.

Each system must have its own system terminal.

The SEND command does not issue a prompt.

## PROGRAM EXECUTION

### Selecting the Target Processor

Description: This procedure selects the proper emulator control software for the microprocessor you are programming.

Procedure: > sel microprocessor

Parameters: **microprocessor**—The OS/40 name of the target processor. See the Emulator Specifics section of this manual for the OS/40 name of your microprocessor.

Comments: OS/40 automatically performs several commands internally to initialize the emulator when you select the target processor. Refer to the discussion of the SEL command in the Command Dictionary section of this manual for further information.

Example: > se1 8085

This command line selects the 8085A emulator on the 8540. (The command also selects 8080A/8085A assembler and compiler software on the 8560, if you are in TERM mode.)

### Selecting the Emulation Mode

Description: This procedure selects the emulation mode in which the the system operates.

Procedure: > em mode

Parameters: **mode**—0, 1, or 2.

Mode 0 is system mode. Execution in this mode uses program memory, 8540 system I/O (through SVCs), and the emulator clock. The system is placed in emulation mode 0 whenever the target processor is selected.

Mode 1 is partial emulation mode. Execution in this mode uses program memory and/or prototype memory (depending on the memory map assignments), SVCs and/or prototype I/O, and the prototype's clock.

Mode 2 is full emulation mode. Execution in this mode uses the prototype's memory, I/O, and clock, and may also use SVCs.

Comments: The emulation mode is set to mode 0 when you enter the SEL command. When executing programs in emulation modes 1 and 2, your prototype must be connected to the system via the prototype control probe.

Example: > em 1

This example selects emulation mode 1 (partial emulation mode).

See also: ● Selecting the target processor

### Executing the Program

Description: This procedure begins program execution at the specified address. The program must already reside in memory.

Procedure: > g addr

Parameters: **addr**—The address of the next instruction to be executed. If this parameter is omitted, execution continues at the address in the emulator's program counter.

Example: > g 300  
 This example starts program execution at address 300.

- See also:
- Downloading a program from the 8560
  - Downloading a program from the host computer
  - Downloading a program from the 8550
  - Downloading a program from the 8002A

### Displaying the Emulator Registers

Description: This procedure displays the contents of the emulator registers.

Procedure: > ds

Comments: For more information on the DS command, see the Emulator Specifics supplement for your microprocessor.

Example: > sel 8085  
 > ds

```
PC=0000 SP=0000 F=00 A=00 B=00 C=00 D=00 E=00 H=00 L=00
SOD=0 SID=0 I7=0 I6=0 I5=0 IE=0 M7=1 M6=1 M5=1
```

- See also:
- Setting the emulator registers

### Setting the Emulator Registers

Description: This procedure changes values in individual emulator registers.

Procedure: > s r1=value1 r2=value2...

Parameters: **r1**—The name of an emulator register you want to set.

**value1**—The value you want to store in the register.

**r2**—The name of another emulator register you want to set.

**value2**—The value you want to store in the register.

Example: > s A=0E1 B=00

This example sets the emulator registers A and B to E1 and 00, respectively.

- See also:
- Displaying the emulator registers

### Creating and Defining a Symbol

Description: This procedure adds a symbol to the symbol table and assigns it a value.

Procedure: > adds symbol=value

Parameters: **symbol**—The name of the symbol that you are creating.

**value**—The value of the new symbol.

Comments: You can enter more than one symbol definition on the same line. If a symbol represents a scalar (not an address), precede it with a -S.

Example: > adds start=100 -s tablesize=50t

This example defines two symbols: **start** and **tablesize** with the values 100 (hexadecimal) and 50 (decimal) respectively.

- See also:
- Downloading symbols from the 8560

### Creating a Virtual Memory Map

Description: This procedure allows your program to access a block of memory locations at virtual addresses rather than at their actual physical addresses in program memory. This procedure is useful if your program accesses addresses beyond the limits of physical program memory.

#### NOTE

*If you are using the Memory Allocation Controller (MAC) option or the 8086/8088 emulator, the AL command has a different syntax and use, so do not use this procedure. Refer to the Emulation section of this manual for an explanation of the MAC option.*

Procedure: > al actual blocks virtual

Parameters: **actual**—The physical address of the first memory location that you want to allocate to a different area.

**blocks**—The number of 4K-byte blocks to be reallocated.

**virtual**—The address that your program uses to refer to the beginning of the reallocated memory area. The addresses **actual** and **virtual** should be multiples of 1000H.

Example: > a1 2000 3 0A000

This example makes the three 4K-byte blocks of memory starting at location 2000 appear to be addressable starting at location A000. See Fig. 2-3.

### Logging Commands to the Line Printer

Description: This procedure sends a copy of your terminal input/output to the line printer.

Procedure: > log LPT

Comments: This command line sends all terminal input/output to both the terminal and the line printer.

Example: > log LPT  
 .  
 (OS/40 commands)  
 .  
 > log CONO  
 (Log CONO turns off the display to the line printer.)

## PROGRAM DEBUGGING

### Turning Symbolic Output On and Off

Description: This procedure turns symbolic display on or off.

Procedure: > synd on  
 or  
 > synd off

Comments: This command only affects the way in which commands like TRA and DI display their output. If SYMD is on, symbols are substituted for address. You can still use symbols in expressions and use commands like SYMB and ADDS even if SYMD is off. For information regarding symbolic debug, refer to the Emulation section of this manual.

See also: ● Tracing all of the program  
 ● Executing the program

### Tracing All of the Program

Description: This procedure displays each instruction as it is executed. Tracing continues until the end of the program is reached or until a break occurs.

Procedure: > tra all  
 > g sirt

Parameters: **strt**—The memory address at which program execution is to begin. If this parameter is omitted, execution begins at the transfer address of the program.

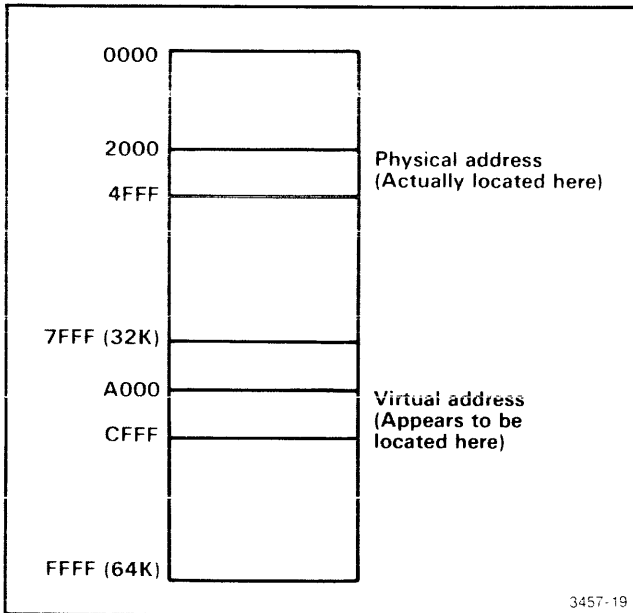


Fig. 2-3. Virtual memory map assignments.

Example: The example in Display 2-1 loads PROG1 from the 8560 and displays each instruction as it is executed.

Comments: The first command line (TRA OFF) cancels any previous trace selections. TRA OFF may be omitted if no other trace selections have been entered.

- See also:
- Tracing part of the program
  - Tracing program branches

Example: See Display 2-2.

- See also:
- Tracing all of the program
  - Tracing program branches

### Tracing Part of the Program

Description: This procedure displays the trace of a specified portion of the program. Instructions outside the specified range are not included in the display.

Procedure: > tra off  
> tra all loadd hiadd  
> g strt

Parameters: **loadd**—The lowest address to be traced.  
**hiadd**—The highest address to be traced.  
**strt**—The memory address at which program execution is to begin.

### Tracing Program Branches

Description: This procedure traces only those instructions that change the normal execution flow of the program (branches, subroutine calls, etc.).

Procedure: > tra jmp  
> g strt

Parameters: **strt**—The memory address at which program execution is to begin.

```

> lo <PROG1
> tra all
> g 100

LOC  INST  MNEM  OPER  SP    F    A  B  C  D  E  H  L  IM  SOD
000100 210005 LXI   H,0500 0000 04 0F 01 00 00 00 05 00 07 0
000103 0605  MVI   B,05 0000 04 0F 05 00 00 00 05 00 07 0
000105 AF    XRA   A      0000 44 00 05 00 00 00 05 00 07 0
000106 86    ADD   M      0000 00 01 05 00 00 00 05 00 07 0
000107 23    INX   H      0000 00 01 05 00 00 00 05 01 07 0

```

Display 2-1.

```

> tra off
> tra all 103 106
> g 100

LOC  INST  MNEM  OPER  SP    F    A  B  C  D  E  H  L  IM  SOD
000103 0605  MVI   B,05 0000 04 0F 05 00 00 00 05 00 07 0
000105 AF    XRA   A      0000 44 00 05 00 00 00 05 00 07 0
000106 86    ADD   M      0000 00 01 05 00 00 00 05 00 07 0
000106 86    ADD   M      0000 04 03 04 00 00 00 05 01 07 0
000106 86    ADD   M      0000 04 0F 01 00 00 00 05 04 07 0

```

Display 2-2.

Example: The example in Display 2-3 loads PROG1 from the 8560 and executes it while tracing all branch instructions.

See also: 

- Tracing all of the program
- Tracing part of the program

See also: 

- Tracing all of the program
- Tracing part of the program

### Stepping Through the Program

Description: This procedure stops program execution after each trace line is displayed. To execute the next instruction, enter the G command.

Procedure:   
 > tra -s all  
> g strt

Parameters: **strt**—The memory address at which program execution is to begin.

Example: The example in Display 2-4 loads and executes PROG1, stopping after each instruction.

### Setting a Breakpoint

Description: This procedure sets a program breakpoint. The breakpoint stops execution if the specified memory address is accessed for an instruction fetch or for any other memory read or write. For most emulators, up to two breakpoints may be specified at a time.

Procedure: > bk n address

Parameters: **n**—The number of the breakpoint. 1 or 2 for most emulators.

**address**—The memory address where program execution is to be interrupted.

```

> lo <PROG1
> tra jmp
> g 100

LOC  INST  MNEM  OPER  SP   F   A  B  C  D  E  H  L  IM  SOD
000109 C20601 JNZ   0106  0000  10  01  04  00  00  00  05  01  07  0
000109 C20601 JNZ   0106  0000  14  03  03  00  00  00  05  02  07  0
000109 C20601 JNZ   0106  0000  10  06  02  00  00  00  05  03  07  0
000109 C20601 JNZ   0106  0000  10  0A  01  00  00  00  05  04  07  0
.
.
.
    
```

Display 2-3.

```

> lo <PROG1
> tra -s all
> g 100

LOC  INST  MNEM  OPER  SP   F   A  B  C  D  E  H  L  IM  SOD
000100 210005 LXI   H,0500  0000  04  0F  01  00  00  00  05  00  07  0
000100 ^BREAK TRACE^

> g

LOC  INST  MNEM  OPER  SP   F   A  B  C  D  E  H  L  IM  SOD
000103 0605  MVI   B,05  0000  04  0F  05  00  00  00  05  00  07  0
000103 ^BREAK TRACE^
.
.
.
    
```

Display 2-4.

Example: The example in Display 2-5 loads and executes PROG1. Execution is interrupted when the program accesses address 504.

See also: 

- Clearing a breakpoint
- Displaying breakpoints

### Clearing a Breakpoint

Description: This procedure cancels a breakpoint set by a previous BK command.

Procedure: > bk n clr

Parameters: **n**—The number of the breakpoint that you want to cancel. For most emulators, valid entries for this parameter are 1, 2, or all.

Examples: > bk 1 clr

This example cancels breakpoint 1.

```
> bk all clr
```

This command clears all breakpoints.

See also: 

- Setting a breakpoint

### Displaying Breakpoints

Description: This procedure displays all breakpoints.

Procedure: > bk

Example:

```
> bk 1 58 wt
> bk 2 47F
> bk
BK 1 000058 WT (Breaks when your program
writes at address 0058.)
BK 2 00047F RD WT (Breaks when your program
reads or writes at 047F.)
```

See also: 

- Setting a breakpoint
- Clearing a breakpoint

### Executing a Segment of Code Repeatedly

Description: This procedure executes a segment of code repeatedly until you press CTRL-C.

Procedure: > bk 1 addr  
> g -r start

Parameters: **addr**—The address of the end of the segment of code that you want to repeat.

**start**—The start address of the segment of code that you want to repeat.

Comments: The segment of code is repeatedly executed until you enter a CTRL-C. Refer to the Command Dictionary of this manual for more information about the **g -r** command.

Example: > bk 1 8F10  
> g -r 100

This example executes the code between addresses 100 and 8F10 repeatedly until you enter CTRL-C.

See also: 

- Executing the program
- Tracing part of the program

## DEBUGGING THE PROGRAM IN THE PROTOTYPE

### Selecting SVCs

Description: This procedure enables the system service call (SVC) features.

Procedure: > svc on

```
> bk 1 504
> g
LOC INST MNEM OPER SP F A B C D E H L IM SOD
000106 86 ADD M 0000 04 0F 01 00 00 00 05 04 07 0
000106 <BREAK BK1>
```

ADD M adds the bytes at address 504 into the accumulator.

Display 2-5.



Comments: When this feature is ON, OS/40 executes SVCs in all emulation modes. If an I/O operation is found that can be translated into an SVC, it is treated as an SVC. Changing to emulation mode 1 or 2 disables the SVC feature, but it can be enabled by performing this procedure.

Example: > em 1  
> svc on

These command lines select emulation mode 1 and then enables the use of SVCs.

See also: 

- Selecting prototype I/O
- Selecting the emulation mode

### Selecting Prototype I/O

Description: This procedure turns the SVC feature OFF. When SVCs are OFF, all SVC invocations are treated as normal I/O instructions. If your program uses SVCs, do not execute it while the SVC feature is OFF unless the extraneous I/O instructions are taken into consideration.

#### NOTE

*Prototype I/O is available only in modes 1 and 2.*

Procedure: > svc off

See also: 

- Selecting the emulation mode

### Displaying Memory Map Assignments

Description: This procedure displays the current memory map assignments.

Procedure: > map

Comments: Each line of the display consists of a memory address range and one of the following parameters:

- PRW — Program memory, read and write access.
- PRO — Program memory, read-only access.

- URW — Prototype memory, read and write access.
- URO — Prototype memory, read-only access.

The program/prototype designation applies only in mode 1; the read-only designation applies only to program memory.

#### NOTE

*Refer to the Emulator Specifics section of this manual for specific information about the MAP command for your microprocessor.*

Example: > map  
  
0000-007F URW  
0080-00FF PRO  
0100-017F URW  
0180-01FF PRW  
  
F980-F9FF URW

See also: 

- Creating a virtual memory map
- Displaying the contents of memory

## USING THE TRIGGER TRACE ANALYZER

#### NOTE

*To perform the procedures in this subsection, you must have the Trigger Trace Analyzer (TTA) option installed in your 8540 Integration Unit. TTA commands are available only after you have used the SEL command to select the emulator software.*

### Breaking on a Designated Instruction

Description: This procedure uses the TTA to produce a breakpoint. The program continues to run until the designated instruction is executed. At that time, a break occurs and OS/40 returns control of the system to you.

Procedure: > eve -s 1 clr a=addr b=f  
  
> g strt

Parameters: **strt**—The starting point for program execution.

**addr**—The address of the designated instruction.

Comments: The EVE command line establishes the conditions of event 1. The CLR parameter erases any previous conditions designated for event 1. The parameters a=addr and b=f set event 1 to trigger on any instruction fetch from **addr**. The -s parameter sets a breakpoint at event 1.

Example: 

```
> eve -s 1 clr a=30E8 b=f
```

In this example, a break occurs when an instruction fetch from 30E8 is performed.

See also: 

- Setting a breakpoint
- Breaking on a specified pass through a loop

### Breaking on a Specified Pass Through a Loop

Description: This procedure uses the TTA to stop execution of a program when the designated instruction is executed the specified number of times.

Procedure: 

```
> eve 2 a=addr b=f
> cou -s 2 clr s=ev2 o=delay v=pass
> gstrt
```

Parameters: **addr**—The address of the designated instruction.

**pass**—The number of times to execute the instruction before stopping: can be any expression that evaluates to a number in that range.

Comments: The s=ev2 parameter counts the number of times that event 2 occurs. The o=delay parameter counts down from the value of **pass** and enables a breakpoint when it reaches 0. The v=pass parameter starts the counter at **pass**.

If the instruction at address **addr** is executed fewer than **pass** times, no break occurs. In that case, the program continues either until termination, or until you type CTRL-C to regain control of

the system. Refer to the Trigger Trace Analyzer Users Manual for more information.

Example: 

```
> eve 2 a=2F04 b=f
> cou -s 2 clr s=ev2 o=delay v=3
> g
```

In this example, the instruction at 2F04 is executed 3 times before a breakpoint occurs.

See also: 

- Breaking on a designated instruction

### Breaking When Execution Proceeds Outside a Designated Range

Description: This procedure uses the TTA to stop execution of a program when the instruction to be executed lies outside the designated address range.

Procedure: 

```
> eve 1 -s clr an=loadd hiadd b=f
> gstrt
```

Parameters: **strt**—The starting point for program execution.

**loadd**—The lowest address that can be executed without causing a break.

**hiadd**—The highest address that can be executed without causing a break.

Comments: The **loadd** address must be less than the **hiadd** address. Also, for meaningful results, the starting address of the program must lie between **loadd** and **hiadd**.

Example: 

```
> eve 1 -s clr an=3100 4100 b=f
```

In this example, program execution halts when the program proceeds outside of the range 3100—4100.

See also: 

- Breaking when the program writes in a designated memory area

### Breaking When the Program Writes in a Designated Memory Area

Description: This procedure uses the TTA to stop execution when an attempt is made to alter the contents of a designated memory area.

Procedure: 

```
> eve -s 1 clr a=loadd hiadd b=m wt
> gstrt
```

Parameters: **strt**—The starting point for program execution.

**loadd**—The lowest address of the designated memory area.

**hiadd**—The highest address of the designated memory area.

Comments: The EVE command line restricts the break to only those situations where a memory write (M WT) is attempted into a memory location inside the designated range. The value of **loadd** must not exceed that of **hiadd**.

You can also use the MAP command to designate an area of program memory as read-only. The first write to that area causes a break to occur. Refer to the Command Dictionary for information on the MAP command.

Example: 

```
> eve -s 1 clr a=0C800 0E800 b=m wt
```

This example halts program execution when the program writes to memory in the range C800—E800.

See also: 

- Breaking when execution proceeds outside a designated range

### Saving an Execution Trace Record Without Interrupting the Program

Description: This procedure uses the TTA to record and display 255 instruction cycles executed before a designated instruction. The program does not terminate when the designated instruction is executed, but proceeds until reaching the breakpoint address.

Procedure: 

```
> eve 4 clr a=addr b=f
> acq all for 1 cyc aftertrig4
> eve -s 1 clr a=quit b=f
> g strt
(Wait for program execution to terminate.)
> disp
```

Parameters: **strt**—The starting point for program execution.

**addr**—The address of the instruction where you want to stop recording.

**quit**—The address at which a break occurs. If you want execution to continue until the program is completed, omit the EVE -S command line, as in the following procedure.

Comments: The EVE command lines describe the conditions that generate events 1 and 4. The DISP command line displays the contents of the Acquisition Memory on the system terminal.

Example: 

```
> eve 4 clr a=4F64 b=f
> acq all for 1 cyc aftertrig4
> eve -s 1 clr a=0FFFF b=f
> g 4000
.
.
.
> disp
```

This example saves a record of the 255 instruction cycles executed before the instruction at 4F64 is executed.

See also: 

- Recording instructions executed before and after a designated instruction
- Recording instructions executed after a designated instruction

### Recording Instructions Executed Before and After a Designated Instruction

Description: This procedure uses the TTA to record and display 127 instruction cycles executed before and 127 instruction cycles after the execution of a designated instruction.

Procedure: 

```
> eve 4 clr a=addr b=f
> acq all for 127 cyc aftertrig4
> g strt
```

Parameters: **strt**—The starting point for program execution.

**addr**—The address of the designated instruction.

Example: 

```
> eve 4 clr a=3A24 b=f
> acq all for 127 cyc aftertrig4
```

This example saves a trace record of 127 instruction cycles before and 127 instruction cycles after the instruction at address 3A24.

- See also:
- Saving an execution trace record without interrupting the program
  - Recording instructions executed after a designated instruction

### Recording Instructions Executed After a Designated Instruction

**Description:** This procedure uses the TTA to record and display up to 255 instruction cycles that were executed after the designated instruction.

**Procedure:**

```
> eve 4 clr a=addr b=f
> acq all for 255 cyc aftertrig4
> g strt
```

**Parameters:** **strt**—The starting point for program execution.

**addr**—The address of the designated instruction.

**Example:**

```
> eve 4 clr a=0B024 b=f
> acq all for 255 cyc aftertrig4
```

This example saves a trace record of 255 instruction cycles after the instruction executed from address B024.

- See also:
- Saving an execution trace record without interrupting the program
  - Recording instructions executed before and after an instruction

### Determining the Execution Time of a Program Segment

**Description:** This procedure uses the TTA to calculate the time elapsed between the execution of two designated instructions. Program execution terminates when the second instruction is executed. Time can be calculated in any time units allowed by the COU command.

**Procedure:**

```
> eve 1 clr a=addr1 b=f
> cou 2 clr s=units o=arm g=seqh v=0
> eve -s 2 clr a=addr2 b=f
> g
(Wait for program execution to terminate.)
> ts -c 2
```

**Parameters:** **units**—Any valid COU unit. Refer to the Emulation section of this manual for information about the COU command.

**addr1**—The address at which you want to start timing.

**addr2**—The address at which you want to stop timing.

**Comments:** The CLR parameter of the EVE commands clears any previously set EVE conditions on triggers 1 and 2. The CLR parameter of the COU command clears any previously set conditions on counter 2. The s=units parameter sets the time units to count. The o=arm parameter causes the counter to count in ascending order. The v=0 parameter starts the counter at 0. The g=seqh parameter tells the counter to start counting when trigger 1 occurs. The EVE 2 command line stops the counter when the instruction at the specified address is fetched. The TS command displays the value in the counter.

**Example:**

```
> eve 1 clr a=3320 b=f
> cou 2 clr s=cyc o=arm g=seqh v=0
> eve -s 2 clr a=5F40 b=f
> g
.
.
.
> ts -c 2
```

This example displays the number of bus cycles that occurred during the execution of the program segment in the address range 3320—5F40.

- See also:
- Measuring the interval between probe events
  - Counting the occurrences of an event

### Counting the Occurrences of an Event

**Description:** This procedure counts the number of times that a specified event occurs.

**Procedure:**

```
> eve 1 clr event
> cou 2 clr s=ev1 o=arm v=0
> g
(Wait for program execution to terminate.)
> ts -c 2
```

**Parameters:** **event**—The sequence of parameters that define the event.

Comments: The parameter `s=ev1` increments the counter when event 1 occurs. The `o=arm` parameter tells the counter to count in ascending order. The `v=0` parameter starts the counter at 0. After the program executes, the TS command line displays the value in counter 2.

Example:

```
> eve 1 clr a=1030 b=f
> cou 2 clr s=ev1 o=arm v=0
> g
.
.
.
> ts -c 2
```

This procedure counts the number of times the program fetches an instruction from address 1030.

See also:

- Breaking on a specified pass through a loop
- Measuring the interval between probe events

## Measuring the Interval Between Probe Events

Description: This procedure uses the TTA test clips to measure the time interval between two external events. The program terminates when the conditions of the second probe event are satisfied.

Procedure: **(Attach the TTA test clips to the desired signal lines in your prototype.)**

```
> eve 1 clr p=begin
> cou 2 clr s=units o=arm g=seqh v=0
> eve -s 2 p=end
> g strt
```

Parameters: **strt**—The starting point for program execution.

**units**—Any valid COU unit. Refer to the Emulation section of this manual for information about the COU command.

**begin**—Eight characters representing the states of the eight test clips when the measurement is to begin. Each character can be 0 (logic low), 1 (logic high), or X (don't care). If the first character is "X", put a zero in front of it.

**end**—Eight characters (0, 1, or X) representing the states of the eight test clips when measurement (and program execution) is to end.

Comments: This procedure is similar to that used for determining the execution time of a program segment. Here, however, the conditions for starting and stopping the counter are defined in terms of the test clip values instead of the designated instruction fetches. The clock and ground lines of the TTA test probe must be attached for the TTA to function properly.

Example: **(Attach the TTA test clips to the desired signal lines in your prototype.)**

```
> eve 1 clr p=0xxxxxxx1
> cou 2 clr s=cyc o=arm g=seqh v=o
> eve -s 2 p=1111xxxx
```

This example measures the number of bus cycles between the time probe clip 0 goes high and the time clips 4-7 become high simultaneously.

See also:

- Breaking on a probe event
- Determining the execution time of a program segment

## Breaking on a Probe Event

Description: This procedure uses the TTA to stop program execution when the conditions of the probe event are satisfied.

Procedure: **(Attach the probe test clips to the desired signal lines in your prototype.)**

```
> eve -s 1 p=begin
> g strt
```

Parameters: **begin**—Eight characters representing the states of the eight test clips when the measurement is to begin. Each character can be 0 (logic low), 1 (logic high), or X (don't care).

**strt**—The starting point for program execution.

Comments: The EVE command line defines event 1 in terms of a probe test clip value. The clock and ground lines of the test clips must be attached for the TTA to function properly.

Example: 

```
> eve -s 1 p=1xxxxx01
```

This example halts program execution when the 0 and 7 clips go high and the 1 clip goes low.

- See also:
- Measuring the interval between probe events

## MEMORY MANIPULATION

### Displaying the Contents of Memory

**Description:** This procedure displays the contents of a selected area of memory.

**Procedure:** > d loadd hiadd

**Parameters:** **loadd**—The lower limit of the memory area to be displayed.

**hiadd**—The upper limit of the memory area to be displayed.

**Example:** In Display 2-6, the system terminal displays 16-byte data blocks from program or prototype memory. The display starts with memory address 0000 and continues to 002F.

- See also:
- Changing the contents of a section of memory
  - Initializing a block of memory

### Changing the Contents of a Section of Memory

**Description:** This procedure modifies consecutive memory locations.

**Procedure:** > p addr hexstring

**Parameters:** **addr**—The starting address of the data in program or prototype memory that you want to change.

**hexstring**—A sequence of hexadecimal digits. The hexstring directly replaces the values in memory starting at the designated address on a byte-by-byte basis. Each pair of digits represents one byte; thus the string must contain an even number of digits. For word-oriented microprocessors, the number of digits in the string must be divisible by 4.

**Example:** In Display 2-7, the contents of memory locations 400 through 40C are replaced with the specified hexadecimal string.

- See also:
- Displaying the contents of memory
  - Initializing a block of memory

```
> d 0 2F
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 21 .....
000010 00 05 06 05 AF 86 23 05 C2 06 01 D3 F7 00 00 00 .....#.....
000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

Display 2-6.

```
> d 400
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000400 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

> p 400 00050605AF862305C20601D3F7

> d 400
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000400 00 05 06 05 AF 86 23 05 C2 06 01 D3 F7 00 00 00 .....

```

Display 2-7.

## Initializing a Block of Memory

**Description:** This procedure initializes memory locations with a specified value.

**Procedure:** > f loadd hiadd hexstring

**Parameters:** **loadd**—The lowest memory address to be initialized.

**hiadd**—The highest memory address to be initialized. Must be greater than or equal to that of **loadd**.

**hexstring**—The value that is to fill the designated memory area. If the target processor is word-oriented, this string must contain a multiple of 4 hexadecimal digits. Otherwise, an even number of digits are required.

**Example:** > f 0 OFF 0000

This command line fills memory locations 0000-00FF with the value 0.

**See also:**

- Displaying the contents of memory
- Changing the contents of a section of memory

## Breaking When the Program Writes in a Designated Memory Block

**Description:** This procedure causes program execution to halt when the program attempts to write to a specified block of program memory.

**Procedure:** > map pro loadd hiadd

**Parameters:** **loadd**—The starting address of the memory range.

**hiadd** The ending address of the memory range.

**Comments:** The value of **loadd** should be less than that of **hiadd**. More than one range of addresses can be specified on the MAP command line.

**Example:**

```
> lo <prog1
> map pro 3A00 4FFF
> g 100
```

If during execution, prog1 attempts to write into a memory location within the range 3A00-4FFF, a break is generated.

**See also:**

- Breaking when the program writes in a designated memory area

## Disassembling the Contents of Memory

**Description:** This procedure translates the object code in the specified address range to its corresponding mnemonics and operands, and displays this information.

**Procedure:** > di loadd hiadd

**Parameters:** **loadd**—The memory address where disassembly is to begin.

**hiadd**—The memory address where disassembly is to stop.

**Example:** > di 100 10E

```
LOC  INST  MNEM  OPER
000100 210005 LXI   H,0500
000103 0605   MVI   B,05
000105 AF    XRA   A
000106 86    ADD   M
000107 23    INX   H
000108 05    DCR   B
000109 C20601 JNZ   0106
00010C D3F7   OUT   F7
00010E 00    NOP
```

The contents of memory locations 0100-010E are translated into mnemonics and operands and are displayed on the system terminal.

**See also:**

- Displaying the contents of memory
- Changing the contents of a section of memory

## SYSTEM I/O

### Displaying the Current Channel Assignments

**Description:** This procedure displays the current I/O channel assignments.

**Procedure:** > stat

**Comments:** This command displays the following information:

- The OS/40 name of the currently selected target processor.
- The present assignments of channels 0-7.

Example:     > stat  
               8085            is the selected processor  
               Channel 6 is assigned to LPT

See also:     • Assigning a channel  
               • Closing a channel

### Assigning a Channel

Description:   This procedure associates a physical device or file with an I/O channel number.

Procedure:     > as n filespec

Parameters:   **n**—A logical I/O channel number. This number must be between 0 and 7, inclusive.

**filespec**—The device or 8560 file to be assigned to the channel number.

Comments:     I/O channels allow your programs to read or write to files or devices via service calls (SVCs).

Example:     > as 3 table1 5 LPT

This example assigns the 8560 file table1 to channel 3 and the line printer to channel 5.

See also:     • Displaying the channel assignments  
               • Closing a channel

### Closing a Channel

Description:   This procedure disconnects a file or device from an I/O channel previously opened with the AS command.

Procedure:     > cl n

Parameters:   **n**—The number of the I/O channel number that you want to close. This number must be between 0 and 7, inclusive.

Example:     > cl 4

This command line disconnects channel 4 from the file or device to which it was assigned.

See also:     • Assigning a channel  
               • Displaying the channel assignments



## Section 3 COMMAND DICTIONARY

	Page
<b>Command Index</b> .....	3-1
<b>Introduction</b> .....	3-2
<b>Other System Commands</b> .....	3-2
<b>Command Syntax</b> .....	3-2
Notation Conventions .....	3-2
Command Line .....	3-2
Strings .....	3-4
Legal Address Expressions .....	3-5
<b>Special Keys</b> .....	3-6
<b>Command Memory Areas</b> .....	3-7
<b>Commands</b> .....	3-8

### TABLES

Table No.		
3-1	System Devices .....	3-12
3-2	Types of Data Transfer .....	3-21
3-3	EX Command Options .....	3-29

### ILLUSTRATIONS

Fig. No.		
3-1	Sample syntax block .....	3-2
3-2	Syntax for string definition .....	3-4
3-3	AL command example .....	3-11

## Section 3

# COMMAND DICTIONARY

### COMMAND INDEX

	Page		Page
<b>Memory Management Commands</b>			
AL—Allocates memory to logical address map . . . . .	3-10	BK—Sets or displays breakpoint condition . . . . .	3-13
COM—Sets up communications with host		CL—Disconnects channel from device or file . . . . .	3-16
computer . . . . .	3-19	CLOCK—Controls program clock . . . . .	3-17
D—Displays memory contents . . . . .	3-24	DI—Translates object code to mnemonics . . . . .	3-26
DEAL—Deallocates memory from logical address		DS—Displays contents of emulator registers . . . . .	3-24
map . . . . .	3-25	EM—Selects emulation mode . . . . .	3-28
EX—Displays or alters memory contents . . . . .	3-29	G—Begins program execution . . . . .	3-32
F—Fills program/prototype memory with data . . . . .	3-30	RD—Reads from emulator port . . . . .	3-45
LO—Loads program into memory . . . . .	3-34	REMS—Removes symbol from symbol table . . . . .	3-46
MAP—Sets or displays memory map assignments . . . . .	3-35	RESET—Reinitializes emulator . . . . .	3-47
MEM—Specifies memory to be available to		S—Assigns value to register or symbol . . . . .	3-51
emulator . . . . .	3-38	SVC—Controls execution of service calls from	
MEMSP—Defines memory space to be used by		user program . . . . .	3-57
memory commands . . . . .	3-39	SYMB—Returns symbolic equivalent of value . . . . .	3-58
MOV—Moves data between program and		SYMD—Controls symbolic display output . . . . .	3-59
prototype memory . . . . .	3-40	SYMLO—Loads symbols into symbol table . . . . .	3-61
NOMEN—Specifies memory unavailable to		TRA—Controls display of executed instructions . . . . .	3-62
emulator . . . . .	3-41	WRT—Writes to emulator I/O port . . . . .	3-65
P—Alters memory contents . . . . .	3-42		
RH—Reads hexadecimal code into memory . . . . .	3-48	<b>Miscellaneous System Commands</b>	
SAV—Saves memory contents in file . . . . .	3-52	A—Aborts user program or command execution . . . . .	3-8
SEA—Searches memory for value or string . . . . .	3-52	CALC—Evaluates arithmetic expression . . . . .	3-15
WH—Saves memory contents in hexadecimal		CO—Resumes execution of suspended command . . . . .	3-18
format . . . . .	3-64	CONFIG—Defines system configuration and host	
X—Loads and executes program . . . . .	3-65	interface . . . . .	3-22
		LOG—Logs terminal input/output to device . . . . .	3-34
		PERMSTR—Stores user-defined string in	
		EEPROM . . . . .	3-43
		ROMPATCH—Updates operating system . . . . .	3-49
		SEL—Selects target processor chip . . . . .	3-54
		STAT—Displays system status . . . . .	3-55
		STR—Displays or deletes temporary user strings . . . . .	3-55
		SUSP—Suspends command . . . . .	3-56
<b>Debugging and Emulation Commands</b>			
ADDS—Adds symbol to symbol table . . . . .	3-9		
AS—Assigns channel to device or file . . . . .	3-12		

## INTRODUCTION

This Command Dictionary describes most OS/40 commands. The Command Index on the preceding page lists all commands described in this section according to their functions. The "Commands" subsection lists the commands alphabetically.

This section is divided into the following subsections:

- **Other System Commands.** Tells where to look for documentation of subsystem commands.
- **Command Syntax.** Describes the notation conventions used in syntax blocks; command lines; and legal address expressions.
- **Special Keys.** Describes the special function keys that the operating system recognizes.
- **Commands.** Describes commands in detail. Each command description consists of the following parts: a syntax block, parameter definitions, an explanation of the function and use of the command, and one or more examples.

## OTHER SYSTEM COMMANDS

Commands that invoke or use major subsystems are fully documented in other manuals, and are not described in this Command Dictionary. The following list shows the manuals that fully describe each subsystem command.

- PROM Programmer commands (CPR, PSTAT, PTYPE, RPR, WPR)—PROM Programmer Specifics supplement
- TTA commands (ACQ, AD, BRE, BUS, CONS, COU, CTR, DATA, DISP, EVE, PRO, QUA, TCLR, TS)—Trigger Trace Analyzer Users Manual

## COMMAND SYNTAX

Each command description includes a syntax block that illustrates the format for a command. This subsection describes the notation conventions used in the syntax blocks; command line format, strings, and legal address expressions.

### Notation Conventions

The syntax block for each command illustrates the command entry: the command name, whether or not each part of the command entry must be included, and the order in which to enter parts of the command entry. Figure 3-1 illustrates a sample syntax block.

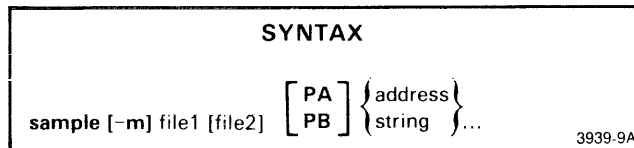


Fig. 3-1. Sample syntax block.

This figure illustrates a syntax block for a fictional operating system command: **sample** is the command name; **-m** is a command modifier; and **file1**, **file2**, **PA**, **PB**, **address**, and **string** are command parameters. The braces, brackets, and trailing dots are for syntactical representation only.

## Command Line

A command line consists of one or more commands and/or string definitions separated by semicolons (;). The maximum length of a command line is 80 characters, including spaces and the terminating carriage return.

If you enter a command line that contains more than 80 characters, a command buffer overflow error will occur. The excess characters will be stored in the type-ahead buffer and used as the beginning of the next command line. You must type CTRL-U or CTRL-C to delete the excess characters before entering another command. (The long command line will not be executed.)

Braces { } in a syntax block surround **required** parts of the command line. Brackets [ ] in a syntax block surround **optional** parts of the command line. When parts are stacked, you choose one part from the ones in the stack. Braces and brackets serve only to represent the syntax, and should not be entered as part of the command line.

**Boldface** letters and other characters in the syntax block are **required** in the command line, and should be entered exactly as they appear in the syntax block.

Three trailing dots in a syntax block show that the preceding element of the command line may be repeated as many times as needed, up to the maximum line length of 80 characters.

Underlined letters within a required parameter represent the acceptable short form of the parameter.

For most parts of a command line, you may use either uppercase or lowercase letters. The exceptions are as follows: String names must appear as they did when created. Command names must be entered in lowercase if you are in TERM mode with an 8560.

## Command Name

A command name is a word that represents an operating system command. Every command must begin with a command name.

## Delimiters

A delimiter separates parts of the command from each other. Allowed delimiters are spaces, commas, or tabs, with one comma being equivalent to one or more spaces or tabs.

## Command Modifiers

A command modifier (a special type of parameter) consists of a dash (-) followed by a letter. Figure 3-1 contains the command modifier **-m**.

Multiple command modifiers (when used) can be strung together: **-A -B -C** can be entered as **-ABC**. A command modifier, if specified, can usually occur anywhere in the command line; its position in the line is not significant except in the following instance.

When a dash modifier is used to modify a parameter instead of the command, the dash modifier must immediately precede the appropriate parameter. For example, in the command line **SEA 0 100 -A HELLO** the third parameter is **-A HELLO**, signifying the ASCII string "HELLO".

Using a modifier affects only one invocation of a command.

Certain commands may produce unexpected results if you specify command modifiers that are invalid or contradictory.

## Parameters

Parameters specify how the command is executed.

Parameters in boldface must be entered exactly as they appear in the syntax block, when used.

Parameters which are not in boldface describe the type of parameter. Acceptable entries for parameter types are described in the PARAMETERS explanation for each command.

Parameters may be required or optional, as described in the following paragraphs.

**Required Parameters.** Required single parameters appear in the command line without braces or brackets. In Fig. 3-1, **file1** is a required parameter.

**Optional Parameters.** Optional parameters are enclosed in brackets [ ] in the syntax block. In Fig. 3-1, **file2** is an optional parameter.

**Omitting Parameters.** To omit an optional parameter, enter two commas in its place. For example, to omit **file2** from the command line, enter:

```
> SAMPLE -M MYFILE, ,PA 100
```

Use three commas when you omit two consecutive parameters. For example, to omit **file2**, and **PA** or **PB** from the command line, enter:

```
> SAMPLE -M MYFILE, , ,100
```

Do not enter commas to omit a command modifier or the final parameter(s) in a command line. For example, to omit **-M** from the command line, you can enter:

```
> SAMPLE MYFILE YOURFILE PA 100
```

**Choice of Parameters.** Parameters are stacked one above another when there is a choice between two or more parameters. If the parameters are stacked within braces { }, one of the parameters **must** be used. In Fig. 3-1, either **address** or **string** must be chosen. If the parameters are stacked within brackets [ ], the selection is optional. In Fig. 3-1, you may select either **PA** or **PB** or neither.

**Repeated Parameters.** When three dots follow a parameter (or a group of parameters enclosed in brackets or braces), the parameter may be repeated any number of times up to the end of the command line. In Fig. 3-1, the choice of **address** or **string** may be repeated as many times as the line length permits.

## Examples

Here are some examples of how the fictitious SAMPLE command (shown in Fig. 3-1) could be used:

```
> SAMPLE MYFILE, ,,"some text"
```

```
> SAMPLE -M YFL MFL PB "txta" 10 OF "txtb"
```

```
> SAMPLE -M HISFILE, ,PB 80 90 OAO OBO
```

## Redirecting Standard Input and Output

In general, the operating system commands take input from standard input and send output to standard output. The system terminal usually serves as both standard input and standard output.

However, you can redirect command input and output. A left arrow (<) redirects input, so the command can read from a file or device other than the system terminal. A right arrow (>) redirects output, so the command can send results to a file or device other than the system terminal.

The I/O redirection arrow can be inserted in a command line anywhere after the command name, and can be applied to any command.

Using more than one input arrow (<) or more than one output arrow (>) in one command entry causes an error.

**Examples of I/O Redirection.** Execute the LO command, and have the LO command download from the 8560 file MYLOFILE, with the following command line:

```
> lo <MYLOFILE
```

Execute the LO command, and have the LO command load from the paper tape reader with the following command line:

```
> LO <PTR
```

Disassemble address range 100-20A, and send the disassembly to the line printer with the following command line:

```
> DI 100 20A >LPT
```

### CAUTION

*Using the wrong redirection symbol may cause your file to be overwritten. For example, if you wanted to load from file LOADFILE but mistakenly entered lo >LOADFILE, then the LO command would attempt to read the load module from the terminal (standard input) and would send its output to LOADFILE, overwriting the current contents.*

## Multiple Commands

You may enter more than one command on an 80-character line, if you separate the commands with semicolons. For example, the first four commands in Display 3-1 are equivalent to the last command:

```
> BK 1 504 RD
> BK 2 10A
> TRA A LOOPST LOOPEND
> G 100

> BK 1 504 RD;BK 2 10A;TRA A LOOPST LOOPEND;G 100
```

Display 3-1.

The commands in a multiple-command line are executed in the order in which they're entered, not in parallel. Command errors occurring along the way will not stop subsequent commands from executing.

If you suspend a command in a multiple-command line using CTRL-C, the subsequent commands are lost.

If you enter a multiple-command line that is more than 80 characters long, the system issues an error message and does not execute any of the command line. Also, if any string substitutions are done on the multiple-command line so as to expand the line beyond 80 characters, the system issues an error message and does not execute any of the remaining commands in the long line.

## Escape Character

The backslash (\) is used to "escape from" or "turn off" any special significance of the character it precedes. For example, OS/40 recognizes the dollar sign (\$) as the first character of a string specification. To include a special character in a command parameter, precede it with a backslash (for example, \\$) to disable the special significance.

A character preceded by a backslash is treated "as is". For example, the operating system recognizes MY\FILE as MY"FILE, MY\FILE as MYFILE, and MY\\FILE as MY\FILE.

To include a single or double quote (' or ") within an ASCII string, precede it with a backslash. The operating system recognizes the string "I said,\"OK\"" as the string: "I said, "OK"".

You can't turn off the special significance of a carriage return with the backslash.

## Strings

You may assign names to strings of characters, and then refer to the strings by name. You can use strings as part of a command line, or a string can be a sequence of commands.

### NOTE

*Strings are volatile and will be lost if the 8540 is restarted or turned off. If you wish to save strings in the permanent string storage area in EEPROM, use the PERMSTR command.*

## Defining Strings

A name is assigned to a string by entering the string name, an equals sign, then the string. To define a string, use the syntax shown in Fig. 3-2.

```
stringname=[ ]string[ ]
```

Fig. 3-2. Syntax for string definition.

3939-10

### PARAMETERS

**stringname** The name you want to assign to the string of characters. May include up to 8 letters or digits, and the first character must be a letter. Uppercase and lowercase letters are considered distinct.

string The value of the string you want to name. The value must be enclosed in single or double quotes if it contains a delimiter such as a comma or space, but the quotes are not part of the string value. The value of the string must be enclosed in single quotes if it contains a backslash, a double quote, or a dollar sign.

In any of these cases, you may avoid using quotes if you "escape" the special character. For information on how to do this and on how to include single quotes within your string, refer to the "Escape Character" discussion in this section.

### Using Strings

You may reference the string anywhere in a command line by entering a dollar sign (\$) followed by the string name.

A string name may be made of stringnames. A string name may also expand into a dash modifier. If a string name is referenced, but no string with that name has been defined, a null value will be substituted.

String substitution is repeated until no more substitutions are possible. For example, if a = "ab" and c = "100" and ab100 = "1234", \$\$a\$c first changes to \$ab100 and finally to 1234.

String substitutions are done sequentially, not in parallel. For example, in the following sequence of commands, the second value of string abc ("efg") is the one that is found by the SEA command.

```
> abc="xyz"
> abc="efg" ; SEA $abc
```

### Using Quotes in Strings

Double quotes (") are used to enclose a string that contains delimiters such as commas or spaces. All characters within the double quotes are treated as a single unit.

Single quotes (') are used to enclose a string of characters that are to be treated "as is", and the single quotes escape (turn off) the special significance of any characters within the single quotes, such as the backslash and dollar sign. For example, if string xyz equals 100, then the string "abc \$xyz def" becomes abc 100 def. However, the string 'abc \$xyz def' remains abc \$xyz def.

### Examples

Define strings REGSET, RANGE1, and RANGE2 with the following command lines:

```
> REGSET="S R1=0 R2=2 R3=3"
> RANGE1="1000,1300"
> RANGE2="2010,3000"
```

Enter the string REGSET (equivalent to entering the S command) with the following command line:

```
> $REGSET
```

Disassemble address range 1000-1300 with the following command line:

```
> DI $RANGE1
```

Fill address range 2010-3000 with zeros with the following command line:

```
> F $RANGE2 00
```

## Legal Address Expressions

Any address parameter in a command line can be represented by an expression: a formula that contains numbers, register names, symbols, and memory space designators, related by the operators + and -. An expression must not contain any spaces.

The following paragraphs describe each of these expression elements.

**Numbers.** Each number is suffixed with the standard character that selects its radix:

H	hexadecimal (default if no suffix is specified)
T	decimal
Q or O	octal
Y	binary

A hexadecimal number cannot begin with a letter. For example, the hexadecimal number A must be entered as 0A.

#### NOTE

*Do not specify a negative value for an address parameter. Negative addresses may produce unexpected results.*

**NOTE**

*Care must be taken if the expression parameter begins with a minus sign. It will be considered a dash modifier unless you precede it with a zero. For example, 0-BIAS.*

**Don't-Cares.** An "X" within an expression can signify a "don't-care" value. Don't-care values are not allowed in decimal expressions (radix selector T); an error will be returned if you try to do so.

An expression beginning with a don't-care X will be mistaken for a program symbol. Precede the X with a zero to begin an expression with a don't-care.

Don't-care values are used mainly in Trigger Trace Analyzer commands.

**Register Names.** Register names are used in expressions to represent the contents of the emulator registers. For example, the command CALC A+B displays the sum of the numbers in registers A and B. Refer to your Emulator Specifics supplement for the register names of your emulator.

**Operators.** The two operators + and - are used in expressions to represent addition (+) and subtraction (-).

**Symbols.** Expressions may also include symbols. Symbols are names that represent values in your program. For example, two types of values that may be represented by symbols are addresses (unsigned numeric values), and scalars (signed numeric values). The primary source of symbols is your assembly language program. After you assemble and link your program, the load file contains the symbols from your source code, and you can use the SYMLO or COM command to download the symbols and their values into the symbol table in system memory.

A secondary source of symbols is the ADDS command, which adds symbols and their values to the symbol table.

You can use the S command to assign a new value to an existing symbol.

A symbol consists of one or more characters beginning with a letter and containing only letters, digits, periods, underscores, or dollar signs. ADDS and REMS issue an error message if the symbol has more than 16 characters. Elsewhere, any characters after the 16th are ignored.

Symbols may be entered in command line expressions in either of their two forms: as "symbol" or "symbolspec". When "symbol" is entered in the command line it is simply the name of the symbol. A "symbolspec" is either: the name of the section in which the symbol is found, followed by a colon and then the symbol; or, simply the symbol. The term "symbolspec" in a syntax block stands for either. For

example, if the symbol ELECTION, representing an address, appears in the label field of the program section CAMPAIGN, then ELECTION is the "symbol" and either CAMPAIGN:ELECTION or just ELECTION is the "symbolspec". Since scalars are not considered to be assigned to a particular section, the "symbolspec" for scalar symbols should not include a section name.

Refer to your Assembler Users Manual for more details about symbols.

**Memory Space Designators.** A number may be prefixed with a memory space designator, which is a pair of letters, followed by a colon. For example, in the Z8001/Z8002 emulator, the term SC:1000 specifies address 1000 in the System Code memory space. Refer to the Emulator Specifics section of this manual to find out if your emulator uses memory space designators.

**Examples.** The following expressions are legal:

```
0
OFF
W+5
X+Y-Z
SC: 1000
OAH+5T-7Q
JUMPSTART
FH (Unacceptable as a hexadecimal number: must
not start with a letter. Acceptable as symbol FH:
e.g., in ADDS command.)
```

The following expressions are illegal:

```
(W+5) (An expression may not contain paren-
theses.)
X*Y/Z (An expression may not contain asterisks
or slashes.)
OAH + 5T (An expression may not contain spaces )
```

## SPECIAL KEYS

The following special function keys and control characters are recognized by the operating system:

- CTRL-C—interrupts command or program execution
- CTRL-Q—continues display after CTRL-S has been used
- CTRL-R—displays all text in type-ahead buffer
- CTRL-S—halts display
- CTRL-U—discards all text in type-ahead buffer
- CTRL-Z—end of file
- BACKSPACE—deletes character
- RUBOUT—deletes character

To enter a control character (CTRL-x), hold down the CTRL key while you press the appropriate key.

## CTRL-C

One CTRL-C alerts the system to accept an additional command. Two CTRL-Cs in a row suspend all suspendable commands.

Entering CTRL-C can cause one of the following effects:

- **One CTRL-C during command line entry** deletes the contents of the type-ahead buffer, and displays a single or double prompt (> or >>) on the following line. A double prompt will be displayed only if one or more commands are currently running. The system then waits for you to enter another command line.
- **One CTRL-C during command execution** causes a single or double prompt (> or >>) to be displayed and allows you to enter an additional command (usually an A command to abort the command already executing). Any active command continues to execute until it needs to access the terminal.
- **One CTRL-C while your program is running** suspends the program and displays a trace line indicating where the break occurred. To resume program execution, use the G command.
- **Two CTRL-Cs during command execution** cause all suspendable commands to be suspended. To abort a process after suspending, use the A command. To continue execution after suspending, use the CO command.

If you enter CTRL-C while a multi-command line is executing, all commands after the one currently executing are lost.

CTRL-C does not send a visible character to standard output.

If the 8540 is in TERM mode with an 8560, CTRL-C suspends your program, if any, and aborts any commands that are executing.

## CTRL-Q

CTRL-Q continues the display on standard output after it has been suspended by CTRL-S. Any character other than CTRL-Q is used as input and also resumes the display.

CTRL-Q does not send a visible character to standard output.

## CTRL-R

CTRL-R causes all characters in the type-ahead buffer to be displayed on the terminal. If you have typed ahead more than one line, all lines typed in will be displayed. Refer to the Learning Guide section of this manual for a description of the type-ahead buffer.

CTRL-R sends the characters ^R to standard output.

## CTRL-S

CTRL-S suspends the console display. Enter any character to continue the display. CTRL-Q is the only character that will resume the display without also being used as input. All other characters are used as input in addition to resuming the display. You may halt and continue the console display as many times as you desire.

CTRL-S does not send a visible character to standard output.

## CTRL-U

CTRL-U causes all characters in the type-ahead buffer to be discarded. If you have typed ahead more than one line, all lines will be discarded. Refer to the Learning Guide section of this manual for a description of the type-ahead buffer.

CTRL-U sends the characters ^U to standard output.

## CTRL-Z

CTRL-Z is an end-of-file indication for ASCII input data. It does not send a visible character to the terminal.

## BACKSPACE

The BACKSPACE key on a CRT terminal deletes the most recently typed character in the type-ahead buffer, erases it from the screen, and moves the cursor one space to the left. Refer to the Learning Guide section of this manual for a description of the type-ahead buffer.

## RUBOUT

The RUBOUT key performs the same actions as the BACKSPACE key.

## COMMAND MEMORY AREAS

Two commands can execute concurrently if they do not run in the same memory area. The OS/40 commands reside in two areas of memory. Six of the commands (A, CO, G, LO, SUSP, and X) are resident; the rest share the system memory overlay area. If you press CTRL-C once while one of the commands in the system memory overlay area is executing, the system prompt becomes a double right arrow (>>), and you can execute one of the resident commands.



## SYNTAX

```

a { command
  -a
  -u }

```

## PARAMETERS

command The name of a command: any command listed in this Command Dictionary.

-a All active commands or programs.

-u The program executing in program memory.

## EXPLANATION

The A command terminates execution of the specified command or program. The A command also closes all channels assigned to the aborted process. After execution of the specified process is aborted, control returns to the operating system, and you may enter commands as usual.

You can only enter the A command when the system prompt is displayed. When previously entered commands or programs are running, you must first return control to the keyboard with CTRL-C.

Entering one CTRL-C alerts the system to accept another command; commands currently running are not disturbed until they need access to the terminal. Entering two successive CTRL-Cs alerts the system to accept another command, and suspends all currently running commands or programs. Refer to the "Special Keys" heading in this section for details on the CTRL-C character. The system prompt will be displayed after you enter one or two CTRL-Cs.

When the system prompt is displayed, you can enter the A command to terminate execution. The system then issues an "Error FE—Process aborted" message, along with the name of the affected command.

To terminate a particular command, enter the A command with the name of the command to be terminated. To terminate all active processes, enter A -A. To close the channels of the program executing in program memory, enter A -U.

## NOTE

*If you abort a displaying command, such as the D command, an extra line of text is displayed after the A command is invoked.*

The following commands may NOT be aborted or suspended:

A	G	SUSP
COM	LO	X
CO	SEL	

Refer to the SUSP command for information on suspending, rather than aborting, command execution.

## EXAMPLES

**Abort a Command**

Start dumping a large listing on the line printer with the following command entry:

```
> D O OFFFF >LPT
```

Enter CTRL-C to receive the system prompt, then enter the following command to abort execution of the D command:

```
> A D
```

**Abort the User Program**

Run your program by entering the G command:

```
> G
```

Enter CTRL-C to suspend your program. A trace line is displayed, then a BREAK message, then the system prompt. Enter the following command line to close the channels opened by your program:

```
> A -U
```

**SYNTAX**

```
adds { [sectionname:] addresssymbol=value }
      -s scalarsymbol=value } ...
```

**PARAMETERS**

-s	Signifies that the following symbol represents a scalar value.
addresssymbol	A user-defined symbol to be used to represent the address.
sectionname	The name of the section with which <b>addresssymbol</b> is to be associated. This parameter is required if the symbol does not lie within the bounds of the section to which it is associated, or when a symbol is added to an absolute section. If <b>sectionname</b> is not indicated, the system will automatically associate the symbol with the section which contains it.
scalarsymbol	A user-defined symbol to be used to represent a scalar value. The symbol may not contain a section name.
value	An expression whose value is assigned to the symbol.

**EXPLANATION**

The ADDS command inserts a user-defined symbol, with its associated value, into the user symbol table. The symbol can then be used in place of the value in any command line during the debug session.

When you reboot or SElect, symbolic debug automatically puts a dummy section entry called "NO.SECTION" in the symbol table. This section encompasses all of memory, and allows you to put address symbols in the symbol table using ADDS without doing a SYMLO.

Refer to the Emulation section of this manual for an overview of symbolic debug.

**NOTE**

*If you define an address symbol with the ADDS command, the address must lie within a currently defined section unless "sectionname:" is included in the command. Section definition information is provided in the load module. You cannot use the ADDS command to define a section.*

**NOTE**

*Like other symbols in the symbol table, symbols created with the ADDS command reside in system memory, and will be lost when the 8540 is restarted or turned off.*

**EXAMPLES****Use a Symbol to Monitor a Memory Address**

Assuming that there is a section containing the address 500H, enter the following command line to create a symbol called COUNTER and assign it a value of 500H:

```
> ADDS COUNTER=500H
```

Enter the following command line to define breakpoint number 1, which suspends execution when a write is made to the address represented by COUNTER:

```
> BK 1 COUNTER WT
```

**Use Symbols to Monitor a Loop**

Enter the following command line to create symbols LOOPST and LOOPEND, and assign them the values of the addresses of the beginning and ending of the loop:

```
> ADDS LOOPST=0106H LOOPEND=0109H
```

Enter the following command line to trace all the instructions within the loop as it executes:

```
> TRA A LOOPST LOOPEND
```

**Add a Symbol Outside Existing Sections**

If you have only one section called "SECTIONA" in memory locations 100-200, and you attempt to add the symbol "SYMBOL" at location 300, as follows:

```
> ADDS SYMBOL=300
```

you will get this error message:

```
Error 8D No section contains specified address
```

Instead, use the section name, too:

```
> ADDS SECTIONA:SYMBOL=300
```

# AL

Allocates memory to logical address map

## SYNTAX

al [actual] [blocks] [logical]

## PARAMETERS

- |         |   |
|---------|---|
| actual  | An expression representing the beginning address of the program memory region that you want to allocate to logical (virtual) addresses. Must be between 0 and the highest address in existing program memory. Defaults to 0.                  |
| blocks  | An expression representing the number of contiguous 4K-byte blocks of program memory you want to allocate. Must be between 1 and 10H. Defaults to the number of blocks from the actual memory address to the end of installed program memory. |
| logical | An expression representing the beginning address of the first logical (relocated) block. Must be between 0 and FFFF. Defaults to the actual memory address.   |

The ALlocate command entered without parameters displays the memory allocation status.

## EXPLANATION

### NOTE

*There are two different versions of the AL command. The version described here is supported by the 8540 itself. The Memory Allocation Controller (MAC) option also supports an AL command which has the syntax:*

al loadr [hiaddr]

*The MAC version of the AL command is described in the Emulation section of this manual.*

*Some emulators (e.g. the 8086/8088) support the MAC version of the AL command even though they do not use the MAC option. Refer to your Emulator Specifics supplement.*

### NOTE

*You must have SElected the target processor before executing the AL command. The SEL command reinitializes logical address assignments to be the same as the physical addresses.*

The AL command allows your program to access a block of program memory at logical addresses instead of their physical addresses.

If you only have 32K of program memory (addresses 0000-7FFF), but your program accesses addresses in the

range 8000-FFFF, you can use the AL command to temporarily reassign a section of program memory from the 0000-7FFF address range to the 8000-FFFF address range.

Initial settings may be restored by entering the following command line:

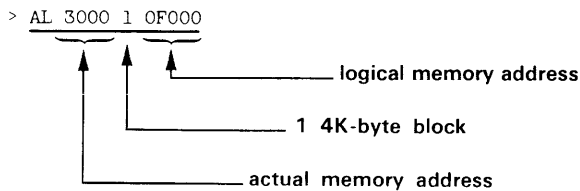
```
> AL 0 10 0
```

**NOTE**

*The AL command affects only program memory, not prototype memory. Memory allocation has no effect in emulation mode 2, and does not affect prototype memory locations in mode 1.*

**EXAMPLE**

Assume your program has a subroutine in the address range F000-FFFF, but you only have 32K of program memory (up to address 7FFF). Also assume your program does not use any addresses in the range 3000-3FFF. Enter the following command to allocate the unused memory to the needed memory:



To display the memory allocation status, enter the AL command. See Display 3-2.

Figure 3-3 shows how your memory map now looks.

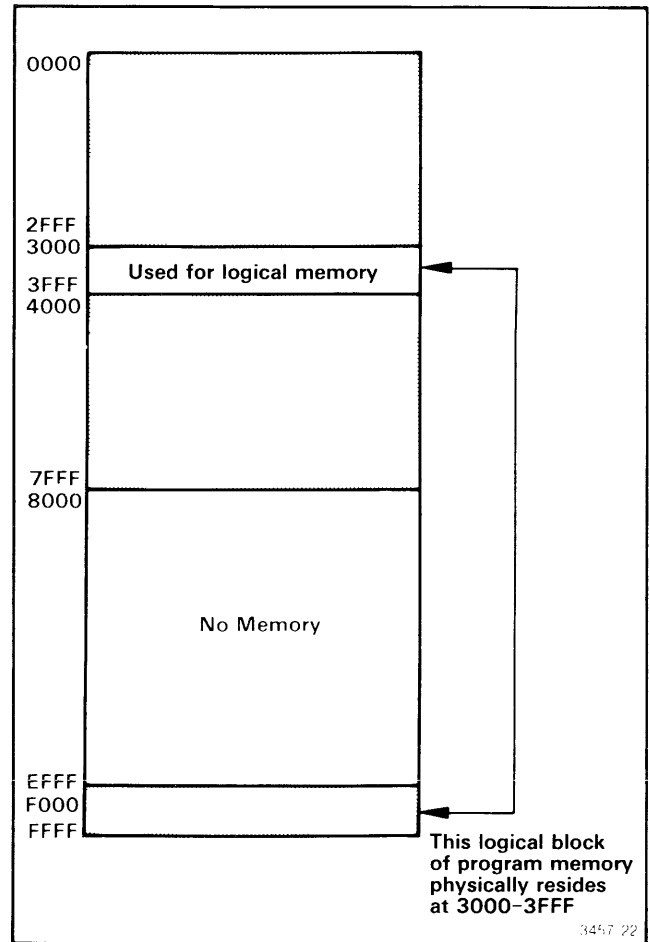


Fig. 3-3. AL command example.

```
> AL
```

Virtual	=	Physical address
0000-0FFF	=	0000-0FFF
1000-1FFF	=	1000-1FFF
2000-2FFF	=	2000-2FFF
3000-3FFF	=	3000-3FFF
4000-4FFF	=	4000-4FFF
5000-5FFF	=	5000-5FFF
6000-6FFF	=	6000-6FFF
7000-7FFF	=	7000-7FFF
8000-8FFF	=	8000-8FFF
9000-9FFF	=	9000-9FFF
A000-AFFF	=	A000-AFFF
B000-BFFF	=	B000-BFFF
C000-CFFF	=	C000-CFFF
D000-DFFF	=	D000-DFFF
E000-EFFF	=	E000-EFFF
F000-FFFF	=	3000-3FFF

← When the program references addresses between F000 and FFFF, the emulator will access physical memory at 3000 to 3FFF.

Display 3-2.

**SYNTAX**

```
as {channel filespec} ...
```

**PARAMETERS**

**channel** An I/O channel number between 0 and 7.

**filespec** The name of the file or device to be assigned to the channel number.

**EXPLANATION**

The AS command assigns the specified device or host file to the specified I/O channel. Channel assignments allow your programs to read from or write to devices or files on the host, using service calls. Refer to the Service Calls section of this manual for further details.

Channels are numbered 0 through 9. Channels 8 and 9 are initially assigned to the system for standard input and standard output, respectively.

Console input (CONI) and console output (CONO) may be assigned to any number of channels at the same time. All other devices are limited to one channel assignment at a time.

Table 3-1 lists the allowable system device names, their meanings, and the connector numbers to which the devices must be attached.

**Table 3-1**  
**System Devices**

Device Name	Meaning	Connector Number
CONO	Console output	J104
CONI	Console input	J104
LPT	Line printer	J103
PPTP	Paper-tape punch	J103
PPTR	Paper-tape reader	J103
REMI	Remote input	J101(DTE), J102(DCE)
REMO	Remote output	J101(DTE), J102(DCE)

To display the channel-to-device assignments and the emulator processor status, enter the STAT command. To disconnect a channel from a device, use the CL command.

The AS command creates a new file if the specified file does not exist.

**NOTE**

*You must have SElected the target processor before executing the AS command.*

*Your 8540 must be in TERM mode in order to assign files on the host.*

*The 8560 also has an AS command, which invokes the TNIX assembler. If your 8540 is operating in TERM mode with an 8560, the 8540 AS command must be preceded by the word "8540," as in the following examples.*

**EXAMPLES****Assign Channels**

Connect channel 0 to the host file named DATAFILE and channel 1 to the local line printer (LPT) with the following command line:

```
> 8540 as 0 DATAFILE 1 LPT
```

Enter the STAT command to display channel assignments and emulator status. See Display 3-3.

**Reassign Channel**

Here's an example of reassigning a channel that's already been assigned once. Try to assign channel 1, which is already assigned to the line printer, to the file OUTFILE with the following command line:

```
> 8540 as 1 OUTFILE  
> AS: Error 18
```

**(Channel already open)**

An error occurs. You must disconnect the assigned channel from the line printer before you can connect that channel to another file or device. Enter the following command line to reassign channel 1:

```
> cl 1  
> 8540 as 1 OUTFILE
```

Now enter the STAT command again. See Display 3-4.

```
> stat

8085 is the selected processor

User program was last loaded into program memory

Channel 0 is assigned to DATAFILE
Channel 1 is assigned to LPT
```

Display 3-3.

```
> stat

8085 is the selected processor

User program was last loaded into program memory

Channel 0 is assigned to DATAFILE
Channel 1 is assigned to OUTFILE
```

Display 3-4.

## SYNTAX

```
bk [ { 1 }
    { 2 }
    all ] clr
or
bk [-c] { 1 } expression [rd]
    { 2 }
```

## PARAMETERS

1	Specifies breakpoint number 1.
2	Specifies breakpoint number 2.
all	Specifies all currently defined breakpoints.
clr	Clears the specified breakpoint.
-c	Continues execution after each breakpoint occurs. If -C is not specified (default condition), the BK command stops execution after a breakpoint occurs. To resume program execution, enter the G command without parameters.
expression	An expression representing the address where program execution is to be interrupted.
rd	Designates that a breakpoint occurs when a memory read operation occurs at the specified address. Defaults to any access (read or write).
wt	Designates that a breakpoint occurs when a memory write operation occurs at the specified address. Defaults to any access (read or write).

## NOTE

*The syntax for the BK command may be different for some emulators. Refer to your Emulator Specifics supplement for details.*

When you enter the BK command without parameters, the currently set breakpoints are displayed. The breakpoint display will contain symbols if breakpoints are set at addresses of loaded symbols.

## EXPLANATION

The BK (BreakPoint) command sets a breakpoint that will suspend program execution when the specified address is read from or written to. Up to two breakpoints may be active at one time.

When a breakpoint is encountered during execution, a trace line is displayed followed by the message "BKPT1" or "BKPT2", depending on which breakpoint occurred. Some emulators offer additional features with the BK command, such as the ability to set up to 3 breakpoints concurrently. See the Emulator Specifics supplement for your emulator for details.

If the optional RD/WT parameter is omitted, the breakpoint will occur whenever the address is read from or written to.

To clear existing breakpoints, use the CLR parameter.

## NOTE

*You must have SElected the target processor before executing the BK command.*

*If you set a breakpoint at an I/O instruction that invokes an SVC, the SVC executes **before** the break occurs. For some emulators, execution of the SVC may prevent the operating system from breaking at the desired instruction.*

## EXAMPLES

## Set and Use a Breakpoint

Display the currently set breakpoints with the following command line:

```
> bk
BK 1 CLR
BK 2 CLR
```

No breakpoint has been set. To set a breakpoint that executes only if your program reads an instruction or other data from address 504, enter the following command line:

```
> bk 1 504 rd
```

Now select emulation mode 0, download the 8560 file OBJ into program memory, and start execution at location 100 with the command sequence shown in Display 3-5.

A breakpoint is encountered when the instruction at location 106 reads data from address 504. A trace line is displayed, followed by the BREAK message.

To display the currently set breakpoints, enter the following command line:

```
> bk
BK 1 000504 RD
BK 2 CLR
```

### Set a Breakpoint Using a Symbol

Assume that address 504 is the end of a data list which your program uses. The following sequence of commands

assigns the symbol ENDLIST to address 504, and sets a breakpoint to occur at the symbol (address 504) during program execution.

Clear any existing breakpoints with the following command line:

```
> BK ALL CLR
```

Assign symbol ENDLIST to address 504 with the following command line:

```
> ADDS ENDLIST=504
```

Set a breakpoint to break on a read from symbol ENDLIST (address 504) with the following command line:

```
> BK 1 ENDLIST RD
```

With SYMD ON, enter the following command line to display the breakpoint conditions:

```
> BK
BK 1 ENDLIST RD
BK 2 CLR
```

```
> em 0
> lo <OBJ
> g 100

LOC  INST  MNEM OPER      SP  F  A  B  C  D  E  H  L  IM SOD
000106 86      ADD  M      0000 04 0F 01 00 00 00 05 04 07 0
000106 <BREAK  BKPT1>
```

Display 3-5.

**SYNTAX**

**calc** [-radix] expression

**PARAMETERS**

radix      A letter that selects the number base (radix) for the result of the calculation. The allowable radix selector letters are:

H          hexadecimal (default value)

T or D     decimal (base 10)

Q or O     octal (base 8)

Y or B     binary (base 2)

expression   Any valid expression as defined under the "Legal Address Expressions" heading earlier in this section.

**EXPLANATION**

The CALC command evaluates an arithmetic expression and displays the result on standard output (usually the system terminal). The result is displayed in the selected radix, as shown by suffix H, T, Q, or Y. A negative result is output in two's-complement notation, except for decimal numbers, where a minus sign is used instead.

A hexadecimal number cannot begin with a letter. For example, the hexadecimal number A must be entered as 0A.

**EXAMPLES**

**Add Numbers**

Calculate the sum of the hexadecimal numbers A, B, and C and display the result in hexadecimal with the following command line:

```
> CALC 0A+0B+0C
21H
```

```
> DS
PC=0011 SP=5500 F=65 A=04 B=0F C=01 D=66 E=00 H=01 L=11
SOD=1 SID=0 I7=1 I6=1 I5=1 IE=1 M7=0 M6=0 M5=0
```

**Display 3-6.**

Calculate the sum of the hexadecimal numbers A, B, and C, but display the result in decimal with the following command line:

```
> CALC -T 0A+0B+0C
33T
```

**Add Contents of Registers**

Display the contents of the emulator registers with the DS command (the selected emulator is the 8085A). See Display 3-6.

When a leading zero is omitted in an expression, the CALC command recognizes the character as a symbol. To calculate the sum of the contents of registers A, B, and C, enter:

```
> CALC A+B+C      (04+0F+01=14H)
14H
```

**Subtract Numbers**

To subtract decimal number 6 from decimal number 5 and display the result in hexadecimal, enter:

```
> CALC 5T-6T
FFFFFFFFH
```

Notice that the result is output in two's-complement notation.

To subtract the decimal number 6 from decimal number 5 and display the result in decimal, enter:

```
> CALC -T 5T-6T
-1T
```

Notice that the negative result is displayed in decimal with a minus sign.



**SYNTAX**

```
cl channel...
```

**PARAMETERS**

channel      An I/O channel number between 0 and 7.

**EXPLANATION**

The CL command disconnects the specified I/O channel(s) from the device or host file previously associated with the channel.

To check the status of I/O channels and the files or devices associated with them, enter the STAT command.

**NOTE**

*You must have SElected the target processor before executing the CL command.*

*When you use the CL command, your 8540 must be in TERM mode in order to access files on the host.*

Refer to the AS command for information on assigning, rather than closing, channels.

**EXAMPLE**

Display the current system status with the STAT command. See Display 3-7.

```
> stat
8085 is the selected processor
User program was last loaded into program memory
Channel 0 is assigned to LPT
Channel 1 is assigned to MYFILE
```

**Display 3-7.**

Disconnect channels 0 and 1 from LPT and MYFILE, respectively, with the following command line:

```
> cl 0 1
```

Display the system status again. See Display 3-8.

```
> stat
8085 is the selected processor
User program was last loaded into program memory
```

**Display 3-8.**

Note that no channel assignments are displayed.

**SYNTAX**

```
clock [ on
       off
       value ]
```

**PARAMETERS**

- on Enables the 100-millisecond program clock.
- off Disables the 100-millisecond program clock.
- value A decimal number between 0 and 65535. Sets the 100-millisecond program clock to the given value.

When you enter the CLOCK command without parameters, the current value of the clock and the elapsed time are displayed.

**EXPLANATION**

The CLOCK command zeros, enables, or disables the program clock. When this clock is ON, it will be incremented every 100 milliseconds only during emulator program execution. The program clock defaults to OFF after system power-up or restart.

While the clock is ON, the emulator runs approximately one percent slower than otherwise.

The Trigger Trace Analyzer provides more precise and versatile timing features. This option is described in the Emulation section of this manual.

**NOTE**

*You must have SElected the target processor before executing the CLOCK command.*

**EXAMPLES****Display Current Clock Values**

Display the current clock counter and elapsed time by entering the CLOCK command without a parameter:

```
> CLOCK
Clock counter is 78
Elapsed time is 47
```

**Time Execution of Program Segment**

Enter the following command to zero the program clock:

```
> CLOCK 0
```

Now enable the program clock with the following command:

```
> CLOCK ON
```

Start execution by entering the G command. See Display 3-9.

Display the program clock and elapsed time again:

```
> CLOCK
Clock counter is 64
Elapsed time is 64
```

The program took 6.4 seconds (6400 milliseconds) to execute the code between address 100 and the first breakpoint. Now resume program execution at the instruction following the breakpoint. See Display 3-10.

Display the program clock again:

```
> CLOCK
Clock counter is 69
Elapsed time is 5
```

One-half second (500 milliseconds) elapsed before the second break. A total of 6.9 seconds of execution time have elapsed since program execution began at address 100.

```
> G 100
LOC  INST  MNEM OPER      SP F  A  B  C  D  E  H  L  IM SOD
000306 86    ADD  M      0000 04 0F 01 01 00 00 05 04 07  0
000306 <BREAK BKPT1>
```

**Display 3-9.**

```
> G
LOC  INST  MNEM OPER      SP F  A  B  C  D  E  H  L  IM SOD
0200 86    ADD  M      0000 04 1F 01 00 00 00 06 0A 07  0
0200 <BREAK BKPT2>
```

**Display 3-10.**

# CO

Resumes execution of suspended command

## SYNTAX

```
co { command }  
   -a
```

## PARAMETERS

**command** The name of a command; any command listed in this Command Dictionary or an allowed short form.

**-a** Designates all suspended commands.

## EXPLANATION

The CO (COntinue) command continues execution of a suspended command. Execution resumes at the point where the process was suspended.

A command may be suspended in either of two ways:

1. by entering CTRL-C twice (suspends all processes), or
2. by entering CTRL-C once and then entering the SUSP command.

## SYNTAX

```
com [e={r}] [l={o}] [p=prompt] [t=delay] [m=parity] [c={t}]
    [f=t] [eol=hexstring] [hs=off] [sub ch1=ch2/ch1=ch2...]
    [b=bytecount]
```

## PARAMETERS

e=	Echo parameter.	c=	Error check parameter.
r	Selects remote echoing of characters entered from the system terminal. (Host echoes character.) Default value.	t	Specifies that COM execution stops if a communication error (lost carrier, parity error, or framing error) occurs on the remote communication port. Default value.
l	Selects local echoing of characters entered from the system terminal. (8540 echoes character.)	i	Ignores communication error and continues COM execution.
l=	Linefeed parameter.	f=	Load Module Format parameter. This parameter specifies the message block format to be used in formatted uploads. If this parameter is omitted, blocks are transmitted in Extended Tekhex format.
o	Specifies that the 8540 does not output a linefeed with a carriage return. (Host provides linefeed.) Default value.	t	Specifies Standard Tekhex format. COM accepts message blocks in either Standard or Extended Tekhex format, regardless of the F parameter setting.
i	Specifies that the 8540 outputs a linefeed with each carriage return.	eol=	End-of-line parameter. This parameter specifies the character sequence used by the host to mark the end of a line of text, such as a host command or a Tekhex message block.
p=	Prompt parameter. Specifies the prompt sequence of the host computer communicating with the 8540.	hexstring	A maximum of 32 hexadecimal digits representing a character string. Defaults to an ASCII carriage return (0D hexadecimal).
prompt	Prompt sequence. The characters that the host computer outputs when it is ready to receive data. The prompt sequence consists of up to 32 hexadecimal digits, representing up to 16 ASCII characters. Defaults to no prompt sequence.	hs=off	Handshaking parameter. The parameter HS=OFF eliminates the ACK/NAK response from the protocol of a formatted transfer. In a download, COM normally responds with an ACK or NAK each time it receives a message block from the host. When handshaking is OFF, COM gives no response. However, checksums are still computed and checked, and if any are found to be incorrect, COM issues an error message when the transfer is complete. In an upload, COM expects no ACK or NAK response from the host. Checksum calculating is also turned OFF.
t=	Turnaround delay parameter. Specifies the preparation time required by the host computer between sending a message or prompt sequence to the 8540 and receiving a message from the 8540 during data transfers.		Turning handshaking off has no effect on the P (prompt sequence) and T (turnaround delay) parameters. COM still looks for the host's prompt sequence (if any) and waits the indicated turnaround delay time (if any) before continuing.
delay	Delay time. A two-digit hexadecimal number representing the delay time in 100-millisecond units. Defaults to 00. (For COM Version 4.1 and later versions, the delay time is specified in 10-millisecond units.)	sub	Substitution parameter. This parameter specifies one or more character substitutions to be performed by COM on ASCII data transmitted to or from the host. Each
m=	Parity parameter. Selects the parity option required by the host computer when it receives or sends data.		
parity	Includes character transmission length and number of stop bits. Refer to a table of the Intersystem Communication section for the		

substitution construct consists of a pair of character codes (represented as two-digit hexadecimal numbers) separated by an equals sign. When the first character is received from the host, COM substitutes the second character. When the second character is to be sent to the host, COM substitutes the first.

Multiple substitution constructs can be strung together, separated by slashes. For example, the parameter entry **SUB aa=bb/cc=dd/ee=ff** will cause the host character aa to be changed to bb; the host character cc to be changed to dd; and the host character ee to be changed to ff, whenever COM receives the characters aa, cc, or ee from the host.

ch1	ASCII character encoded as two hexadecimal digits.
ch2	ASCII character encoded as two hexadecimal digits.
b=	Byte count parameter. Specifies the number of bytes of object code that COM will place in each data block during formatted uploads. (Each byte is encoded as two hexadecimal digits.) This parameter is supported only for COM Version 4.1 and later versions. This parameter is ignored if you specify <b>F=T</b> .
bytecount	A two-digit hexadecimal number in the range 01-76. Defaults to 20 (32 decimal). Any number outside the range 01-76 yields a byte count of 76 (118 decimal).

Any or all of the COM parameters may be entered in the command line, in any order. If the same parameter is entered in the command line more than once, the most recent entry overrides any previous entry.

When you enter the COM command without parameters, communication is started using the default values, which is the same as entering:

```
> COM E=R L=0 T=00 M=4 C=T EOL=OD B=20
```

The COM (COMmunication) command initiates communication between the 8540 and a host computer. After you execute the COM command, you can enter host computer commands on the system terminal.

**Transferring Data.** COM is used for transferring data between files on the host computer and program/prototype memory or I/O ports on the 8540.

Refer to the Intersystem Communication of this manual for a complete description of COM command parameters, and for a detailed explanation of each type of transfer. Refer to the Operating Procedures of this manual for examples of each type of transfer.

A transfer is accomplished by the following procedure:

- Enter the host computer command that transfers the data between the terminal and the appropriate file on the host system. (Do **not** enter a carriage return.)
- Enter the sequence of keystrokes (transfer sequence) that tells the 8540 what to send or where to store what it receives. The transfer sequence begins with a null character.

Appropriate transfer sequences are listed in Table 3-2.

**Obtaining COM Status.** Status information about the connection between the 8540 and the host computer is maintained in the 8-bit Remote Port Status Register. To display the contents of the Remote Port Status Register, enter the null character, then press the S key and enter a carriage return. The system terminal displays the register contents as two hexadecimal digits. Refer to the Intersystem Communication section for details about the Remote Port Status Register.

**Aborting a Data Transfer.** To abort a data transfer, press the BREAK key. You may then attempt another data transfer, enter another host command, or exit from COM.

**Exiting from COM.** To exit from COM, enter the null character and then press the ESC key.

**Table 3-2**  
**Types of Data Transfer**

<b>Name</b>	<b>Type of Data</b>	<b>Type of Transfer</b>	<b>Source</b>	<b>Destination</b>	<b>Transfer Sequence<sup>a</sup></b>
Formatted Download	Object Code	Tekhex to Binary	Host File	8540 Program Memory	(null)(cr)
Formatted Upload	Object Code	Binary to Tekhex	8540 Program Memory	Host File	(null) addresses <sup>b</sup> (cr)
Unformatted ASCII Download	ASCII	Copy	Host File	8540 Device	(null) >device <sup>c</sup> (cr)
Unformatted Binary Download	Binary	Copy	Host File	8540 Device	(null) =device <sup>c</sup> (cr)
Unformatted ASCII Upload	ASCII	Copy	8540 Device	Host File	(null) <device <sup>c</sup> (cr)
Unformatted Binary Upload	Binary	Copy	8540 Device	Host File	(null) +device <sup>c</sup> (cr)

<sup>a</sup> (null)—null character (ASCII 00, usually CTRL-@)  
(cr)—carriage return

<sup>b</sup> addresses—the starting and ending addresses of one or more blocks of program memory to be uploaded. The last pair of addresses may optionally be followed by the transfer address of the object code.

<sup>c</sup> device—the 8540 I/O device that is to provide or receive the data.

```

SYNTAX

config local
or
config term [e={r}] [i={o}] [m=parity] [j={r}] [t=hexstring]
```

t= Timeout parameter. This is a multiplier that affects TERM interface protocol timeouts. For example, if t=5, then all the timeouts will be five times as long as their default durations.

hexstring The multiplier. Can range from 1 to FF; the default is 1. If the multiplier starts with a letter (A-F) it must be preceded by a 0.

### PARAMETERS

local	Places the 8540 in LOCAL mode.
term	Places the 8540 in terminal (TERM) mode.
e=	Echo parameter. Applicable in TERM mode.
r	Remote echoing. Characters entered at the terminal are echoed by the host computer. Default value.
l	Local echoing. Characters entered at the terminal are echoed by the 8540.
i=	Linefeed parameter. Applicable in TERM mode.
o	Specifies that the 8540 will not output a linefeed with a carriage return. Default value.
i	Specifies that the 8540 will output a linefeed with a carriage return.
m=	Parity parameter. Selects the parity option required by the host computer when it receives or sends data. Applicable when i=r (RS-232-C).
parity	Includes character transmission length and number of stop bits. Refer to a table of the Intersystem Communication section for the possible values of this parameter and what those values represent. Defaults to 6.
i=	Interface parameter. Applicable in TERM mode.
r	Specifies that the 8540-host interface is the REMI/REMO port.
h	Specifies that the 8540-host interface is the HSI port. This is the default.

### EXPLANATION

The CONFIG (CONFIGuration) command puts the 8540 in one of two possible system configurations when the 8540 is connected to a host computer. In one configuration, TERM mode, the system acts just as if the terminal were connected directly to the host computer. In the other configuration, LOCAL mode, the system acts as if the terminal were connected only to the 8540.

### TERM and LOCAL Modes

In TERM mode, all characters entered at the system terminal are sent directly to the host computer. This mode is particularly powerful when the host computer is an 8560 Multi-User Software Development Unit. Refer to the Intersystem Communication section of this manual for more information on the use of TERM mode when the 8540 is connected to an 8560. Refer also to the 8560 System Users Manual for details about using an 8540 connected to an 8560.

In LOCAL mode, all characters entered at the system terminal are sent directly to the 8540 to be processed as OS/40 commands. The CONFIG LOCAL command produces the same effect as when the 8540 isn't connected to a host at all.

Unless your host is an 8560, you should use the COM command (rather than CONFIG) to transfer data between the host and the 8540.

### Startup Configurations

When you power up the 8540, you are in LOCAL mode. You may also achieve the effects of booting in TERM mode, using the startup command string. This feature is controlled by switch position number 1 (switch number S1100) on the 8540 System Controller board.

If the switch is closed (0), the 8540 executes the startup command string named \$STARTUP, residing in EEPROM.

If the switch is open (1), \$STARTUP is not executed.

If you want the 8540 to boot up in TERM mode, close the switch and include a CONFIG TERM command, with appropriate parameters, as the **first command** in the \$STARTUP string.

## Changing Configurations

The following list shows how to change the 8540/host computer system configuration:

1. From TERM mode to LOCAL mode: Enter the "config local" command (8560) or re-boot the 8540.
2. From LOCAL mode to TERM mode: Enter a CONFIG command, including TERM and other appropriate parameters.



# D

Displays memory contents

## SYNTAX

```
d [-b] [-w] loaddr [hiaddr]
```

## PARAMETERS

- b           Selects byte-oriented output.
- w           Selects word-oriented output.
- (default)   The default byte/word output format depends on the currently selected emulator. Refer to the Byte/Word Parameter discussion in the Emulator Specifics supplement for your emulator processor.
- loaddr       An expression representing the address of the first byte in program/prototype memory to be displayed.
- hiaddr       An expression representing the address of the last byte in program/prototype memory to be displayed. Defaults to (loaddr+OFH) or the top of memory, whichever is less.

## EXPLANATION

The D (Dump) command sends the contents of program or prototype memory to standard output (usually directed to the system terminal).

D displays the hexadecimal representation of the data alongside the ASCII representation. In the ASCII representation, periods represent data bytes with hexadecimal values less than 20 or greater than 7E.

The emulation mode and memory map assignments determine whether program or prototype memory data is displayed. Refer to the EM and MAP command discussions.

## EXAMPLES

For 8-bit processors, or if -B is specified, the D command output is in the format shown in Display 3-11.

For 16-bit processors, or if -W is specified (where each address represents one byte), the D output is in the format shown in Display 3-12.

For 16-bit processors with external word addressing (where each address represents one word), the D output is in the format shown in Display 3-13.

```
> D 3 16
   3 4 5 6 7 8 9 A B C D E F 0 1 2
000003 32 31 00 00 00 00 00 00 41 42 43 00 00 00 00 21 . . . . . ABC . . . . .
000013 00 30 31 32 . . . . . .012
```

Display 3-11.

```
> D 4 18
   4 6 8 A C E 0 2
000004 3100 0000 0000 0041 4243 0000 0000 0000 1 . . . . . ABC . . . . .
000014 3031 3233 0000 0123 . . . . .
```

Display 3-12.

```
> D 4 0E
   4:C 5:D 6:E 7:F 8:0 9:1 A:2 B:3
000004 3100 0000 0000 0041 4243 0000 0000 0000 1 . . . . . ABC . . . . . [words 4--B]
00000C 3031 3233 0000 0123 . . . . . [words C--E]
```

Display 3-13.

```

SYNTAX

deal {-aloaddr [hiaddr]}
```

**PARAMETERS**

- a Deallocate all program memory blocks.
- loaddr An expression representing the beginning address of the logical address block to be deallocated. If no memory space designator is used, all memory spaces in the given range will be deallocated.
- hiaddr An expression representing the ending address of the logical address block to be deallocated. Defaults to the ending address of the 4K-byte block specified by the **loaddr** parameter.

**EXPLANATION**

*NOTE*

*This command deallocates memory assignments made using the Memory Allocation Controller (MAC) version of the AL command, which is described in the Emulation section of this manual. If memory allocation was done using the version of the AL command described in this section, refer to the AL command for methods of restoring initial memory allocation settings. Refer to your Emulator Specifics supplement for further information.*

The DEAL (DEALlocate) command deallocates logical memory assignments. If memory had been allocated, DEAL causes physical memory to be made available for allocation to any logical address.

An attempt to access a non-allocated program memory address will cause an error to occur.

**EXAMPLES**

- > DEAL -A  
Deallocates all program memory. Makes all of program memory unavailable to your program.
- > DEAL 1000 1FFF  
Deallocates addresses 1000-1FFF.
- > DEAL 1000  
Deallocates all program memory from address 1000 to the end of the block. Effectively the same as the previous example.
- > DEAL PROMEMLO PROMEMHI  
Deallocates memory from the beginning of the block containing the address PROMEMLO to the end of the block containing the address PROMEMHI.

## SYNTAX

```
di [loadr] [hiaddr] [lines]
```

## PARAMETERS

loadr	An expression representing the program/prototype memory address where disassembly begins. Defaults to 0000.
hiaddr	An expression representing the program/prototype memory address where disassembly ends. Defaults to the end of memory.
lines	The number of lines to be disassembled. If this parameter and <b>hiaddr</b> are omitted, disassembly continues until the end of memory is reached or until you enter CTRL-C.

When you enter the DI command without parameters, disassembly starts at address 0000, and continues until the end of memory is reached or until you enter CTRL-C.

## EXPLANATION

The DI (Disassemble) command translates the object code in the specified memory area back into its corresponding assembly language mnemonics and operands, and sends this information to standard output, usually the system terminal.

If you enter only a single address parameter, disassembly proceeds from that address to the end of memory, or until you enter CTRL-C. If you have only 32K of program memory, addresses 8000 through the end of memory are considered to be all zeros.

The disassembly is erroneous if the specified **loadr** does not contain the first byte of an instruction. The disassembly terminates when either the **hiaddr** is reached or the **lines** parameter is satisfied.

The memory map assignments and emulation mode determine whether program or user prototype memory is

disassembled. Refer to the MAP and EM command discussions.

## NOTE

*You must have SElected the target processor before executing the DI command.*

## EXAMPLES

**Disassemble Between Addresses**

To disassemble addresses 100-102, enter:

```
> DI 100 102
```

```
LOC  INST  MNEM  OPER
000100 210005 LXI  H,0500
```

**Disassemble a Number of Lines**

Enter the following command line:

```
> DI 100 10E 4
```

```
LOC  INST  MNEM  OPER
000100 210005 LXI  H,0500
000103 0605  MVI  B,05
000105 AF    XRA  A
000106 86    ADD  M
```

DI disassembles only to address 106: Only the first four lines are disassembled. This example is the same as entering DI 100,,4.

**Disassemble to an Ending Address**

Enter the following command:

```
> DI 100 10E 30
```

```
LOC  INST  MNEM  OPER
000100 210005 LXI  H,0500
000103 0605  MVI  B,05
000105 AF    XRA  A
000106 86    ADD  M
000107 23    INX  H
000108 05    DCR  B
000109 C20601 JNZ  0106
00010C D3F7  OUT  F7
```

Only 9 lines are disassembled, because the **hiaddr** parameter is reached. This example is the same as entering DI 100 10E.

**SYNTAX**

ds [-l]

**PARAMETERS**

- l Long display. Displays all the emulator registers.
- (default) If -L is not specified, the DS command displays only the most frequently used emulator registers. Default and long displays are the same for some emulators.

**EXPLANATION**

The DS (Display Status) command displays the current status of the emulator. The display line varies for each

emulator; however, each display line includes the following information:

- the address of the next instruction to be executed by the emulator.
- the emulator register contents.

Refer to the Emulator Specifics supplement for the DS display format for your emulator.

*NOTE*

*You must have SElected the target processor before executing the DS command.*

**EXAMPLES**

Select the 8085A emulator and display the current status of the emulator registers with the command sequence shown in Display 3-14.

```

> SEL 8085
> DS

program counter shows address of next instruction to be
executed: initialized to 0000 when emulator is selected
└─┬──
   │
   ▼
PC=0000 SP=0000 F=00 A=00 B=00 C=00 D=00 E=00 H=00 L=00
SOD=0 SID=0 I7=0 I6=0 I5=0 IE=0 M7=1 M6=1 M5=1
    
```

Display 3-14.

**SYNTAX**`em [mode]`**PARAMETERS**

- mode One of the following modes:
- 0 System mode. Execution in this mode uses program memory and the emulator clock and uses SVCs for I/O. Mode 0 is set when an emulator is initially selected.
  - 1 Partial emulation mode. Execution in this mode uses program memory and/or user prototype memory, according to the map assignments. (Refer to the MAP command). Execution also uses the prototype's clock and may use either prototype I/O or (if available) SVCs.
  - 2 Full emulation mode. Execution in this mode uses the prototype's memory, clock, and I/O. Mode 2 SVCs are available for some emulators.

When you enter the EM command without parameters, the current emulation mode is displayed.

**EXPLANATION**

The EM (EMulate) command selects the emulation mode. In emulation mode 1 or 2, your prototype must be connected to the 8540 via the prototype control probe.

**NOTE**

*You must have SElected the target processor before executing the EM command.*

**CAUTION**

*Be careful when writing to program memory in emulation mode 1: Memory write operations performed in program memory are also performed in user prototype memory, for most emulators.*

**EXAMPLES****Choose Emulator and Emulation Mode**

Select the 8085A emulator and emulation mode 2 with the following command lines:

```
> SEL 8085
> EM 2
```

**Display Current Mode**

To display the current emulation mode, enter:

```
> EM
8085 is in emulation mode 2
```

**SYNTAX**

ex  $\left[ \begin{matrix} -b \\ -w \end{matrix} \right] [-n] \text{ address}$

**PARAMETERS**

- b           Selects a byte-oriented input and output format.
- w           Selects a word-oriented input and output format.
- (default)   The default byte/word input/output format depends on the currently selected emulator. Refer to the Byte/Word Parameter discussion in the Emulator Specifics supplement for your emulator.
- n           Suppresses the read-back check for the command.
- address     An expression representing the address of the data in program/prototype memory that you wish to display or alter. Must be an even value if the -W option is used.

**EXPLANATION**

The EX (EXamine) command permits you to display and (optionally) alter the contents of the specified address, as well as the contents of the subsequent addresses, on the system terminal.

The EX command displays the specified address, an equal sign, the contents of the address, and a space. The contents of each address appear as two hexadecimal digits for byte processors, or four digits for word processors. Table 3-3 lists your options when this information is displayed.

The read-back check compares prototype memory contents after the command is finished to see if the memory was

**Table 3-3**  
**EX Command Options**

Entry	Function
Space Bar	Displays the next address and its contents.
BACKSPACE	Displays the previous address and contents.
LINEFEED or RUBOUT	Displays the current address and contents.
RETURN or CTRL-C	Terminates the command. Memory locations altered before entering CTRL-C or pressing the RETURN key remain altered.
A hexadecimal data string (2 or 4 digits)	Replaces the current data. The next data byte or word then appears on a new line.

properly altered. If there is a difference between memory contents and what was sent to memory, an error is returned. You may want to suppress the read-back check when you know it's going to fail anyway, for example after you've written to prototype I/O. No read-back check is performed on writes to program memory.

The emulation mode and memory map assignments determine whether program or prototype memory is selected. Refer to the EM and MAP command discussions.

**Alternative Commands**

To display blocks of memory contents, use the D command. To modify a sequence of bytes without checking their current contents, use the P command.

**EXAMPLES**

The example shown in Display 3-15 illustrates the various EX command actions that you can take.

```
> EX 500
000500=09 spacebar    Displays the next address and
                          contents on next line.
000501=21 backspace   Displays the previous address and
                          its contents on next line.
000500=09 linefeed    Displays the current address and
                          its contents on next line.
000500=09 01           Replaces current data and displays
                          next address and its contents on next line.
000501=21 backspace   Displays the previous address and
                          its contents on next line.
000500=01 return      Terminates the EX command.
```

Display 3-15.

## SYNTAX

```
f [-b] [-w] [-n] loaddr hiaddr { hexstring }
  [-a string ]
```

## PARAMETERS

-b	Selects byte-oriented filling of memory.
-w	Selects word-oriented filling of memory.
(default)	The default byte/word format depends on the currently selected emulator. Refer to the Byte/Word Parameter discussion in the Emulator Specifics supplement for your emulator processor.
-n	Suppresses the read-back check for the command.
loaddr	An expression representing the lower bound of the address range filled with the hexstring or string. <b>Loaddr</b> must represent an even value if the <b>-W</b> mode is used.
hiaddr	An expression representing the upper bound of the address range filled with the hexstring or string. <b>Hiaddr</b> must be greater than or equal to <b>loaddr</b> , and must represent an odd value if the <b>-W</b> mode is used.
hexstring	An <b>even</b> number of hexadecimal digits that represent a value to be stored in the specified memory area. If the current emulator is a 16-bit processor or the <b>-W</b> parameter is selected, there must be a multiple of four hexadecimal digits; otherwise, there must be a multiple of two hexadecimal digits.
-a	Specifies that the next string in the command line is to be an ASCII string.
string	An ASCII string to be stored in the specified memory area. If the current emulator is a 16-bit processor or the <b>-W</b> parameter is selected, there must be an even number of ASCII characters.

## EXPLANATION

## NOTE

*In this discussion, the term "byte string" refers to the sequence of bytes or words that is the "filler" for the duration of the F command. The byte string is created by concatenating all hexstrings and ASCII strings in the command invocation into a single string.*

The F (Fill) command fills the specified memory area with the specified byte string. The lower memory address receives the first byte or word of the byte string, the next address receives the next byte or word, and so on until the memory area is filled.

If the end of the byte string is reached before the upper limit of the address range is reached, filling continues with the beginning of the byte string. This process continues until all addresses within the specified range have been filled.

When the memory area to be filled is not an exact multiple of the byte string length, filling stops at the specified upper address and a message indicating a truncation error appears on the system terminal. All previous addresses, however, retain their contents (already filled with the byte string).

The read-back check compares prototype memory contents after the command is finished to see if memory was properly altered. If there is a difference between memory contents and what was sent to memory, an error is returned. No read-back check is performed on writes to program memory. You may want to suppress the read-back check when you know it will fail anyway, for example after you've written to prototype I/O.

For 8-bit processors, you must enter each hexstring as a series of hexadecimal pairs. For 16-bit processors, the hexstring must contain a multiple of four hexadecimal digits. For example, you would enter the value B as 0B for an 8-bit processor, and 000B for a 16-bit processor.

The memory area filled with the specified byte string may lie in program memory, user prototype memory, or both, depending on the memory map assignments and the emulation mode. Refer to the MAP and EM command discussions.

## EXAMPLES

## Fill with Zeros

Enter the following command line to fill addresses 0-F with zeros:

```
> F 0 F 0
F: Error 53 - Symbol not found
```

An error occurs because the hexadecimal number was interpreted as a symbol, F, which does not exist. Try again:

```
> F 0 0F 0
F: Error 37 - Invalid hexadecimal string
```

Another error. A hexstring must contain an even number of hexadecimal digits.

```
> F 0 0F 00
```

This time, no errors. Now check the contents of addresses 0-F with the DUMP command line, as shown in Display 3-16.

**Fill with a Hexstring**

To fill memory locations 00-03 with the hexstring ABCD and display the contents, enter the command sequence shown in Display 3-17.

**Fill with an ASCII String**

Fill memory locations 00 to 03 with the string "ABCD", and display the contents, with the command sequence shown in Display 3-18.

Notice that the memory values of the ASCII string are not the same as those of the hexstring.

**Fill with a Long String**

Fill memory locations 00 to 03 with the hexstring 1234567890 and display the contents with the command sequence shown in Display 3-19.

Notice that although the hexstring is truncated, all previous addresses retain their filled contents.

**Fill with Both Hex and ASCII String**

Fill memory locations 00 to 0A with the combination of hexstring ABCD and ASCII string 'AB CD \$EF' and display the contents with the command sequence shown in Display 3-20.

```
> D 0
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Display 3-16.

```
> F 0 3 ABCD ← a hexstring may start with a letter
> D 0
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000000 AB CD AB CD 00 00 00 00 00 00 00 00 00 00 00 .....
```

Display 3-17.

```
> F 0 3 -A ABCD
> D 0
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000000 41 42 43 44 00 00 00 00 00 00 00 00 00 00 00 ABCD.....
```

Display 3-18.

```
> F 0 3 1234567890
F: Error 56 - Truncation error
> D 0
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000000 12 34 56 78 00 00 00 00 00 00 00 00 00 00 00 .4Vx.....
```

Display 3-19.

```
> F 0 0A ABCD -A 'AB CD $EF'
> D 0
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000000 AB CD 41 42 20 43 44 20 24 45 46 00 00 00 00 00 ..AB CD $EF.....
      ↑
hexstring ABCD      ASCII string
                     AB CD $EF
```

Display 3-20.



## SYNTAX

```
g  $\begin{bmatrix} -r \\ -l \end{bmatrix}$  [address]
```

## PARAMETERS

- r** Causes this G command to be reinvoked each time a breakpoint is encountered. A break message is displayed at each break. This continues indefinitely until you enter CTRL-C.
- l** Same as -R, except trace and break lines are suppressed.
- (default)** If neither -R nor -L is specified, the G command stops execution after the first break.
- address** An expression representing the address where program execution begins. Defaults to the current value of the program counter, which is:
1. The address following the last instruction executed, if any; or
  2. The transfer address, if it exists and the program has not already been started; or
  3. Address 0000, if there is no transfer address and the program has not already been started.

Entering the G command without parameters starts the program at the default address, as described above. Execution stops at the first breakpoint.

## EXPLANATION

The G (Go) command starts program execution, using the currently selected emulator.

To interrupt program execution when your program reads from or writes to specified addresses, set breakpoints with the BK command. To suspend program execution, enter CTRL-C.

If -R or -L is used, the G command repeats itself when the breakpoint is encountered. However, each time execution begins again, the emulator registers are **not** reset to the values they held when the G command was originally entered.

For example, with the Z8001/Z8002 emulator, you may begin execution at an address in the System Code memory space. During the execution of the program and before the breakpoint is reached, the memory space being used may switch from System Code to another one, such as Normal Code. The emulator's register contents will have changed to reflect the change in memory space being used, so when the G command repeats according to the -R or -L modifier, execution begins in the wrong memory space (Normal Code instead of System Code.)

## NOTE

*If you have used the BK -C command to specify that execution is to continue after a breakpoint, then the -R or -L modifiers of the G command have no effect. That is, BK -C has precedence over G -R and G -L.*

**EXAMPLES**

Select the 8085A emulator and emulation mode 0 with the following command:

> SEL 8085

Set a breakpoint at a read from address 504 and start program execution at address 100 with the command sequence shown in Display 3-21.

A break occurs when the instruction at location 106 reads data from address 504.

If you specify the -R modifier in the G command line, the G command will be executed repeatedly. See Display 3-22.

```

> BK 1 504 RD
> G 100

LOC  INST  MNEM  OPER      SP  F  A  B  C  D  E  H  L  IM  SOD
000106 86    ADD   M      0000 B0 E0 01 00 00 00 05 04 07  0
000106 <BREAK BKPT1>
    
```

Display 3-21.

```

> G -R 100

LOC  INST  MNEM  OPER      SP  F  A  B  C  D  E  H  L  IM  SOD
000106 86    ADD   M      0000 B0 E0 01 00 00 00 05 04 07  0
000106 <BREAK BKPT1>
000106 86    ADD   M      0000 B0 E0 01 00 00 00 05 04 07  0
000106 <BREAK BKPT1>
000106 86    ADD   M      0000 B0 E0 01 00 00 00 05 04 07  0
000106 <BREAK BKPT1>
    (Enter CTRL-C to stop the
    repetition of G command execution.)
    
```

Display 3-22.

## LO

Loads program into memory

## LOG

Logs terminal input/output to device

Command Dictionary—8540 System Users

### SYNTAX

```
lo [parameters]
or
lo -o offset [parameters]
```

### PARAMETERS

- parameters** One or more values to pass to the program being loaded. The program executes the Get Execution Line Parameter SVC (function code 1C) to obtain these parameters one at a time. Refer to the Service Calls section for a description of the Get Execution Line Parameter SVC.
- o** Specifies load with offset.
- offset** The amount to offset each block of object code from its designated memory location. Also adds an offset to the transfer address, if one is present.

### EXPLANATION

The LO (LOad) command loads a load module from standard input into program/prototype memory. The input should be of the type created by an A-Series or B-Series assembler or linker.

Each block of object code in the file is loaded into program/prototype memory starting at the location

specified in the object code (plus any specified offset). The transfer address (if any) is loaded to the emulator's program counter. If there is no transfer address in the input, the emulator's program counter will be set to zero.

The standard redirection symbol (<) is used to indicate the load file on the host, which can be an object file created by an assembler, or a load file created by the SAV command or by a linker.

The memory map assignments and emulation mode determine whether program memory or prototype memory contents are affected. Refer to the EM and MAP command discussions.

### NOTE

*When you use the LO command, your 8540 must be in TERM mode in order to access files on the host.*

### EXAMPLES

```
$ lo <CALORIES PIZZA BEER
```

Downloads the contents of the 8560 load file CALORIES into memory. The parameters PIZZA and BEER can be accessed through the Get Execution Line Parameter SVC in the program.

```
$ lo -o <PROBLEMS 100 YES NO
```

Downloads the contents of the 8560 load file PROBLEMS into memory. Each block is loaded at the address equal to the location specified in the object code plus 100. The parameters YES and NO can be accessed through the Get Execution Line Parameter SVC in the program.

### SYNTAX

```
log filespec
```

### PARAMETERS

- filespec** The 8540 device that is to receive a copy of terminal input/output.

### EXPLANATION

The log command causes a copy of most terminal input/output to be sent to the specified device or file. The LOG command is not intended for use in TERM mode.

To turn off display to the logging device, enter the LOG CONO command.

### EXAMPLES

```
> LOG LPT
   (sends all terminal input/output to
   both the terminal and the line printer)
:
(OS/40 commands)
:
:
> LOG CONO
   (turns off display to the line printer)
```

## SYNTAX

```
map [-m]
or
map option {loadr [hiaddr]} ...
```

## PARAMETERS

-m	Displays the current memory map assignments in matrix form.
(default)	If -M is not specified, the MAP command displays the current memory map assignments in vertical tabular form.
option	One of the following options:
P	Assigns the specified memory area to program memory. This option does not affect the memory protect status of these blocks.
U	Assigns the specified memory area to user prototype memory. This option does not affect the memory protect status of these blocks.
PRW	Assigns the specified memory area to program memory and sets the memory protect status to read and write.
PRO	Assigns the specified memory area to program memory and sets the memory protect status to read only.
URW	Assigns the specified memory area to prototype memory and sets the memory protect status to read and write.
URO	Same as URW in emulation modes 1 and 2; same as PRO in emulation mode 0. <b>OS/40 does not support write protection for prototype memory.</b>
RO	Assigns the specified memory area to be read only (ROM simulation) when mapped to program memory. This option does not affect the program/prototype assignments of this area.
RW	Assigns the specified memory area to be read/write memory when mapped to program memory. This option does not affect the program/prototype assignments of this area.

loadr An expression representing the lower bound of the address range (starting at the specified address rounded down to a 128-byte multiple) assigned to program or user prototype memory.

hiaddr An expression representing the upper bound of the address range (starting at the specified address rounded up to a 128-byte multiple) assigned to program or user prototype memory. **Hiaddr** must be greater than or equal to **loadr**. Defaults to the end of the 128-byte block that contains the lower address.

When you enter the MAP command without parameters, the current memory map assignments are displayed in tabular form.

## EXPLANATION

## NOTE

*The syntax and use of the MAP command may be different for some emulators (e.g. the 16-bit emulators). Refer to your Emulator Specifics supplement for details.*

The MAP command sets memory map assignments or displays the current memory map status. The memory map is an internal table maintained by OS/40 that indicates which portions of memory used by the emulator are in program memory and which are in user prototype memory. When you SElect an emulator, the map is initialized so that all memory is mapped to program memory (PRW option).

The addressable memory space for most emulators is made up of 512 blocks of 128 bytes each. Refer to the Emulator Specifics supplement for the block size for your emulator. Each block may be assigned to either program or user prototype memory. Each block also has a memory protect status that either allows read and write operations or restricts access to read only. This memory protection feature is implemented only for program memory.

The program/prototype assignments and memory protect status are independent. Therefore, in emulation mode 0 (program memory only), options U, URW, and URO are interpreted as P, PRW, and PRO, respectively. In emulation mode 2 (prototype memory only), options P, PRW, and PRO are interpreted as U, URW, and URW, respectively. Notice that URO is the same as URW in emulation modes 1 and 2. Refer to the EM command for descriptions of the three emulation modes.

If a write access occurs in a block of program memory that you have specified as read-only, an execution break will occur and no write operations will be performed.

You may specify either a single address or an address range. When you enter a single address, one block is mapped. When you specify any portion of a block, the whole block is affected. Refer to the examples.

The MAP command allows you to display the current memory map status (program/prototype memory assignment and memory protect status) in either of two forms. When you include the -M parameter in the command, the display is in matrix form: Each element represents one block of user prototype or program memory. The MAP command entered without parameters displays a vertical table that includes addresses or address ranges.

Before setting memory map assignments, type EM 1 to enter emulation mode 1. Otherwise, the system terminal displays the message, WARNING—NOT IN EMULATOR MODE 1. (The settings still take effect, however.)

**NOTE**

*You must have SElected the target processor before executing the MAP command.*



*In emulation mode 1, memory write operations performed in program memory are also performed in the corresponding bytes of user prototype memory for most emulators.*

**EXAMPLES**

**Display Mapped Memory**

When you SElect an emulator, the memory map is initialized so that all memory is mapped PRW. Enter the following command to display the memory map assignments:

```
> MAP
0000-FFFF PRW
```

**Set Memory Map Assignments**

Select emulation mode 1 before setting memory map assignments with the following command line.

```
> EM 1
```

Now you may set the memory map assignments. Notice that when you specify any portion of a block, the whole block is affected.

```
> MAP U 735 17A0 80 100
      0700      0080
      to       to
      17FF      017F
```

The preceding command line assigns memory within address ranges 0700-17FF and 0080-017F to user prototype memory.

```
> MAP PRO 0 55 5000,,5200 617F
```

The preceding command line assigns memory within address ranges 0000-007F, 5000-507F, and 5200-617F to program memory, and sets the memory protect status to read only.

```
> MAP URO 0A000 0BFFF
```

The preceding command line assigns memory within address range A000-BFFF to user prototype memory, and sets the memory protect status to read-only.

Since OS/40 does not support write protection for prototype memory, this address range is write-protected only in emulation mode 0, when all memory is automatically mapped to program memory.

**Display Map Assignments**

Display the resulting memory map assignments in a vertical table, as shown in Display 3-23.

Display the same memory map assignments in matrix form, as shown in Display 3-24.

```
> MAP
0000-007F PRO
0080-017F URW
0180-06FF PRW
0700-17FF URW
1800-4FFF PRW
5000-507F PRO
5080-51FF PRW
5200-617F PRO
6180-9FFF PRW
A000-BFFF URO
C000-FFFF PRW
```

Display 3-23.

```
> MAP -M
PRW(-), PRO(=), URW(*), URO(.)
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 == * - - - - - - - - - - ** ** ** ** ** ** ** ** ** ** ** ** *
1000 ** ** ** ** ** * ** ** * ** ** * - - - - - - - - - -
2000 - - - - - - - - - - - - - - - - - - - - - -
3000 - - - - - - - - - - - - - - - - - - - - - -
4000 - - - - - - - - - - - - - - - - - - - - - -
5000 = - - - = = = = = = = = = = = = = = = = = =
6000 = = - - - - - - - - - - - - - - - - - - - -
7000 - - - - - - - - - - - - - - - - - - - - - -
8000 - - - - - - - - - - - - - - - - - - - - - -
9000 - - - - - - - - - - - - - - - - - - - - - -
A000 . . . . . . . . . . . . . . . . . . . . . .
B000 . . . . . . . . . . . . . . . . . . . . . .
C000 - - - - - - - - - - - - - - - - - - - - - -
D000 - - - - - - - - - - - - - - - - - - - - - -
E000 - - - - - - - - - - - - - - - - - - - - - -
F000 - - - - - - - - - - - - - - - - - - - - - -
```

Display 3-24.

# MEM

Specifies memory available to emulator

## SYNTAX

```
mem [loadr [hiaddr]]
```

## PARAMETERS

**loadr** An expression representing the beginning of the block of memory the emulator is allowed to access.

**hiaddr** An expression representing the end of a block of memory the emulator is allowed to access.

When you enter the MEM command with no parameters, the current MEM status is displayed. If your emulator has separate memory spaces, there will be a separate display for each address space.

## EXPLANATION

### NOTE

*Most emulators do not support the MEM command. Most emulators that do support the MEM command require the MAC option. Refer to your Emulator Specifics supplement for this information.*

The MEM (MEMory) command is used to reverse the effects of a previous NOMEM command. The NOMEM command designates prototype memory as nonexistent. The MEM command designates prototype memory as existent. The default condition is for the entire range of prototype memory to exist.

If your program tries to access memory that you have declared nonexistent, the system issues a break.

## EXAMPLES

### Designate Existent Memory

```
> MEM 2000 2FFF
```

This command designates user prototype memory addresses 2000–2FFF to be available to the emulator.

```
> MEM SC:2000
```

This command designates Z8001/Z8002 System Code addresses 2000 to the end of the block to be available to the emulator.

```
> MEM NS:2400
```

This command designates Z8001/Z8002 Normal Stack addresses 2400 to the end of the block to be available to the emulator.

**SYNTAX**

```
memsp [ s memspace
       m memspace ... ]
```

**PARAMETERS**

- s Specifies the default memory space for single-memory-space commands (see list below).
- m Specifies the default memory spaces for multiple-memory-space commands (see list below).
- memorspace A two-character symbol that represents a specific memory space. Your Emulator Specifics supplement lists the correct memory space symbols for your emulator, if applicable. Default address space symbols are emulator-dependent.

Entering the MEMSP command without parameters displays the currently set default memory spaces.

**EXPLANATION**

**NOTE**

*For most emulators, the MEMSP command is not applicable. In addition, you may need the MAC option if you wish to use memory space designators when referring to program memory. Refer to your Emulator Specifics supplement.*

The MEMSP (MEMory SPace) command specifies which memory space will be used when no memory space is specified in a command line. After the default spaces have been defined, you can enter commands without having to specify an address space with each address expression. The MEMSP command is only applicable for those emulators with memory spaces.

Commands that accept a single memory space in the address expression use the default set by the MEMSP S command. The following commands are influenced by the MEMSP S command:

COM	DI	MOV	RH	SEA	WPR
CPR	EX	P	RPR	SVC	WRT
D	F	RD	SAV	WH	

Commands that accept multiple memory spaces in the address expression(s) use the default set by the MEMSP M command. The following commands are influenced by the MEMSP M command:

AL	MEM
BK	NOMEN
MAP	

The MEMSP default setting may be overridden if a command would otherwise perform a meaningless operation on the memory space; for example, trying to disassemble a stack (DI command).

**EXAMPLES**

```
> MEMSP S UD
> D 400 4FF
```

The MEMSP command selects User Data as the default 68000 memory space for single-memory-space commands. The D command dumps the contents of User Data addresses 400-4FF.

```
> MEMSP M ND NS NC
> BK 1 2000
```

The MEMSP command selects Normal Data, Normal Stack, and Normal Code memory spaces for multiple-memory-space commands for the Z8001/Z8002 emulator. The BK command sets a breakpoint for any access to location 2000 in the Normal Data, Normal Stack, or Normal Code memory spaces.

```
> MEMSP
Default single memory space.....ND
Default multiple memory spaces...ND NS NC
```

Displays current memory space defaults.



<b>SYNTAX</b>	
$\left. \begin{matrix} uu \\ up \\ pu \\ pp \end{matrix} \right\}$	<code>mov loaddr hiaddr destaddr</code>

**EXPLANATION**

The MOV command copies the specified data block from either program or user prototype memory to a new location in either memory.

If an error occurs when you try to move data to RAM which exists in user prototype memory, your prototype control probe may be malfunctioning.

The MOV command is not affected by the current emulation mode, memory map assignments, and write protection settings.

**PARAMETERS**

- uu Both data source and data destination are user prototype memory.
- up Data source is user prototype memory. Data destination is program memory.
- pu Data source is program memory. Data destination is user prototype memory.
- pp Both data source and data destination are program memory.
- loaddr An expression representing the lower address of the data block copied.
- hiaddr An expression representing the upper address of the data block copied. **Hiaddr** must be greater than or equal to **loaddr**.
- destaddr An expression representing the lower address of the memory area to which the data block is copied.

**EXAMPLES**

Display the contents of program memory addresses 100-10E as shown in Display 3-25.

Copy the contents of program memory locations 100-10E to program memory addresses 300-30E. Then display the contents of memory addresses 300 through 30E. Refer to Display 3-26.

The contents of memory addresses 100-10E remain unchanged.

It is possible to copy the specified data block to memory locations within that same block. Refer to Display 3-27.

The contents of program memory 2FF-303 are overwritten by the old contents of program memory 300-304.

```
> D 100 10E
  0 1 2 3 4 5 6 7 8 9 A B C D E F
000100 21 00 05 06 05 AF 86 23 05 C2 06 01 D3 F7 00 !.....#.....
```

**Display 3-25.**

```
> MOV PP 100 10E 300
> D 300 30E
  0 1 2 3 4 5 6 7 8 9 A B C D E F
000300 21 00 05 06 05 AF 86 23 05 C2 06 01 D3 F7 00 !.....#.....
```

**Display 3-26.**

```
> MOV PP 300 304 2FF
> D 2F0 30E
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0002F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 21 !.....!
000300 00 05 06 05 05 AF 86 23 05 C2 06 01 D3 F7 00 !.....#.....
```

**Display 3-27.**

**SYNTAX**

```
nomem [loadr [hiaddr]]
```

**PARAMETERS**

- loadr** An expression representing the beginning of a block of memory the emulator is not allowed to access. May include memory space designators.
- hiaddr** An expression representing the end of a block of memory the emulator is not allowed to access.

Entering the **NOMEM** command with no parameters displays the list of memory blocks which are currently not available. There is a separate display for each memory space.

**EXPLANATION****NOTE**

*Most emulators do not support the **NOMEM** command. Most emulators that do support the **NOMEM** command require the **MAC** option. Refer to your Emulator Specifics supplement for this information.*

The **NOMEM** (NO MEMory) command designates 4K-byte blocks of prototype memory as nonexistent. The default condition is for all prototype memory to exist.

If the emulator tries to access nonexistent memory, the system generates a break.

The **MEM** command is used to reverse the effects of the **NOMEM** command.

**EXAMPLES**

```
> NOMEM 2000 2FFF
```

Designates user prototype memory addresses 2000–2FFF as nonexistent.

```
> NOMEM SC:2000
```

Designates Z8001/Z8002 system code memory address 2000 to the end of the block as nonexistent.

```
> NOMEM NS:0
```

Designates Z8001/Z8002 normal stack memory address 0 to the end of that block as nonexistent.

**SYNTAX**

```
p [-b] [-w] [-n] address {hexstring} [-a string] ...
```

**PARAMETERS**

- b           Selects byte-oriented processing.
- w           Selects word-oriented processing.
- (default)   The default byte/word format depends on the currently selected emulator. Refer to the Byte/Word Parameter discussion in the Emulator Specifics supplement for your emulator.
- n           Suppresses the read-back check for the command.
- address     An expression representing the address where code alteration begins. If word mode is chosen (-W), the **address** must represent an even value.
- hexstring   An even number of hexadecimal digits: the sequence of bytes to be patched into memory. If -W is selected, or the current emulator is a 16-bit processor, the **hexstring** must be a multiple of 4 hexadecimal digits; otherwise, it must be a multiple of 2 hexadecimal digits.
- a           Specifies that the next string in the command line is to be an ASCII string.
- string      A string of ASCII characters that is patched into memory. If -W is selected, or the current emulator is a 16-bit processor, there must be an even number of ASCII characters.

**EXPLANATION**

The P (Patch) command replaces the memory contents beginning at the specified address with the byte sequence represented by the ASCII string(s) and/or hexstring(s).

For byte-oriented processing, a hexstring must contain an even number of bytes. For word-oriented processing, the number of bytes in a hexstring must be a multiple of 4, and an ASCII string must have an even number of characters.

You may enter a combination of hexstrings and ASCII strings in one P command entry.

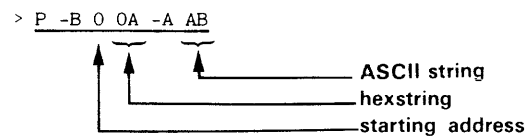
The read-back check compares prototype memory contents after the command is finished to see if memory was properly altered. If there is a difference between memory contents and what was sent to memory, an error is returned. You may want to suppress the read-back check when you know it will fail anyway, for example after you've written to prototype I/O. No read-back check is performed on writes to program memory.

The previously set memory map assignments and emulation mode determine whether program memory or prototype memory contents are affected. Refer to the MAP and EM command discussions.

**EXAMPLES**

**Patch with Hexstring and ASCII string**

Enter the following command line to replace the data starting at address 0 with the hexstring 0A and ASCII string AB:



Display the contents of addresses 0-0F, as shown in Display 3-28.

```
> D 0
000000 0A 41 42 00 00 00 00 00 00 00 00 00 00 00 00 .AB.....
          ^   ^
          |   |
hexstring 0A  ASCII string AB
```

Display 3-28.

**SYNTAX**

```
permstr -l
or
permstr -d [stringname ...]
or
permstr stringname ...
```

**PARAMETERS**

-l Lists the currently defined permanent user strings.

-d Deletes the specified permanent string(s). If no stringname is given, all currently defined permanent strings will be deleted.

stringname The user-assigned name for the string.

**EXPLANATION**

The PERMSTR command allows you to store strings (which have been previously defined) in EEPROM, to display the strings stored in EEPROM, and to delete one or more of these strings. For information on how to declare and use strings, refer to the discussion on "Strings" near the beginning of this Command Dictionary.

An error occurs if any attempt to create more permanent strings when sufficient space is not available on EEPROM. Use the PERMSTR -D command to delete unwanted strings. A permanent string must be deleted before its value can be redefined.

Permanent strings may compete for space in EEPROM with operating system patches created using the ROMPATCH command. Both EEPROM chips must be installed in order to use the PERMSTR command.

**EXAMPLES**

First, declare two temporary strings:

```
> reg set="S R1=0 R2=2 R3=5"
> range1="1000,2000"
```

You can then list these temporary strings with the STR command:

```
> STR -L
regset S R1=0 R2=2 R3=5
range1 1000,2000
```

Now store the strings in EEPROM with PERMSTR, and list them:

```
> PERMSTR reg set range1
> PERMSTR -L
regset S R1=0 R2=2 R3=5
range1 1000,2000
```

Delete one of the permanent strings, and list them again:

```
> PERMSTR -D reg set
> PERMSTR -L
range1 1000,2000
```

Your temporary string definitions still exist, though, until you delete them with the STR -D command, or turn off the power to the 8540:

```
> STR -L
regset S R1=0 R2=2 R3=5
range1 1000,2000
```

# QUERY

Turns query mode on and off

## SYNTAX

```
query [ on ]  
      [ off ]
```

## PARAMETERS

on            Turns query mode on.  
off           Turns query mode off.

When you enter the QUERY command without parameters, the current system-wide query status is displayed.

## EXPLANATION

The QUERY command turns query mode on or off. When query mode is on, commands that have a -Q option will query. When the mode is off, those commands will query only if the -Q option is invoked. Query mode is initially on when you reboot or start up the system.

## EXAMPLES

After starting up the system, check the query mode setting by entering the command without parameters:

```
> QUERY  
ON
```

Now remove the symbol SYMBOL1 using the REMS command:

```
> REMS SYMBOL1  
Remove (scalar):SYMBOL1                    ?Y
```

The REMS command asks for confirmation before removing the symbol. You type "Y" for yes. To turn query mode off, enter:

```
> QUERY OFF
```

Now use the REMS command again to remove SYMBOL2. It will remove the symbol without asking for confirmation.

```
> REMS SYMBOL2
```

To turn query mode back on, enter:

```
> QUERY ON
```

## SYNTAX

```
rd [-s] [-b] [-m] [-w] portnum ...
```

## PARAMETERS

- s** Specifies special read as opposed to normal read. Not valid with **-M** modifier. Refer to your Emulator Specifics supplement to see whether your emulator can use this modifier.
- m** Specifies that the value of **portnum** is a memory address (meaningful for memory-mapped I/O). This is the default for emulators that only have memory-mapped I/O. Refer to your Emulator Specifics supplement to see whether you can use the **-M** modifier.
- If this modifier is omitted, the **portnum** parameter is assumed to be a fixed port. The emulators that support both memory-mapped I/O and fixed-port I/O default to a fixed-port read.
- b** Specifies byte-oriented reading. Default value.
- w** Specifies word-oriented reading.
- portnum** If **-M** is not used: an expression designating a fixed I/O port. Refer to your Emulator Specifics supplement for the correct **portnum** values for your emulator.

If **-M** is used: an expression designating a memory location. The expression may include only one memory space designator.

## EXPLANATION

The RD (ReaD) command reads a byte or a word from an I/O port on the emulator. This command is not available for most emulators. Refer to your Emulator Specifics supplement for details.

If more than one **portnum** value is entered, reads will be performed and results displayed in the order the **portnum** parameters were entered.

## EXAMPLES

To read from Z8001/Z8002 emulator ports 70, 73, and 7A, and display the values, enter the following command:

```
> RD 70 73 7A
0070=23
0073=00
007A=1F
```

To perform a special byte-oriented read from Z8001/Z8002 port symbolized by the symbol **TERMACIA**, and display the value, enter the following command:

```
> RD -S TERMACIA
00F0=23
```

To read from the Z8001/Z8002 memory address symbolized by **TERMACIA** in normal data memory space, enter the following command:

```
> RD -MW ND:TERMACIA
ND:0000F0=0023
```

## SYNTAX

```
rems [-n] [-q] {sectionname:* } symbolspec ...
```

## PARAMETERS

- q Turns query mode on for the duration of this command.
- n Turns query mode off for the duration of this command.
- (default) The default query mode depends on the current system-wide query status. Refer to the QUERY command.
- sectionname:\* Represents all the symbols within the specified section.
- symbolspec The symbol to be removed from the symbol table. The **symbolspec** may include the **sectionname** if the symbol represents an address.

## EXPLANATION

The REMS (REMove Symbol) command removes the specified symbol(s) from the symbol table.

If only the second part of the symbol is included in the symbolspec parameter (no section name), the REMS command searches for the symbol first in the current section, then in the entire symbol table, and removes only the first occurrence of that symbol. Refer to the Emulation section of this manual for more information on symbolic debug.

When query mode is on, the REMS command queries you before removing the symbol.

## EXAMPLES

```
> REMS ENDLIST
```

This command line removes symbol ENDLIST from the symbol table. No section name is specified.

The following command removes symbols APPLAUSE and KISSBABY, associated with sections SPEECH and ELECTION, respectively, from the symbol table. Both symbol-specs include both a section (the name before the colon) and a symbol which represents the address value (the name after the colon).

```
> REMS SPEECH:APPLAUSE ELECTION:KISSBABY
```

The following command removes all the symbols from the section WRITERS.

```
> REMS WRITERS:*
```

**SYNTAX**

reset

*NOTE*

*You must have SElected the target processor before executing the RESET command.*

**EXPLANATION**

The RESET command simulates a hardware reset by reinitializing the values of the emulator registers.

The RESET command is emulator-dependent. Refer to the Emulator Specifics section for details on how the RESET command affects specific registers in your processor.

**EXAMPLES**

Check the register contents of an 8085A, enter the RESET command, and check the registers again, as shown in Display 3-29.

```

~ DS
PC=0111 SP=5500 F=65 A=04 B=49 C=88 D=66 E=87 H=01 L=22
SOD=1 SID=0 I7=1 I6=1 I5=1 IE=1 M7=0 M6=0 M5=0

~ RESET

> DS
↓
PC=0000 SP=5500 F=65 A=04 B=49 C=88 D=66 E=87 H=01 L=22
SOD=0 SID=0 I7=0 I6=0 I5=0 IE=0 M7=1 M6=1 M5=1
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

```

(The arrows show the changed registers.)

Display 3-29.



## SYNTAX

```
rh [ i
    m
    t ] [offset]
```

## PARAMETERS

-i	Input is Intel-formatted hexadecimal object code.
-m	Input is Motorola-formatted hexadecimal object code.
-t	Input is Standard Tekhex format.
(default)	If no modifier is specified, input is Extended Tekhex format.
offset	An expression representing the amount to offset the load address of each data block. Defaults to 0.

When you enter the RH command without parameters, Extended-Tekhex-formatted object code is read from the standard input into program/prototype memory without an offset.

## EXPLANATION

The RH (Read Hexadecimal code) command reads hexadecimal object code from standard input, converts the code to binary, and loads the binary code into program or prototype memory.

Each message block read by the RH command must be in the specified load format. Refer to the Tables section for a discussion of the Tekhex, Intel, and Motorola formats.

The RH command will not read or load any program symbols. Refer to the ADDS, SYMLO and COM commands for information on loading symbols into the symbol table.

Each block of code is loaded into memory starting at the message block load address. Any offset amount is added to the load address for each data block transferred.

The emulation mode and the memory map assignments determine whether program memory or prototype memory contents are affected. Refer to the EM and MAP command discussions.

To write rather than read hexadecimal object code, use the WH command.

## NOTE

*When you use the RH command, your 8540 must be in TERM mode in order to access files on the host.*

## EXAMPLES

```
· rh · PROG.THEX
```

Reads Extended Tekhex formatted object code from the 8560 file PROG.THEX, converts it to binary, and writes the object code into memory.

```
· rh -i · PROG.IHEX 100
```

Reads the Intel-formatted hexadecimal object code from the 8560 file PROG.IHEX and converts it to binary. Adds an offset amount of 100 to each load address. For example, when the load address specified by the message block is 100, the data block is loaded at location 200. Specifying the offset parameter does not affect the transfer address, if one is given.

**SYNTAX**

```
rompatch checksum patchid patchaddress filespec hexstring
or
rompatch -d checksum patchid
or
rompatch -l
```

**CAUTION**

*The ROMPATCH command is intended to be used to update your operating system. Do not use this command unless directed to do so by Tektronix. Erroneous entries may invalidate your system. Certain options of the ROMPATCH command are not described in this manual.*

**PARAMETERS**

-d	Deletes the last defined patch.
-l	Lists all the patches in EEPROM.
checksum	Helps insure that the patch has been entered correctly. The ROMPATCH command calculates a checksum value for the characters of all parameters in the ROMPATCH command line except the <b>checksum</b> parameter. The resulting value is then compared with the <b>checksum</b> parameter. If they are not the same, an error occurs and the command aborts.
patchid	The sequence number of the patch being entered or deleted. This parameter is compared to the last <b>patchid</b> number saved in EEPROM. If it is not one greater than the stored value, then an error occurs and the command aborts.
patchaddress	The address where the patch will be made relative to the load address of the file named.

filespec	The name of the ROM file (or command) to be patched. The <b>filespec</b> must contain the ROM file level as well as the filename. (For example, in "/DEFLT/EX", "/DEFLT" is the ROM file level and "/EX" is the filename.) If the ROM file is not found, an error occurs and the command aborts.
hexstring	The data to be patched into memory: an even number of hex digits.

**EXPLANATION**

The 8540 contains two EEPROM chips on which updates to the operating system can be stored using the ROMPATCH command. These updates are then inserted in the command modules when they are loaded. Normally, the user will not invoke this command except when directed to do so by Tektronix. Usually this will be done via explicit written instructions.

The ROMPATCH command performs several checks on your input to guard against accidental or intentional erroneous entries.

ROM patches may compete for space in EEPROM with permanent strings created using the PERMSTR command. It may be necessary to delete some strings in order to enter a new ROM patch.

**EXAMPLES****Adding a Patch**

The following command line causes the hexstring 3FC24A to be entered as the 26th patch in the EEPROM. It patches the module /DEFLT/EX at address 1124 relative to the load address of the module.

```
ROMPATCH 31 26 1124 /DEFLT/EX 3FC24A
```

**Deleting a Patch**

This command line deletes the 157th patch (which must be the last entered) from EEPROM.

```
ROMPATCH -D 3C 158
```

## Listing Patches

You can use the -L modifier to list all the patches currently in EEPROM. In the example shown in Display 3-30,

- PATCH ID corresponds to "patchid" in the syntax block,
- ROM FILE NAME corresponds to "filespec",

- PART NUMBER is the Tektronix part number excluding the 020 prefix,
- ADDRESS corresponds to "patchaddress", and
- PATCH corresponds to "hexstring".

```
> ROMPATCH -L
PATCH ID  ROM FILE NAME  PART NUMBER  ADDRESS  PATCH
1          /KERNL/K1          238501      45A8     878DAEDF9FB50F98C1BD
2          /DEFLI/ROMPATCH  239700      C240     404040B3B405D2EF217A7B204376
          234BD7C7
3          /8085 /DS          253200      0024     6C725A7183
4          /Z8001/BK         318402      28C4     9C02DEF6A8BD2A4C2CC03065
```

Display 3-30.

### SYNTAX

s symbolspec=expression ...

### PARAMETERS

**symbolspec** The name of the symbol or register that receives a new value. May be a standard register name or a user-created symbol that is already in the symbol table.

**expression** Any valid expression as defined in the "Legal Address Expressions" discussion in this section.

### EXPLANATION

The S (Set) command assigns a new value to each register or symbol specified. Refer to the Emulator Specifics section of this manual for the specific registers associated with your emulator.

### EXAMPLES

#### Assign Value to Register

Select the 8085A emulator and display the contents of the emulator registers, as shown in Display 3-31.

Reassign values to registers SP, F, and A with the following command:

```
> S SP=0FFFF F=11 A=B+5
```

Display the contents of the 8085A emulator registers again. See Display 3-32.

#### Assign Value to Symbol

Symbolic debug allows you to declare a symbol for a value using the ADDS command. This name can then be used in the S command:

```
> ADDS SUM=1224H
```

```
.
```

```
> S SUM=5
```

Display the value of SUM with the following command:

```
> CALC SUM
5H
```

```
> SEL 8085
> DS
PC=0000 SP=0000 F=00 A=00 B=00 C=00 D=00 E=00 H=00 L=00
SOD=0 SID=0 I7=0 I6=0 I5=0 IE=0 M7=1 M6=1 M5=1
```

Display 3-31.

```
> DS
      ↓   ↓   ↓
PC=0000 SP=FFFF F=11 A=05 B=00 C=00 D=00 E=00 H=00 L=00
SOD=0 SID=0 I7=0 I6=0 I5=0 IE=0 M7=1 M6=1 M5=1
```

(The arrows show the changed registers.)

Display 3-32.

## SYNTAX

```

sav  $\begin{bmatrix} -s \\ -l \end{bmatrix}$  ;loaddr hiaddr} ... [transfer]
    
```

### PARAMETERS

- s            Specifies A-series (small address space) format.
- l            Specifies B-series (large address space) format.
- (default)   Defaults to B-series format if the selected target processor can address more than 64K of memory; otherwise, defaults to A-series format.
- loaddr       An expression representing the lower address of a block of program/prototype memory to be transferred to the file.
- hiaddr       An expression representing the upper address of a block of program/prototype memory to be transferred to the file. **Hiaddr** must be greater than or equal to **loaddr**.

transfer     An expression representing the address of the first instruction to be executed in the program. Defaults to the first **loaddr**.

### EXPLANATION

The SAV command directs the specified contents of program/prototype memory to standard output in A Series or B Series load module format, suitable for loading with the LO or X commands.

The memory map assignments and emulation mode determine whether program memory or prototype memory contents are affected. Refer to the MAP and EM command discussions.

#### NOTE

*Your 8540 must be in TERM mode in order to create files on the host.*

### EXAMPLE

Display the contents of addresses 100-10E on the system terminal. See Display 3-33.

Enter the following command to upload the contents of addresses 100-10E into the 8560 load file named LOAD.

```
> sav -l >LOAD 100 10E
```

```

> D 100 10E
   0 1 2 3 4 5 6 7 8 9 A B C D E F
000100 21 00 05 06 05 AF 86 23 05 C2 06 01 D3 F7 00  !.....#.....
    
```

**Display 3-33.**

**SYNTAX**

```
sea [-b] [-w] [-r] loadr [hiaddr] { value [precision] }
                                { -a string }
```

**PARAMETERS**

- b           Selects a byte-oriented search.
- w           Selects a word-oriented search, and requires an even starting address, and an odd ending address.
- (default)   The default byte/word format depends on the currently selected emulator. Refer to the Byte/Word Parameter discussion in the Emulator Specifics supplement for your emulator.
- r           Specifies to search for repeated occurrences of the specified string.
- (default)   If -R is not specified, the SEA command stops searching after one match is found.
- loadr       An expression representing the address in program/prototype memory where searching is to start. Must be even if the -W mode is used.
- hiaddr      An expression representing the address in program/prototype memory where searching is to stop. **Hiaddr** must be greater than **loadr**, and odd if the -W mode is used. Defaults to the highest address accessible by the target processor.
- value       An expression that represents a value for which you want to search. Up to four bytes of precision are possible.

- precision   The number of bytes required to match the specified search **value**. This number must be between 1 and 4. Defaults to 1 for byte mode, 2 for word mode.
- a           Specifies a search for an ASCII string. Must be followed by the target string.
- string      The ASCII string for which you are searching. The maximum length of the string is limited only by the maximum length of the command line.

**EXPLANATION**

The SEA command searches the specified memory area for the given value or string. SEA will display the starting address of the first occurrence of the value or string; or of every occurrence if the repetitive search modifier (-R) is chosen.

The memory map assignments and emulation mode determine whether program memory or prototype memory contents are affected. Refer to the MAP and EM command discussions.

**EXAMPLES**

Assume that addresses 0-0F contain the data shown in Display 3-34.

Display 3-35 shows some examples of searching for the value 34 with different **precision** parameters:

Search for the ASCII string "45" with the following command:

```
> SEA 0 0F -A 45
000DH
```

The starting address of the first occurrence of ASCII 45 (hexadecimal 3435) is 000D.

```
> D 0
0 1 2 3 4 5 6 7 8 9 A B C D E F
000000 34 00 34 00 00 34 00 00 00 34 00 00 00 34 35 00 4 4 . . 4 . . 4 . . 4 5 .
```

Display 3-34.

```
> SEA 0 0F 34           (Default number of bytes of precision is 1)
0000H           (First occurrence of 34 is at 0000H)
> SEA 0 0F 34 2       (Search for the 2-byte value 0034)
0001H
> SEA 0 0F 34 3       (3 bytes of precision:000034)
0003H
> SEA 0 0F 34 4       (4 bytes of precision: 00000034)
0006H
```

Display 3-35.

# SEL

Selects target processor chip

## SYNTAX

```
sel [chip]
```

## PARAMETERS

chip           The name of the target processor.

When you enter the SEL command without parameters, the currently selected processor name is displayed.

## EXPLANATION

The SEL command specifies the command set for your processor. After you select a processor, all emulator commands will be appropriate for that specific processor. When the 8560 is host, SEL also causes the 8560 to select the appropriate assembler and high level language commands for the desired microprocessor.

Entering a SEL command with a chip parameter closes any channels assigned to your program. Also, when you enter the SEL command with a chip parameter, the symbol table is cleared.

The SEL command selects the target processor. The system then acts as if the following commands had been executed:

```
AL 0 10 0           (for most emulators)
BK ALL CLR
CLOCK OFF
EM 0
MAP PRW 0000 FFFF   (for most emulators)
RESET
SVC ON 40 OF0       (for most emulators)
SYMD -LS ON; SYMD OFF
TCLR -X
TRA OFF
```

You should use the SEL command to select a target processor before you execute any of the following commands:

ACQ	CONS	MAP
AD	COU	PRO
AL	CTR	QUA
AS	DATA	RESET
BK	DI	SVC
BRE	DISP	TCLR
BUS	DS	TRA
CL	EM	TS
CLOCK	EVE	

At system start up, the command set is selected from the command group called DEFLT.

## EXAMPLES

Display the currently selected processor with the following command:

```
> SEL
no emulator
```

Now, select the processor:

```
> SEL 8085
```

Display the currently selected processor again:

```
> SEL
8085
```

**SYNTAX****stat****EXPLANATION**

The STAT command displays the status of the emulator on standard output. The status information displayed includes:

- name of the selected processor
- name of the last program loaded into program memory
- I/O channel assignments

**EXAMPLES**

Display 3-36 is an example of STAT command output.

```
> stat
8085 is the selected processor

User program was last loaded into program memory

Channel 0 is assigned to DATAFILE
```

Display 3-36.

**SYNTAX**

```
str -l
or
str -d stringname ...
```

**PARAMETERS**

- l Lists the currently defined temporary user strings.
- d Deletes the specified string(s). If no string-name is specified, all currently defined temporary strings will be deleted.
- stringname The user-assigned name for the string.

**EXPLANATION**

The STR command allows you to display all the currently defined temporary user strings or to delete one or more of

these strings. For information on how to declare and use strings, see the discussion of "Strings" near the beginning of this Command Dictionary.

**EXAMPLES**

First, declare two strings:

```
> regset="S R1=0 R2=2 R3=5"
> range1="1000,2000"
```

List your currently defined strings with the following command:

```
> STR -L
regset S R1=0 R2=2 R3=5
range1 1000,2000
```

Now delete the first string, and list the strings again with the following sequence of commands:

```
> STR -D regset
> STR -L
range1 1000,2000
```



# SUSP

Suspends commands

## SYNTAX

```
susp { command }  
      { -a }
```

## PARAMETERS

- command    The name of a OS/40 command: any command listed in this Command Dictionary.
- a         All active commands.

## EXPLANATION

The SUSP command suspends execution of the specified command. To suspend a particular command, first press CTRL-C and then enter the SUSP command with the name of the command. The command remains suspended until you abort it with the A command, or continue it with the CO command. To suspend all active processes, enter SUSP -A.

Refer to the A command explanation for a list of commands that may not be suspended.

**SYNTAX**

`svc [ on ] [ address ] [ port ]`

**PARAMETERS**

- on            Turns on user SVCs.
- off           Turns off user SVCs.
- (default)    If neither ON nor OFF is entered, the current SVC ON/OFF status remains unchanged.
- address      An expression specifying where in memory the SRB vector will begin. The default value (usually 40H) is emulator-dependent. Refer to your Emulator Specifics supplement for details.  
  
If the **address** parameter is omitted, the location of the SRB vector remains unchanged.
- port          An expression representing the lowest I/O port address to be used by your program to initiate service calls. The I/O port range defaults to F0-F7. The **port** parameter should be a multiple of 10H, thus specifying a range of the form n0-n7, where n is a hexadecimal number between 0 and F. If the **port** parameter is omitted, the I/O port range remains unchanged.

When you enter the SVC command without parameters, the current SVC capability (ON or OFF), SRB vector address, and SVC I/O port range are displayed.

**EXPLANATION**

*NOTE*

*SVC command default values may be different for some emulators. Refer to your Emulator Specifics supplement for details.*

The SVC (SerVice Calls) command turns the SVC capability on and off in all three emulation modes. Refer to the Service Calls section of this manual for an explanation of SVCs.

Entering emulation mode 0 automatically turns SVCs on. Entering emulation modes 1 or 2 automatically turns SVCs off.

If SVCs are ON, any I/O instruction in your program that can be interpreted as an SVC request will initiate the appropriate SVC. If SVCs are OFF, such I/O instructions will be treated as normal code by the emulator, and no SVC request will be generated.

You may also use the SVC command to specify where the SRB vector will begin and which I/O ports can be used to initiate SVCs.

Initial SVC settings may be restored at any time by executing the following command:

```
> SVC ON 40 OF0
```

*NOTE*

*You must have SElected the target processor before executing the SVC command.*

*Certain SVC features, such as the parameters of the SVC command and the capacity for SVCs in modes 1 and 2, are emulator-dependent. Refer to the Emulator Specifics section of this manual for this information.*

**EXAMPLES**

Enter the following command to display the current SVC settings:

```
> SVC
User SVCs are ON .
The SRB vector is 0040.
The I/O address is F0.
```

Change the location of the SRB vector to 200 and the SVC I/O port range to 0-7 with the following command:

```
> SVC, ,200 0
```

Display the SVC setting again:

```
> SVC
User SVCs are ON .
The SRB vector is 0200.
The I/O address is 00.
```

# SYMB

Returns symbolic name for address

## SYNTAX

`symb` expression

The format of the returned expression is influenced by the `-S` and `-L` options previously specified with the `SYMD` command. `SYMB` temporarily turns `SYMD ON` when returning the expression.

## PARAMETERS

expression    An expression whose symbolic equivalent you want to display, if one exists.

## EXPLANATION

Given an expression, `SYMB` returns an equivalent symbolic expression of the form "addresssymbol" or "section + offset", where "offset" is the displacement of the address relative to the start of the section. Symbols representing scalar values will not appear in the returned symbolic expression.

To do the opposite, that is, to find the value associated with a symbol, use the `CALC` command.

## EXAMPLES

If there is an symbol which represents an address and matches the specified expression, and the `SYMD -SL` options are enabled, the `SYMB` command will display the symbol as shown in the following command line:

```
> SYMB 1234H  
1234H=LABEL001
```

If no address symbol corresponds to the specified expression, the system responds with the section name and byte offset of the given value:

```
> SYMB 132AH  
132AH=SECTION3+00AH
```

**SYNTAX**

symd off  
or  
symd [-s] [-l] on

**PARAMETERS**

- off           Turns off symbolic debug output.
- on            Turns on symbolic debug output.
- l            Enables address symbol (label) substitutions only. Turns off offset substitution.
- s            Enables the substitution (for each hexadecimal address) of its offset relative to the start of its section (shown in the location counter field of your assembler listing). Turns off address symbol substitution.
- sl           Enables the substitution of both symbols representing addresses and section names with offset.

If SYMD ON is entered without modifiers, SYMD is enabled with the most recently selected features.

**EXPLANATION**

The SYMD command affects the output of commands such as SYMB, TRA and DI. If SYMD is on, symbols are

substituted for the appropriate addresses in the command output.

SYMD does not effect your ability to use symbols in expressions, or to use the ADDS command.

SYMD is automatically turned Off whenever you SElect an emulator.

No "sectionname + offset" will appear for absolute sections.

For more information about symbolic debug, refer to the Emulation section of this manual.

**EXAMPLES**

Display 3-37 illustrates output produced by the TRA command when SYMD is ON and options -S and -L are both enabled. (Exact displays are emulator-dependent.)

The address symbols, such as LABEL001, are symbols from the label field in your assembly language source code. The byte offset addresses (+000, +003, etc.) correspond to those in the location counter field of your assembler listing. Thus, symbols in your trace output can make it easier to find trouble spots in your source code.

For comparison, Display 3-38 shows output produced by the TRA command when SYMD is OFF. (Again, exact displays are emulator-dependent.)

LOC	INST	MNEM	R	OPER	X/PC	EADD	RA	RB	XREG	SP	CC
SECTION:ALPHA											
+000	CE0010	LDX		0010			00	00	0010	0001	D0
+003	5F	CLR	B				00	00	0010	0001	D4
LABEL001	A600	LDA	A	00	+0010=0010		00	00	0010	0001	D4
+006	AB01	ADD	A	01	+0010=0011		00	00	0010	0001	D4
LABEL002	C900	ADC	B	00			00	00	0010	0001	D4
+00A	AB02	ADD	A	02	+0010=0012		00	00	0010	0001	D4
+00C	7E1055	JMP		1055		1055	00	00	0010	0001	D4
SECTION:BETA											
LABEL003	C900	ADC	B	00			00	00	0010	0001	D4
+007	AB03	ADD	A	03	+0010=0013		00	00	0010	0001	D4

Display 3-37.

```
> TRA ALL
> TRA JMP 0106 0110
> G 100

LOC   INST  MNEM  OPER   SP    F    A  B  C  D  E  H  L  IM  SOD
000100 210005 LXI   H,0500 0000 04 0F 01 00 00 00 05 00 07 0
000103 0605  MVI   B,05  0000 04 0F 05 00 00 00 05 00 07 0
000105 AF   XRA   A      0000 44 00 05 00 00 00 05 00 07 0
000109 C20601 JNZ   0106  0000 10 01 04 00 00 00 05 01 07 0
000109 C20601 JNZ   0106  0000 14 03 03 00 00 00 05 02 07 0
000109 C20601 JNZ   0106  0000 10 06 02 00 00 00 05 03 07 0
000109 C20601 JNZ   0106  0000 10 0A 01 00 00 00 05 04 07 0
000106 86   ADD   M      0000 04 0F 01 00 00 00 05 04 07 0
```

**Display 3-38.**

**SYNTAX**

```
symlo [-a] [-g] [-s] [sectionname] ...
```

**PARAMETERS**

- a Appends new symbols to the table. If -a is omitted, symbols currently in the table will be overwritten. Do not use -a in the first SYMLO after a SElect or reboot since you must overwrite the dummy section "NO. SECTION" with which the system is initialized.
- g Loads global symbols only. Works with B-Series linker output only. (Global symbols are explained in your Assembler Core Users Manual.)
- s Loads symbols representing scalars as well as those representing addresses.
- sectionname Name of a section of object code from which the symbols will be loaded. If no section names are specified, symbols from all sections are loaded.

**EXPLANATION**

SYMLO (SYMBOL LOad) loads symbols from standard input into the symbol table for use in symbolic debug. The input should be of the type produced by an A-Series or B-Series linker. Refer to the Emulation section of this manual for more information about symbolic debug.

**NOTE**

*When you use the SYMLO command, your 8540 must be in TERM mode in order to access files on the host.*

**EXAMPLE**

To load all the symbols in the load file BOL23 into the symbol table, including global symbols and scalar symbols, enter:

```
> symlo -S <BOL23
```

**SYNTAX**

```

tra [-n]
  [-l]
or
tra [-s] [-n] [-l] { all
                    jmp
                    off } [loaddr] [hiaddr]
    
```

**PARAMETERS**

- s Stops execution after each trace line is displayed. If -S is not specified, the program continues execution after each trace line is displayed.
- l Selects long display. Shows all registers on subsequent trace output.
- n Selects normal display. The contents of only the most important registers are reported. If the microprocessor has few registers, all registers are shown whether or not -L is specified.
- (default) If -L and -N are omitted, the most recent selection remains in effect.
- all Every instruction is displayed after its execution.
- jmp Displays only jump instructions: unconditional jumps, subroutine calls, and conditional branches where the specified condition is satisfied.

- off Disables trace display.
- loaddr An expression representing the lower bound of the address range to which ALL, JMP, or OFF applies. Defaults to 0.
- hiaddr An expression representing the upper bound of the address range to which ALL, JMP, or OFF applies. Defaults to the top of memory. **Hiaddr** must be greater than **loaddr**. If both **loaddr** and **hiaddr** are specified and **loaddr** does not contain the first byte of an instruction, tracing starts at the next instruction after **loaddr**.

When you enter the TRA command without parameters, the current trace conditions are displayed.

**EXPLANATION**

The TRA command establishes the conditions for displaying trace lines during program execution. Each trace line contains the address of the instruction, the instruction itself in hexadecimal, the mnemonic for the instruction, the operand, and certain processor register contents. The trace line display format varies for each microprocessor type. Refer to the Emulator Specifics section for the TRA command description that corresponds to your emulator.

Up to three TRA selections may be active at a time. The most recently entered selection takes precedence over any previously entered commands. A TRA ALL or TRA JMP command entered without address parameters replaces all existing selections. If you enter a TRA OFF command without address parameters, all previously entered TRA selections are deleted.

You may enter TRA -L or TRA -N at any time, no matter how many other selections are currently active. The -L and -N forms apply to the entire memory space regardless of the ranges specified in any previous TRA command lines.

*NOTE*

*You must have SElected the target processor before executing the TRA command.*

*When any TRA selections are in effect, your program executes at less than normal speed, even in those parts of the program that are not traced.*

**EXAMPLES**

Display 3-39 is an example of 8085A TRA command output.

When you enter the TRA command without parameters, the current trace conditions are displayed on standard output, as in the following example:

```
> TRA
TRA          ALL,000000,00FFFF
TRA          JMP,000106,000110
```

In this example, TRA ALL conditions are set for addresses 0000 to 0105, TRA JMP conditions are in effect from 0106 to 0110, and TRA ALL conditions are again in effect from 0111 to FFFF.

```
> TRA ALL
> TRA JMP 0106 0110
> G 100

LOC  INST  MNEM  OPER  SP    F  A  B  C  D  E  H  L  IM  SOD
000100 210005 LXI   H,0500 0000 04 0F 01 00 00 00 05 00 07 0 } TRA ALL
000103 0605  MVI   B,05 0000 04 0F 05 00 00 00 05 00 07 0 } in effect
000105 AF    XRA   A      0000 44 00 05 00 00 00 05 00 07 0
000109 C20601 JNZ   0106 0000 10 01 04 00 00 00 05 01 07 0
000109 C20601 JNZ   0106 0000 14 03 03 00 00 00 05 02 07 0 } TRA JMP
000109 C20601 JNZ   0106 0000 10 06 02 00 00 00 05 03 07 0 } in effect
000109 C20601 JNZ   0106 0000 10 0A 01 00 00 00 05 04 07 0
000106 86    ADD   M      0000 04 0F 01 00 00 00 05 04 07 0 } TRA ALL
                                     in effect
.
.
.
```

Display 3-39.



**SYNTAX**

```
wh [ -i
    -m
    -t ] {loadr hiaddr} ... [transfer]
```

**PARAMETERS**

- i Output is Intel-formatted hexadecimal object code.
- m Output is Motorola-formatted hexadecimal object code.
- t Output is in Standard Tekhex format.
- (default) If none of the output options (-i, -m, or -t) is specified, output is in Extended Tekhex format.
- loadr An expression representing the lower address of a data block in program/prototype memory to be converted and transferred to the file or device.
- hiaddr An expression representing the upper address of a data block in program/prototype memory to be converted and transferred to the file or device.
- transfer An expression representing the starting address of the program. Defaults to 0.

**EXPLANATION**

The WH (Write Hexadecimal code) command converts the specified data blocks within program or prototype memory from binary to hexadecimal format. The converted data is

then transferred to standard output. You may specify a file or device using the redirectional arrow (>).

For examples of WH command output in Tekhex, Intel, and Motorola formats, see the Tables section.

The WH command does not write out the symbols from the symbol table.

The contents of memory are not affected by the WH command.

The memory map assignments and emulation mode determine whether program memory or prototype memory contents are written. Refer to the EM and MAP command discussions.

To read hexadecimal object code from a file or device, rather than to write it, use the RH command.

*NOTE*

*When you use the WH command, your 8540 must be in TERM mode in order to access files on the host.*

**EXAMPLES**

Check the contents of memory 40-10F with the D command. See Display 3-40.

Convert the data in program memory locations 40-42 and 100-10E to Standard Tekhex format, and upload the converted code to the 8560 file named DEMO.TEX with the following command:

```
> wh -t >DEMO.TEX 40 42 100 10E 100
```

When the data is again read into program memory, the emulator program counter will be set to the transfer address (100).

```
> D 40 10F
  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 42 1A 00 00 00 00 00 00 00 00 00 00 00 00 00  .B.....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
000100 21 00 05 06 05 AF 86 23 05 C2 06 01 D3 F7 00 00  !.....#.....
```

**Display 3-40.**

## SYNTAX

```
wrt [-s][[-w]
      [-m][[-b] portnum value
```

## PARAMETERS

- s** Specifies special write as opposed to normal write. Not valid with **-M** modifier. Refer to your Emulator Specifics supplement to see if your emulator can use this modifier.
- m** Specifies that the value of **portnum** is a memory address (valid for memory-mapped I/O.) **-M** is the default for emulators with only memory-mapped I/O. Refer to your Emulator Specifics supplement to see if you can use the **-M** modifier.
- If this modifier is omitted, the **portnum** parameter is assumed to be a fixed port. The emulators which support both memory-mapped I/O and fixed-port I/O default to a fixed-port write.
- b** Specifies byte-oriented writing.
- w** Specifies word-oriented writing.
- (default) If neither **-B** nor **-W** is specified, the default is byte-oriented writing.

**portnum** If **-M** is not used: an expression designating a fixed I/O port. Refer to your Emulator Specifics supplement for the correct **portnum** values for your emulator.

If **-M** is used: an expression designating a memory location. The expression may include only one memory space designator.

**value** A valid expression. Its value must not exceed the maximum possible value of the indicated word or byte.

## EXPLANATION

The WRT (WRiTe) command writes a byte or a word to an I/O port on the emulator. This command is not available for most emulators: Refer to your Emulator Specifics supplement for details.

## EXAMPLES

```
> WRT 70 00
```

Writes the value 00 to emulator port 70.

```
> WRT -S TERMACIA 88
```

Performs a special byte-oriented write to the Z8001/Z8002 port represented by the symbol TERMACIA.

## SYNTAX

```
x [parameters]
```

## PARAMETERS

- parameters** One or more values to pass to the program. The program executes a Get Execution Line Parameter SVC (function code 1C) for each of these parameters. Refer to the Service Calls section for a description of the Get Execution Line Parameter SVC.

## EXPLANATION

The X (eXecute) command performs the same function as entering the LO and G commands sequentially.

The standard redirection symbol (<) is used to indicate the load file on the host, which can be an object file created by an assembler, or a load file created by the SAV command or by a linker.

## NOTE

*When using the X command, your 8540 must be in TERM mode in order to access files on the host.*

## EXAMPLES

Before initiating program execution, you must select the desired processor and the emulation mode.

```
> sel 8085
> em 1
```

Now you can execute the program in the 8560 load file CHECKBOOK by entering:

```
> x <CHECKBOOK DEPOSIT 200
```

This command loads CHECKBOOK into program memory and begins program execution. The parameters DEPOSIT and 200 can be accessed through Get Execution Line Parameter SVCs in the program.

## Section 4 INTERSYSTEM COMMUNICATION

	Page
<b>Introduction</b> .....	4-1
<b>Terminology</b> .....	4-1
<b>8540/8560 TERM Interface</b> .....	4-1
Setup .....	4-1
Command Execution .....	4-2
Precautions .....	4-2
Summary .....	4-2
<b>COM Interface</b> .....	4-2
Types of Transfers .....	4-3
Performing a Data Transfer .....	4-4
The Null Character .....	4-4
<b>8540/8550 Interface</b> .....	4-4
<b>8540/8002A Interface</b> .....	4-6
<b>8540/General Host Interface</b> .....	4-7
Unformatted Transfers .....	4-7
Formatted Transfers .....	4-9
Tektronix Hexadecimal Format (Tekhex) .....	4-9
Standard Tekhex .....	4-10
Extended Tekhex .....	4-11
Protocols for Formatted Transfers .....	4-15
Host Software Requirements for Formatted Transfers .....	4-16
The COM Command .....	4-22
Hardware Requirements .....	4-23
Establishing Communication .....	4-25
Exiting from COM .....	4-25
Data Transfers .....	4-25
Troubleshooting .....	4-27
Commands for Intersystem Communication .....	4-29

## TABLES

Table No.		Page
4-1	Data Transfer Types .....	4-3
4-2	Characteristics of Unformatted Transfers ...	4-8
4-3	Standard Tekhex Data Block Format .....	4-10
4-4	Standard Tekhex Termination Block Format .....	4-11
4-5	Standard Tekhex Abort Block Format .....	4-11
4-6	Extended Tekhex Header Field .....	4-12
4-7	Character Values for Checksum Computation .....	4-12
4-8	Extended Tekhex Data Block Format .....	4-12
4-9	Extended Tekhex Termination Block Format .....	4-12
4-10	Extended Tekhex Symbol Block Format ...	4-14
4-11	Extended Tekhex Symbol Block: Section Definition Field .....	4-14
4-12	Extended Tekhex Symbol Block: Symbol Definition Field .....	4-14
4-13	Character Transmission Formats Corresponding to Values for M .....	4-22
4-14	Signals at 8540 Remote Port .....	4-24
4-15	Remote Port Status Register Bits .....	4-25

## ILLUSTRATIONS

Fig. No.		Page
4-1	Command execution in TERM mode .....	4-2
4-2	Protocol for unformatted ASCII uploads ...	4-8
4-3	Protocol for unformatted binary uploads ...	4-8
4-4	Protocol for unformatted downloads .....	4-9
4-5	Protocol for formatted uploads .....	4-15
4-6	Protocol for formatted downloads .....	4-16
4-7	Algorithm for Tekhex transmit program ...	4-17
4-8	Algorithm for Tekhex receive program ...	4-18
4-9	Algorithm for Standard Tekhex conversion program .....	4-19
4-10	Algorithm for Extended Tekhex conversion program .....	4-20

## Section 4

# INTERSYSTEM COMMUNICATION

## INTRODUCTION

This section is designed to help you establish communication between the 8540 and an external computer system.

In this section, the term "intersystem communication" refers to the general process of communication between an 8540 and an external computer. A method for intersystem communication is necessary whenever you have information on one computer system that must be speedily and accurately transferred to another computer system.

This section includes basic definitions and examples, algorithms for use in writing required software, and troubleshooting techniques. This section is organized as follows:

- **Terminology.** Defines the terms used in the section.
- **8540/8560 TERM Interface.** Briefly describes the special interface between an 8540 and an 8560 Multi-User Software Development Unit.
- **COM Interface.** Summarizes the types of data transfers possible between an 8540 and a computer other than an 8560.
- **8540/8550 Interface.** Describes how an 8550 Microcomputer Development Lab operates as a host computer to an 8540.
- **8540/8002A Interface.** Describes how an 8002A  $\mu$ Processor Lab operates as a host computer to an 8540.
- **8540/General Host Interface.** Describes how the 8540 communicates with a computer that is not an 8560, 8550, or 8002A. Includes communication protocols, Standard and Extended Tekhex load module formats, COM command parameters, and hardware and software requirements for the host computer.

## TERMINOLOGY

The terms "host" and "host computer" refer to a computer system that directs or presides over one or more 8540s. Programs are created on the host, then downloaded to the 8540 to be executed and debugged.

"Upload" refers to the transfer of data from an 8540 to a host computer. "Download" refers to the transfer of data from a host computer to an 8540.

A "TERM interface" is a special arrangement between an 8540 and a host 8560 in which you have access to all the resources of the 8540 and the 8560 simultaneously. (You initiate a TERM interface by entering the OS/40 command CONFIG TERM.)

A "COM interface" is a setup that is used principally for the transfer of object code between an unspecified host computer and an 8540. (You initiate a COM interface by entering the COM command on the 8540.)

## 8540/8560 TERM INTERFACE

The 8560 System Users Manual contains a detailed discussion of the 8560's TERM interface with an 8540 or 8550. This subsection is a brief summary of that discussion.

## Setup

When operating in TERM mode, the 8540 and 8560 communicate using a high-speed interface (HSI) over a line that runs from an HSI I/O connector on the 8560 to the HSI connector on the 8540. (The 8540 and 8560 can also communicate using RS-232-C protocol, as through a modem. In this case, the communications line connects to the DTE jack (J101) on the 8540.) Your system terminal may be connected to another HSI I/O connector on the 8560, or to the 8540 (as in the stand-alone configuration for the 8540). For the purposes of this discussion, we will assume that the terminal is attached to the 8540, and that the 8540 and 8560 are connected via an HSI line.

You establish communication by placing the 8540 in TERM mode with the OS/40 command CONFIG TERM. In TERM mode, the 8540 passes every character you type directly to the 8560, and passes 8560 responses directly to the terminal. Once you log in to TNIX (the 8560 operating system), you can intermix OS/40 commands and TNIX commands, execute OS/40 commands from TNIX shell files, and download and upload programs between the 8540 and the 8560. In order for SVCs in your program to access files on the 8560, the 8540 must be in TERM mode.

### Command Execution

Every command line you enter is processed first by the 8560. If it is an OS/40 command, the 8560 passes the command back to the 8540. The 8540 executes the command, perhaps accessing 8560 files in the process, and then sends its response to you back through the 8560. This process is illustrated in Fig. 4-1.

### Precautions

Because TNIX processes every line you enter in TERM mode, an OS/40 command line that is acceptable to an 8540 in LOCAL mode will not necessarily be processed correctly in TERM mode. In TERM mode, for example, OS/40 command names must be entered in lowercase; the use of commas as delimiters is restricted; and certain characters that have special meaning to the TNIX shell must be "escaped" (made to appear non-special) so that the shell does not perform unwanted transformations on the command line.

### Summary

An 8540/8560 TERM interface gives you simultaneous access to the 8540 and the 8560, and provides a convenient means for combining the resources of the two systems. For more information on the hardware configurations and software considerations of this interface, refer to the 8560 System Users Manual.

### COM INTERFACE

The OS/40 COM command sets up communication between the 8540 and a host computer. To keep the interface as simple as possible, the 8540 uses the same RS-232-C-compatible communication protocol as a terminal attached to the host. You use the parameters of the COM command to specify the protocol that is appropriate for your host. A communications interface established via the COM command is referred to as a **COM interface**.

#### NOTE

*Your 8540 must have the optional COM Interface Package installed in order to support a COM interface with any computer. Throughout the rest of this section, it is assumed that this COM option is installed in your 8540.*

*The 8540 cannot execute the COM command while in TERM mode.*

Once you establish communication via the COM command, you can:

- execute host commands from the 8540 system terminal just as if the 8540 weren't there; and
- transfer data between the host and the 8540.

The 8550 Microcomputer Development Lab and 8002A  $\mu$ Processor Lab can also host COM interfaces with the 8540. These special interfaces are discussed later in this section.

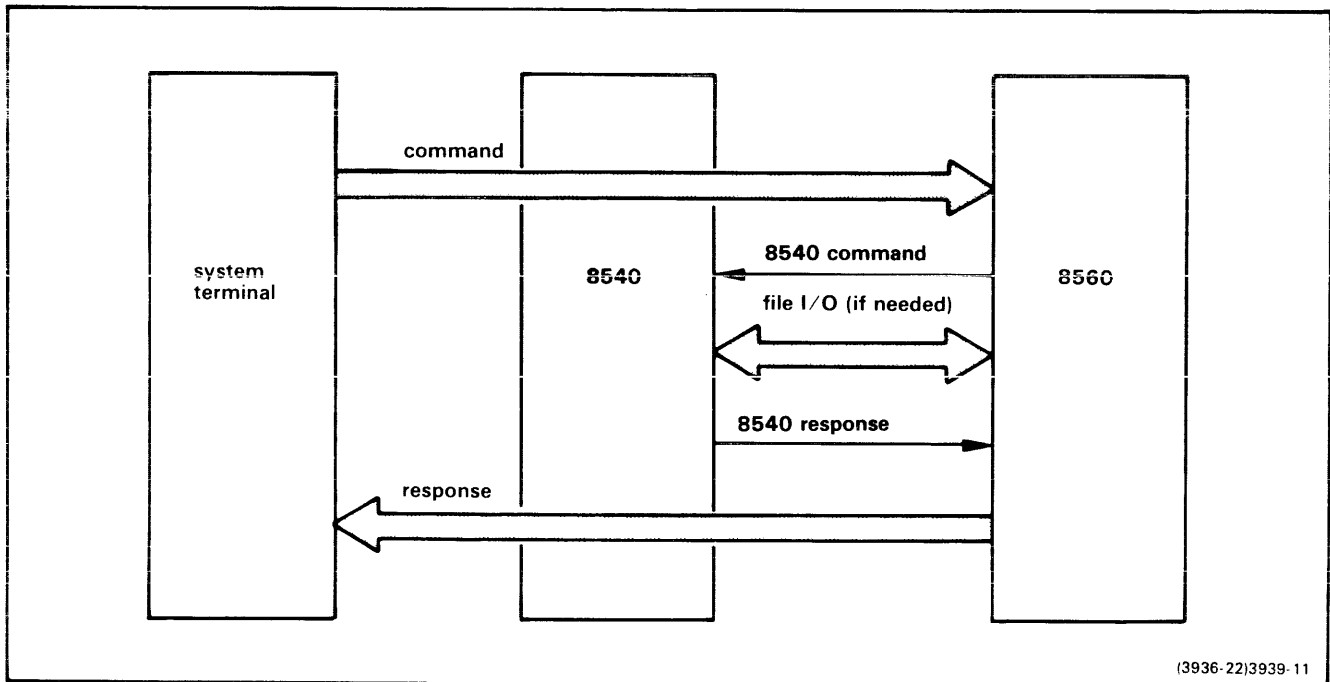


Fig. 4-1. Command execution in TERM mode.

(3936-22)3939-11

## Types of Transfers

Two categories of data transfers are possible with a COM interface: formatted and unformatted. Table 4-1 summarizes the types of formatted and unformatted transfers.

### Formatted (Object Code) Transfers

A formatted transfer copies blocks of object code in Tektronix Hexadecimal (Tekhex) format between a file on the host and program/prototype memory on the 8540. Checksums in the Tekhex format permit verification of each block of data as it is received. When you want to execute a program that you have created on the host, you use a **formatted download** to load the program into 8540 program memory.

There are two forms of Tekhex: **Standard Tekhex**, which is appropriate for microprocessors that address 64K of memory or less; and **Extended Tekhex**, which provides for

the transfer of program symbols (for use in symbolic debug) and addresses of up to 64 bits. Both Tekhex formats are explained in detail in the subsection "8540/General Host Interface" later in this section.

### Unformatted (File) Transfers

An unformatted transfer copies data between a file on the host and an I/O device (such as a line printer or paper tape reader/punch) on the 8540. The data is transmitted and received with no error checking other than parity.

A file may be transferred in **ASCII** mode or in **binary** mode. You use ASCII mode to transfer text files, and binary mode to transfer files of binary data.

At this time, unformatted transfers are of limited use on the 8540.

Table 4-1  
Data Transfer Types

Name	Type of Data	Type of Transfer	Source	Destination	Host's <sup>a</sup> Transfer Command	8540's Transfer Sequence
Formatted Download	Object Code	Tekhex to Binary	Host File	8540 Program Memory		(null) (cr)
Formatted Upload	Object Code	Binary to Tekhex	8540 Program Memory	Host File		(null) addresses (cr) <sup>b</sup>
Unformatted ASCII Download	ASCII	Copy	Host File	8540 Device		(null) >device (cr)
Unformatted Binary Download	Binary	Copy	Host File	8540 Device		(null) =device (cr)
Unformatted ASCII Upload	ASCII	Copy	8540 Device	Host File		(null) <device (cr)
Unformatted Binary Upload	Binary	Copy	8540 Device	Host File		(null) +device (cr)

<sup>a</sup> Fill in the commands that you use with your host.

<sup>b</sup> addresses—the starting and ending addresses of one or more blocks of program/prototype memory to be uploaded. The last pair of addresses may be followed by the transfer address of the program.

## Performing a Data Transfer

You initiate a data transfer by entering a command line that contains the following items:

1. a host-specific command that tells the host to transmit or receive the desired data (no carriage return here); and
2. a sequence of characters that tells the 8540 what to transmit or where to put what it receives. This 8540 transfer sequence begins with a null character and ends with a carriage return. The last column of Table 4-1 gives the 8540 transfer sequence for each type of data transfer.

Detailed descriptions of all six types of data transfers are given later in this section.

## The Null Character

The null character (ASCII 00) is a special command line delimiter to an 8540 operating in a COM interface. The null character signifies that everything preceding it is a command to the host computer, and that everything following it (including the null itself) is a command to the 8540.

Characters entered at the terminal are transmitted directly to the host computer until the null is encountered. When a carriage return is entered at the end of a command line that contains the null character, the 8540 processes its portion of the command line, prepares to receive or send data, and then transmits the carriage return to the host. The host computer then processes the portion of the command line that was sent to it, and the data transfer takes place. (If you specified a special end-of-line character sequence with the EOL parameter of the COM command, that end-of-line is sent to the host in place of each carriage return.)

On most terminals, the null character can be generated as a CTRL-@, that is, by pressing the CTRL key at the same time you press the keys needed to generate the @ symbol. In this section the null character is represented by **(null)**. The null character is not displayed on the terminal.

A null character can be sent from the 8540 terminal to the host by entering two consecutive nulls on the 8540 terminal.

## 8540/8550 INTERFACE

This subsection explains how to use an 8550 to host formatted (Tekhex) transfers with an 8540. (For information on how the 8550 hosts both formatted and unformatted transfers, refer to your 8550 System Users Manual.) When the 8550 is host, all communication procedures and data formats are handled automatically, so you do not need any special knowledge of Tekhex, the COM

command, or communication protocols. For details on this type of information, refer to the "8540/General Host Interface" discussion later in this section.

## Setup

To set up intersystem communication between an 8550 and an 8540, you must connect an RS-232-C cable to the remote communication ports on the 8301 (jack J102) and the 8540 (jack J101). Set the MODE SELECT switches on the two systems to CNTL (L). Set the REMOTE BAUD switches to 2400.

## Establishing Communication

When the physical connection has been established, you can initiate communication between the systems by first entering the COM command on the 8540 and then entering the HOST command on the host 8550. If you are performing one or more formatted uploads, and you want the object code to be stored in Standard Tekhex rather than Extended Tekhex format, enter:

```
> COM P=7E7E F=T <CR>
```

Otherwise, enter:

```
> COM P=7E7E <CR>
```

The HOST command has no parameters.

## HOST Command Operation

After you have entered the COM command on the 8540 and the HOST command on the 8550, a message appears on the 8550 terminal that says

```
HOST Vx.x
```

and a message appears on the 8540 terminal that says

```
Enter U or D; A, B, or H; and 8550 filespec:
```

At this point you are ready to initiate a data transfer. The discussion under "Data Transfers" describes how to perform both types of formatted transfers. While the HOST command is active, you cannot execute other DOS/50 commands, and all input to both the 8540 and the 8550 is through the 8540's terminal.

## Exiting from HOST

To exit from HOST, perform the following steps:

1. Press the 8540 terminal's BREAK key to abort any data transfer that is taking place.
2. Enter a null character and press the ESC key on the 8540 to exit from the COM command.
3. Type CTRL-C on the 8550 terminal to obtain the DOS/50 prompt.

4. Enter the command "A -A" (Abort All) on the 8550 terminal.

```
(CTRL-C)
>>A -A <CR>
```

## Data Transfers

The commands to initiate data transfers are entered on the 8540 terminal in response to the prompts sent to the 8540 terminal from the 8550. Two prompts are displayed. The first prompt is:

Enter U or D; A, B, or H; and 8550 filespec:

This prompt requests information for the 8550 about the type and direction of the desired transfer: U (upload) or D (download); A (unformatted ASCII transfer), B (unformatted binary transfer), or H (Tekhex formatted transfer). The 8550 filespec specifies the 8550 file to or from which the transfer is to take place. No embedded spaces are allowed between the parameters.

After you have entered the requested information, the following prompt appears on the 8540's terminal:

Enter transfer sequence, beginning  
with null character:

This prompt requests information for the 8540. The null character indicates to the 8540 that the rest of the command line is a command to the 8540.

The transfer command to the 8550 must match the transfer command to the 8540.

### NOTE

*Check the 8550 system terminal for error messages after you initiate each transfer. The informational message "Error FF - End of file" appears after each successful transfer.*

After a transfer has been completed, there may be a delay of up to 30 seconds before the next prompt appears and the 8550 is ready for another transfer.

## Formatted Download

The following command sequence downloads a Tekhex load module to the 8540. The object code is placed in 8540 program/prototype memory, and the program symbols, if any, are placed in the symbol table in 8540 system memory.

Enter U or D; A, B, or H; and 8550 filespec:  
DHfilespec <CR>

**D** indicates to the 8550 that the transfer is a download.  
**H** indicates to the 8550 that the transfer is a formatted Tekhex transfer.

**filespec** is the 8550's Tekhex file from which the data is to be transmitted.

After you've entered this command line, you tell the 8540 the type of transfer.

Enter transfer sequence, beginning  
with null character:  
(null) <CR>

**(null)** indicates that the rest of the command line is a command to the 8540. Since there are no commands following the null character, the 8540 recognizes that the transfer is a formatted download into program memory.

Each time the 8550 sends a message block, an asterisk is displayed on the 8540's terminal. If the block is transferred correctly, the starting address and length of the block are displayed on the 8540 terminal. If the 8540 detects an error and issues a negative acknowledgement (NAK), the 8550 resends the block and another asterisk is displayed on the 8540 terminal. (If the 8540 has COM Version 4.1 or a later version, the 8540 displays an asterisk for each successfully transferred block and an underscore for each NAK.)

When the transfer is complete, the 8540 terminal displays the initial prompt requesting information for the host computer:

Enter U or D; A, B, or H; and 8550 filespec:

and the 8550 terminal displays

HOST: Error FF - End of file

## Formatted Upload

The following command sequence uploads blocks of object code from 8540 program/prototype memory to a file on the 8550. No program symbols are uploaded.

Enter U or D; A, B, or H; and 8550 filespec:  
UHfilespec <CR>

**U** indicates to the 8550 that the transfer is an upload.  
**H** indicates to the 8550 that the transfer is a formatted Tekhex transfer.

**filespec** is the 8550 file to which the data is to be transferred.

After you've entered this command line, you tell the 8540 the type of transfer.

Enter transfer sequence, beginning  
with null character:  
(null)lowadd1 hiadd1 lowadd2  
hiadd2 ... transadd <CR>



(null) indicates that the rest of the command line is a command to the 8540.

**lowadd1** is the low address of a block of 8540 program memory to be transferred.

**hiadd1** is the high address of a block of 8540 program memory to be transferred.

**lowadd2** is the low address of another block of 8540 program memory to be transferred.

**hiadd2** is the high address of another block of 8540 program memory to be transferred.

**transadd** is the beginning execution address for the code in the transferred blocks.

Each time the 8540 sends a block, the 8540 terminal displays

```
*BLOCK SENT AT: address
```

An asterisk is displayed each time a block is sent. If more than one asterisk appears, the 8550 has detected an error and issued a negative acknowledgement (NAK), and the block has been resent.

When the transfer is complete, the 8540 terminal displays the initial prompt requesting information for the 8550:

```
Enter U or D; A, B, or H; and 8550 filespec:
```

and the 8550 terminal displays

```
HOST: Error FF - End of file
```

## Troubleshooting

Symptom	Cannot establish initial communication.
Check	<ol style="list-style-type: none"> <li>1. Verify that the optional COM interface software has been installed in your 8540.</li> <li>2. Verify that the transmit and receive baud rates of the 8550 and the 8540 correspond.</li> <li>3. Verify that the RS-232-C cable is plugged into the correct jacks on the 8550 and on the 8540.</li> <li>4. Verify that the switches controlling the REMOTE jacks on the 8301 and 8540 are positioned correctly.</li> </ol>
Symptom	No reaction after typing transfer request.
Check	<ol style="list-style-type: none"> <li>1. Verify that your commands to the 8550 and 8540 are appropriate, and that you are using the correct file name.</li> <li>2. Verify that when you entered the COM command on the 8540, you specified the appropriate prompt sequence (P=7E7E).</li> <li>3. On a formatted download, verify that the Tekhex file on the 8550 is in proper format.</li> </ol>

## 8540/8002A INTERFACE

This subsection explains how to use the TEKDOS SEND command to download a program from an 8002A  $\mu$ Processor Lab to 8540 program/prototype memory. The program must be in Standard Tekhex format, as produced by the TEKDOS WHEX command.

For details about Tekhex, the COM command, and communications protocols, refer to the "8540/General Host Interface" discussion later in this section.

### Setup

The 8002A and 8540 are connected via an RS-232-C line that runs from jack J101 on the 8002A rear panel to jack J102 (DTE) on the 8540 rear panel. On the 8540, be sure that the REMOTE BAUD switch is set to 2400 and that the MODE SELECT switch is set to CNTL (L). On the 8002A System Communication board, verify that the J1 jumper is set to the lower position, and that the PORT1 baud rate jumper is set to 2400.

Each system must have its own system terminal.

### Establishing Communication

To establish communication between the 8540 and the 8002A, enter the following COM command on the 8540 system terminal:

```
> COM P=3F E=L L=I <CR>
```

Then enter the SEND command on the 8002A system terminal:

```
> SEND <CR>
```

The SEND command does not issue any prompt.

### Performing a Download

To download a program from a Tekhex file on the 8002A, type the name of the file, a null character, and a carriage return on the 8540 terminal:

```
filename(null) <CR>
```

Each time the 8002A sends a message block to the 8540, an asterisk is displayed on the 8540 terminal. If the block is transferred correctly, the starting address and length of the block are displayed on the 8540 terminal. If the 8540 detects an error and issues a negative acknowledgement (NAK), the 8002A resends the block and another asterisk is displayed. (If the 8540 has COM Version 4.1 or a later version, the 8540 displays an asterisk for each successfully transferred block and an underscore for each NAK.) When the download is complete, the 8540 terminal displays the message "DNLOAD:".

## Terminating Communication

To terminate communication between the 8540 and the 8002A, perform the following steps:

1. If a download is in progress, press the BREAK key twice on the 8540 terminal.
2. Enter a null character and press the ESC key on the 8540 terminal.

## Troubleshooting

Symptom	Cannot establish initial communication.
Check	<ol style="list-style-type: none"> <li>1. Verify that the optional COM interface software has been installed in your 8540.</li> <li>2. Verify that the RS-232-C cable is plugged into the correct jacks on the 8540 and the 8002A.</li> <li>3. Verify that the REMOTE BAUD and MODE SELECT switches on the 8540 rear panel are positioned correctly.</li> <li>4. Verify that the J1 jumper and PORT1 baud rate jumper are positioned correctly on the 8002A System Communication board.</li> </ol>
Symptom	Unexpected reaction to download request.
Check	<ol style="list-style-type: none"> <li>1. Verify that you entered the correct parameters for the COM command.</li> <li>2. Verify that the 8002A file you specified is in Standard Tekhex format.</li> </ol>

## 8540/GENERAL HOST INTERFACE

This subsection provides the detailed information required to establish communication between an 8540 and a host computer that is not an 8560, 8550, or 8002A. Refer to the "COM Interface" discussion earlier in this section for a summary of the types of data transfers possible between the 8540 and a host.

If you are sufficiently familiar with the way your host computer communicates with a terminal, you can use the information in this subsection to:

- establish the appropriate RS-232-C connections and switch settings on the 8540 and the host
- determine the parameters of the COM command line you will use to initiate communication with the host
- determine the host command line that is appropriate for each type of data transfer
- write programs for the host that create, transmit, or receive files in Tekhex format.

It is recommended that you make this host-specific information available to other users of your 8540, using the spaces provided at the end of this section and in the Operating Procedures section of this manual.

This subsection is divided into the following topics:

- **Unformatted Transfers.** Explains the protocols for unformatted transfers and identifies the types of host computer commands you can use to perform these transfers.
- **Formatted Transfers.** Describes Standard and Extended Tekhex formats, explains the protocols for formatted transfers, and gives algorithms for programs to create, transmit, and receive Tekhex files.
- **The COM Command.** Explains the parameters of the COM command.
- **Hardware Requirements.** Describes the intersystem communication hardware on the 8540.
- **Establishing Communication.** Tells how to connect the 8540 to the host and establish communication.
- **Exiting from COM.** Tells how to terminate communication.
- **Data Transfers.** Describes in detail how to perform each type of data transfer.
- **Troubleshooting.** Describes possible communication problems and ways to solve them.

## Unformatted Transfers

An unformatted transfer conveys data between a file on the host and an I/O device on the 8540. You cannot use an unformatted transfer for direct loading or dumping of 8540 program memory.

This discussion covers the utilities required on the host computer to accomplish unformatted transfers, the kinds of unformatted transfers, and protocols for unformatted transfers.

Unformatted transfers are simpler than formatted transfers. In most cases, no special host computer software is required and standard host computer operating system utilities are used.

### Host Utility Requirements for Unformatted Transfers

Two host computer utilities are required for unformatted transfers. These utilities must accomplish the following tasks:

1. For a download, the utility must be able to copy information from a file on the host computer to a terminal.
2. For an upload, the utility must be able to copy information from a terminal to a file on the host computer.

Simple copy commands, text editors, and rapid text entry modes are examples of utilities that might be used.

### Kinds of Unformatted Transfers

Four kinds of unformatted transfers are possible with the 8540: ASCII uploads, ASCII downloads, binary uploads, and binary downloads. Table 4-2 shows the characteristics of each type of unformatted transfer.

**Table 4-2**  
Characteristics of Unformatted Transfers

Transfer Type	Maximum Recommended Baud Rate <sup>a</sup>	Unit of Transfer	Significant Bits Per Data Byte
ASCII Upload	2400	ASCII line	7 <sup>b</sup>
ASCII Download	2400	Entire file	7 <sup>b</sup>
Binary Upload	2400	Entire file	8
Binary Download	2400	Entire file	8

<sup>a</sup> Unformatted transfers may also proceed successfully at 4800 or 9600 baud for some hosts.

<sup>b</sup> The high-order bit of each data byte is cleared by the 8540.

### Protocols for Unformatted Transfers

With unformatted transfers, the transmitting computer does not receive confirmation that data was correctly received by the receiving computer. Instead, a prompt sequence (defined by the P parameter of the COM command) signals the 8540 that the host is ready for input (either more data or a new command).

**Upload Protocol.** An unformatted upload transfers data from a device on the 8540 (such as the paper tape reader) to a file on the host, either line by line (ASCII upload) or as an uninterrupted stream of bytes (binary upload).

When the upload is complete, the 8540 prints the message RIOT:

on the terminal, but does not transmit any end-of-file indicator. You must enter the appropriate end-of-file sequence when you see the RIOT: message, or the host must keep track of the time elapsed after each byte is sent and terminate the transfer when the 8540 appears to have stopped transmitting.

**ASCII Upload Protocol.** In an unformatted ASCII upload, the host sends the prompt to the 8540 when the host is ready to receive a line of data. The 8540 then waits the amount of

time indicated by the T parameter of the COM command and then sends the next line of data. Omit the P (prompt) parameter unless the host actually issues a prompt every time it is ready to receive a line.

On the 8540, every carriage return (ASCII 0D) in the data is assumed to be the end of a line. If you specify an end-of-line sequence with the EOL parameter of the COM command, the 8540 sends that sequence in place of each carriage return.

The protocol for an unformatted ASCII upload is given in Fig. 4-2.

**Binary Upload Protocol.** In an unformatted binary upload, the host sends the prompt (if any) to the 8540 when the host is ready to receive the data. The 8540 then waits the amount of time indicated by the T parameter of the COM command, and then sends the data to the host as an uninterrupted stream of bytes.

The protocol for an unformatted binary upload is given in Fig. 4-3.

**Download Protocol.** In an unformatted download, a file on the host is copied to an I/O device on the 8540 (such as the line printer). The host transmits the entire file without interruption, then sends the prompt to indicate that the end of the file has been transmitted and that the host can accept input from the 8540.

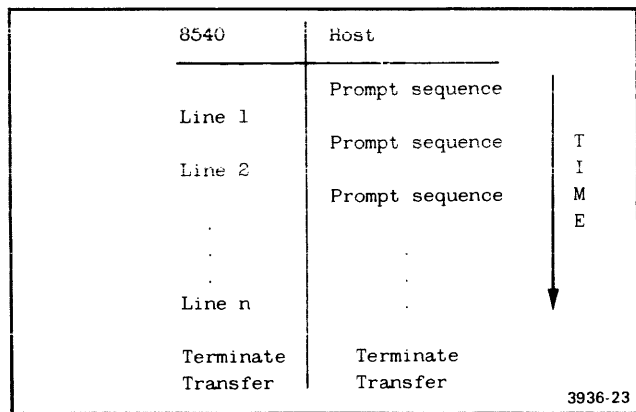


Fig. 4-2. Protocol for unformatted ASCII uploads.

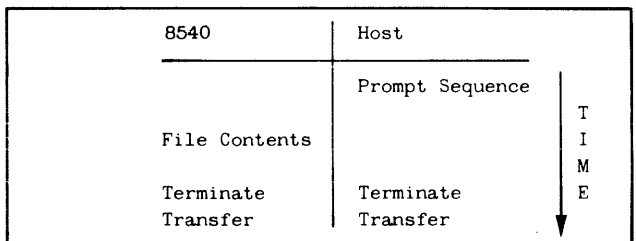


Fig. 4-3. Protocol for unformatted binary uploads.

3936-24

For unformatted downloads, it is usually appropriate for the P (prompt) parameter of the COM command to match the host's system prompt. However, be sure that the prompt sequence you specify does not appear anywhere in the file to be downloaded, since the prompt marks the end of the data to be downloaded.

The protocol for an unformatted download is given in Fig. 4-4.

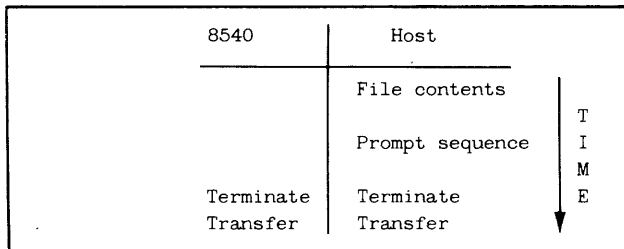


Fig. 4-4. Protocol for unformatted downloads. 3936-25

## Formatted Transfers

A formatted transfer conveys object code between a file on the host and program/prototype memory on the 8540. The file may also contain program symbols to be placed in the 8540's system memory for use in symbolic debug. Formatted transfers are more complex than unformatted transfers because the data must be put into predefined message blocks before it is transferred. The message blocks are structured according to the rules defining Tektronix Hexadecimal Format (Tekhex).

This discussion covers the two types of Tekhex format, protocols for formatted transfers, and algorithms for software required on the host computer.

### Tektronix Hexadecimal Format (Tekhex)

Standard Tekhex and Extended Tekhex are two ways of encoding a load module as lines of printable ASCII characters. A load module contains the following information:

1. blocks of **object code** to be loaded into memory;
2. a **load address** for each block of object code that tells where in memory to load that block;

3. a **transfer address**, which is the address where program execution should begin; and
4. for use in symbolic debug, **program symbols** and their values.

Extended Tekhex format can convey all four types of information, while Standard Tekhex conveys only the first three types. Both Tekhex formats also contain additional information, such as headers and checksums, which are used by the program that loads the object code.

An 8540 or 8550 can accept a module that is encoded in either format, or in a mixture of the two. An 8002A (or an 8550 running DOS/50 Version 1) supports only Standard Tekhex.

You need to use Extended Tekhex instead of Standard Tekhex if any of the following conditions are true:

- you need to specify a load address larger than FFFF; or
- you need to convey symbols and symbol values for symbolic debug; or
- you want to transmit more than thirty bytes of object code per block.

**Message Block Delimiters.** A line of Tekhex information is referred to as a **message block**. Each message block begins with a slash (/—for Standard Tekhex) or a percent sign (%—for Extended Tekhex) and ends with an end-of-line character sequence. Tektronix development systems use a single carriage return as an end-of-line. If your host computer uses a different end-of-line, it must be specified with the EOL parameter of the COM command, which is discussed later in this section.

**Hexadecimal Digits.** The letters A through F, when used as hexadecimal digits in Tekhex message blocks, must be uppercase.

### Standard Tekhex

Standard Tekhex uses three types of message blocks:

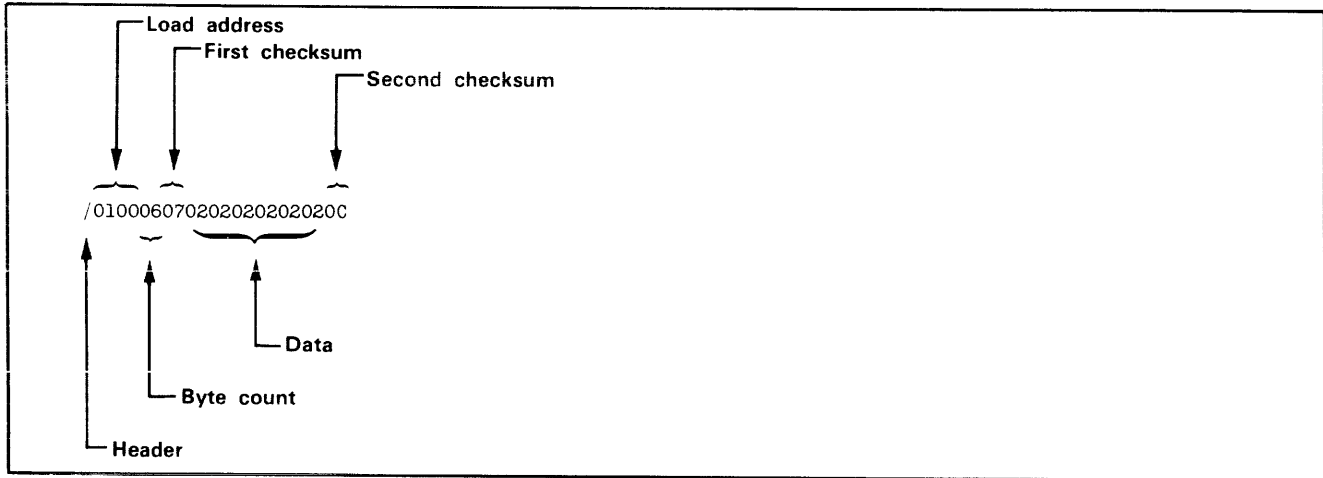
1. A **data** block contains object code.
2. A **termination** block contains the transfer address and marks the end of the load module.
3. An **abort** block is used to terminate transmission when an unrecoverable error occurs.

**Data Blocks.** A data block contains the load address for the object code in the block, a byte count, 30 or fewer bytes of object code, and checksum information. The format for a data block is given in Table 4-3.

Display 4-1 is an example of a data block.

**Table 4-3**  
**Standard Tekhex Data Block Format**

Field	Number of ASCII Characters	Description
/	1	The slash specifies that the block is in Standard Tekhex format.
Load Address	4	The address where the object code is to be loaded (high-byte, low-byte format).
Byte Count	2	Number of data bytes in the data field of the block.
First Checksum	2	The sum, mod 256, of the six hex digits of the load address and byte count.
Data	2n (2 to 60)	n data bytes, each represented as two hex digits. Maximum of 30 data bytes.
Second Checksum	2	Sum, mod 256, of the 2n hex digits of the data field.



Display 4-1.

**Termination Blocks.** A termination block contains only a transfer address (instead of a load address), a byte count of zero, and a checksum of the address and byte count. Because there is no data in the block, there is no second checksum. The format for a termination block is given in Table 4-4.

**Table 4-4**  
Standard Tekhex Termination Block Format

Field	Number of ASCII Characters	Description
/	1	The slash specifies that the block is in Standard Tekhex format.
Transfer Address	4	Starting execution address (high-byte, low-byte format) of the code transmitted in the data blocks.
Byte	2	Always 00 in a termination block.
Checksum	2	Sum, mod 256, of the six hex digits of the transfer address and byte count.

Display 4-2 is an example of a termination block.

**Abort Blocks.** An abort block contains two slashes followed by a message. This block tells the 8540 that an unrecoverable transmission error has occurred and to

terminate the transmission. When this block is received, the message in the block is displayed on the 8540's terminal. The format for an abort block is given in Table 4-5.

**Table 4-5**  
Standard Tekhex Abort Block Format

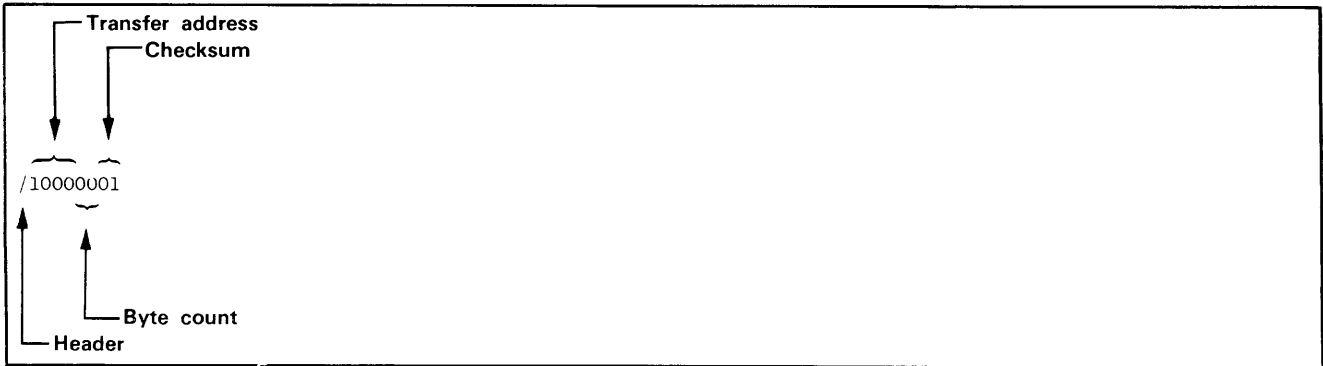
Field	Number of ASCII Characters	Description
/	1	The slash specifies that the block is in Standard Tekhex format.
/	1	Another slash to identify the abort block.
Message	1 to 69	A message to be displayed when the transfer is aborted.

Display 4-3 is an example of an abort block.

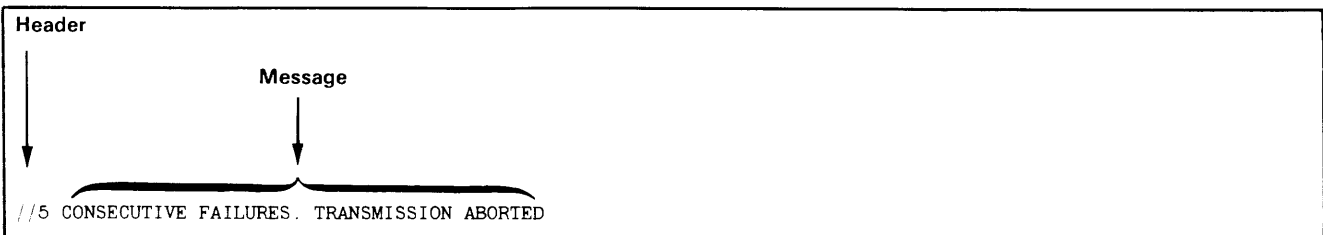
**Extended Tekhex**

Extended Tekhex uses three types of message blocks:

1. A **data** block contains object code.
2. A **symbol** block contains information about a program section and the symbols associated with it. This information is needed only for symbolic debug.
3. A **termination** block contains the transfer address and marks the end of the load module.



Display 4-2.



Display 4-3.

**NOTE**

*Extended Tekhex has no specially defined abort block. To abort a formatted transfer, use a Standard Tekhex abort block, as defined earlier in this section.*

Each block begins with a six-character header field and ends with an end-of-line character sequence (on the 8540, a carriage return). A block can be up to 255 characters long, not counting the end-of-line. A header field has the format shown in Table 4-6.

**Table 4-6**  
Extended Tekhex Header Field

Item	Number of ASCII Characters	Description
%	1	A percent sign specifies that the block is in Extended Tekhex format.
Block Length	2	The number of characters in the block: a two-digit hex number. This count does not include the leading % or the end-of-line.
Block Type	1	6 = data block 3 = symbol block 8 = termination block
Checksum	2	A two-digit hex number representing the sum, mod 256, of the values of all the characters in the block, except the leading %, the checksum digits, and the end-of-line. Table 4-7 gives the values for all characters that may appear in Extended Tekhex message blocks.

**Table 4-7**  
Character Values for Checksum Computation

Characters	Values (Decimal)
0..9	0..9
A..Z	10..35
\$	36
%	37
. (period)	38
_ (underscore)	39
a..z	40..65

**Variable-Length Fields.** In Extended Tekhex, certain fields may vary in length from 2 to 17 characters. This practice enables you to compress your data by eliminating leading zeros from numbers and trailing spaces from symbols. The first character of a variable-length field is a hexadecimal digit that indicates the length of the rest of the field. The digit 0 indicates a length of 16 characters.

For example, the symbols START, LOOP, and KLUDGE-STARTSHERE are represented as 5START, 4LOOP, and OKLUDGESTARTSHERE. The values 0, 100H, and FF0000H are represented as 10, 3100, and 6FF0000.

**Data and Termination Blocks.** If you do not intend to transfer program symbols with your object code, you can do without symbol blocks. Your load module can consist of one or more data blocks, followed by a termination block. Table 4-8 gives the format of a data block, and Table 4-9 gives the format of a termination block.

**Table 4-8**  
Extended Tekhex Data Block Format

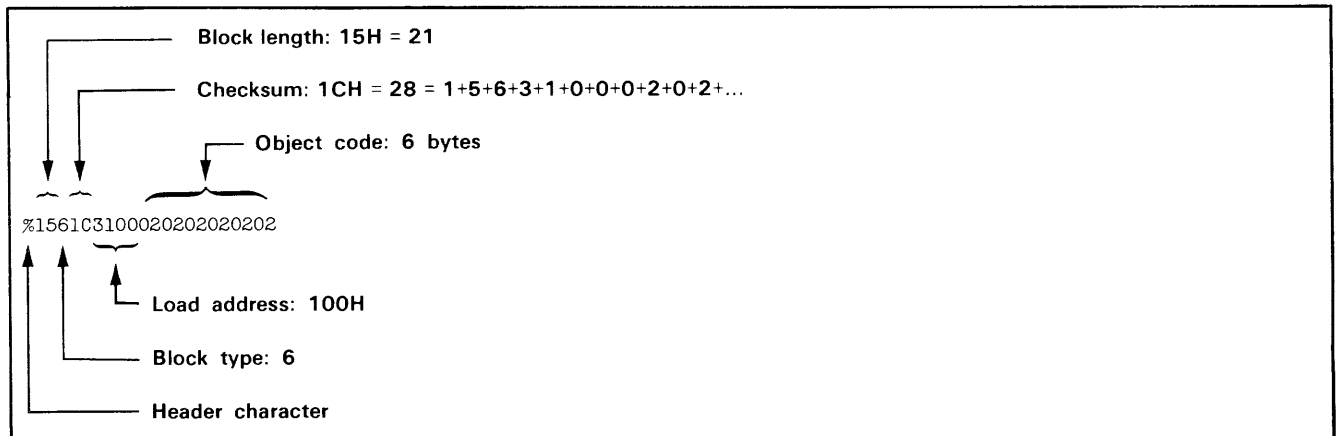
Field	Number of ASCII Characters	Description
Header	6	Standard header field. Block type = 6.
Load Address	2 to 17	The address where the object code is to be loaded: a variable-length number.
Object Code	2n	n bytes, each represented as two hex digits.

**Table 4-9**  
Extended Tekhex Termination Block Format

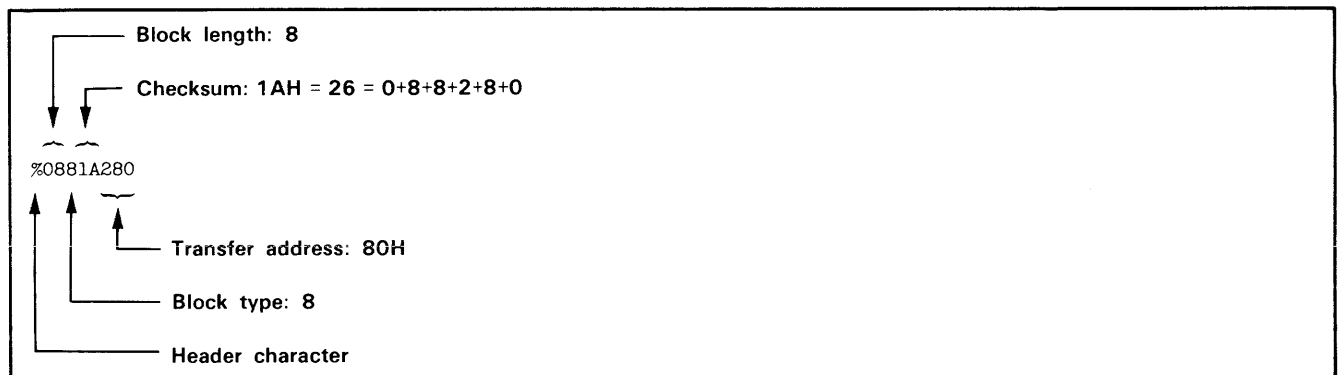
Field	Number of ASCII Characters	Description
Header	6	Standard header field. Block type = 8.
Transfer Address	2 to 17	The address where program execution is to begin: a variable-length number.

Display 4-4 is an example of a data block.

Display 4-5 is an example of a termination block.



Display 4-4.



Display 4-5.

**Symbol Blocks.** A symbol used in symbolic debug has the following attributes:

1. the symbol itself: 1 to 16 letters, digits, dollar signs, periods, or underscores. The first character of the symbol can be a letter or (if the symbol is a section name) a percent sign.
2. a value: up to 64 bits (16 hexadecimal digits).
3. a type: address or scalar. (A scalar is any number that is not an address.) An address may be further classified as a **code address** (the address of an instruction) or a **data address** (the address of a data item). Symbolic debug does not currently use the code/data distinction, so the address/scalar distinction is sufficient for standard applications of Extended Tekhex.
4. a global/local designation. This designation is of limited use in a load module, and is provided for future development. The concept of global symbols is discussed in the Assembler Core Manuals for TEKTRONIX A Series and B Series assemblers. If the global/local distinction is not important for your purposes, simply call all your symbols global.

5. section membership. A section may be thought of as a named area of memory. Each address in your program belongs to exactly one section. A scalar belongs to no section. The concept of sections is discussed in detail in the Assembler Core Manuals for both A Series and B Series assemblers. The significance of sections with regard to symbolic debug is illustrated in the Learning Guide of this manual.

The symbols in your program are conveyed in **symbol blocks**. Each symbol block contains the name of a section and a list of the symbols that belong to that section. (You may include scalars with any section you like.) More than one block may contain symbols for the same section. For each section, exactly one symbol block should contain a **section definition** field, which defines the starting address and length of the section.

If your object code has been generated by an assembler or compiler that does not deal with sections, simply define one section called (for example) MEMORY, with a starting address of 0 and a length greater than the highest address used by your program; and put all your symbols in that section.



Table 4-10 gives the format of a symbol block. Tables 4-11 and 4-12 give the formats for section definition fields and symbol definition fields, which are parts of a symbol block.

**Table 4-10**  
**Extended Tekhex Symbol Block Format**

Field	Number of ASCII characters	Description
Header	6	Standard header field. Block type = 3.
Section Name	2 to 17	The name of the section that contains the symbols defined in this block: a variable-length symbol.
Section Definition	5 to 35	This field must be present in exactly one symbol block for each section. This field may be preceded or followed by any number of symbol definition fields. Table 4-11 gives the format for this field.
Symbol Definition(s)	5 to 35 each	Zero or more symbol definition fields, as described in Table 4-12.

**Table 4-11**  
**Extended Tekhex Symbol Block:**  
**Section Definition Field**

Item	Number of ASCII Characters	Description
0	1	A zero signals a section definition field.
Base Address	2 to 17	The starting address of the section: a variable-length number.
Length	2 to 17	The length of the section: a variable-length number, computed as 1 + (high address - base address).

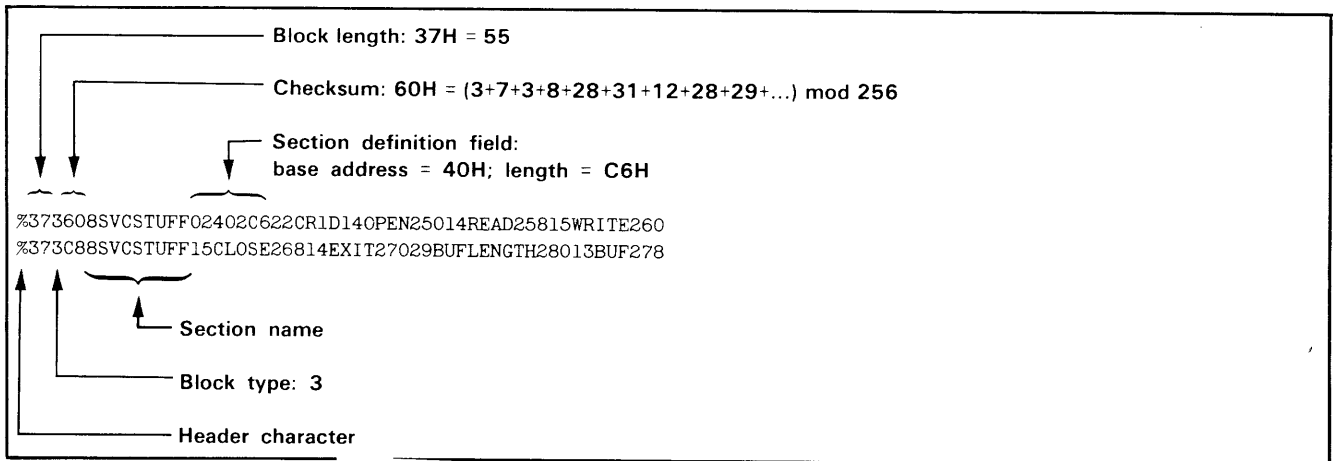
**Table 4-12**  
**Extended Tekhex Symbol Block:**  
**Symbol Definition Field**

Item	Number of ASCII Characters	Description
Type	1	A hex digit that indicates the global/local designation of the symbol, and the type of value the symbol represents: 1 = global address 2 = global scalar 3 = global code address 4 = global data address 5 = local address 6 = local scalar 7 = local code address 8 = local data address
Symbol	2 to 17	A variable-length symbol.
Value	2 to 17	The value associated with the symbol: a variable-length number.

*Symbol Block Example.* Suppose a section called SVCSTUFF occupies memory locations 40H-105H and contains the symbols shown in the following table, and that you don't care about global/local and code/data distinctions.

Symbol	Value	Type	Symbol Definition Field
CR	0DH	scalar	22CR1D
OPEN	50H	address	14OPEN250
READ	58H	address	14READ258
WRITE	60H	address	15WRITE260
CLOSE	68H	address	15CLOSE268
EXIT	70H	address	14EXIT270
BUFLNGTH	80H	scalar	29BUFLNGTH280
BUF	78H	address	13BUF278

Display 4-6 shows how this information might be encoded in Extended Tekhex symbol blocks. (All this information could be encoded in a single 96-character block. It is divided into two blocks for purposes of illustration.)



Display 4-6.

### Protocols for Formatted Transfers

Formatted transfers require a formal communications protocol. ASCII characters are used to positively acknowledge or negatively acknowledge the receipt of Tekhex message blocks. The character "O" (ASCII 30H) is used as a positive acknowledgment (ACK) and the character "7" (ASCII 37H) is used as a negative acknowledgment (NAK). Each ACK or NAK is followed by an end-of-line.

**Upload Protocol.** After the 8540 transmits a message block to the host, the host computer performs whatever error checking is desired. If no transmission error occurs, the host sends an ACK followed by a prompt to the transmitting 8540. The transmitting 8540 then continues with the formatted transfer by sending the next message block. If the host detects a transmission error, it responds by sending a NAK followed by a prompt to the transmitting 8540. When the transmitting 8540 receives a NAK, it re-sends the message block that was incorrectly received. Figure 4-5 illustrates the protocol for a formatted upload.

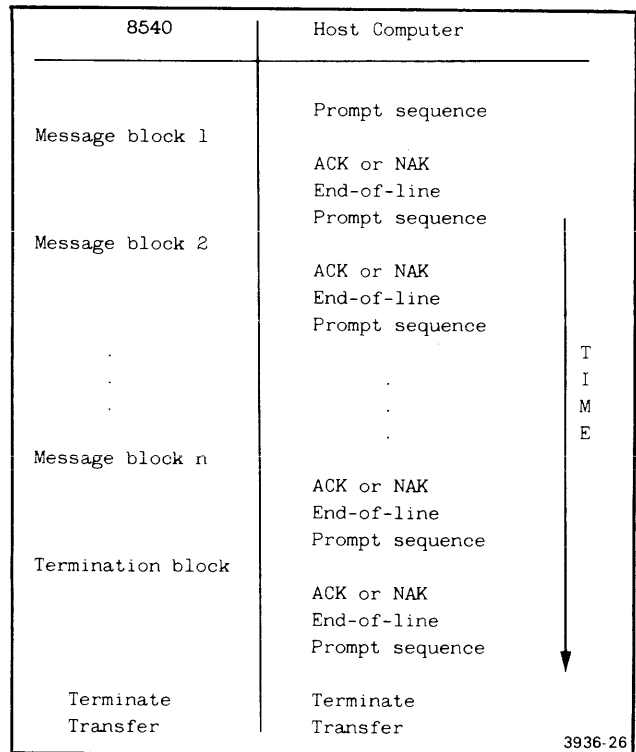


Fig. 4-5. Protocol for formatted uploads.

**Download Protocol.** After the host sends a message block followed by a prompt sequence to the receiving 8540, the receiving 8540 performs the appropriate error checking. If no transmission error occurs, the receiving 8540 responds by sending an ACK to the transmitting host. The host then continues with the transfer by sending the next message block. If the receiving 8540 detects a transmission error, it responds by sending a NAK to the host. When the transmitting host receives a NAK, it re-sends the message block that was incorrectly received. Fig. 4-6 illustrates the protocol for a formatted download.

**Suppressing Handshaking.** The COM command parameter HS=OFF eliminates the ACK/NAK response ("handshaking") from the protocol of a formatted transfer. Eliminating handshaking simplifies a data transfer at the risk of undetected errors. In an upload, when the host receives a block, its response must consist only of the prompt sequence if handshaking is off. In a download, the 8540 does not respond at all when it receives a block. However, the 8540 still verifies each checksum, and if any are found to be incorrect, the 8540 issues an error message when the transfer is complete. In Version 4.1 and later versions, COM displays an asterisk (signifying a good block) or an underscore (bad block) on the system terminal each time it receives a block, regardless of the HS parameter.

### Host Software Requirements for Formatted Transfers

Because formatted transfers require a formal protocol, a host computer used with the 8540 must be capable of

satisfying that protocol. In particular, the following programs are required on the host computer:

1. A **transmit program** that will transfer Tekhex blocks to the 8540.
2. A **receive program** that will accept Tekhex blocks from the 8540.
3. If you have object files in non-Tekhex format that you would like to transfer and use on the 8540, you must have a **conversion program** that will translate the object files into Tekhex format.

Notice that without handshaking, the protocol for a formatted transfer is the same as for an unformatted ASCII transfer, as described earlier in this subsection. Therefore, a simple copy command could serve as a transmit or receive program. However, the 8540 generally takes longer to process a message block than the host requires to send the next one, so the transmit program must pause enough between lines that no data is lost.

One way for the host to achieve the necessary delay is to transmit "fill" characters at the beginning of each message block. COM finishes processing one message block while it receives (and ignores) the fill characters at the beginning of the next block. You may use any characters you want for fill, so long as they differ from the host's end-of-line characters(s) and from the leading character of the message block. For transfers at 2400 baud, four to six fill characters should be sufficient; the actual number required depends on your host and baud rate.

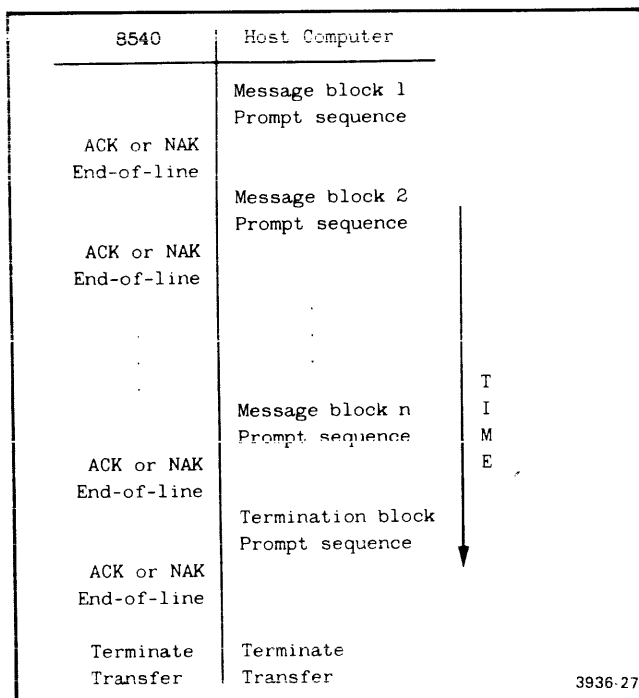


Fig. 4-6. Protocol for formatted downloads.

**Transmit Program.** This program downloads Tekhex-formatted files to the 8540, using the download protocol described earlier in this section. The general algorithm is as follows:

1. Read a Tekhex block from a file (error checking for valid format is optional).
2. Transmit the block followed by the predefined prompt sequence, and wait for an acknowledgement. The prompt sequence is defined in the COM command line, and the 8540 will not respond until it receives the prompt.
3. If the reply is a negative acknowledgement (NAK):
  - a. Increment the negative acknowledgement counter (this counter keeps track of how many successive transmission failures have occurred).
  - b. If the negative acknowledgement counter exceeds a maximum value that you have defined, terminate the transfer by sending an abort block.
  - c. Prepare to re-transmit the block that was incorrectly received and go to step 2.
4. If the reply is a positive acknowledgement (ACK):
  - a. If the block sent was a termination block, terminate the transfer.
  - b. Otherwise, go to step 1.

This algorithm is presented in pseudo-code in Fig. 4-7.

**Receive Program.** This program receives Tekhex files from the 8540, using the upload protocol described earlier in this section. The general algorithm is as follows:

1. Read a Tekhex block from the 8540.
2. Verify proper format. A Standard Tekhex block must begin with a slash and must not exceed 71 characters in length (not counting the end-of-line). An Extended Tekhex block must begin with a percent sign and must not exceed 255 characters in length (not counting the end-of-line).
3. If the block is a termination block,
  - a. Verify the checksum.
  - b. If an error is encountered, send a NAK followed by the pre-defined prompt sequence, and increment the NAK counter.
4. If the block is not a termination block,
  - a. Verify the byte count.
  - b. Verify the checksum(s).
  - c. If an error is encountered, increment the NAK counter and test to see if it exceeds a maximum value that you've selected. If it exceeds that value, terminate the transmission with an abort block; otherwise, send a NAK followed by the predefined prompt sequence to the 8540. The prompt sequence is defined in the COM command line, and is used by the 8540 to determine when the host computer is ready for input.
  - d. If no error is encountered, write the block to a file and send an ACK followed by the predefined prompt-sequence to the 8540.

```

END-OF-FILE = FALSE
WHILE NOT END-OF-FILE
.  READ TEKHEX BLOCK FROM FILE
.  IF TEKHEX BLOCK IS A TERMINATION BLOCK THEN END-OF-FILE = TRUE
.  NUMBER-OF-NAKS = 0
.  GOODTX = FALSE
.  REPEAT
.  .  TRANSMIT BLOCK
.  .  TRANSMIT PROMPT SEQUENCE
.  .  GET REPLY
.  .  IF REPLY IS AN ACK
.  .  THEN GOODTX = TRUE
.  .  ELSE
.  .  .  NUMBER-OF-NAKS = NUMBER-OF-NAKS + 1
.  .  .  IF NUMBER-OF-NAKS >= 5
.  .  .  THEN
.  .  .  .  TRANSMIT ABORT BLOCK
.  .  .  .  GOODTX = TRUE
.  .  .  .  END-OF-FILE = TRUE
.  .  .  END
.  .  END
.  UNTIL GOODTX = TRUE
END
EXIT

```

3936-28

Fig. 4-7. Algorithm for Tekhex transmit program.

This general algorithm is presented in pseudo-code in Fig. 4-8.

**Conversion Programs.** This discussion gives two algorithms for converting a file of object code into Tekhex format. The first algorithm is for conversion to Standard Tekhex,

and the second is for conversion to Extended Tekhex. Because of the wide variation in object file formats from assembler to assembler, it is possible to give only very general algorithms.

```

NAK-COUNTER = 0
END-OF-TRANSMISSION = FALSE
WHILE NOT END-OF-TRANSMISSION
. CLEAR ERROR FLAG
. READ A TEKHEX BLOCK FROM THE RECEIVE LINE
. IF THE BLOCK STARTS WITH A "/" AND BLOCK-LENGTH <= 71
. OR THE BLOCK STARTS WITH A "%" AND BLOCK-LENGTH <= 255
. THEN
. . IF BLOCK IS A TERMINATION BLOCK
. . THEN
. . . VERIFY CHECKSUM
. . . IF ERROR
. . . THEN SET ERROR FLAG
. . . ELSE
. . . . **NO ERROR IN TERMINATION BLOCK**
. . . . WRITE BLOCK TO FILE
. . . . END-OF-TRANSMISSION = TRUE
. . . . TRANSMIT ACK, END-OF-LINE
. . . . TRANSMIT PROMPT SEQUENCE
. . . . END
. . END
. . IF BLOCK IS A NOT A TERMINATION BLOCK
. . THEN
. . . VERIFY BYTE COUNT
. . . VERIFY CHECKSUM(S)
. . . IF ERROR
. . . THEN SET ERROR FLAG
. . . ELSE
. . . . **NO ERROR IN DATA OR SYMBOL BLOCK**
. . . . WRITE BLOCK TO FILE
. . . . TRANSMIT ACK, END-OF-LINE
. . . . TRANSMIT PROMPT SEQUENCE
. . . . NAK-COUNTER = 0
. . . . FND
. . . . END
. . END
. . ELSE
. . . **BLOCK FORMAT IS BAD**
. . . SET ERROR FLAG
. . . END
. . IF ERROR FLAG SET
. . THEN
. . . NAK-COUNTER = NAK-COUNTER + 1
. . . IF NAK-COUNTER >= NAK-LIMIT
. . . THEN
. . . . **IF NUMBER OF NAKS EXCEEDS NAK-LIMIT,
. . . . . ABORT THE TRANSMISSION**
. . . . TRANSMIT ABORT BLOCK
. . . . END-OF-TRANSMISSION = TRUE
. . . . END
. . . ELSE
. . . . TRANSMIT NAK, END-OF-LINE
. . . . TRANSMIT PROMPT SEQUENCE
. . . . END
. . . END
. . END
END
EXIT

```

3936-29

Fig. 4-8. Algorithm for Tekhex receive program.

*Standard Tekhex Conversion Program.*

1. Discard any header information and scan for the first block of code.
2. Read the block of code.
3. If there are no more blocks:
  - a. Generate a termination block.
  - b. Stop execution.
4. Strip off any non-object code information from the block.
5. Check the block for valid input format.
6. Start the output block with a slash.
7. Immediately after the slash, append the 4-digit load address for the block.
8. Determine the byte count *n*, where *n* is the number of bytes of object code in the block. Place the byte count in the output block immediately following the load address.
9. Calculate the first checksum as the sum of the individual hexadecimal digits of the load address and the byte count. Append this value immediately following the byte count.
10. Put the *n* bytes of object code into the block, immediately following the first checksum. While performing this step, calculate the second checksum as the sum, mod 256, of the  $2n$  hexadecimal digits in the object code.
11. Write the block to the output file.
12. Go to step 2.

This general algorithm is presented in pseudo-code in Fig. 4-9.

```

DISCARD HEADER INFORMATION
END-OF-FILE = FALSE
WHILE NOT END-OF-FILE
. READ BLOCK FROM INPUT FILE
. IF NO MORE BLOCKS
. THEN END-OF-FILE = TRUE
. ELSE
. . IF BLOCK IS A TERMINATION BLOCK
. . THEN
. . . SAVE TRANSFER ADDRESS FOR TEKHEX TERMINATION BLOCK
. . . END-OF-FILE = TRUE
. . . END
. . ELSE
. . . SAVE LOAD ADDRESS
. . . STRIP NON OBJECT CODE INFORMATION FROM BLOCK
. . . CHECK FOR VALID FORMAT
. . . IF ERROR
. . . THEN
. . . . OUTPUT ERROR MESSAGE
. . . . END-OF-FILE = TRUE
. . . . END
. . . ELSE
. . . . OUTPUT-BLOCK [0] = "/"
. . . . DETERMINE LOAD ADDRESS
. . . . OUTPUT-BLOCK [1..4] = 4-DIGIT LOAD ADDRESS
. . . . DETERMINE BYTE COUNT
. . . . OUTPUT-BLOCK [5..6] = 2-DIGIT BYTE COUNT
. . . . CALCULATE FIRST CHECKSUM
. . . . OUTPUT-BLOCK [7..8] = 2-DIGIT CHECKSUM
. . . . OUTPUT-BLOCK [9..2n+8] = OBJECT CODE
. . . . CALCULATE SECOND CHECKSUM
. . . . OUTPUT-BLOCK [2n+9..2n+10] = 2-DIGIT CHECKSUM
. . . . WRITE OUTPUT-BLOCK TO FILE
. . . . END
. . . END
. . END
. END
END
GENERATE TEKHEX TERMINATION BLOCK
EXIT

```

3936-30A

Fig. 4-9. Algorithm for Standard Tekhex conversion program.

*Extended Tekhex Conversion Program.* This algorithm makes the following assumptions:

- The object file may contain program symbols and section information as well as executable object code.
- All the usable information in each block is output to the Tekhex file before the next block is read from the object file. (You may find it more useful to save the program symbols in a table as you proceed through the object file, and generate your symbol blocks from the symbol table at the end of the program.)
- At most one data block and/or one symbol block is generated for each block read from the object file: there are no checks to verify that the maximum block length (255) has not been exceeded.
- The variable *n* is used to represent the number of digits in a variable-length number. The value of *n* may be constant throughout the program, or it may be different for each number you encode (for example, if you choose to eliminate leading zeros). Similarly, the variable *s* represents the number of characters in a variable-length symbol.

Here is the algorithm:

1. Discard any header information and scan for the first block of code or symbols.
2. Read the block.
3. If there are no more blocks:
  - a. For every undefined section, generate a symbol block with a section definition field.
  - b. Generate a termination block.
  - c. Stop execution.
4. Strip off any extraneous information.
5. Check the block for valid input format.
6. If the block contains executable object code:
  - a. Start the output block with a percent sign.
  - b. Set the block type to 6 (data block).
  - c. Place the load address in the block.
  - d. Place the object code in the block.
  - e. Determine the block length and checksum and put them in the block header.
  - f. Write the block to the output file.
7. If the block contains program symbols or section information:
  - a. Start the output block with a percent sign.
  - b. Set the block type to 3 (symbol block).
  - c. Place the name of the section that contains these symbols in the block.
  - d. If the base address and length of the section are available and have not been output previously, create a section definition field after the header field.
  - e. For each symbol in the block, create a symbol definition field containing the symbol, its type, and its value.
  - f. Determine the block length and checksum and place them in the block header.
  - g. Write the block to the output file.
8. Go to step 2.

This algorithm is presented in pseudo-code in Fig. 4-10.

```

DISCARD HEADER INFORMATION
END-OF-FILE = FALSE
WHILE NOT END-OF-FILE
.  READ BLOCK FROM INPUT FILE
.  IF NO MORE BLOCKS
.  THEN END-OF-FILE = TRUE
.  ELSE
.  .  IF BLOCK IS A TERMINATION BLOCK
.  .  THEN
.  .  .  SAVE TRANSFER ADDRESS FOR TEKHEX TERMINATION BLOCK
.  .  .  END-OF-FILE = TRUE
.  .  END
.  .  ELSE
.  .  .  CHECK FOR VALID FORMAT
.  .  .  IF ERROR
.  .  .  THEN
.  .  .  .  OUTPUT ERROR MESSAGE
.  .  .  .  END-OF-FILE = TRUE
.  .  .  END
.  .  END
.  END

```

3936.31

Fig. 4-10. Algorithm for Extended Tekhex conversion program. (part 1 of 2)

```

. . . ELSE
. . . IF BLOCK CONTAINS EXECUTABLE CODE
. . . THEN **GENERATE DATA BLOCK**
. . . . OUTPUT-BLOCK [0] = "%"
. . . . OUTPUT-BLOCK [3] = "6"
. . . . OUTPUT-BLOCK [6] = n
. . . . OUTPUT-BLOCK [7..6+n] = n-DIGIT LOAD ADDRESS
. . . . OUTPUT-BLOCK [7+n..6+n+2m] = m BYTES OF OBJECT CODE
. . . . OUTPUT-BLOCK [1..2] = 6+n+2m **2-DIGIT BLOCK LENGTH**
. . . . OUTPUT-BLOCK [4..5] = 2-DIGIT CHECKSUM
. . . . WRITE OUTPUT-BLOCK TO FILE
. . . . END
. . . IF BLOCK CONTAINS SYMBOLS OR SECTION INFORMATION
. . . THEN **GENERATE SYMBOL BLOCK**
. . . . OUTPUT-BLOCK [0] = "%"
. . . . OUTPUT-BLOCK [3] = "3"
. . . . OUTPUT-BLOCK [6] = s
. . . . OUTPUT-BLOCK [7..6+s] = s-CHARACTER SECTION NAME
. . . . p = 7+s **p POINTS TO NEXT AVAILABLE SPOT IN OUTPUT-BLOCK**
. . . . IF SECTION NOT ALREADY DEFINED
. . . . . THEN IF BASE ADDRESS AND LENGTH ARE AVAILABLE
. . . . . THEN **GENERATE SECTION DEFINITION FIELD**
. . . . . . OUTPUT-BLOCK [p] = "0"
. . . . . . OUTPUT-BLOCK [p+1] = n
. . . . . . OUTPUT-BLOCK [p+2..p+1+n] = n-DIGIT BASE ADDRESS
. . . . . . p = p+n+2
. . . . . . OUTPUT-BLOCK [p] = n
. . . . . . OUTPUT-BLOCK [p+1..p+n] = n-DIGIT SECTION LENGTH
. . . . . . p = p+n+1
. . . . . . MARK SECTION AS DEFINED
. . . . . . END
. . . . . . ELSE MARK SECTION AS UNDEFINED
. . . . . **GENERATE SYMBOL DEFINITION FIELDS**
. . . . . FOR EACH SYMBOL
. . . . . . OUTPUT-BLOCK [p] = SYMBOL TYPE
. . . . . . OUTPUT-BLOCK [p+1] = s
. . . . . . OUTPUT-BLOCK [p+2..p+1+s] = s-CHARACTER SYMBOL
. . . . . . p = p+s+2
. . . . . . OUTPUT-BLOCK [p] = n
. . . . . . OUTPUT-BLOCK [p+1..p+n] = n-DIGIT VALUE
. . . . . . p = p+n+1
. . . . . . END
. . . . . . OUTPUT-BLOCK [1..2] = p-1 **2-DIGIT BLOCK LENGTH**
. . . . . . OUTPUT-BLOCK [4..5] = 2-DIGIT CHECKSUM
. . . . . . WRITE OUTPUT-BLOCK TO FILE
. . . . . . END
. . . . . END
. . . . END
. . . END
. . . END
. . . END
END
FOR EACH UNDEFINED SECTION
GENERATE SYMBOL BLOCK WITH SECTION DEFINITION FIELD
GENERATE TEKHEX TERMINATION BLOCK
EXIT

```

3939-31

Fig. 4-10. Algorithm for Extended Tekhex conversion program. (part 2 of 2)



## The COM Command

The COM command specifies the details of the communication protocol that the 8540 is to use with your host computer. This discussion explains the parameters of the COM command.

All COM parameters are optional and may be entered in any order and in any combination that will tailor the response of the 8540 to the host computer.

### Echo Parameter (E=)

This parameter selects remote or local echoing of characters entered from the 8540 system terminal. **E=R** (remote echo) indicates that the host computer echoes the characters entered from the 8540 system terminal. **E=L** (local echo) indicates that the host computer does not echo characters entered from the 8540 system terminal, and thus the echo must be provided locally (by the 8540). The default value is **E=R**.

### Linefeed Parameter (L=)

This parameter allows the 8540 system terminal to include or omit a linefeed on the system terminal after a carriage return is sent to the terminal. **L=I** (include linefeed) indicates that the host computer does not output a linefeed to the terminal after a carriage return is entered, so the 8540 supplies the linefeed. **L=O** (omit linefeed) indicates that the host computer sends a linefeed to the terminal after a carriage return is entered. The default value is **L=O**.

### Prompt Parameter (P=)

This parameter specifies the prompt sequence used by the host computer communicating with the 8540. The prompt sequence represents the characters that the host computer

sends when it is ready to receive data. This is not necessarily the system prompt of the host computer. The default value for this parameter is **no prompt sequence**. If you omit the prompt parameter, the 8540 does not wait to receive a prompt sequence from the host computer before sending data.

The prompt sequence consists of an even number of up to 32 hexadecimal digits, representing up to 16 ASCII characters. You may not enter spaces between the digits.

The 8540 ignores the prompt sequence except during data transfers.

### Turnaround Delay Parameter (T=)

Some computers require a delay between displaying their prompt sequence and accepting input. The turnaround delay parameter allows you to ensure that, after receiving the prompt sequence from the host, the 8540 will wait for the appropriate amount of time before it sends a reply.

The T parameter has a range of 00 to FF (hexadecimal), where each unit has a value of 100 milliseconds. For example, **T=04** sets the turnaround delay to 400 milliseconds. The default value for this parameter is **T=00**. (For COM Version 4.1 and later versions, the delay is specified in 10-millisecond units, so **T=04** represents 40 milliseconds.)

### Character Transmission Format (M=)

This parameter specifies the character transmission format required by the host computer when it receives or transmits data. Table 4-13 lists the transmission characteristics corresponding to values for M. The default value for this parameter is **M=4**.

**Table 4-13**  
Character Transmission Formats Corresponding to Values for M

M	Start Bits	Stop Bits	Data Bits	Significant Bits	Parity	Parity Bits	Total Transmitted Bits
0	1	2	7	7	even	1	11
1	1	2	7	7	odd	1	11
2	1	1	7	7	even	1	10
3	1	1	7	7	odd	1	10
4	1	2	8	7 or 8 <sup>a</sup>	none	0	11
5	1	1	8	7 or 8 <sup>a</sup>	none	0	10
6	1	1	8	7 or 8 <sup>a</sup>	even	1	11
7	1	1	8	7 or 8 <sup>a</sup>	odd	1	11

<sup>a</sup> High-order bit is set to 0 by the 8540 in ASCII transfers.

### Error Check Parameter (C=)

This parameter allows you to override the COM error checking facilities on the remote communication port. If you specify **C=I** (ignore errors), the 8540 ignores any condition that causes a communication error, and continues execution. If you enter **C=T** (terminate on error), control leaves COM and passes to OS/40 if any communication error occurs. The default value for this parameter is **C=T**.

### Load Module Format (F=)

This parameter specifies the message block format to be used in formatted uploads. If this parameter is omitted, blocks are transmitted in Extended Tekhex format. **F=T** specifies Standard Tekhex format. COM accepts message blocks in either Tekhex format, regardless of the F parameter.

### End-of-Line Parameter (EOL=)

This parameter specifies the character sequence used by the host to mark the end of a line of text, such as a host command, a line of ASCII data, or a Tekhex message block.

The character sequence is specified as an even number of up to 32 hexadecimal digits, representing 16 characters. The default end-of-line parameter is **EOL=0D** (a carriage return).

### Handshaking Parameter (HS=)

The parameter **HS=OFF** eliminates the ACK/NAK response from the protocol of a formatted transfer. In a download, COM normally responds with an ACK or NAK each time it receives a message block from the host. When handshaking is OFF, COM gives no response. However, checksums are still computed and checked, and if any are found to be incorrect, COM issues an error message when the transfer is complete. In an upload, COM expects no ACK/NAK response from the host.

Turning off handshaking has no effect on the P (prompt sequence) and T (turnaround delay) parameters. COM still looks for the host's prompt sequence (if any) and waits the indicated turnaround delay (if any) before continuing.

### Character Substitution Parameter (SUB xx=yy)

This parameter specifies one or more character substitutions to be performed by COM on data transmitted to or from the host. Each substitution construct consists of a pair of character codes (represented as two-digit hexadecimal numbers) separated by an equal sign. When the first character is received from the host, COM substitutes the second character. When the second character is to be sent to the host, COM substitutes the first. Multiple substitution constructs can be strung together, separated by slashes — **SUB 74=24/75=25/78=30**, for example.

### Byte Count Parameter (B=)

This parameter specifies the number of bytes of object code that COM will place in each data block during formatted uploads. (Each byte is encoded as two hexadecimal digits.) This parameter is supported only for COM Version 4.1 and later versions. This parameter is ignored if you specify **F=T**. The byte count is specified as a two-digit hexadecimal number in the range 01-76. The default count is 20 (32 decimal). Any number outside the range 01-76 yields a byte count of 76 (118 decimal).

## Hardware Requirements

### General Information

The 8540 I/O device for intersystem communication is an ACIA (Motorola 6850 Asynchronous Communication Interface Adapter) connected to REMOTE jacks J101 and J102 on the back of the 8540. Both jacks are RS-232-C compatible; these are the only jacks for which COM interface procedures are implemented.

Jack J101 is a 25-pin male plug designed as an interface for data terminal equipment (DTE). Jack J102 is a 25-pin female plug designed as an interface for data communication equipment (DCE). The jack you select and the setting of the MODE SELECT switch on the 8540 rear panel determine how the 8540 uses the various RS-232-C signals. See Table 4-14.

**Table 4-14**  
**Signals at 8540 Remote Port**

Signal Description	Jack J101 (DTE) MODE SELECT Switch Setting			Jack J102 (DCE) MODE SELECT Switch Setting	
	DTE1	DTE2	CNTL (L)	DCE	CNTL (L)
Pin 1—GRD	Protective Ground				
Pin 2—TX	Output: Data from 8540	Output: Data from 8540	Output: Data from 8540	Input: Data from external computer	Input: Data from external computer
Pin 3—RX	Input: Data from external computer	Input: Data from external computer	Input: Data from external computer	Output: Data from 8540	Output: Data from 8540
Pin 4—RTS	Output: Goes high when 8540 has data to send	Output: Goes high when 8540 has data to send	Output: Always high	Input: Must be high before 8540 accepts data	Input: Ignored
Pin 5—CTS	Input: Must be high before 8540 sends data	Input: Ignored	Input: Ignored	Output: Goes low when 8540 sends data	Output: Always high
Pin 6—DSR	Input: Ignored	Input: Must be high before 8540 sends data	Input: Ignored	Output: Always high	Output: Always high
Pin 7—GRD	Signal Ground				
Pin 8—DCD	Input: Must be high before 8540 accepts data	Input: Must be high before 8540 accepts data	Input: Ignored	Output: Always high	Output: Always high
Pin 20—DTR	Output: Always high	Output: Always high	Output: Always high	Input: Must be high before 8540 sends data	Input: Ignored

The type of equipment you have determines which jack you will use. In general, if the external computer's RS-232-C I/O port has a female plug, you'll use jack J101 on the 8540 and set the MODE SELECT switch to DTE1 or DTE2. If the external computer's I/O jack has a male plug, you'll use jack J102 on the 8540 and set the MODE SELECT switch to CNTL (L) or DCE.

**Baud Rate**

The baud rate at which the REMOTE jacks operate is switch selectable from 110 baud to 9600 baud. The baud switch is located immediately to the right of jack J102 on the 8540 rear panel.

The maximum recommended transfer rate for an unformatted transfer is 2400 baud. Formatted transfers should proceed successfully at up to 9600 baud, depending on the limitations of the host.

**Modem Considerations**

When using a modem to connect the 8540 with an external computer, be sure that the modem is operating in full duplex mode.

**Remote Port Status Register**

An 8-bit status register is associated with the ACIA device. When you are not transferring files, you can display the contents of the status register by entering the command line

```
(null)S <CR>
```

The 8540 will respond with a 2-digit hexadecimal number representing the status of the ACIA. Table 4-15 describes what each of the bits in the status register represents.

**Table 4-15**  
Remote Port Status Register Bits

Bit No. <sup>a</sup>	Set by	Cleared by	Use
0	Byte received by ACIA.	Reading received byte in ACIA.	Determine whether byte received.
1	Byte moved to ACIA shift-out register.	Writing byte into transmit data register.	Determine whether transmit data register ready for next byte.
2	Data carrier signal lost. No connection at REMOTE port.	Carrier signal detected at ACIA.	Test of communications connection.
3	Incoming CTS signal low.	Incoming CTS signal high.	Determine whether external device is ready to receive data.
4	Byte in ACIA has error in stop bits (framing error).	Reading byte or receiving next byte from remote communication port.	Test for valid data byte.
5	Byte received from external device and previous byte not yet read. Previous byte is overwritten.	Reading received data byte	Test for lost data.
6	Parity error in received byte.	Reading current byte or receiving next byte.	Test for a valid data byte.
7	Transmit data register empty or receipt of byte into ACIA and interrupt enabled.	For a transmit interrupt, writing the next byte clears the interrupt. For a receive interrupt, reading the received byte clears the interrupt.	Test for occurrence of interrupts.

<sup>a</sup> Bit number 7 is the most significant bit.

## Establishing Communication

To prepare the 8540 for operation with a host computer, you must first establish a physical connection between the two computers. To do this, plug one end of an RS-232-C cable into a host computer RS-232-C compatible terminal port. Connect the other end of the RS-232-C cable to the REMOTE port on the 8540 rear panel. (Refer to the "Hardware Requirements" discussion earlier in this subsection to determine which jack and MODE SELECT switch setting are appropriate.)

Now use the COM command to establish communication with the host. After you have established initial communication, the 8540 acts as a terminal to the host. You can now transfer data to and from the 8540; see the following discussion of "Data Transfers."

## Exiting from COM

When the 8540 is acting as a terminal to the host, you may exit the COM subsystem at any time by entering the command

```
(null)(escape)
```

where (escape) is the ESC key on your terminal.

If you are performing a transfer and want to exit the COM subsystem, you must first abort the transfer by pressing the BREAK key. This will cause the 8540 to resume acting as a terminal to the host. You can then exit COM by entering (null)(escape).

## Data Transfers

Once your 8540 is acting as a terminal to the host computer, you can initiate the transfer of data between the two computers. The general format for uploading and downloading data is given in the following paragraphs. You can abort a data transfer at any time by pressing the BREAK key on the 8540 system terminal.

### Formatted Transfers

The following procedures transfer data between Tekhex files on the host and program/prototype memory in the 8540.

**Formatted Download.** The following command line downloads a Tekhex load module to 8540 program/prototype memory. Program symbols in the load module are placed in the 8540's symbol table, for use in symbolic debug.

```
host-command(null) <CR>
```

**host-command** is the host computer command to initiate the execution of your host computer program for formatted downloads.

**(null)** indicates that the rest of the command line is a command to the 8540. Since there is no command following the null character, the 8540 recognizes that the transfer is a formatted download.

With COM Version 4.1 and later versions, COM displays an asterisk on the system terminal and sends an ACK to the host each time a block is received successfully. If the block contains errors, COM displays an underscore and sends a NAK to the host.

In earlier versions, COM displays an asterisk each time it receives a block. If the block contains errors, COM sends a NAK to the host; otherwise COM sends an ACK and displays the load address and byte count of the block.

When COM receives a symbol block successfully, it reports the section name to the system terminal. If the block contains errors, COM may store erroneous symbols or values in the symbol table before detecting the error. If you see COM NAK in a symbol block, beware of errors in the symbol table. You may need to reSElect the emulator (to clear the symbol table) and download the file again.

When the transfer is complete, the following message appears on the 8540 system terminal:

```
DNLOAD:
```

**Formatted Upload.** The following command line uploads selected blocks of 8540 program/prototype memory to the host in Tekhex format.

```
host-command(null)lowadd1 hiadd1  
lowadd2 hiadd2 ... transadd <CR>
```

**host-command** is the host computer command that initiates the execution of your host computer program for formatted uploads.

**(null)** indicates that the rest of the command line is a command to the 8540.

**lowadd1** is the low address of a block of 8540 program memory to be transferred.

**hiadd1** is the high address of a block of 8540 program memory to be transferred.

**lowadd2** is the low address of another block of 8540 program memory to be transferred.

**hiadd2** is the high address of another block of 8540 program memory to be transferred.

**transadd** is the beginning execution address for the code in the transferred blocks.

No program symbols are uploaded from the 8540.

When the transfer is complete, the following message appears on the 8540 system terminal:

```
UPLOAD:
```

## Unformatted Transfers

The following procedures copy data between files on the host and I/O devices connected to the 8540.

**Unformatted ASCII Download.** Here is the command syntax for an unformatted ASCII download:

```
host-command(null)>device <CR>
```

**host-command** is the host computer command to display a text file on the terminal.

**(null)** indicates that the rest of the command line is a command to the 8540.

**>** indicates that the transfer is an unformatted ASCII download.

**device** is the 8540 device (for example, LPT) to which the downloaded data is to be written.

When the transfer is complete, the following message appears on the 8540 system terminal:

```
RIOT:
```

**Unformatted Binary Download.** Here is the command syntax for an unformatted binary download:

```
host-command(null)=device <CR>
```

**host-command** is the host computer command to dump a binary file to the terminal.

**(null)** indicates that the rest of the command line is a command to the 8540.

**=** indicates that the transfer is an unformatted binary download.

**device** is the 8540 device (for example, PPTP) to which the binary data is to be written.

When the transfer is complete, the following message appears on the 8540 system terminal:

```
RIOT:
```

**Unformatted ASCII Upload.** Here is the format for an unformatted ASCII upload from the 8540 to the host computer:

```
host-command(null)<device <CR>
```

**host-command** is the host computer command that reads text from the terminal and transfers the input to a file on the host.

**(null)** indicates that the rest of the command line is a command to the 8540.

**<** indicates that the transfer is an unformatted ASCII upload.

**device** is the name of the 8540 device (for example, PPTR) that provides the data to be transferred to the host computer.

When the transfer is complete, the following message appears on the 8540 system terminal:

```
RIOT:
```

The 8540 does not send an end-of-file message to the host. When you see the RIOT: message, enter your host's end-of-file character at the system terminal.

**Unformatted Binary Upload.** Here is the format for an unformatted binary upload from the 8540 to the host computer:

```
host-command(null)+device <CR>
```

**host-command** is the host computer command that reads binary data from the terminal and transfers the input to a file on the host.

**(null)** indicates that the rest of the command line is a command to the 8540.

**+** indicates that the transfer is an unformatted binary upload.

**device** is the name of the 8540 device (for example, PPTR) that provides the data to be transferred to the host computer.

When the transfer is complete, the following message appears on the 8540 system terminal:

```
RIOT:
```

The 8540 does not send an end-of-file message to the host. When you see the RIOT: message, enter your host's end-of-file character at the system terminal.

## Troubleshooting

### Establishing Initial Communication

Symptom	Cannot establish initial communication.
Check	<ol style="list-style-type: none"> <li>1. Verify that the optional COM interface software has been installed in your 8540.</li> <li>2. Check the COM command line for correct syntax. Be sure that the parameters were specified correctly.</li> </ol>
Symptom	Cannot log on or communicate with host computer.
Check	<ol style="list-style-type: none"> <li>1. Check the REMOTE port baud rate switch on the 8540 rear panel. The rate selected must match that of the modem or output port of the host computer.</li> <li>2. Verify that the MODE SELECT switch is set properly for operation with your equipment.</li> <li>3. Check all RS-232-C control signals/lines in the interface.</li> <li>4. Verify the connection to the correct 8540 rear panel REMOTE port, J101 or J102.</li> <li>5. Be sure that the M parameter in the COM command corresponds to the host computer's data transmission format.</li> <li>6. Exit from COM and return to OS/40 by</li> </ol>

entering (null)(escape) on the 8540's terminal. Then restart the system.

Symptom	Random characters displayed on the terminal, or improper display when attempting to log on to the host computer.
---------	--

Check	<ol style="list-style-type: none"> <li>1. Check for proper baud rate selection on the 8540 rear panel REMOTE port.</li> <li>2. Be sure that the M parameter in the COM command corresponds to the host computer's data transmission format.</li> </ol>
-------	--

### Attempting a Formatted Download to the 8540

Symptom	No reaction after typing download transfer request.
Check	<ol style="list-style-type: none"> <li>1. Check the download transfer request for compatibility with the required host computer command.</li> <li>2. If E=R was specified in the COM command, check to see if the host computer is echoing all carriage returns sent to it. If the host computer does not echo the carriage returns at the end of the download transfer request command, the 8540's download routine will not execute properly.</li> </ol>
Symptom	Host computer's download program is inoperative.
Check	<ol style="list-style-type: none"> <li>1. Check the download program I/O assignments. If the program output is not assigned to the terminal corresponding to the line to the 8540, the 8540 will not receive the output.</li> <li>2. Verify that the download command line ended with a null character.</li> <li>3. Verify that the download program in the host computer is actually starting execution.</li> <li>4. Verify that the file being downloaded is in Tekhex format.</li> <li>5. Verify that the host computer is sending an end-of-line sequence (as specified by the EOL parameter of the COM command) at the end of each Tekhex block.</li> </ol>
Symptom	Download program transmits one or more blocks and then hangs up.

Check

1. Verify that the prompt sequence was correctly specified in the COM command.
2. Test the download program by omitting the null character from the command line used to start the download. The output from the host computer should now be displayed on the terminal. Enter positive and negative acknowledgments manually and step the program through the download sequence.
3. Verify that the turnaround delay parameter, T, is correctly specified in the COM command.

2. Verify that the byte count logic in the host computer upload program is correct.
3. Verify that the checksum logic in the host computer upload program is correct.
4. Verify that the input buffer in the host upload program is of sufficient length.

Symptom

The 8540 repeatedly sends negative acknowledgments for the first block and causes the download program to abort.

Check

1. Verify that the Tekhex block contains the correct checksum(s).

### Attempting an Unformatted ASCII or Binary Download to the 8540

Symptom

No reaction after typing download transfer request.

Check

1. Verify that the host computer's portion of the command is a valid host-computer-to-terminal copying command.
2. Verify that the 8540's portion of the command has no embedded spaces and is of the form

(null)>device <CR> (ASCII)  
 or  
(null)=device <CR> (binary)

### Attempting a Formatted Upload from the 8540

Symptom

No reaction after typing transfer request.

Check

1. Verify that the transfer request command was entered correctly.
2. If E=R was specified in the COM command, check to see if the host computer is echoing all carriage returns sent to it. If the host computer does not echo the carriage returns at the end of the download transfer request command, the 8540's upload routine will not execute properly.

Symptom

Host computer's upload program is inoperative.

Check

1. Verify that the prompt sequence was correctly specified in the COM command.
2. Verify that the upload program on the host computer is sending a positive or negative acknowledgement after the first message block is received.

Symptom

The upload program on the host computer repeatedly sends negative acknowledgements, and eventually aborts.

Check

1. Verify that the protocol in the host computer upload program is correct.

3. If using a modem, check to see if any data is being received by the modem.
4. All of the data may have been transmitted, but the 8540 may not have received the the end-of-transmission prompt. Recall that a unique prompt sequence must be transmitted after the file has been transmitted. That prompt sequence must match the prompt sequence that you defined when you entered the the COM command. If the 8540 is waiting for the end-of-transmission prompt, you can manually exit the transfer process by pressing the BREAK key.

Symptom

Download begins, but at some point the file starts appearing on the 8540 system terminal.

Check

1. Verify that the prompt sequence that you defined when you entered the COM command appears only at the end of the file to be downloaded. If the prompt sequence appears anywhere within the body of the file to be transferred, the 8540 will stop writing information to the selected device and start writing information to the system terminal.

## Attempting an Unformatted ASCII or Binary Upload from the 8540

**Symptom** No reaction after typing upload transfer request.

**Check** 1. Verify that the host computer's portion of the command is a valid terminal-to-host-computer transfer command.  
2. Verify that the 8540's portion of the command has no embedded spaces and is of the form

**or** (null)<device <CR> (ASCII)

**or** (null)+device <CR> (binary)

3. Verify that the prompt sequence specified in the COM command is correct. If you specify a prompt sequence on an ASCII upload, the host must respond with that prompt every time it receives a line from the 8540.  
4. Verify that a valid 8540 device was specified as the source of the upload.  
5. Restart the 8540.

**Symptom** Partial data transfer with the loss of the first part of the data. Also, large gaps in the received file.

**Check** 1. Verify that the turnaround delay parameter, T, is big enough. If the T parameter was not specified, it may have to be specified to prevent loss of the beginning of the data. Insufficient turnaround delay causes the loss of an initial portion of each transmission made.  
2. Verify that the proper P (prompt sequence) parameter was specified in the COM command line. If no prompt sequence is specified, the 8540 may be transmitting information before the host computer is ready to accept input.

## Commands for Intersystem Communication

These spaces are provided for you to write the command sequences that you will use with your 8540 and your host computer.

### Establishing Initial Communication

### Formatted Download

### Formatted Upload

### Unformatted ASCII Download

### Unformatted Binary Download

### Unformatted ASCII Upload

### Unformatted Binary Upload



## Section 5 EMULATION

	Page		Page
<b>Introduction</b> .....	5-1	<b>The Trigger Trace Analyzer</b> .....	5-13
<b>The Emulator</b> .....	5-1	Overview .....	5-13
<b>Emulation Modes</b> .....	5-1	Events and Triggers .....	5-14
<b>Selecting the Target Processor</b> .....	5-3	Breakpoints .....	5-16
<b>Basic Emulation Tasks</b> .....	5-3	General Purpose Counters .....	5-16
Memory .....	5-3	Acquisition Memory .....	5-17
Executing the Program .....	5-4	TTA Summary .....	5-19
Monitoring the Program .....	5-4		
Modifying the Program .....	5-5		
Timing .....	5-5		
SVCs .....	5-6		
Interrupts .....	5-6		
<b>Considerations for Modes 1 and 2</b> .....	5-6		
Memory .....	5-7		
Executing the Program .....	5-7		
SVCs .....	5-7		
Interrupts .....	5-7		
<b>Symbolic Debug</b> .....	5-7		
Using Symbolic Debug .....	5-9		
Other Symbolic Debug Commands .....	5-9		
<b>The Memory Allocation Controller</b> .....	5-9		
Memory Spaces .....	5-11		
Memory Allocation .....	5-11		
MAC Commands .....	5-11		

### TABLES

Table No.		
5-1	Emulation Modes .....	5-3
5-2	Counter Output Options .....	5-17

### ILLUSTRATIONS

Fig. No.		
5-1	Emulation modes 0, 1, and 2 .....	5-2
5-2	Trace display without symbolic debug .....	5-8
5-3	Trace display with symbolic debug .....	5-8
5-4	The top plane bus .....	5-10
5-5	68000 and Z8001/Z8002 memory spaces ..	5-10
5-6	Generating a trigger signal .....	5-14
5-7	Acquisition Memory .....	5-18

## Section 5

# EMULATION

### INTRODUCTION

The 8085A emulator demonstration run in the Learning Guide of this manual showed you how to load, execute, and monitor a program in emulation mode 0. This section, Emulation, discusses the following topics:

- the three emulation modes
- the basic emulation tasks, many of which were presented in the Learning Guide
- additional considerations for emulation modes 1 and 2
- use of symbolic debug
- the Memory Allocation Controller (MAC) option
- the Trigger Trace Analyzer (TTA) option

This section describes the common uses of many OS/40 commands. For information on the syntax and limitations of these commands, and for examples of their use, refer to the Command Dictionary of this manual. For a more detailed description of the TTA, refer to the Trigger Trace Analyzer Users Manual.

### THE EMULATOR

The **emulator** (or **emulator processor**) is one or more circuit boards that reside in the 8540. The emulator

contains an **emulating microprocessor** on which your program executes, plus control circuitry that allows you to start, stop, and monitor program execution using OS/40 commands. For most emulators, the emulating microprocessor is of the same type as the microprocessor being emulated.

The term **emulator registers** refers to the registers of the emulating microprocessor. The emulator is said to be "halted" whenever the emulating microprocessor is not executing your program.

As your program executes, the emulator monitors the emulating microprocessor and intervenes when a break or service call (SVC) occurs.

### EMULATION MODES

In order to execute a microprocessor-based program, you normally need the following pieces of hardware:

- a microprocessor on which to execute the program
- a clock to drive the microprocessor
- memory for the program to reside in
- I/O facilities: external circuitry to control or respond to the microprocessor.

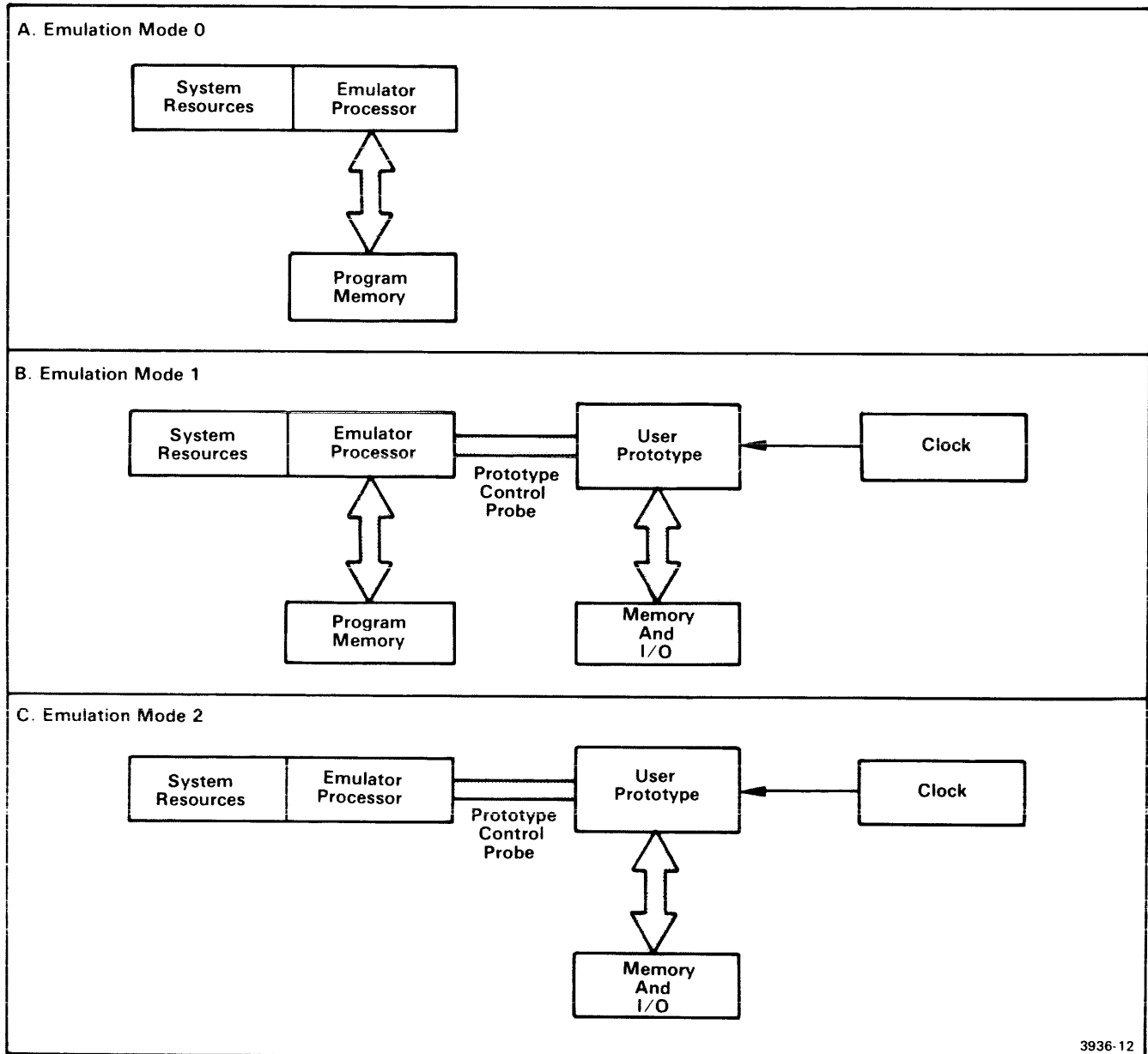


Fig. 5-1. Emulation Modes 0, 1, and 2.

This figure illustrates the differences between the three emulation modes. Mode 0 uses the emulator clock, program memory, and SVCs (which are part of the system resources). Mode 1 uses the prototype clock, either program or prototype memory, and either SVCs or prototype I/O. Mode 2 uses the prototype clock, prototype memory, and either SVCs or prototype I/O.

So that you may execute your program even before your prototype hardware is available, the 8540 provides or simulates all of the necessary hardware. Once your prototype is built, you can begin transferring the necessary hardware functions from the 8540 to your prototype. You

indicate which of these functions are handled by the 8540 and which are handled by the prototype by specifying the **emulation mode**. Refer to Fig. 5-1 during the following discussion.

The three emulation modes are:

- Mode 0 (system mode). Mode 0 uses the emulator clock and program memory. Program I/O is handled through service calls (SVCs). Until your prototype is connected to the emulator, your program can execute only in mode 0.
- Mode 1 (partial emulation mode). Mode 1 uses the prototype's clock, and is the only mode in which the program may access both program memory and prototype memory. An internal memory map determines whether a particular address refers to program memory or prototype memory. Program I/O is handled with prototype I/O facilities, SVCs, or both.
- Mode 2 (full emulation mode). Mode 2 uses the prototype's clock, memory, and I/O facilities. Some emulators support the use of SVCs in mode 2.

In all three emulation modes, the emulator takes the place of the microprocessor that will eventually reside in the working prototype. In modes 1 and 2, a prototype control probe must connect the prototype to the emulator.

**NOTE**

*Some emulators do not support SVCs in emulation modes 1 or 2. Refer to the Emulator Specifics section of this manual to determine whether your emulator supports the use of SVCs in modes 1 and 2.*

Table 5-1 summarizes the three emulation modes.

The "Basic Emulation Tasks" subsection describes features that are common to all three emulation modes. Special considerations for modes 1 and 2 are presented later in the section.

## SELECTING THE TARGET PROCESSOR

Because your 8540 System ROM Board may contain software for more than one microprocessor, you must specify which microprocessor your program is written for before you attempt to execute a program.

The SEL command selects the system software that is appropriate for your microprocessor. For example, the command SEL 8085 specifies that subsequent microprocessor-dependent commands refer to the 8085A emulator. Certain emulator-related OS/40 commands cannot be executed until an emulator has been selected. Those commands are listed in the discussion of the SEL command, in the Command Dictionary of this manual.

## BASIC EMULATION TASKS

This subsection discusses the following aspects of emulation:

- memory
- executing the program
- monitoring the program
- modifying the program
- timing
- SVCs
- interrupts

Except where otherwise stated, this information applies to all three emulation modes. Special considerations for modes 1 and 2 are presented later in this section.

To select emulation mode 0 (system mode), enter the following command:

```
> EM 0
```

In mode 0, your program executes in program memory independently of the prototype hardware. Through SVCs in your program, you can use the I/O facilities of the 8540 to simulate the real-world interfaces of the prototype microprocessor.

### Memory

In emulation mode 0, the 8540's program memory takes the place of the memory in your prototype.

**Table 5-1**  
Emulation Modes

Emulation Mode	Memory	I/O Facilities	Clock	Required Hardware
0	Program	SVCs	Emulator	Emulator
1	Program and/or Prototype, depending on memory map	Prototype and (if supported) SVCs	Prototype	Emulator, Prototype Control Probe
2	Prototype	Prototype and (if supported) SVCs	Prototype	Emulator, Prototype Control Probe

In order to simulate ROM modules, you can use the RO parameter of the MAP command to designate blocks of program memory as read-only. If your program tries to store data into a read-only area, OS/40 generates a break and the protected area remains unchanged.

If your 8540 has only 32K bytes of program memory, your program can still access addresses in the range 8000-FFFF. You can use the AL command to make blocks of existing program memory represent addresses in the 8000-FFFF range.

If you have a Z8001/Z8002 emulator or 68000 emulator, you should have the **Memory Allocation Controller (MAC)** option installed in your system in order to use addresses outside the range of existing program memory. The MAC option is discussed later in this section.

## Executing the Program

The demonstration run in your Emulator Specifics supplement contains instructions for assembling a source program on an 8560 or other host computer. Use your host computer to prepare binary object code from your source program, then download the object code to your 8540. Use the G command to begin program execution. You can use any of several methods to stop program execution:

- Insert a HALT instruction (or the corresponding instruction, if any, for your microprocessor) into your program.
- Insert an SVC such as Exit Program (function 1A) or Abort Program (function 1F) into your program.
- Set a breakpoint, using the OS/40 BK command. (If you have the TTA option installed, you may use the TTA events and breakpoints.)
- Type CTRL-C while the program is executing.

After the program has been stopped by one of these methods, use the G command to resume execution.

If you need to execute a program segment repeatedly to detect an intermittent bug, use the -R or -L parameter of the G command.

## Monitoring the Program

In case your program does not execute as expected, OS/40 provides several ways to locate the problem. The following paragraphs describe some methods of monitoring the program.

## The TRA Command

You can use the TRA (TRAcE) command to produce a running display of the principal registers as the program executes. Each time your program executes an instruction, OS/40 displays a trace line that lists the instruction and the resulting contents of the registers.

Once you have narrowed the problem down to one area of the program, you can specify that only that area be traced. By using the -S option of the TRA command, you can step through critical areas one instruction at a time: you type G when you are ready to execute the next instruction.

You can obtain an abbreviated trace that shows the logical flow of your program. The TRA JMP option lets you trace only jump instructions (unconditional jumps, subroutine calls, and conditional jumps when the conditions are satisfied).

### NOTE

*When any TRAcE selections are in effect, your program executes at less than normal speed, even in those parts of the program that are not traced.*

## The BK Command

You can use the BK (BreaK) command to cause your program to break when a particular address is accessed. By setting a breakpoint at the end of a program segment, you can suspend execution to verify that the segment has executed correctly. If you suspect that your program is straying outside a particular block of data or instructions, you can find out by setting breakpoints at the boundaries of the block. To clear a breakpoint, use the CLR parameter of the BK command.

## TTA Option

If you have a Trigger Trace Analyzer, you have many more tracing and breakpoint options available to you.

The TTA automatically maintains a trace of the processor's address bus, data bus, and control signals. The TTA can monitor up to 64 signals at a time. You can view this trace at any time by using the DISP command.

You can also define breakpoints and other actions based on criteria that would be too complicated to define if you used only the BK command. For more information on the TTA, refer to the "Trigger Trace Analyzer" discussion later in this section.

## Diagnostic Messages

You can insert high-level language statements or assembly language SVCs (such as Write ASCII and Wait) into your source code, so that your program issues its own diagnostic messages as it executes.

## Other Commands

We've looked at several methods of monitoring the program. Here are some additional commands you can use:

- The DS (Display Status) command displays the full set of registers. (For some emulators, this information is part of the standard trace display. For others, you can obtain this information by specifying the `-L` option with the TRA command.)
- The D (Dump) command displays the contents of a specified section of memory in both hexadecimal and ASCII formats.
- The DI (Disassemble) command translates object code in memory back into assembly language mnemonics. DI is used to display blocks of instructions; D is used to display blocks of data.

## Modifying the Program

Once you have located a problem in your program, you have several ways to fix it. One way is to correct the source code and compile, assemble, link, and download the program over again. Often, however, you can first verify the effect of the change by modifying the object code or registers directly.

To change the object code or other information in memory, you can use one of the following commands:

- The P (Patch) command stores a string of bytes or words at a specified location in memory.
- The EX (EXAm) command lets you modify an area of memory byte by byte or word by word. Use EX instead of P if you want to change only selected bytes, or if you want to see the previous contents of each byte before you change it.
- The F (Fill) command fills an area of memory with a specified value or string of values. One common use of F is to zero out a block of memory.

To change register contents, use the S (Set) or RESET commands. S changes only the registers you select; RESET simulates a reset signal to your microprocessor.

You can obtain a working program by modifying your object code, then using the SAV or COM command to upload the modified code into a load file. Be sure to update your source

code to match your object code. One way to keep track of the modifications to your object code is to use the LOG command to create a log of your debugging session.

## Timing

In emulation mode 0, the clock signal to the emulating microprocessor is provided by the emulator. For most emulators, the clock frequency in mode 0 is the same as the maximum frequency allowed for the prototype clock in modes 1 and 2. Refer to your Emulator Specifics supplement for this clock information.

### Timing a Program Segment

In order to optimize the execution time of your program, it is useful to be able to measure the execution times of program segments. You can obtain an accurate approximation of a program segment's execution time even before the prototype is available. Simply time the segment in mode 0 and adjust for the difference between the emulator clock rate and the prototype clock rate.

To time a segment, use the 8540's **program clock** or the TTA's **general purpose counters**.

- The program clock increments every 100 milliseconds during program execution. The CLOCK command turns this clock on and off and displays the clock value. The clock value can also be accessed through the Read Program Clock SVC (function 11).
- The general purpose counters, which are explained later in this section, provide more precision and versatility than the program clock.

#### NOTE

*When the program clock is running, your program executes at slightly less than normal speeds.*

### Factors That Affect Timing

The following activities cause the emulating microprocessor to wait or halt, and so may disrupt the timing of a program segment:

- Executing an SVC halts the emulating microprocessor until the SVC is completed.
- Tracing with the TRA command causes the emulator to execute your program one instruction at a time. After each instruction, the emulating microprocessor pauses while OS/40 determines whether a trace line is to be displayed.
- A keystroke on the terminal momentarily pauses the emulator.

If you must time a program segment that contains such interruptions, use a general purpose counter of the TTA and specify a very small time unit, such as clock cycles. The general purpose counter will not count during the SVC or trace pause.

## SVCs

Service calls (SVCs) allow your program to read and write text and binary data, and perform many other OS/40 functions. Depending on your application, you may be able to use SVCs to help simulate the prototype circuitry your program will have to deal with.

The Service Calls section of this manual describes each type of SVC function in detail, and also describes the four parts of an SVC request. These parts are:

- an I/O instruction that invokes the SVC;
- a pointer in the Service Request Block (SRB) vector, which provides a logical link from each SVC invocation to the appropriate SRB;
- a Service Request Block (SRB) that specifies the function to be performed; and
- an I/O buffer, if needed.

The Service Request Block (SRB) vector is a table of addresses that point to the beginning of each SRB. For most emulators, the SRB vector is assumed to begin at byte 40 of program memory. If your program needs that memory area for other uses, you can choose a different area for the SRB vector. Use the SVC command to notify OS/40 of the new location.

Similarly, the I/O instruction that initiates the SVC usually must specify a port in the range F0-F7, but you can use the SVC command to allocate a different range of ports for this purpose.

## Phasing Out SVCs

Another use of the SVC command is to turn SVCs ON and OFF. Since your prototype should handle its own I/O, you will eventually want to eliminate the SVCs from your program. The command SVC OFF suppresses execution of SVCs: SVC invocations are simply executed as I/O instructions.

The SVC OFF feature is most useful for programs that use SVCs for purely diagnostic purposes. If the SVCs perform some vital function for the program, such as obtaining required input, you must develop an alternate form of the program in which the SVC functions are taken over by the prototype.

### NOTE

*When you set a breakpoint at an instruction that invokes an SVC, the break occurs **after** the SVC is executed.*

*For some emulators, execution of the SVC may prevent OS/40 from breaking at the desired instruction.*

## Interrupts

In mode 0, you have no way of triggering an interrupt line during program execution. However, here is one way to simulate a randomly timed interrupt:

1. Start the program.
2. Type CTRL-C to break the program at a random location.
3. You will want your program to return to this location when it has finished processing the interrupt. Obtain the value of the program counter from the break display line; use a command such as P to store this value where your program can retrieve it (on the stack, for example).
4. Use the G command to jump to a section of the program that simulates the interrupt. Afterward, your program retrieves the old program counter value and continues from where you typed CTRL-C.

Notice that the emulator is always halted when such an "interrupt" occurs.

## CONSIDERATIONS FOR MODES 1 AND 2

In emulation modes 1 and 2, your program executes in the prototype under 8540 control. The prototype must be connected to the emulator via the prototype control probe.

### CAUTION

*Before you begin emulation in modes 1 and 2, refer to your Emulator Processor Installation Manual for the following information:*

- a. *electrical specifications for the probe-prototype interface: power, impedance, drive, etc.; and*
- b. *instructions on how to connect the prototype control probe to the emulator and to the prototype.*

*Also refer to the Emulator Specifics section of this manual for possible timing discrepancies between the prototype control probe and the microprocessor that it replaces.*

Once your prototype is connected to the emulator, and the prototype and 8540 are turned on, verify that your program still executes correctly in mode 0 under this arrangement. If it does not, the prototype may be causing the problem, or the prototype control probe may be damaged. If you suspect damage, refer to your Emulator Processor Installation Manual for verification procedures.

Next, use mode 1 to verify that the memory modules in your prototype are working correctly. When the prototype's memory and I/O are working correctly, you can proceed to mode 2 to solve the more subtle problems.

## Memory

OS/40 maintains an internal table, called the **memory map**, that indicates which address blocks refer to program memory and which blocks refer to prototype memory. You use the MAP command to control this memory map. For most emulators, memory is mapped in 128-byte blocks.

OS/40 consults the program/prototype memory assignment of the memory map only in emulation mode 1. In mode 0, only program memory is used; in mode 2, only prototype memory is used. However, changing the emulation mode does not change the memory map or memory contents.

The program/prototype memory assignment (defined by the emulation mode and memory map) is observed by OS/40 when it executes your program, and also when it executes any command that accesses memory (for example, LO, P, or D).

When you change the memory map, be sure to move the corresponding blocks of object code between program and prototype memory. You can use the LO or COM command to reload your entire program, or you can use MOV to copy selected areas of memory.

### CAUTION

*In emulation mode 1, memory-write operations performed in program memory are also performed in the corresponding bytes of user prototype memory, for most emulators.*

## Memory Protection

Unlike program memory, prototype memory cannot be write-protected by the MAP command, for most emulators. The procedure "Breaking When the Program Writes in a Designated Memory Area," in the Operating Procedures section of this manual, shows you how to use the TTA to detect a write to a read-only area of prototype memory.

## Executing the Program

In modes 1 and 2, no break occurs when your program executes a HALT instruction (or the corresponding instruction, if any, for your microprocessor), even if you set a breakpoint at that instruction.

## SVCs

Refer to the Emulator Specifics section of this manual to determine whether your emulator supports SVCs in modes 1 or 2.

In mode 1, the instruction sequence that invokes the SVC can reside in either program memory or prototype memory. The Service Request Block (SRB) vector, Service Request Block (SRB), and optional I/O buffer(s) must reside in program memory.

In mode 2, all parts of the SVC must be located in prototype memory.

In modes 0 and 1, the I/O instruction that invokes the SVC must be followed by one NOP instruction. In mode 2, the I/O instruction that invokes the SVC must be followed by two NOP instructions.

## Interrupts

Your prototype may attempt to interrupt the emulating microprocessor when it is stopped during a break, SVC, or TRAcE activity. Most emulators do not detect such interrupts.

## SYMBOLIC DEBUG

### NOTE

*Some emulators do not support symbolic debug. Refer to your Emulator Specifics supplement for this information.*

Symbolic debug allows you to use the symbols from your assembly language program in place of hexadecimal numbers when you are debugging. You can use your symbols as parameters in all OS/40 commands that accept expressions as parameters. For example, let's assume you have a segment of assembly language code having label START at its starting address (100H) and label FINISH at its ending address (220H). Then the following two forms of the TRA command are equivalent:

```
> TRA ALL 100 220
```

```
> TRA ALL START FINISH
```



The command line may also include the name of the assembler section in which the symbol is found. This is useful if the same symbol is used in more than one section and it is not defined as a global symbol. (Refer to your Assembler Users Manual for information about sections and global symbols.) For example, if `START` is in section `TEST1` and `FINISH` is in section `TEST2`, the `TRA` command could be entered:

```
> TRA ALL TEST1:START TEST2:FINISH
```

In the output display, symbolic debug replaces each hexadecimal address with either a symbol (taken from the label field of your program), or its position relative to the

start of the current section (as shown in the location counter field of your assembler listing.) Figure 5-2 contains an example of a trace display without symbolic debug. Figure 5-3 contains an example of a trace display with symbolic debug.

The "section + offset" feature is not implemented for absolute sections. (Refer to your Assembler Core Users Manual for information about absolute sections.)

For most 8-bit emulators, symbols are truncated to eight characters when displayed.

LOC	INST	MNEM	R	OPER	X/PC	EADD	RA	RB	XREG	SP	CC
001000	CE0010	LDX		0010			00	00	0010	0001	D0
001003	5F	CLR	B				00	00	0010	0001	D4
001004	A600	LDA	A	00	+0010=0010		00	00	0010	0001	D4
001006	AB01	ADD	A	01	+0010=0011		00	00	0010	0001	D4
001008	C900	ADC	B	00			00	00	0010	0001	D4
00100A	AB02	ADD	A	02	+0010=0012		00	00	0010	0001	D4
00100C	7E1055	JMP		1055		1055	00	00	0010	0001	D4
001055	C900	ADC	B	00			00	00	0010	0001	D4
001057	AB03	ADD	A	03	+0010=0013		00	00	0010	0001	D4

3936-13

Fig. 5-2. Trace display without symbolic debug.

This is an example of a trace display without symbolic debug.

LOC	INST	MNEM	R	OPER	X/PC	EADD	RA	RB	XREG	SP	CC
SECTION: (Section1)											
+000	CE0010	LDX		0010			00	00	0010	0001	D0
+003	5F	CLR	B				00	00	0010	0001	D4
LABEL001	A600	LDA	A	00	+0010=0010		00	00	0010	0001	D4
+006	AB01	ADD	A	01	+0010=0011		00	00	0010	0001	D4
LABEL002	C900	ADC	B	00			00	00	0010	0001	D4
+00A	AB02	ADD	A	02	+0010=0012		00	00	0010	0001	D4
+00C	7E1055	JMP		1055		1055	00	00	0010	0001	D4
SECTION: (Section2)											
LABEL003	C900	ADC	B	00			00	00	0010	0001	D4
+007	AB03	ADD	A	03	+0010=0013		00	00	0010	0001	D4

3936-14

Fig. 5-3. Trace display with symbolic debug.

With symbolic debug, each hexadecimal address is replaced with either a label or its offset from the beginning of a section. In this example, Section1 begins at 1000H and section2 begins at 1050H.

## Using Symbolic Debug

To make your program symbols available for use in symbolic debug, you must perform the following steps:

1. If you are using a Tektronix assembler:
  - a. All assembly source files that have symbols to be referenced by symbolic debug must include the assembler directive LIST DBG. This directive causes the assembler to output the symbols to the object module, which is used by the linker.
  - b. Include the DEBUG or `-d` command option when invoking the linker. For the A Series linker, enter:

```
> LINK
* LOAD loadfile
* DEBUG
```

For the B Series linker, enter:

```
$ link -o loadfile -d . . .
```

This directive causes the symbols and their values to be stored in the load module.

- c. Enter the SYMLO command, which downloads the symbols and their values from the load module to the symbol table in 8540 system memory. SYMLO accepts files produced by B Series assemblers, the B Series linker, or the A Series linker, but does not accept unlinked files produced by A Series assemblers.

2. If you are not using a Tektronix assembler, your program symbols and their values must be placed in Extended Tekhex symbol blocks and downloaded with the rest of your load modules, using the COM command.

## Enabling Symbolic Debug

The SYMD command enables and disables symbolic debug in the displays of commands such as TRA and DI. Refer to the Command Dictionary for syntax and parameters of the SYMD command.

## Other Symbolic Debug Commands

The following paragraphs describe the other commands used with symbolic debug:

- The SYMLO command extracts your symbols from the load module and loads them into the symbol table. (The symbol table holds approximately 1000 symbols.) You can specify that the symbols be loaded from selected sections only.

- The ADDS command enables you to add symbols to the table during a debugging session. You may not add section names with this command. If you have not already loaded your symbols from the load module using SYMLO or COM, any symbols you add with ADDS are placed in a default section called NO.SECTION. If you later use SYMLO, the values in NO.SECTION are lost.

- The REMS command deletes symbols from the symbol table.

- The SYMB command gives the symbolic equivalent of a hexadecimal address. For example, using values from Fig. 5-3:

```
> SYMB 1055H <CR>
```

```
1055H=LABEL003
```

```
> SYMB 1057H <CR>
```

```
1057H=SECTION2+007
```

- The CALC command finds the value associated with a symbol. For example:

```
> CALC symbol
```

```
symbol=30FFH
```

Refer to the Command Dictionary for details of the parameters and syntax of symbolic debug commands.

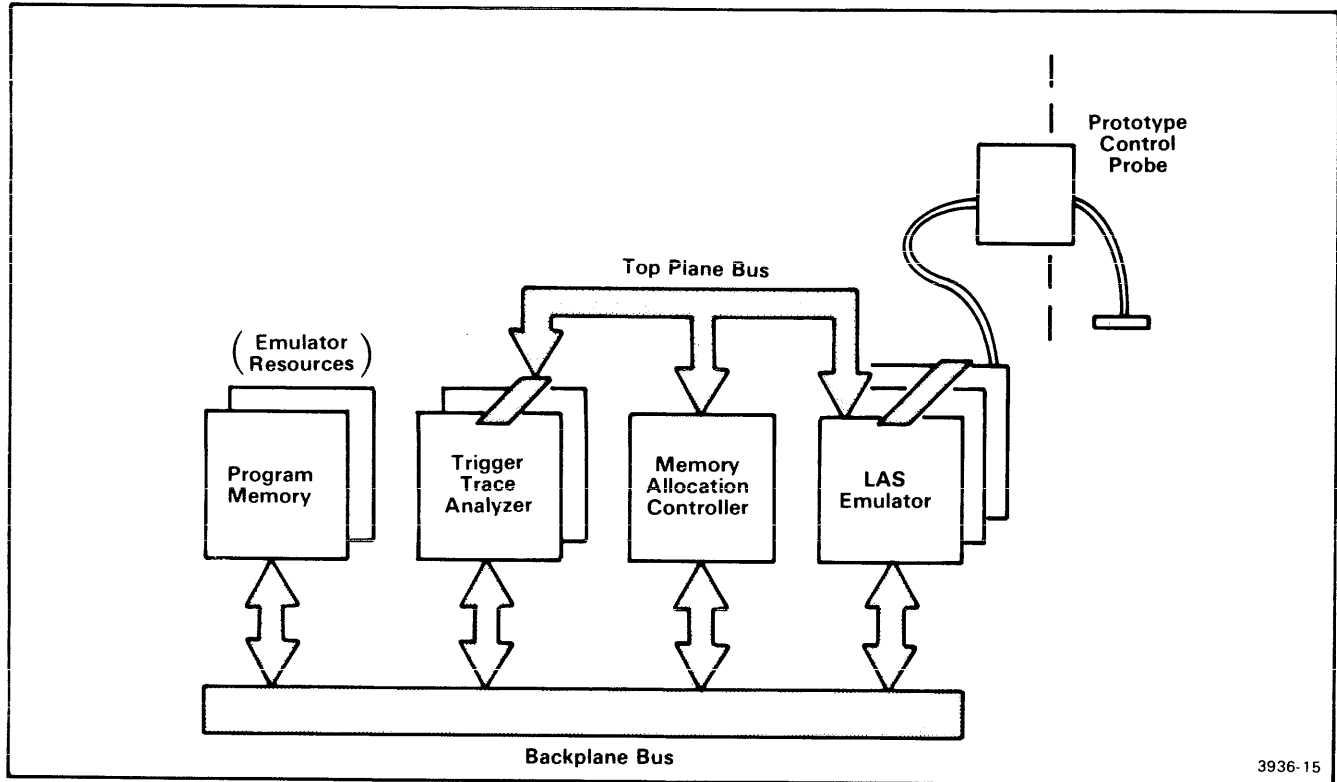
## THE MEMORY ALLOCATION CONTROLLER

The Memory Allocation Controller (MAC) is a hardware option that permits the expanded addressing capabilities of emulators such as the Z8001/Z8002 and 68000 to operate within the confines of 8540 program memory. (The 8086 emulator performs this function internally and does not need the MAC board to execute the memory allocation commands.)

The MAC option consists of:

- the MAC circuit board that inserts into the 8540 mainframe
- the top plane bus that connects the MAC board to the emulator and the Trigger Trace Analyzer (TTA) option

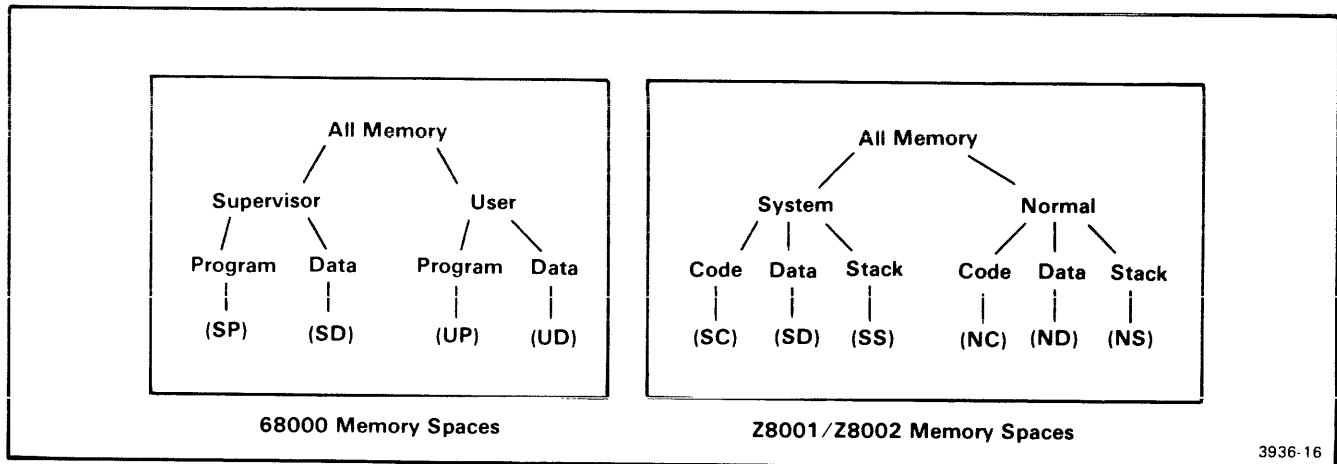
Figure 5-4 illustrates the position of the top plane bus.



3936-15

Fig. 5-4. The top plane bus.

The top plane bus carries signals used by Large Address Space emulators, the Memory Allocation Controller, and the Trigger Trace Analyzer circuit boards.



3936-16

Fig. 5-5. 68000 and Z8001/Z8002 Memory Spaces.

This figure illustrates the memory spaces available with the 68000 and Z8001/Z8002 microprocessors. You can specify up to four memory spaces for the 68000, and up to six memory spaces for the Z8001/Z8002.

## Memory Spaces

Both the 68000 and Z8001/Z8002 emulators allow you to designate memory areas for system (supervisor) use only or for user (normal) use only. You can further specify that these system or user areas be reserved for data, program code, or (for the Z8001/Z8002), the stack. These divisions are called **memory spaces**. Figure 5-5 illustrates how the memory spaces are divided and named for the 68000 and Z8001/Z8002 emulators. Figure 5-5 also includes the abbreviations used in the memory allocation commands. Refer to your 68000 or Z8001/Z8002 Emulator Specifics supplement for more information on memory spaces.

The memory spaces are differentiated by control signals from the microprocessor. For this reason, two or more blocks of memory assigned to different memory spaces can have the same addresses.

## Memory Allocation

Program memory is allocated in 4K-byte blocks. Each 32K of program memory in your system consists of eight blocks. The MAC board maintains an internal allocation map to translate **logical addresses** from the emulator into **physical addresses** in program memory.

## MAC Commands

The following paragraphs describe the commands associated with memory allocation. AL and DEAL affect program memory; MEM and NOMEM affect prototype memory.

You can set a default memory space value with the MEMSP command. If you enter one of the memory allocation

commands without a memory space qualifier, the command defaults to the MEMSP value.

**The AL (Allocate) command** assigns logical addresses to 4K-byte blocks of program memory. The command line can specify one or more memory space qualifiers, or none. The following examples use a 68000 emulator in a system configured with 64K of program memory.

```
> AL 38000
1 BLOCK(S) ALLOCATED      038000      038FFF
```

This command allocates one 4K-byte block of program memory to logical addresses beginning at 38000H. OS/40 returns the number of blocks allocated, and the block boundaries.

Entering AL without parameters displays a list of the allocated memory, including the memory space assignments: SP and SD are supervisor program and supervisor data, respectively. UP and UD are user program and user data, respectively. In these examples, the default allocation includes all memory spaces.

```
> AL
038000      038FFF      SP SD UP UD
1 BLOCK(S) ALLOCATED      15 BLOCK(S) FREE
```

The display includes the number of blocks allocated and the number of blocks still available.

Allocations are cumulative. The command sequence in Display 5-1 allocates one 4K-byte block to supervisor program and supervisor data, and a different 4K-byte block to user program and user data. The display from the AL command includes the allocation from the previous example.

```
> AL SP:SD:4000 4FFF
1 BLOCK(S) ALLOCATED      004000      004FFF
> AL UD:UP:4000 4FFF
1 BLOCK(S) ALLOCATED      004000      004FFF
> AL
004000      004FFF      SP SD .. ..
004000      004FFF      .. .. UP UD
038000      038FFF      SP SD UP UD } ← previously allocated
3 BLOCK(S) ALLOCATED      13 BLOCK(S) FREE
```

Display 5-1

The command sequence shown in Display 5-2 allocates one block of memory to each memory space, though each memory space has the same addresses: 1000 to 1FFF.

If you attempt to allocate more memory than you have available, the following error message is displayed:

No more Program memory available

When you allocate more than one block of memory in one command, AL displays the allocation block by block. In the example in Display 5-3, assume that you have no other memory allocated.

You must use the DEAL command to deallocate one or more blocks before you try to allocate again.

The **DEAL (Deallocate) command** removes a previous allocation, and makes one or more 4K blocks available to be

```

> AL SP:1000
1 BLOCK(S) ALLOCATED      001000  001FFF

> AL SD:1000
1 BLOCK(S) ALLOCATED      001000  001FFF

> AL UP:1000
1 BLOCK(S) ALLOCATED      001000  001FFF

> AL UD:1000
1 BLOCK(S) ALLOCATED      001000  001FFF

> AL
001000  001FFF      SP  ..  ..  ..
001000  001FFF      ..  SD  ..  ..
001000  001FFF      ..  ..  UP  ..
001000  001FFF      ..  ..  ..  UD
004000  004FFF      SP  SD  ..  ..
004000  004FFF      ..  ..  UP  UD
038000  038FFF      SP  SD  UP  UD } ← previously allocated

7 BLOCK(S) ALLOCATED      9 BLOCK(S) FREE
    
```

Display 5-2

```

> AL SP:UP:0 2FFF
3 BLOCK(S) ALLOCATED      000000  002FFF

> AL SD:UD:0 2FFF
3 BLOCK(S) ALLOCATED      000000  002FFF

> AL
000000  000FFF      SP  ..  UP  ..
000000  000FFF      ..  SD  ..  UD
001000  001FFF      SP  ..  UP  ..
001000  001FFF      ..  SD  ..  UD
002000  002FFF      SP  ..  UP  ..
002000  002FFF      ..  SD  ..  UD

6 BLOCK(S) ALLOCATED      10 BLOCK(S) FREE
    
```

Display 5-3

reallocated. You can remove all previous allocations with the `-A` parameter, or you can specify the address range to be deallocated.

*NOTE*

*DEAL does not use the default memory space values from the MEMSP command: if no memory spaces are specified with an address, DEAL deallocates all memory spaces.*

*The default condition for AL and DEAL is that no memory is allocated.*

The **NOMEM (No Memory) command** informs the emulator that no memory exists in the prototype for a given block of addresses. Like the `AL` command, the `NOMEM` command accepts memory space qualifiers, and designates memory in 4K-byte blocks. Entering `NOMEM` without parameters displays the current state of non-existent memory. Both `NOMEM` and `MEM` display the status in terms of memory spaces. For example:

```
> NOMEM 201000
> NOMEM
```

```
INVALID USER MEMORY ADDRESSES
UD:  201000    201FFF
UP:  201000    201FFF
SD:  201000    201FFF
SP:  201000    201FFF
```

If your program attempts to access non-existent memory, a break is generated and the trace line includes the message: "NO-PMEM" or "NO-UMEM".

The **MEM (Memory) command** informs the emulator that the prototype **does** contain memory at a given block of addresses. The `MEM` command is used to reverse a previous `NOMEM` command. You can use address parameters and memory space qualifiers with the `MEM` command. Entering `MEM` without parameters displays the current state of existent memory.

*NOTE*

*The default condition for MEM and NOMEM is that all prototype memory exists.*

## THE TRIGGER TRACE ANALYZER

This subsection summarizes the use of the Trigger Trace Analyzer (TTA). For more detailed information and some examples of how the TTA is used, refer to the Trigger Trace Analyzer Users Manual.

The TTA is an optional hardware product that enhances the emulation capabilities of your 8540. The TTA consists of the following modules:

- two TTA circuit boards that insert into the 8540 mainframe
- two TTA interconnect cables that physically connect the two boards
- a Data Acquisition Interface unit that mounts into the 8540 rear panel
- a Data Acquisition Probe that connects the Data Acquisition Interface to user-selected circuits in the prototype
- the optional top plane bus that connects the TTA to the MAC board and certain emulators. (Refer back to Fig. 5-4, which illustrates the top plane bus.)

The Trigger Trace Analyzer Installation Guide explains how to install the TTA circuit boards and Data Acquisition Interface unit. The Trigger Trace Analyzer Users Manual explains how to connect the test clips of the Data Acquisition Probe to your prototype.

## Overview

The following paragraphs are an overview of the TTA option. The TTA features are discussed in detail later in this subsection.

The TTA is a real-time debugging tool. Your program executes at normal speed while the TTA monitors the buses and certain control signals. The TTA allows you great control in specifying which combinations of signals constitute an **event**, and what actions follow when an event occurs. The TTA can store up to 255 bus transactions in its Acquisition Memory, allowing you to capture a window of data during program execution. The TTA is capable of precise timing with a variety of source signals.

The TTA includes the following features:

- the ability to define four **events** based on combinations of input data from the buses and signals monitored;
- four **triggers** that can break program execution, start or stop counters, or signal other instruments;
- four general purpose **counters**; and
- an **Acquisition Memory** that can store up to 255 bus transactions.

We'll discuss each of these features in turn.

In this subsection, the term "input data" refers collectively to the following 64 bits of information monitored by the TTA:

- the address bus (up to 24 bits)
- the data bus (8 or 16 bits)
- the 8 test clips on the Data Acquisition Probe (up to 8 bits)
- the emulator-dependent bus signal interface (up to 11 bits from the emulator)
- the external event qualifier, via a BNC input on the Data Acquisition Interface (1 bit)
- counter output signals (4 bits)

## Events and Triggers

Many actions performed by the TTA are based on **events** and **triggers**. An event is the simultaneous occurrence of one or more specified conditions in the input data. The EVE command and several related commands are used to define up to four events.

Each trigger has an associated event comparator and general purpose counter, which share the trigger channel number. A trigger is enabled by the output of its event

comparator ANDed with the output of its general purpose counter. Figure 5-6 illustrates the relationships between an event, the counter, and the trigger. Each trigger can cause a breakpoint, increment a counter, or signal an external device (such as a logic analyzer) via a BNC output on the Data Acquisition Interface panel.

## Defining An Event

You can select any of several event control commands to specify which input data constitutes an event. These event control commands correspond to the parameters of the EVE command. Once you are experienced with the TTA, you can use the EVE command to specify all the input data on one command line. For the full syntax of TTA commands, refer to the Command Dictionary in the Trigger Trace Analyzer Users Manual.

## Event Control Commands

Each of the following commands can use any or all of the four trigger channels. All of the commands can use the -S and -C breakpoint flags. (Breakpoints are discussed later in this subsection.)

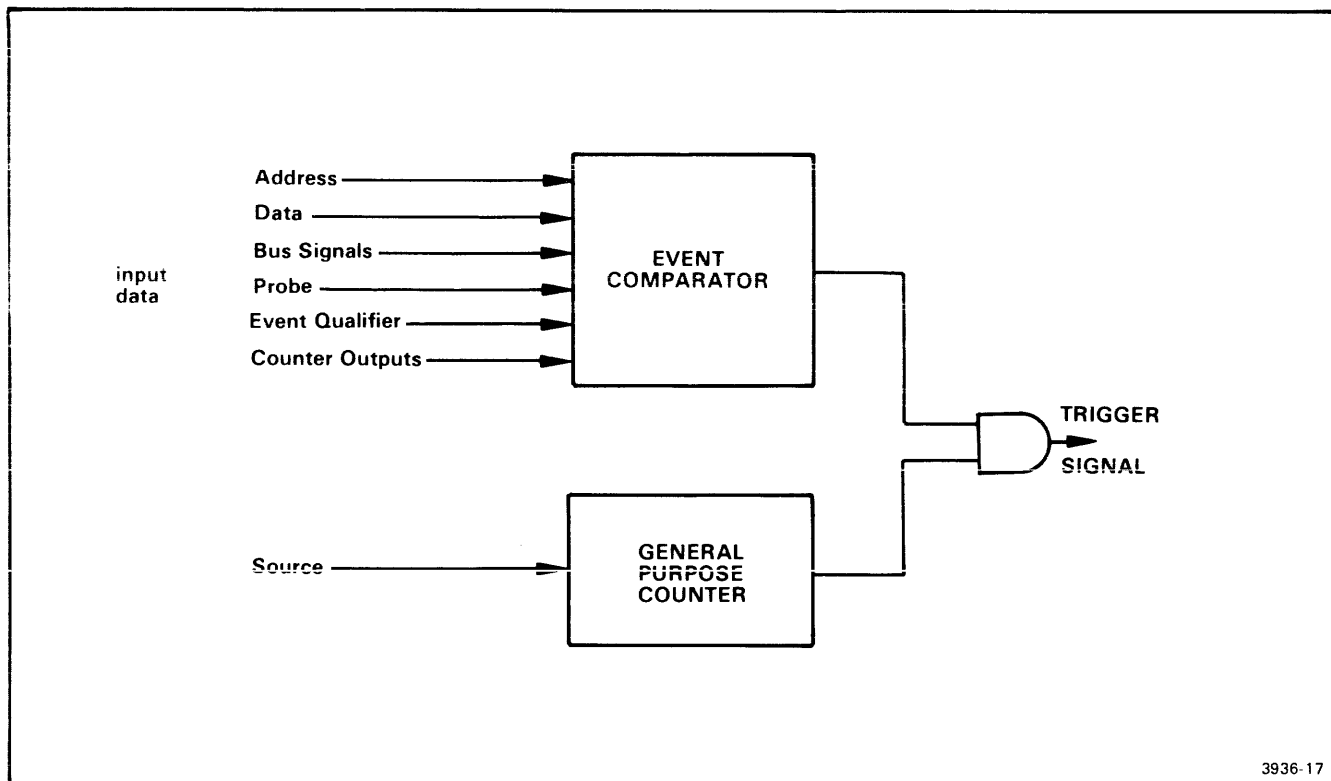


Fig. 5-6. Generating a trigger signal.

The output of the event comparator is ANDed with the output of the general purpose counter to produce the trigger signal.

The **AD command** is used to define a specific address or range of addresses as an event.

- > AD 1 105E
- > AD 2 500 530

These commands specify that event 1 occurs whenever the program accesses address 105E, and that event 2 occurs whenever the program accesses an address within the range 500 to 530, inclusive.

The AD command can include a **-N** command modifier that defines the event as anything other than the value(s) specified.

- > AD -N 4 1000 10FF

This command defines event 4 as any address outside the range 1000 to 10FF.

The **BUS command** allows you to select which bus signals are recognized as an event. The signals are emulator-specific. All of the signals selected are ANDed together—watch out for impossible combinations such as **F I** (fetch AND I/O operation simultaneously).

- > BUS 3 NF

This command defines event 3 as any non-fetch bus transaction.

The **CTR command** defines an event as a pattern of the output of the four counters. The pattern can include 1's, 0's or X's (don't-cares). For example, the command

- > CTR 1 10X0

causes event 1 to occur when counter 1 is high, and counters 2 and 4 are low.

The **DATA command** defines an event as the occurrence on the data bus of a value or one of a range of values. Like the AD command, it can include a **-N** command modifier that defines the event as anything other than the value(s) specified.

- > DATA 3 0F3
- > DATA -N 2 10 3FF

In the first command, event 3 occurs whenever the data bus contains the value F3. In the second command, event 2 occurs whenever the data bus contains a value outside the range 10-3FF.

The **PRO command** refers to the probe inputs via the Data Acquisition Probe and the Data Acquisition Interface Panel. The argument can be a two-digit hexadecimal number, or eight individual bits. You may include "don't-care" digits or bits (X) in hexadecimal or binary values, respectively. Entering **PRO CLR** makes all bits "don't-cares." The default radix for the PRO command is binary.

- > PRO 3 5XH
- > PRO 1 0X1XX0X10

#### NOTE

*Any number that does not begin with a digit in the range 0-9 must include a leading 0; otherwise OS/40 will attempt to evaluate the expression as a symbol and an error will be generated.*

The **QUA command** detects an input from the Event Qualifier BNC on the Data Acquisition Interface Panel. This input is user-controlled. The parameter for this command is a 0 or 1, or any expression that evaluates to a 0 or 1.

### The EVE Command

The EVE command lets you specify all event control conditions on one command line. Each EVE command parameter corresponds to one of the event control commands we've discussed. The EVE command may include the following parameters:

- **A** = Address—Corresponds to the AD command
- **AN** = Address Not—Corresponds to the AD -N command
- **B** = Bus—Corresponds to the BUS command
- **C** = Counter Output—Corresponds to the CTR command
- **D** = Data—Corresponds to the DATA command
- **DN** = Data Not—Corresponds to the DATA -N command
- **P** = Probe—Corresponds to the PRO command
- **Q** = Qualifier—Corresponds to the QUA command

The parameters of an event are cumulative. When a parameter is set, either with the EVE command or one of the event control commands, it remains in effect until cleared. The command

- > EVE 1 B=M WT

changes only the B (Bus) parameter. All other parameters remain unchanged. The command

- > EVE 1 CLR B=M WT

clears all previously specified conditions for event 1 and defines event 1 as any memory-write bus signal.

### Displaying Trigger Status

The **TS (Trigger Status) command** displays the current programming of the triggers and their associated events and counters. The **-E** parameter causes only the event portion of the trigger(s) to be displayed. The **-C** parameter causes only the counter portion of the trigger(s) to be displayed. You can select the triggers to be displayed. TS entered with no parameters displays the status of all the triggers. Refer to your Trigger Trace Analyzer Users Manual for TS command examples.



## Clearing All Trigger Conditions

The **TCLR (Trigger CLearR)** Command returns to the default condition the EVE, COU, and BRE commands associated with a given trigger channel. For example, the command **TCLR 1** is equivalent to entering

```
> EVE 1 CLR; COU 1 CLR; BRE 1 CLR
```

The command line may specify more than one trigger channel to be cleared, or that ALL be cleared.

The **-X** option resets the entire TTA. The command **TCLR -X** is equivalent to entering

```
> EVE ALL CLR; COU ALL CLR; BRE ALL CLR;
> CONS CLR; ACQ ALL
```

The contents of the Acquisition Memory are not altered by this command.

## Consecutive Events

In addition to triggering on individual events, it is possible to trigger on the consecutive occurrence of different events. By using the **CONS** command, events can be linked together so that the occurrence of an event arms the comparator of the following event. If the following event(s) do not occur on the specified bus cycle, then all events are reset and detection resumes with the first event in the sequence. The trigger is generated on the occurrence of the last event in the sequence.

The **CONS** command requires you to select one bus mode in which all the events are considered. The bus modes are:

- **CYC**—all bus cycles are allowed;
- **FET**—only fetch cycles are considered;
- **EMU**—only emulator-dependent bus cycles are considered.

### NOTE

*FET and EMU mode are not implemented on all emulators. Refer to your Emulator Specifics supplement to determine whether these functions are available on your emulator.*

As an example, the following command sequence detects a low-to-high transition in the prototype, following an instruction fetch:

```
> EVE 1 CLR B=F P=0XXXXX0XX
> EVE 2 CLR P=0XXXXX1XX -S
> CONS CYC 12
```

The first command defines EVE 1 as a fetch AND a logic low on probe input 2. EVE 2 is defined as a logic high on probe input 2. The **CONS** command says that EVE 2 will trigger a

break (the **-S** parameter) **only** if EVE 2 occurs on the cycle following EVE 1.

You can direct the TTA to trigger on any of the following event sequences:

```
12 123 1234
23 234 2341
34 341 3412
41 412 4123
```

You may specify two sequences concurrently, such as 23 and 41. The sequences are independent of each other; however both must use the same bus mode.

## Breakpoints

You can set breakpoints with the **BRE** command or with the **-S** or **-C** options of the various event control commands. The **-S** option produces a standard break, which stops execution of your program. The **-C** option prints a trace line and your program continues execution.

To remove breakpoints, you must use the **CLR** parameter of the **BRE** command, in conjunction with a trigger number. You can clear all the breakpoints by entering:

```
> BRE ALL CLR
```

To view the status of the breakpoints, enter the **BRE** command without parameters.

## The General Purpose Counters

Each of the four triggers has a general purpose counter associated with it. The counters are controlled by the **COU** command. Each counter has a **value** which you display using the **TS** command, and an **output** which must be high in order for the associated trigger to be generated.

## The COU Command

The **COU** command has five parameters. They are:

- **V** = Value
- **S** = Source
- **O** = Output
- **G** = Gate
- **R** = Restart

**Value (V)** is used to initialize the counter to the designated value. **V** can be any expression that evaluates to the range 0 to 65535 if incrementing, or 1 to 65535 if decrementing. The default radix for **V** is decimal.

**Source (S)** is **what** the counter counts. You may select any one of the following sources for each counter:

- time intervals: 200 nsec, 2  $\mu$ sec, 20  $\mu$ sec, 200  $\mu$ sec, 2 msec
- occurrences of: EV1, EV2, EV3, or EV4
- occurrences of: TRIG1, TRIG2, TRIG3, or TRIG4
- ACQ: any transaction stored in Acquisition Memory

In addition, there are 3 auxiliary counter sources to which special restrictions apply: Only one of these auxiliary sources can be selected at one time, though more than one counter can use the selected auxiliary source. When you select a second auxiliary source, it replaces the one previously selected. The auxiliary counter sources are:

- CYC: all bus cycles
- EMUCLK: clock on the emulator
- QUA: low-to-high transitions on the Event Qualifier BNC input

The ACQ command uses these counter sources with its FOR parameter.

**Output (O)** controls when the counter output will be high. A trigger is generated when an event occurs and the counter output is high. See Table 5-2.

**Table 5-2**  
**Counter Output Options**

OPTIONS	COUNTS	Counter Outputs		
		While Counting	At <sup>a</sup> Terminal Value	After Counting
O=Arm	Up	1	1	1
O=Disarm	Up	0	0	0
O=Pulse	Down	0	1	0
O=Delay	Down	0	0	1
O=Timeout	Down	1	1	0

<sup>a</sup> For ARM and DISARM, the terminal value is 65535. For PULSE, DELAY, and TIMEOUT, the terminal value is 0.

ARM and DISARM increment the counter from the initial value which is set with the V parameter. PULSE, DELAY, and TIMEOUT decrement the counter from the initial value. After the counter reaches its terminal value (of 65535 or 0000, depending on the output option), it stops counting until you reprogram it, either with the COU command or by entering G **with** an address.

*NOTE*

*The G command entered with an address reinitializes all the counters, including the one associated with the Acquisition Memory. G entered without an address leaves all the counters in their current states.*

**Gate (G)** controls **when** a counter can count its programmed source. It is only available with counters 2, 3, or 4. The G options are:

- G=OFF      Removes any previous gate restrictions
- G=CTR      Counter N counts only when counter N-1 remains high
- G=TRIGH    Counter N counts only when trigger N-1 remains high
- G=TRIGL    Counter N counts only when trigger N-1 remains low
- G=SEQH     Counter N begins counting when trigger N-1 first goes high
- G=SEQL     Counter N begins counting when trigger N-1 first goes low
- G=SELF     Counter N counts only when trigger N remains high

**Restart (R)** is allowed only if the gate parameter has been specified. The R options are ON and OFF. R=ON causes the counter to reinitialize to the value specified by the V parameter, and begin counting again each time the gate function becomes true. R=OFF turns this feature off.

The COU command, like the EVE command, is cumulative. Use the CLR parameter to remove previous conditions. For example, the command sequence

```
> COU 3 CLR V=100 S=ACQ O=DELAY
```

counts 100 acquisitions of Acquisition Memory and then outputs a high to Trigger 3.

## Acquisition Memory

The TTA maintains a record of input data in a 255-by-62-bit buffer called the Acquisition Memory. The ACQ command selects which bus transactions are to be stored. Acquisition Memory contains the 255 most recent samples of input data. (The Acquisition Memory stores slightly different input data from that monitored by the event comparators.) The Acquisition Memory is similar to the buffer memory of a logic analyzer. Figure 5-7 shows how information is stored in the TTA Acquisition Memory.

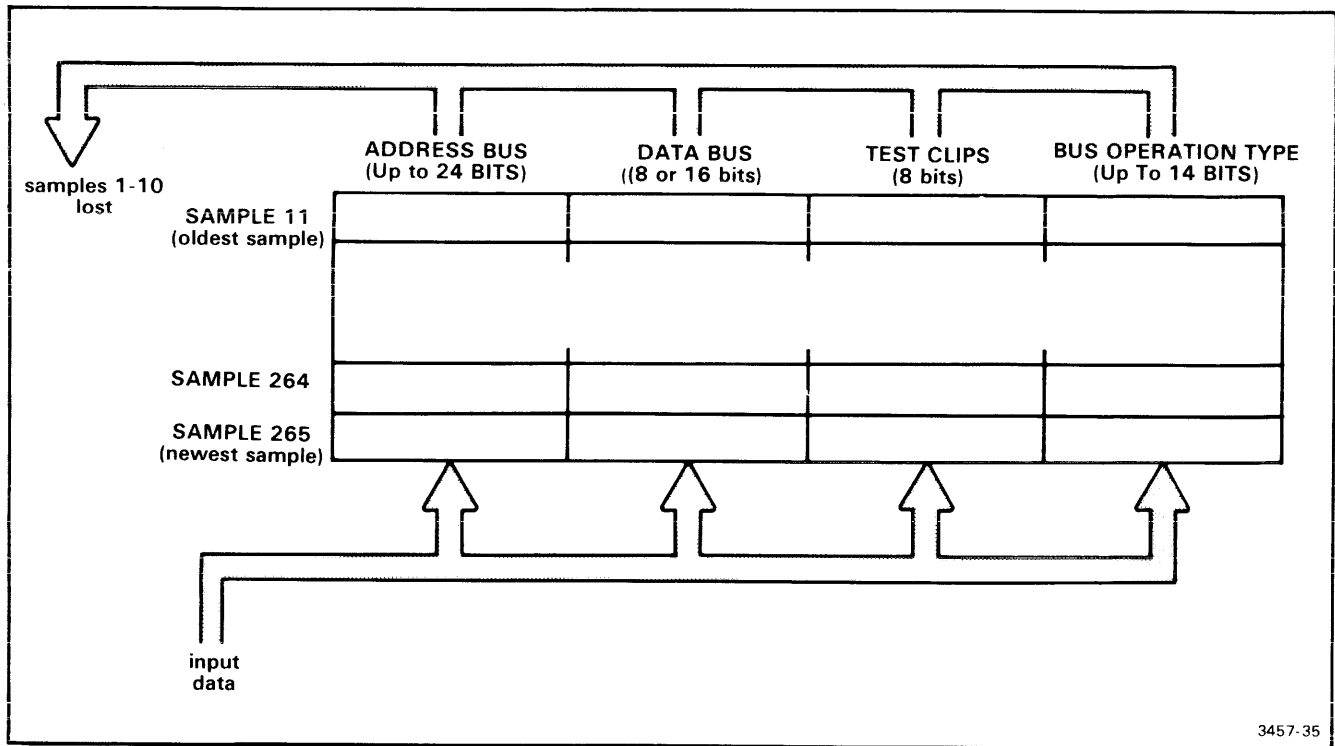


Fig. 5-7. Acquisition Memory.

This figure shows the contents of the Acquisition Memory after 265 samples of input data have been taken. Only the most recent 255 samples are stored. The DISP command is used to display the contents of the Acquisition Memory.

The **ACQ** command selects which bus transactions are to be stored in Acquisition Memory. The parameters are:

- ACQ ALL—all bus transactions are stored
- ACQ EV4—only the transactions defined as Event 4 are stored

The Acquisition Memory begins storing when program execution begins and stops storing when the emulator halts. You may want to halt storage in the buffer before the program has finished execution. There are two ways to do so:

```
> ACQ ALL FOR {value} {source}
> ACQ EV4 FOR {value} {source} AFTERTRIG4
```

You select one of the sources available with the COU command, and a value that equals the number of transactions you want to capture. For example,

```
> ACQ EV4 FOR 10 EV1
```

stores the transactions defined as event 4 until the tenth occurrence of event 1.

```
> ACQ ALL FOR 120 CYC AFTERTRIG4
```

"AFTERTRIG4" delays the start of counting until the conditions of TRIG4 have been met. In this example, acquisition stops 120 cycles after TRIG4 occurs.

Entering the ACQ command without parameters returns the parameters currently selected. If the FOR clause is included, the number of counts remaining to be acquired is also given.

```
> ACQ
```

```
Acquire EV4 for 00020T MSEC
Counts remaining: 00018T
```

The **DISP** command displays the contents of the Acquisition Memory. Entering DISP ALL displays the entire buffer, beginning with the oldest buffer contents. You may also use DISP to display a specified number of most recent transactions. DISP without parameters displays the transactions that have occurred since the emulator was last halted.

## TTA Summary

The following OS/40 commands control your TTA:

- ACQ—selects the type of operation to be stored in the Acquisition Memory
- AD—selects the address portion of an event
- BRE—defines a breakpoint in terms of TTA events
- BUS—selects the emulator-specific bus transactions that define an event
- CONS—causes a trigger after a sequence of events has occurred on consecutive bus cycles.
- COU—selects the counting units and output modes for the general purpose counters
- CTR—defines an event as a pattern of the four counter outputs
- DATA—selects the data portion of an event
- DISP—displays the contents of the Acquisition Memory
- EVE—defines an event in terms of the input data
- PRO—selects signals from the Data Acquisition Probe
- QUA—selects and defines an input from the Event Qualifier
- TCLR—clears the EVE, COU, and BRE conditions associated with a given channel
- TS—displays the status of the TTA or of selected triggers

For more detailed explanations of these commands, refer to the Trigger Trace Analyzer Users Manual. For descriptions of common procedures that use these commands, refer to the subsection "Using the TTA" in the Operating Procedures section of this manual.

## Section 6 SERVICE CALLS

	Page		Page
<b>Introduction</b> .....	6-1	Function 14—Get Device Type .....	6-12
<b>Overview</b> .....	6-1	Function 16—Get Last CONI Character .....	6-12
When Should You Use SVCs? .....	6-1	Function 17—Load Overlay .....	6-13
What is an SVC? .....	6-1	Function 19—Suspend Program .....	6-16
Program Instructions .....	6-3	Function 1A—Exit Program .....	6-12
Service Request Block (SRB) Vector .....	6-3	Function 1C—Get Execution Line Parameter .....	6-12
Service Request Block (SRB) .....	6-4	Function 1F—Abort Program .....	6-11
I/O Buffer .....	6-4	Function 22—Overwrite ASCII and Wait .....	6-14
<b>Limitations of File Handling SVCs</b> .....	6-4	Function 24—Seek Relative to Byte In File .....	6-15
<b>I/O Channels</b> .....	6-4	Function 30—Open for Read .....	6-14
<b>SVC Applications</b> .....	6-5	Function 41—Read Binary and Wait .....	6-15
Assign a Device or File to a Channel .....	6-5	Function 42—Write Binary and Wait .....	6-17
Read a Line from the System Terminal or a File .....	6-5	Function 44—Seek to Byte in File .....	6-16
Write a Line to the System Terminal or a File .....	6-5	Function 50—Open for Write .....	6-14
Time a Program Segment .....	6-5	Function 57—Load Overlay with Bias .....	6-13
Read Binary Data from a File .....	6-6	Function 62—Overwrite Binary and Wait .....	6-14
Write Binary Data to a File .....	6-6	Function 64—Seek to Byte in File Relative to EOF .....	6-16
Read a Single Byte from the System Terminal .....	6-6	Function 70—Open for Read or Write .....	6-14
Write a Single Byte to the System Terminal .....	6-6	Function 81—Read ASCII and Proceed .....	6-14
<b>SVC Demonstration</b> .....	6-6	Function 82—Write ASCII and Proceed .....	6-17
Program Description .....	6-9	Function 90—Create File .....	6-11
Data Declaration .....	6-9	Function A2—Overwrite ASCII and Proceed .....	6-14
Executable Code .....	6-9	Function C1—Read Binary and Proceed .....	6-15
<b>SVC Functions</b> .....	6-10	Function C2—Write Binary and Proceed .....	6-17
Terminology .....	6-10	Function E2—Overwrite Binary and Proceed .....	6-14

### SVC FUNCTION INDEX

Function 01—Read ASCII and Wait .....	6-14
Function 02—Write ASCII and Wait .....	6-17
Function 03—Close Channel .....	6-11
Function 04—Rewind File .....	6-15
Function 07—Special Function .....	6-16
Function 09—Log Error Message .....	6-13
Function 10—Assign Channel .....	6-11
Function 11—Read Program Clock .....	6-15
Function 13—Get Command Line Parameter .....	6-12

### TABLES

Table No.		
6-1	Summary of SAS and LAS formats .....	6-2
6-2	Device Identification and Type .....	6-13

### ILLUSTRATIONS

Fig. No.		
6-1	SVC memory layout, Small Address Space (SAS) format .....	6-2
6-2	SVC memory layout, Large Address Space (LAS) format .....	6-3
6-3	SVC demonstration program flowchart .....	6-7
6-4	8080A/8085A SVC demonstration program listing .....	6-8

# Section 6

## SERVICE CALLS

### INTRODUCTION

Service calls (SVCs) allow your program to perform various system operations. This section is divided into the following subsections:

- **SVC Overview**—Describes the general features of SVCs.
- **Limitations of File-Handling SVCs**—Describes the conditions under which SVCs can access files on the host.
- **I/O Channels**—Describes the I/O channels used by some SVCs.
- **SVC Applications**—Illustrates some useful SVC applications. A brief description accompanies each application.
- **SVC Demonstration**—Includes an 8080A/8085A assembly language program that incorporates several SVCs. A similar program is provided in the Emulator Specifics supplement for each emulator supported by the 8540.
- **SVC Functions**—Gives the function code and format for each SVC. The functions are listed alphabetically in the text; the Table of Contents contains a numerical index.

### OVERVIEW

OS/40 supports a number of service calls (SVCs). Some SVCs handle the transfer of data between your program

and OS/40 peripherals (such as the system terminal or line printer), or files on the 8560. SVCs also instruct OS/40 to perform system operations, such as aborting the program or reading the system clock.

### When Should You Use SVCs?

SVCs are especially useful during the initial stages of software development. While developing software, you may use SVCs to transfer data between your program and the 8540 system peripherals or files on the 8560. The SVCs permit your program to accept and display data.

Once you are certain that your program functions correctly, you may replace the SVCs with I/O instructions in your microprocessor's assembly language. This allows you to check out your program using prototype memory, clock, and I/O.

### What is an SVC?

An SVC consists of four parts: (1) the program instructions that generate the call; (2) a pointer within the Service Request Block vector; (3) a Service Request Block (SRB); and (4) an I/O buffer (for some SVCs).

The format of the SRB vector and the Service Request Block (SRB) is determined by the type of microprocessor you are emulating. In general, microprocessors that can address a maximum of 64K bytes use Small Address Space (SAS) format, while microprocessors that can address more than 64K require Large Address Space (LAS) format. Refer to the Emulator Specifics section to determine which format your microprocessor uses. Table 6-1 summarizes the difference in SRB formats. Figure 6-1 illustrates the SAS memory layout. Figure 6-2 illustrates the LAS memory layout.

All emulators support the use of SVCs in mode 0; some emulators support the use of SVCs in modes 1 and 2. In mode 1, the program instructions may be located in either program or prototype memory; the other parts of the SVC must reside in program memory. In mode 2, the SVC is located entirely in prototype memory. Refer to the Emulator Specifics section to determine whether your emulator supports SVCs in modes 1 and 2.

**Table 6-1**  
Summary of SAS and LAS Formats

SRB Field Name	Bytes Used	
	Small Address Space Format (SAS)	Large Address Space Format (LAS)
Function	1	1
Channel	2	2
Status	3	3
Fourth Byte	4	4
Byte Count	5	5-6
Buffer Length	6	7-8
Buffer Pointer	7-8	9-12

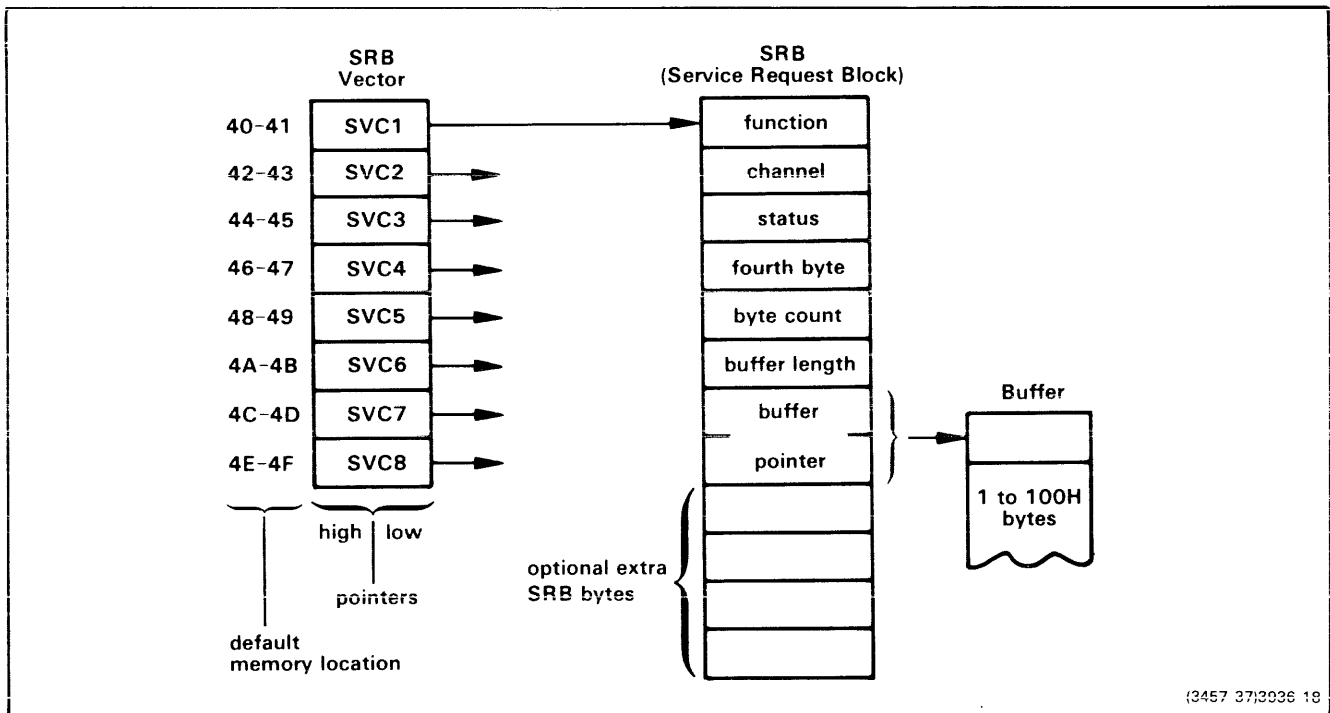


Fig. 6-1. SVC memory layout, Small Address Space (SAS) format.

An instruction sequence within your program determines which of the eight SRB pointers is used. The SRB addressed by the selected pointer contains the parameters needed to perform the SVC. Depending on the function specified in the first field of the SRB, an I/O buffer may be needed; in that case, the buffer length and buffer pointer fields indicate the length and location of the buffer.

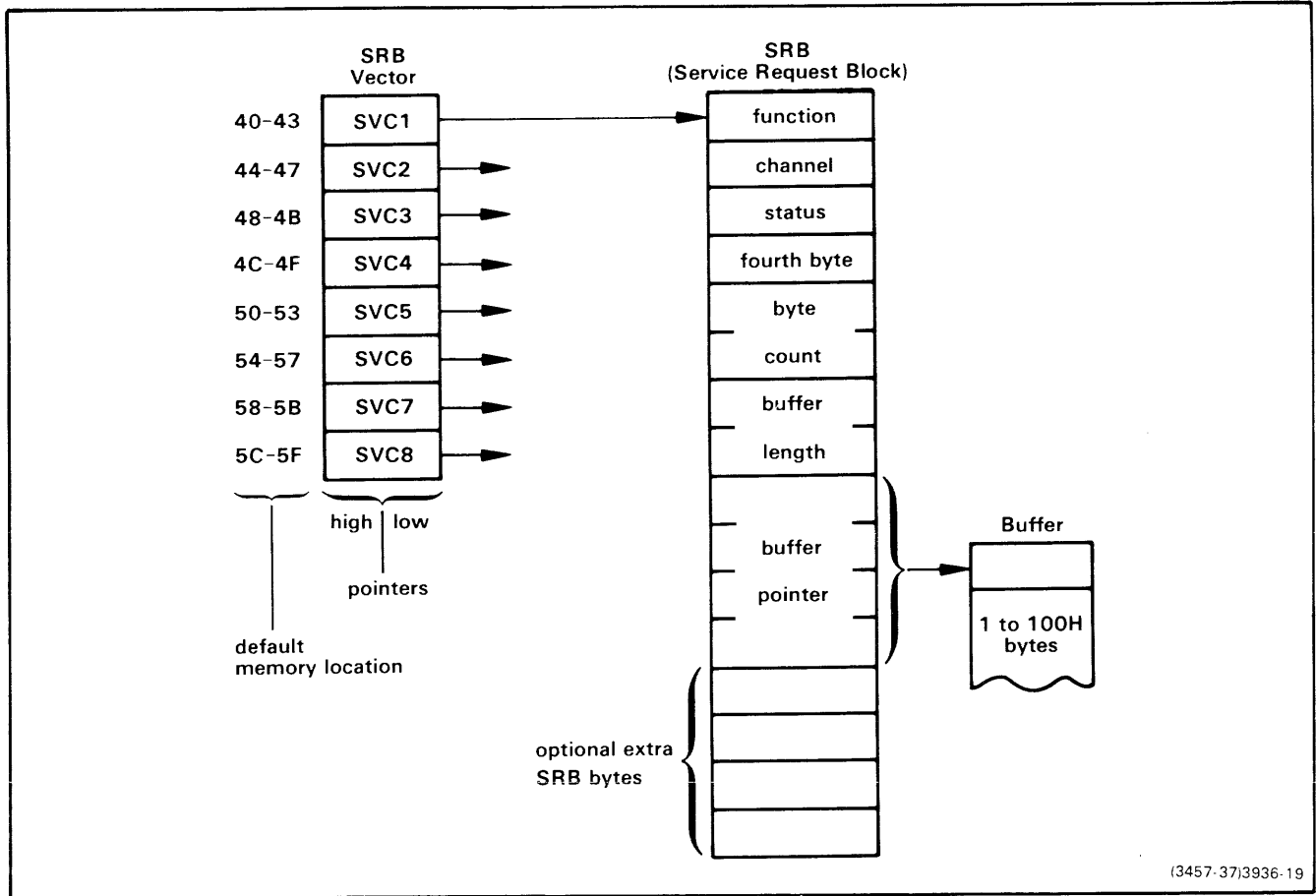


Fig. 6-2. SVC memory layout, Large Address Space (LAS) format.

An instruction sequence within your program determines which of the eight SRB pointers is used. The SRB addressed by the selected pointer contains the parameters needed to perform the SVC. Depending on the function specified in the first field of the SRB, an I/O buffer may be needed; in that case, the buffer length and buffer pointer fields indicate the length and location of the buffer.

**Program Instructions**

The service call is triggered by a microprocessor-dependent instruction sequence. In modes 0 and 1, this sequence usually consists of an I/O instruction followed by one no-operation instruction. (In mode 2, the sequence is an I/O instruction followed by two no-operation instructions.) When the SVC instruction sequence is executed, the 8540 hardware interrupts the system so that OS/40 may take the appropriate action.

Execution of the SVC does not affect the emulator registers. However, the instruction sequence that triggers the SVC may contain an instruction (such as IN) that modifies one or more registers.

Eight instruction sequences are defined for each emulator. These instruction sequences are labeled SVC1 through SVC8. Refer to the Emulator Specifics section for the specific instruction sequence used by your emulator.

The I/O addresses used in the microprocessor I/O instruction are in the range F0 to F7 for most emulators. F7 corresponds to SVC1, F6 to SVC2, and so on. You can use

the OS/40 SVC command to change the I/O address range to any group of eight consecutive I/O addresses with least significant digits in the range of 0 to 7.

**Service Request Block (SRB) Vector**

The Service Request Block (SRB) vector is a table of eight addresses beginning at address 40. Each address points to a different SRB. Although there are only eight SRB pointers, your program may use more than eight SVCs by changing the addresses in the SRB vector to point to additional SRBs.

When the SVC instruction sequence is executed, OS/40 examines the SRB pointer to determine the location of the SRB to be used. For example, if the instruction sequence for SVC1 is executed by an emulator that uses SAS format, OS/40 examines memory locations 40 and 41 to determine the location of the SRB for that call. Memory address 40 contains the high byte of the SRB address, and memory address 41 contains the low byte. If the instruction sequence for SVC1 is executed by an emulator that uses LAS format, OS/40 examines memory locations 40 through 43 to determine the location of the SRB for that



call. The high-order byte is in location 42; the low-order byte is in location 43.

To prevent conflict with fixed memory locations required by your program, you may change the location of the SRB vector. The OS/40 SVC command allows you to place the SRB vector nearly anywhere in memory. If you do not use the default SRB vector location, you must reenter this SVC command each time you reselect the emulator.

### Service Request Block (SRB)

The Service Request Block (SRB) contains the information that OS/40 needs to perform the service call. The SRB also reserves areas for OS/40 to return control information and data after the call is completed. The SRB may be located anywhere in memory. Refer to Table 6-1 as you read the following paragraphs.

The **function field** of the SRB identifies the function of the call. This byte controls how OS/40 interprets the rest of the SRB. SVC functions are described individually later in this section. SVC functions not listed in this section are unsupported and may yield unpredictable results.

The **channel field** usually contains an OS/40 channel number. Channels are discussed in the "I/O Channels" subsection, following this overview.

OS/40 uses the third byte to return status. This **status field** contains 00 if the operation was successfully completed. A non-zero value returned in this byte indicates an abnormal condition, which may or may not be an error: status values 06, 07, 29, 76, 7F, and FF are often not considered errors. These status conditions are listed by status code in the Error Messages section of this manual.

The status field can also be used to return non-status information. Such uses are discussed in the description of the appropriate SVC function.

The **fourth byte** is used by some functions in SAS format. The specific use of this byte is covered in the individual function description. The fourth byte is reserved for system use in LAS format.

The **byte count field** indicates the number of bytes transferred during an I/O operation. This field is also used by SVCs that do not perform data transfer. Refer to the specific SVC function description for details.

The remaining bytes in the SRB describe an I/O buffer. This buffer is explained under the next heading.

Some SVCs use additional bytes to form an extended SRB. The use of these bytes is described with the particular function that uses them.

In an SRB, any byte that is not specifically designated to return data may contain unspecified values (garbage) when the SVC is completed.

### I/O Buffer

The I/O buffer is used when an I/O transfer is performed. Information that is read from a device or host file is placed in the buffer after the SVC is called. Information to be written to an external device or host file is placed in the buffer before the call is made. I/O buffers may also hold ASCII strings (such as filespecs) used by some SVCs.

The **buffer length** may be 1 to 256 bytes. In SAS format, the value 00H is used to indicate a length of 256 bytes, because the buffer length field is only one byte long. In LAS format, the value 100H indicates a buffer length of 256 bytes.

The **buffer pointer field** contains the address of the first byte of the buffer, with the high-order byte or word given first.

The buffer length and buffer pointer fields are used by some functions that do not perform I/O operations. The use of these bytes is discussed in the function description.

## LIMITATIONS OF FILE HANDLING SVCs

The 8540 supports file handling SVCs (such as Create File—function 90) when the 8540 is operating in TERM mode with an 8560. If you are not in TERM mode, SVCs such as Assign, Read, Write, Open, and Close can only access 8540 peripherals.

The following filespecs denote 8540 peripherals: CONI, CONO, REMI, REMO, LPT, PPTR, PPTP. Other filespecs are assumed to denote host files.

## I/O CHANNELS

A channel is an independent data path for I/O operations. A channel may only be assigned to one device or file at a time. A channel can be disassociated ("closed") from the device or file and then reassigned. OS/40 provides 10 channels: channels 0 through 7 may be assigned and closed by your programs. Channels 8 and 9 are assigned by the system to standard input and output, respectively, and may also be used in your programs. OS/40 closes and reopens channels 8 and 9 each time the G or X command is used.

In TERM mode, channels assigned to CONI and CONO are mapped into standard input and standard output, respectively.

### NOTE

*No more than five channels at a time may be open to files.*

*It is possible to assign so many channels that OS/40 cannot continue to operate. In that case, error code 5C will be issued; the only way to recover from this condition is to enter the command A —A.*

*To avoid this condition, be sure to close a channel when you are finished with the file to which the channel is assigned. To close one or more selected channels, use the OS/40 CL command or the Close Channel SVC. To close all channels, use the OA/40 A command or the Abort Program SVC.*

*OS/40 may allow you to read from a write-only channel, such as standard output, but the results of such a read are undefined.*

## SVC APPLICATIONS

This subsection describes some common tasks that use service calls. Each SVC application describes the steps required to accomplish the task. The following applications are included here:

- Assign a device or file to a channel.
- Read a line from the system terminal or a file.
- Write a line to the system terminal or a file.
- Time a program segment.
- Read binary data from a file.
- Write binary data to a file.
- Read a single byte from the system terminal.
- Write a single byte to the system terminal.

### Assign a Device or File to a Channel

This SVC application is used to associate a device or file with a channel. The device or file may then be used for data entry or display.

1. Create a buffer in memory containing the name of the device or file that you want to assign. The name must be terminated with a RETURN character.
2. Establish an SRB containing the following information:
  - a. Function: 10 (Assign).
  - b. Channel: a number between 0 and 7, inclusive.
  - c. Buffer Pointer: pointing to the location containing the first character of the device or file name.
3. Establish a pointer in the SRB vector and execute the corresponding SVC.
4. Examine the status byte to verify that the SVC operated properly.

### Read a Line from the System Terminal or a File

This SVC application reads a line from the system terminal or a file, and stores the characters in memory.

1. Assign CONI (system terminal input) or the desired filespec to an available channel.
2. Create a buffer in memory. Make sure that the buffer is

as long as any line you expect to read, including the RETURN character.

3. Establish an SRB containing the following information:
  - a. Function: 01 (Read ASCII and Wait).
  - b. Channel: the channel to which CONI or the file was assigned.
  - c. Buffer Length: the length of the buffer that you created.
  - d. Buffer Pointer: pointing to the first byte of your buffer.
4. Establish a pointer in the SRB vector and execute the corresponding SVC.
5. Examine the status byte to verify that the SVC operated properly.

### Write a Line to the System Terminal or a File

This SVC application displays a line on the system terminal, or writes the line to a file. The line must be stored as a sequence of ASCII characters terminated by a RETURN character.

1. Assign CONO (system terminal output) or the desired file to an available channel.
2. Create a buffer in memory that contains the line of characters to be written, terminated with a RETURN character.
3. Establish an SRB containing the following information:
  - a. Function: 02 (Write ASCII and Wait).
  - b. Channel: the channel to which CONO or the file was assigned.
  - c. Buffer Length: the number of characters to be transferred, including the RETURN character.
  - d. Buffer Pointer: pointing to the first character of the line.
4. Establish a pointer in the SRB vector, and execute the corresponding SVC.
5. Examine the status byte to verify that the SVC operated properly.

### Time a Program Segment

This SVC application allows you to determine the execution time of a program segment. You must first enable the program clock with the OS/40 CLOCK ON command. Once you have done so, perform the following steps:

1. Establish an SRB with the following information:
  - a. Function: 11 (Read Program Clock).
2. Set up the appropriate pointer in the SRB vector and execute the corresponding SVC.
3. In SAS format, save the contents of bytes 4 and 5 as a 16-bit integer; byte 4 contains the high byte of the program clock. In LAS format, bytes 5 and 6 contain the program clock.

4. Execute the program segment to be timed.
5. Execute the SVC again, as described in steps 2 and 3.
6. Subtract the old 16-bit value from the new 16-bit value to obtain the execution time in 100-millisecond units.

## Read Binary Data from a File

This SVC application is used to read binary data from a file.

1. Assign the file to an available channel.
2. Create a buffer in memory. Make sure that the buffer is as long as any block of data you plan to read.
3. Establish an SRB with the following information:
  - a. Function: 41 (Read Binary and Wait).
  - b. Channel: the channel assigned to the file.
  - c. Buffer Length: the number of bytes to be read from the file.
  - d. Buffer Pointer: pointing to the address where the first byte will be placed.
4. Establish a pointer in the SRB vector, and execute the corresponding SVC.
5. Examine the status byte to verify that the SVC operated properly.

## Write Binary Data to a File

This SVC application is used to write binary data to a file.

1. Assign the file to an available channel.
2. Create a buffer in memory that contains the data to be written.
3. Establish an SRB with the following information:
  - a. Function: 42 (Write Binary and Wait).
  - b. Channel: the channel assigned to the file.
  - c. Buffer Length: the number of bytes to write to the file.
  - d. Buffer Pointer: pointing to the first byte to be written.
4. Establish a pointer in the SRB vector, and execute the corresponding SVC.
5. Examine the status byte to verify that the SVC operated properly.

## Read a Single Byte from the System Terminal

On ASCII reads, OS/40 waits to receive a RETURN character before returning control to your program. However, in some instances, you may want to regain control after each character is typed. This SVC application allows you to read a single byte from the system terminal then regain control.

### NOTE

*On a binary read, the character typed is not echoed; this is the responsibility of your program. Note also that all of the normal editing functions, such as BACKSPACE, are ignored for a binary read; again, it is the responsibility of your program to perform these operations on a binary read.*

1. Assign CONI (system terminal input) to an available channel.
2. Establish an SRB with the following information:
  - a. Function: 41 (Read Binary and Wait),
  - b. Channel: the channel to which CONI is assigned.
  - c. Buffer Length: 01 (this designates a single character to be read).
  - d. Buffer Pointer: pointing to the location that the character is to be placed after it is read.
3. Establish a pointer in the SRB vector, and execute the corresponding SVC.
4. Examine the status byte to verify that the SVC operated properly.

## Write a Single Byte to the System Terminal

This SVC application is used to write a single byte to the system terminal. On ASCII writes, OS/40 waits until it has written a RETURN character before returning control to your program. However, in some instances, you may want to regain control after each character is written. To write one character to the system terminal, perform the following steps:

1. Assign CONO (system terminal output) to an available channel.
2. Create an SRB containing the following information:
  - a. Function: 42 (Write Binary and Wait).
  - b. Channel: the channel assigned to CONO.
  - c. Buffer Length: 01 (for a single character).
  - d. Buffer Pointer: pointing to the character to be displayed.
3. Establish a pointer in the SRB vector, and execute the corresponding SVC.
4. Examine the status byte to verify that the SVC operated properly.

## SVC DEMONSTRATION

This subsection contains a demonstration program that illustrates some of the techniques for using SVCs. The program demonstrates the use of SRB pointers, SRBs, buffers, and four SVCs: Assign Channel, Read ASCII, Write ASCII, and Abort. Figure 6-3 is a flowchart of the demonstration program.

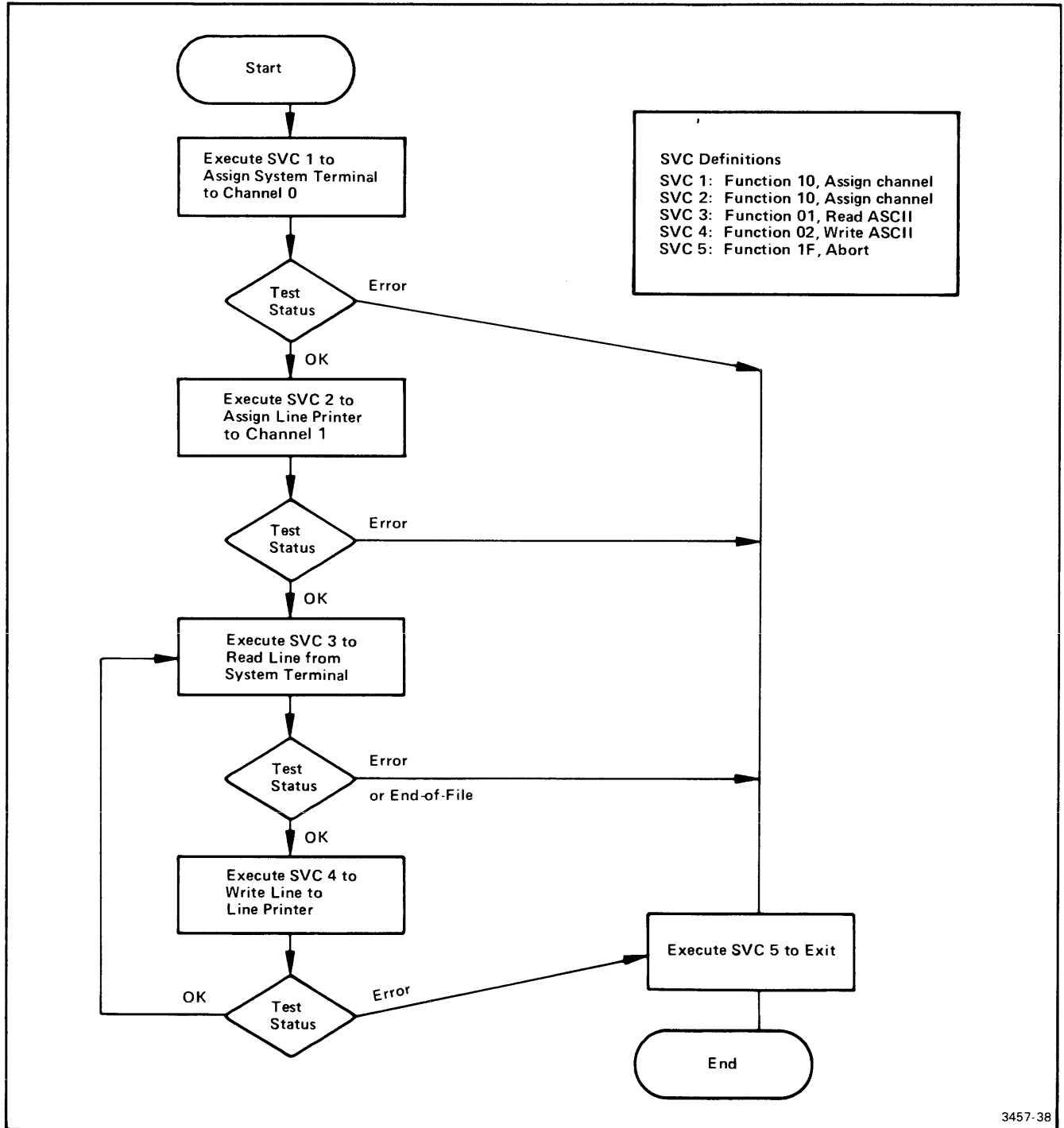


Fig. 6-3. SVC demonstration program flowchart.

This flowchart represents the algorithm for the SVC demonstration program. The instructions for the 8080A/8085A version of this program are given in Fig. 6-4. SVC1 and SVC2 assign the system terminal and the line printer to channels 0 and 1, respectively. SVC3 reads a line and SVC4 copies it to the line printer. To stop the program, enter a CTRL-Z when the terminal is awaiting input.

The demonstration program is presented here in 8080A/8085A assembly language. A similar program is provided in the Emulator Specifics supplement for each

emulator supported by the 8540. Figure 6-4 is an 8080/8085A listing of the demonstration program.

```

; SSSS V V CCCC
; S V V C
; SSSS V V C DEMONSTRATION. 8080A/8085A EMULATOR
; S V V C
; SSSS V CCCC

ORG 40H ; BEGINNING OF SRB VECTOR
BYTE HI(SRB1FN),LO(SRB1FN)
BYTE HI(SRB2FN),LO(SRB2FN)
BYTE HI(SRB3FN),LO(SRB3FN)
BYTE HI(SRB4FN),LO(SRB4FN)
BYTE HI(SRB5FN),LO(SRB5FN)
; END OF SRB VECTOR
;
ORG 100H ; SET UP SRB AREAS
; SRB1 = ASSIGN 'CONI' TO CHANNEL 0
SRB1FN BYTE 10H ; ASSIGN
; TO CHANNEL 0
SRB1ST BLOCK 01H ; STATUS RETURNED HERE
BLOCK 02H ; BYTES 4 AND 5 NOT USED
BYTE 05H ; LENGTH OF 'CONI'+<CR>
BYTE HI(CONI) ; POINTER TO
BYTE LO(CONI) ; 'CONI'+<CR>
; END OF SRB1
; SRB2 = ASSIGN 'LPT' TO CHANNEL 1
SRB2FN BYTE 10H ; ASSIGN
; TO CHANNEL 1
SRB2ST BLOCK 01H ; STATUS RETURNED HERE
BLOCK 02H ; BYTES 4 AND 5 NOT USED
BYTE 04H ; LENGTH OF 'LPT'+<CR>
BYTE HI(LPT) ; POINTER TO
BYTE LO(LPT) ; 'LPT'+<CR>
; END OF SRB2
; SRB3 = READ ASCII LINE FROM CONI (CHANNEL 0)
SRB3FN BYTE 01H ; READ ASCII
; FROM CHANNEL 0
SRB3ST BLOCK 01H ; STATUS RETURNED HERE
BLOCK 01H ; BYTE 4 NOT USED
BLOCK 01H ; BYTE COUNT RETURNED HERE
BYTE 00H ; 256 BYTES IN OUR BUFFER
BYTE HI(BUFFER) ; POINTER TO
BYTE LO(BUFFER) ; OUR BUFFER
; END OF SRB3
; SRB4 = WRITE ASCII LINE TO LPT (CHANNEL 1)
SRB4FN BYTE 02H ; WRITE ASCII
; TO CHANNEL 1
SRB4ST BLOCK 01H ; STATUS RETURNED HERE
BLOCK 01H ; BYTE 4 NOT USED
BLOCK 01H ; BYTE COUNT RETURNED HERE
BYTE 00H ; 256 BYTES IN OUR BUFFER
BYTE HI(BUFFER) ; POINTER TO
BYTE LO(BUFFER) ; OUR BUFFER
; END OF SRB4

```

3939-12

Fig. 6-4. 8080A/8085A SVC demonstration program listing (part 1 of 2).

```

;      SRB5 = ABORT (CLOSE ALL CHANNELS AND TERMINATE)
SRB5FN BYTE  1FH          ; ABORT
        BLOCK  07H          ; BYTES 2 THROUGH 8 NOT USED
;      END OF SRB5
;
;
BUFFER  BLOCK  100H        ; OUR I/O AREA
CONI    ASCII  'CONI'      ; ASCII OF 'CONI'
        BYTE   0DH          ; + <CR>
LPT     ASCII  'LPT'       ; ASCII OF 'LPT'
        BYTE   0DH          ; + <CR>
;      END OF DATA DEFINITIONS
;
;
;      BEGINNING OF EXECUTABLE CODE
;
START   ORG    1000H        ; ENTRY POINT INTO PROGRAM
        OUT    0F7H        ; CALL SVC1
        NOP                    ; TO ASSIGN 'CONI'
        LDA    SRB1ST       ; CHECK THE STATUS TO SEE
        CPI    00H          ; IF ALL WENT WELL
        JNZ    ABORT        ; NO? STOP EVERYTHING
        OUT    0F6H        ; YES? CALL SVC2
        NOP                    ; TO ASSIGN 'LPT'
        LDA    SRB2ST       ; CHECK THE STATUS TO SEE
        CPI    00H          ; IF ALL WENT WELL
        JNZ    ABORT        ; NO? STOP EVERYTHING
LOOP    OUT    0F5H        ; CALL SVC3
        NOP                    ; TO READ A 'CONI' LINE
        LDA    SRB3ST       ; INTO 'BUFFER'
        CPI    00H          ; ALL OK?
        JNZ    ABORT        ; NO? STOP EVERYTHING
        OUT    0F4H        ; CALL SVC4
        NOP                    ; TO WRITE 'LPT'
        LDA    SRB4ST       ; CHECK TO SEE IF
        CPI    00H          ; ALL IS O.K.
        JZ     LOOP        ; YES? BACK TO READ ANOTHER LINE
;      NO? FALL THROUGH TO TERMINATION
;
ABORT   OUT    0F3H        ; CALL SVC5
        NOP                    ; TO DO THE ABORT
        HLT                    ; SHOULD NEVER REACH HERE
        END    START

```

3939-13

Fig. 6-4. 8080A/8085A SVC demonstration program listing (part 2 of 2).

## Program Description

The demonstration program consists of two major parts: the data declarations and the executable code.

### Data Declaration

This part of the program begins by defining the five entries in the SRB vector. Each entry points to a function byte of an SRB. These entries are examined by OS/40 when an SVC is invoked. In this demonstration program, only one SRB is used for each SVC, so the SRB contains the number of the SVC as part of its labels.

After the SRBs are defined, the I/O buffer space is allocated. The device names (used for channel assignment) are also defined.

### Executable Code

This part of the program begins by invoking SVC1. This SVC invocation instructs OS/40 to examine the address stored in memory locations 40 and 41 of the SRB vector. These addresses point to the beginning of SRB1. SVC1 assigns channel 0 to CONI (system terminal input).

After SVC1 is executed, the program checks the status byte. OS/40 sets this byte to indicate whether the operation was completely and properly performed. If this byte is non-zero, some unwanted result has occurred, and the program branches to an exit routine.

Next, the program executes SVC2 to assign the line printer (LPT) to channel 1. The program then tests the status byte as before, branching to the exit routine upon any error.

The program then initiates a read from channel 0. Because the system terminal input (CONI) was assigned to this channel, SVC3 causes the program to be delayed until a line has been typed in at the terminal. When the RETURN character has been entered, OS/40 resumes the program. The line is read into the buffer designated in the SRB for SVC3.

The program then tests the status byte for a possible termination condition. This termination condition is indicated by a non-zero value in the status byte of SRB3. The program is exited when either of the following conditions occurs:

- An error occurs during the I/O transfer; this generates the appropriate I/O error value in the status byte, or
- A CTRL-Z is typed in at the terminal; this generates an end-of-file condition, and returns FF, the end-of-file status value.

If neither termination condition has occurred, SVC4 outputs the line to channel 1 (which was assigned to LPT.) The line itself does not need to be moved, because the same buffer is used for input and output. The status is checked again after SVC4 is completed. If OS/40 indicates no error, the program loops back to read another line.

When any error occurs, or CTRL-Z is entered at the terminal, the program branches to the exit routine. This routine executes SVC5, which is defined by SRB5 to perform an abort (stop program execution, close all open channels).

#### NOTE

*The program in Figure 6-4 is written for a B Series assembler (as provided for an 8560). To modify this program for an A Series assembler (as provided for an 8550), change each single quote (') to a double quote (").*

## SVC FUNCTIONS

The remainder of this section describes each of the SVC functions individually. The first byte of an SRB indicates the function of the service call. Each function is described in the following format:

Identification	The name of the function being described, and its function number. The function number is in byte 1 of the SRB at the time of the call.
Description	One or more paragraphs discussing the operation of the function and its limits or extensions.
Parameters Passed	A list of values that you must put into the SRB or buffer before the call is made. Because the function byte must always be loaded before any call, it is not included in this parameter list.
Information Returned	A list of all items that OS/40 will return in the various parts of the SRB and buffer. Because the status byte is always returned, it is not included in the list unless it has a special significance.

The functions are described in **alphabetic** order. Refer to the Section 6 Table of Contents for a numeric index of the functions. The following paragraphs briefly define the terms used in these descriptions.

### Terminology

**ASCII** and **binary** are two types of data transfers. An **ASCII** transfer consists of a single line of text containing zero to 255 characters, followed by a RETURN character. The most significant bit of each character is cleared on any ASCII transfer.

A **binary** transfer consists of 1 to 256 bytes; the length of the data item is determined only by the buffer size. The most significant bit of each byte is not affected during a binary transfer.

Under OS/40, **proceed** SVCs are equivalent to **wait** SVCs: both complete the I/O operation before your program is allowed to continue.

SRB Field Name	Bytes Used	
	Small Address Space Format (SAS)	Large Address Space Format (LAS)
Function	1	1
Channel	2	2
Status	3	3
Fourth Byte	4	4
Byte Count	5	5-6
Buffer Length	6	7-8
Buffer Pointer	7-8	9-12

Under OS/40, **overwrite** SVCs are equivalent to **write** SVCs: both operations set an end-of-file mark that destroys any information past that point in the file.

The **file pointer** is a logical position in the file. All reads from the file and writes to the file are performed beginning at the current file pointer. The file pointer is updated to the end of the data item read or written. **Seek** SVCs reposition the file pointer. An independent file pointer is maintained for each channel.

**NOTE**

*When the Byte Count, Buffer Length, and Buffer Pointer fields are used for non-I/O operations, they are discussed by byte number.*

*Differences between SAS and LAS formats are noted in the function descriptions.*

*On all two-byte and four-byte values used by SVCs, the high byte is given first.*

## Abort Program—Function 1F

This function terminates the currently running program. All open channels are automatically closed.

**NOTE**

*If this SVC is executed when TRAcE is off, the program counter will contain the address where execution began on the most recent G command.*

*Any errors that result from closing the open channels are displayed on the system terminal.*

### Parameters Passed

None.

### Information Returned

None.

## Assign Channel—Function 10

This function assigns a file or device to the designated channel. Store the filespec (terminated by a RETURN character) in the buffer before invoking this SVC. If you designate a file that does not already exist, the file is created.

### Parameters Passed

**Buffer.** Contains the filespec, terminated by a RETURN character.

**Channel.** The channel number assigned to the file or device designated by the filespec.

**Buffer Pointer.** The location of the first byte of the buffer.

### Information Returned

None.

## Close Channel—Function 03

This function closes a previously opened channel.

### Parameters Passed

**Channel.** The channel to be closed.

### Information Returned

None.

## Create File—Function 90

This function creates an empty file with the specified filespec, and assigns a channel to the new file. If the specified file already exists, its previous contents are lost.

### Parameters Passed

**Buffer.** The filespec of the file to be created, terminated by a RETURN character.

**Channel.** The channel number to which the file will be assigned.

**Buffer Pointer.** The location of the first byte of the buffer.

### Information Returned

None.



SRB Field Name	Bytes Used	
	Small Address Space Format (SAS)	Large Address Space Format (LAS)
Function	1	1
Channel	2	2
Status	3	3
Fourth Byte	4	4
Byte Count	5	5-6
Buffer Length	6	7-8
Buffer Pointer	7-8	9-12

## Exit Program—Function 1A

This function stops the execution of your program. OS/40 will respond with a prompt for another command. Use the G command to resume execution. (Do not use the CO command in this situation.) Any channels currently open remain open. This function is identical to the Suspend Program SVC (function 19).

### Parameters Passed

None.

### Information Returned

None.

## Get Command Line Parameter—Function 13

This function (identical to the Get Execution Line Parameter SVC, function 1C) returns a parameter of the command line used to load or execute the current program. The parameters are numbered with consecutive non-negative integers, beginning with 01. For example, for the OS/40 command line "X <MYFILE VWX+3", parameter 01 is V and parameter 03 is X+3. The selected parameter is returned in the buffer. If the parameter number is greater than the number of parameters in the command line, the value FF is returned in the buffer in place of a parameter.

### Parameters Passed

**SAS Byte 4, LAS Bytes 5-6.** The number of the requested parameter.

**Buffer Length.** The maximum number of characters in the parameter.

**Buffer Pointer.** The location of the first byte of the buffer.

## Information Returned

**Buffer.** The parameter, terminated by a RETURN character.  
**SAS Byte 5** The length of the parameter, including the RETURN character. This information is not returned in LAS format.

## Get Device Type—Function 14

This function returns two device-dependent values that define the type and general capabilities of the device or file assigned to the designated channel. Table 6-2 lists the values that are associated with each device.

In TERM mode, CONI and CONO are treated as files (device identification FF, type code 43).

### Parameters Passed

**Channel.** The channel number to which the device or file is currently assigned.

### Information Returned

**SAS Byte 4, LAS Byte 7.** The device identification number (as defined in Table 6-2).

**SAS Byte 5, LAS Byte 8.** The device type code (as defined in Table 6-2).

## Get Execution Line Parameter—Function 1C

This function is identical to the Get Command Line Parameter SVC, function 13.

## Get Last CONI Character—Function 16

This function returns the ASCII value of the last character entered at the system terminal.

### Parameters Passed

None.

SRB Field Name	Bytes Used	
	Small Address Space Format (SAS)	Large Address Space Format (LAS)
Function	1	1
Channel	2	2
Status	3	3
Fourth Byte	4	4
Byte Count	5	5-6
Buffer Length	6	7-8
Buffer Pointer	7-8	9-12

**Information Returned**

**SAS Byte 4, LAS Byte 6.** The last character entered at the system terminal.

**Load Overlay—Function 17**

This function loads a load file (in A Series or B Series Load Module format) into program or prototype memory. The buffer must contain the name of the file when the SVC is invoked.

**Parameters Passed**

**Buffer.** The filespec, terminated by a RETURN character.  
**Buffer Pointer.** The location of the first byte of the buffer.

**Information Returned**

**SAS Bytes 4-5, LAS Bytes 5-8.** The transfer address of the load file: the high-order byte is given first.

**Load Overlay with Bias—Function 57**

This function loads a load file (in A Series or B Series Load Module format) into program or prototype memory. The file

is loaded with a specified bias (offset) from each of its addresses. The bias is specified in the bytes following the buffer pointer, and is added to the starting address of each block in the load file.

**Parameters Passed**

**Buffer.** The filespec, terminated by a RETURN character.  
**Buffer Pointer.** The location of the first byte of the buffer.  
**SAS Bytes 9-10, LAS Bytes 13-16.** The bias (high byte first) to be added to the starting address of each block in the load file.

**Information Returned**

**SAS Bytes 4-5, LAS Bytes 5-8.** The transfer address of the load file: the high-order byte is given first.

**Log Error Message—Function 09**

This function displays an error message on the system terminal. (All system error messages are listed in the Error Messages section of this manual.) The message includes the name of the last program loaded; this identifies that the message was generated by a user program.

Use this SVC when your program is not prepared to handle the error. You can use the same SRB that returned the error code by changing the function byte.

**Parameters Passed**

**Status.** The error message number returned by the previous call for this SRB.

**Information Returned**

None.

**Table 6-2  
Device Identification and Type**

Name	Description	Device Identification	Type Code	Type Description
CONI	Console input	01	01	ASCII read
CONO	Console output	02	02	ASCII write
LPT	Line printer	03	02	ASCII write
PPTR	Paper tape reader	08	01	ASCII read
PPTP	Paper tape punch	09	02	ASCII write
REMI	Remote input	0A	01	ASCII read
REMO	Remote output	0B	02	ASCII write
(file)	File	FF	43	Binary read/write

SRB Field Name	Bytes Used	
	Small Address Space Format (SAS)	Large Address Space Format (LAS)
Function	1	1
Channel	2	2
Status	3	3
Fourth Byte	4	4
Byte Count	5	5-6
Buffer Length	6	7-8
Buffer Pointer	7-8	9-12

## Open for Read—Function 30

This function assigns an existing file or device to the designated channel. An error is generated if the file or device does not exist, or if a write to the channel is attempted later.

### Parameters Passed

**Buffer.** The filespec, terminated by a RETURN character.

**Channel.** The channel number to which the file is assigned.

**Buffer Pointer.** The location of the first byte of the buffer.

### Information Returned

None.

## Open for Read or Write— Function 70

This function assigns an existing file or device to the designated channel. An error is generated if the file or device does not exist.

### Parameters Passed

**Buffer.** The filespec, terminated by a RETURN character.

**Channel.** The channel number to which the file is assigned.

**Buffer Pointer.** The location of the first byte of the buffer.

### Information Returned

None.

## Open for Write—Function 50

This function assigns an existing file or device to the designated channel. An error is generated if the file or device does not exist, or if a read from the channel is attempted later.

### Parameters Passed

**Buffer.** The filespec, terminated by a RETURN character.

**Channel.** The channel number to which the filespec is assigned.

**Buffer Pointer.** The location of the first byte of the buffer.

### Information Returned

None.

## Overwrite ASCII and Proceed— Function A2

This function is identical to the Write ASCII and Wait SVC, function 02.

## Overwrite ASCII and Wait— Function 22

This function is identical to the Write ASCII and Wait SVC, function 02.

## Overwrite Binary and Proceed— Function E2

This function is identical to the Write Binary and Wait SVC, function 42.

## Overwrite Binary and Wait— Function 62

This function is identical to the Write Binary and Wait SVC, function 42.

## Read ASCII and Proceed— Function 81

This function is identical to the Read ASCII and Wait SVC, function 01.

## Read ASCII and Wait— Function 01

This function reads one ASCII line from the designated channel into the buffer.

### Parameters Passed

**Channel.** The channel assigned to the device or file from which the line is read.

SRB Field Name	Bytes Used	
	Small Address Space Format (SAS)	Large Address Space Format (LAS)
Function	1	1
Channel	2	2
Status	3	3
Fourth Byte	4	4
Byte Count	5	5-6
Buffer Length	6	7-8
Buffer Pointer	7-8	9-12

**Buffer Length.** The maximum number of characters to read, including the RETURN character.

**Buffer Pointer.** The location of the first byte of the buffer.

### Information Returned

**Byte Count.** The number of characters read, including the RETURN character.

**Buffer.** The ASCII line read, including the RETURN character.

## Read Binary and Proceed—Function C1

This function is identical to the Read Binary and Wait SVC, function 41.

## Read Binary and Wait—Function 41

This function reads binary data from the designated channel into the buffer.

### Parameters Passed

**Channel.** The channel assigned to the file or device from which the buffer is read.

**Buffer Length.** The number of bytes to read.

**Buffer Pointer.** The location of the first byte of the buffer.

### Information Returned

**Byte Count.** The number of bytes actually read; the same as buffer length, unless the end-of-file was reached.

**Buffer.** The bytes read.

## Read Program Clock—Function 11

This function reads the OS/40 program clock, which is a 16-bit counter controlled by the OS/40 CLOCK command.

### Parameters Passed

None.

### Information Returned

**SAS Bytes 4-5, LAS Bytes 5-6.** The 16-bit program clock; the high-order byte is given first.

## Rewind File—Function 04

This function moves the file pointer to the beginning of the file assigned to the designated channel.

### Parameters Passed

**Channel.** The channel number of the file to rewind.

### Information Returned

None.

## Seek Relative to Byte in File—Function 24

This function seeks to a byte in the file assigned to the designated channel. The byte number is given as a signed 4-byte value. The first byte of the file is byte number 0; successive bytes are numbered in ascending order.

An offset from the current file pointer is given as a signed 4-byte value. This offset is added to the current file pointer to produce the new file pointer.

The new file pointer is returned to verify that it has indeed been located at the requested position; the new position cannot be beyond the current end-of-file, nor before the beginning of the file. To obtain the value of the current file pointer, pass an offset of 00000000 to this SVC.

### Parameters Passed

**Channel.** The channel assigned to the file on which the seek is performed.

**SAS Bytes 5-8, LAS Bytes 9-12.** A signed 4-byte value (most significant byte first) containing the offset to add to the current file pointer.

### Information Returned

**SAS Bytes 5-8, LAS Bytes 9-12.** The new position of the file pointer.

SRB Field Name	Bytes Used	
	Small Address Space Format (SAS)	Large Address Space Format (LAS)
Function	1	1
Channel	2	2
Status	3	3
Fourth Byte	4	4
Byte Count	5	5-6
Buffer Length	6	7-8
Buffer Pointer	7-8	9-12

## Seek to Byte in File—Function 44

This function seeks to a byte in the file assigned to the designated channel. The byte number is given as a signed 4-byte value. The first byte of the file is byte number 0; successive bytes are numbered in ascending order.

The new file pointer is returned to verify that it has indeed been located at the requested position. The new position cannot be beyond the current end-of-file, or before the beginning of the file.

A seek to byte 0 is equivalent to the Rewind SVC, function 04.

### Parameters Passed

**Channel.** The channel assigned to the file on which the seek is performed.

**SAS Bytes 5-8, LAS Bytes 9-12.** A signed 4-byte value (most significant byte first) containing the byte position to seek to.

### Information Returned

**SAS Bytes 5-8, LAS Bytes 9-12.** The new position of the file pointer; should be the same as the position passed, unless a seek is attempted past the beginning or the end of the file.

## Seek to Byte in File Relative to EOF—Function 64

This function seeks to a byte in the file assigned to the designated channel. An offset from the current end-of-file is given as a signed 4-byte value (usually zero or negative).

You may position the file pointer at the end-of-file without knowing the file's length; to do so, pass a value of 00000000 to this SVC. After this seek operation, you may append additional information to the file.

The new file pointer is returned to verify that it has indeed been located at the requested position; the new position cannot be beyond the current end-of-file, or before the beginning of the file.

### Parameters Passed

**Channel.** The channel assigned to the file on which the seek is performed.

**SAS Bytes 5-8, LAS Bytes 9-12.** A signed 4-byte value (most significant byte first) containing the offset. This offset is added to the number of bytes in the file to obtain the new value of the file pointer.

### Information Returned

**SAS Bytes 5-8, LAS Bytes 9-12.** The new position of the file pointer.

## Special Function—Function 07

This function performs a special device-dependent operation on the device or file currently assigned to the designated channel. At the time of this writing, only one special function code (14) is defined. Its action is identical to the Get Device Type SVC (function 14). Other special functions may be defined at some future date.

### Parameters Passed

**Channel.** The channel number assigned to perform the special function.

**SAS Byte 5, LAS Bytes 5-6.** The special function code (currently defined only for 14).

### Information Returned

(Dependent on special function.)

## Suspend Program—Function 19

This function is identical to the Exit Program SVC, function 1A.

SRB Field Name	Bytes Used	
	Small Address Space Format (SAS)	Large Address Space Format (LAS)
Function	1	1
Channel	2	2
Status	3	3
Fourth Byte	4	4
Byte Count	5	5-6
Buffer Length	6	7-8
Buffer Pointer	7-8	9-12

### Write ASCII and Proceed— Function 82

This function is identical to the Write ASCII and Wait SVC, function 02.

### Write ASCII and Wait— Function 02

This function writes one ASCII line from the buffer to the designated channel.

**Buffer Length.** The maximum number of characters to be written; this count includes the RETURN character.

**Buffer Pointer.** The location of the first byte of the buffer.

### Information Returned

**Byte Count.** The number of characters written.

### Write Binary and Proceed— Function C2

This function is identical to the Write Binary and Wait SVC, function 42.

### Write Binary and Wait— Function 42

This function writes binary data from the buffer to the designated channel.

### Parameters Passed

**Buffer.** The binary data to be written to the channel.

**Channel.** The channel assigned to the file or device to which the buffer is written.

**Buffer Length.** The number of bytes to be written.

**Buffer Pointer.** The location of the first byte of the buffer.

### Information Returned

**Byte Count.** The number of bytes actually written; should be the same as buffer length.

## Section 7

# EMULATOR SPECIFICS

Processor-specific information is contained in the Emulator Specifics supplement that accompanies each emulator. Each supplement is designed to be inserted into this manual.

These Emulator Specifics supplements are numbered as if they were separate sections of this manual. For example, the 8080A supplement is labeled "Section 7A," and the first illustration is numbered "Fig. 7A-1." Similarly, other supplements are labeled Sections 7B, 7C, etc. Figures, pages, and tables are numbered accordingly.

Each supplement presents the following information:

- A general summary of the emulator's capabilities.
- OS/40 commands, parameters, and displays that are specific to that emulator.

- Service call (SVC) information for that emulator, including a sample program parallel to the 8080A/8085A program presented in the Service Calls section of this manual.
- Any special considerations that should be noted.
- Timing discrepancies between the prototype control probe and the microprocessor that it replaces.
- A demonstration run that parallels the one in the Learning Guide of this manual.

Each supplement has its own table of contents.



This manual supports the following TEKTRONIX products:

8300E04    Option 01  
8300P04  
8002F18  
8002F33  
8001F03

This manual supports a software/firmware module that is compatible with:

DOS/50 Version 2 (8550)  
OS/40 Version 1 (8540)

**PLEASE CHECK FOR CHANGE INFORMATION  
AT THE REAR OF THIS MANUAL**

**8550**  
MODULAR MDL SERIES  
**Z80**  
EMULATOR SPECIFICS  
USERS MANUAL

INSTRUCTION MANUAL

Tektronix, Inc.  
P.O. Box 500  
Beaverton, Oregon 97077

Serial Number \_\_\_\_\_



## LIMITED RIGHTS LEGEND

Software License No. \_\_\_\_\_

Contractor: Tektronix, Inc.

Explanation of Limited Rights Data Identification Method

Used: Entire document subject to limited rights.

Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

## RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Copyright © 1981, 1982 Tektronix, Inc. All rights reserved. Contents of this publication may not be reproduced in any form without the written permission of Tektronix, Inc.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and  are registered trademarks of Tektronix, Inc. TELEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Printed in U.S.A. Specification and price change privileges are reserved.

Printed in U.S.A. Specification and price change privileges are reserved.

# Section 7C

## CONTENTS

### Z80 EMULATOR SPECIFICS

	<b>Page</b>
<b>Introduction</b> .....	7C-1
<b>General Information</b> .....	7C-1
Emulator Hardware Configuration .....	7C-1
Microprocessors Supported .....	7C-1
Emulation Modes .....	7C-1
Z80A Clock Rate .....	7C-2
Z80B Clock Rate .....	7C-2
Symbolic Debug .....	7C-2
<b>Emulator-Specific Commands, Parameters, and Displays</b> .....	7C-2
SEL—Selecting an Emulator .....	7C-2
Byte/Word Parameter .....	7C-2
MAP—Mapping Memory .....	7C-2
Setting Breakpoints .....	7C-3
Memory Allocation Commands .....	7C-3
Port Commands .....	7C-3
CONS Command Modes .....	7C-3
Register Designators .....	7C-3
BUS and EVE—Bus Operation Designators .....	7C-5
DS—Sample Z80 Emulator Status Display .....	7C-6
RESET—Resetting Z80 Emulator Status .....	7C-6
DI—Sample Z80 Disassembled Code .....	7C-7
TRA—Sample Z80 TRAcE Display .....	7C-7
<b>Service Calls</b> .....	7C-9
SVCs in Modes 1 and 2 .....	7C-9
SRB Format .....	7C-9
SVC Demonstration .....	7C-9
<b>Z80 Special Considerations</b> .....	7C-13
<b>Z80A Jumpers</b> .....	7C-13
Z80A Emulator Board .....	7C-13
Z80A Driver/Receiver Board .....	7C-13
<b>Z80B Jumpers</b> .....	7C-14
Z80B Emulator Board .....	7C-14
Z80B Driver/Receiver Board .....	7C-15
<b>Z80A Emulator Timing</b> .....	7C-16
<b>Reducing Delay Through The Prototype Control Probe (Z80B Only)</b> .....	7C-16
<b>Z80B Emulator Timing</b> .....	7C-19
<b>Z80A Probe/Prototype Interface Diagram</b> .....	7C-22
<b>Z80B Probe/Prototype Interface Diagram</b> .....	7C-22

## CONTENTS (cont)

<b>Installing Your Z80 Emulator Software</b> .....	7C-25
8540 Firmware Installation Procedure .....	7C-25
8550 Software Installation Procedure .....	7C-25
<b>Z80 Demonstration Run</b> .....	7C-29
Introduction .....	7C-29
Examine the Demonstration Program .....	7C-32
Assemble and Load the Demonstration Program .....	7C-33
Case 1: Assemble and Load on the 8550 .....	7C-33
Case 2: Assemble on the 8560; Download to the 8540 .....	7C-37
Case 3: Download from Your Host to the 8540 .....	7C-42
Case 4: Patch the Program into Memory .....	7C-44
Run the Demonstration Program .....	7C-46
Monitor Program Execution .....	7C-49
Summary of Z80 Emulator Demonstration Run .....	7C-54

### ILLUSTRATIONS

Fig. No.		Page
7C-1.	Flag register bit configuration in the Z80 emulator .....	7C-5
7C-2.	Z80 SVC demonstration program listing .....	7C-11
7C-3.	Connecting the prototype clock input directly to the Prototype Control Probe. ....	7C-16
7C-4.	Z80 microprocessor bus timing .....	7C-21
7C-5.	Block diagram of Z80A emulator/prototype interface .....	7C-23
7C-6.	Block diagram of Z80B emulator/prototype interface .....	7C-24
7C-7.	System configurations .....	7C-29
7C-8.	Demonstration program .....	7C-30
7C-9.	Demonstration program: Extended Tekhex format .....	7C-31
7C-10.	Host computer commands for preparing demonstration program .....	7C-43

### TABLES

Table No.		Page
7C-1.	Z80 Registers and Flags .....	7C-4
7C-2.	Z80 Bus Operation Designators .....	7C-5
7C-3.	Z80 Service Calls .....	7C-10
7C-4.	Z80A Emulator/Z80A Microprocessor Timing Differences .....	7C-17
7C-5.	Z80B Emulator/Z80B Microprocessor Timing Differences .....	7C-19
7C-6.	Basic 8560 Editing Commands .....	7C-39

# Section 7C

## Z80 EMULATOR SPECIFICS

### INTRODUCTION

This section supports the Z80A Emulator Processor and Prototype Control Probe as well as the newer Z80B Emulator Processor and Probe. While the Z80A Emulator Processor can emulate only Z80 and Z80A microprocessors, the Z80B Emulator Processor can emulate the Z80, Z80A, and Z80B microprocessors.

In this section, the term "Z80 emulator" is used in presenting information that applies to both the older Z80A emulator and the newer Z80B emulator. The term "Z80A emulator" is used for information that applies only to the Z80A Emulator Processor. The term "Z80B emulator" is used for information that applies only to the newer Z80B Emulator Processor.

This Emulator Specifics section is to be inserted into Section 7 of the 8550 System Users Manual (DOS/50 Version 2) or the 8540 System Users Manual. It explains the features of the 8550 and 8540 systems that are unique to the Z80A and Z80B Emulators. Throughout the section, "your System Users Manual" refers to the 8550 System Users Manual or 8540 System Users Manual. The Z80 Demonstration Run is designed to be used with Section 1 (the Learning Guide) of your System Users Manual; the rest of this section contains reference material.

### GENERAL INFORMATION

#### Emulator Hardware Configuration

Throughout this Emulator Specifics section, the term "Z80 emulator" refers to a Z80 Emulator Processor board configured with a Z80 Prototype Control Probe or mobile microprocessor. In emulation Mode 0, the mobile microprocessor may be inserted directly into the emulator board. In Modes 1 and 2, the mobile microprocessor must be installed in the prototype control probe and the prototype control probe must be connected to both the emulator and your prototype. For instruction on installing the emulator board, mobile microprocessor, and probe, refer to the Z80 Emulator Processor and Prototype Control Probe Installation Service Manual.

#### Microprocessors Supported

The Z80A emulator emulates the Zilog, Z80 and Z80A microprocessors. The Z80B emulator emulates the Zilog Z80, Z80A, and Z80B microprocessors.

#### Emulation Modes

The Z80 emulator supports Emulation Modes 0, 1, and 2, as described in the Emulation section of your System Users Manual. The Z80 emulator supports service calls (SVCs) in all three modes.

## Z80A Clock Rate

In Mode 0 emulation, the emulator clock rate is 2 or 4 MHz, depending on the setting of jumper J1. In Mode 1 emulation, the maximum recommended rate for the prototype clock is 4 MHz.

## Z80B Clock Rate

In Mode 0 emulation, the emulator clock rate is 4 or 6 MHz, depending on the setting of jumper J3002. In Mode 1 emulation, the clock rate of 6 MHz may be used only with 8500-series systems. At this 6 MHz rate, one wait state will be inserted.

### NOTE

*When used with the Trigger-Trace Analyzer (TTA) or the Real-Time Prototype Analyzer (RTPA), the Z80B Emulator Processor provides an output that is one-half of the emulator operating clock rate. Therefore, the clock count stored in the TTA or RTPA buffers will be one-half of the actual emulator clock count.*

## Symbolic Debug

The Z80 emulator supports the use of symbolic debug.

## EMULATOR-SPECIFIC COMMANDS, PARAMETERS, AND DISPLAYS

### SEL—Selecting an Emulator

The SEL (SElect) command allows you to select the emulator you want to use with your system. The following command line selects the Z80 emulator and assembler:

```
> SEL Z80
```

### Byte/Word Parameter

Several commands offer you the choice of operating on memory on a byte-oriented or word-oriented basis. In affected commands, this choice is represented by the —B or —W parameter. For the Z80 emulator, the default value is —B (Byte).

### MAP—Mapping Memory

The Z80 addresses a 64 K memory space, arranged in 512 blocks of 128 bytes each. The MAP command enables you to assign blocks of memory to either program memory or prototype memory, and to designate blocks of program memory as read-only. Refer to the Command Dictionary for details on the syntax, parameters, and use of the MAP command.

## Setting Breakpoints

The Z80 emulator allows you to specify up to two breakpoints with the BK command.

## Memory Allocation Commands

The Memory Allocation Controller (MAC) option cannot be used with the Z80 emulator. The Z80 does not use the MEMSP command, and does not support memory space qualifiers or expressions. The Z80 emulator supports the AL (ALlocate) command, as described in the Command Dictionary of your System Users Manual. The DEAL, MEM, and NOMEM commands are not supported.

## Port Commands

The Z80 emulator does **not** support the RD or WRT commands.

## CONS Command Modes

The Z80 emulator supports the FET mode of the CONS command of the Trigger Trace Analyzer. The Z80 does **not** support the EMU mode of the CONS command.

## Register Designators

Table 7C-1 alphabetically lists the symbols used by DOS/50 and OS/40 to designate the registers and flags used by the Z80. The table provides the following information for each symbol:

- a description of the register or flag that the symbol represents;
- the size of the register or flag;
- the value assigned to the register or flag by the RESET command;
- whether the register or flag can be assigned a value by the S (Set) command.

Figure 7C-1 shows the contents of the Z80 flag register.

**Table 7C-1**  
**Z80 Registers and Flags**

<b>Symbol</b>	<b>Description</b>	<b>Size in Bits</b>	<b>Value After RESET<sup>a</sup></b>	<b>Alterable by S Command?</b>
A	Register A	8	NC	yes
AA	Alternate register A	8	NC	yes
AB	Alternate register B	8	NC	yes
ABC	Alternate registers B & C	16	NC	yes
AC	Alternate register C	8	NC	yes
AD	Alternate register D	8	NC	yes
ADE	Alternate registers D & E	16	NC	yes
AE	Alternate register E	8	NC	yes
AF	Alternate flag register	8	NC	yes
AH	Alternate register H	8	NC	yes
AHL	Alternate registers H & L	16	NC	yes
AL	Alternate register L	8	NC	yes
B	Register B	8	NC	yes
BC	Registers B & C	16	NC	yes
C	Register C	8	NC	yes
CY	Carry flag <sup>b</sup>	1	NC	yes
D	Register D	8	NC	yes
DE	Registers D & E	16	NC	yes
E	Register E	8	NC	yes
F	Flag register <sup>b</sup>	8	NC	yes
H	Register H	8	NC	yes
HC	Auxiliary carry flag <sup>b</sup>	1	NC	yes
HL	Registers H & L	16	NC	yes
I	Interrupt page address register	8	00	yes
IFF1	Interrupt flip-flop 1	1	0	yes <sup>c</sup>
IFF2	Interrupt flip-flop 2	1	0	yes <sup>c</sup>
IM	Interrupt mode	1	0	yes
IX	Index register X	16	NC	yes
IY	Index register Y	16	NC	yes
L	Register L	8	NC	yes
N	Subtract flag <sup>b</sup>	1	NC	yes
O	Overflow flag <sup>b</sup>	1	NC	yes
P	Parity flag <sup>b</sup>	1	NC	yes
PC	Program counter	16	0000	no
R	Memory refresh register	8	00	yes
S	Sign flag <sup>b</sup>	1	NC	yes
SP	Stack pointer	16	NC	yes
Z	Zero flag <sup>b</sup>	1	NC	yes

<sup>a</sup> NC = not changed by RESET

<sup>b</sup> The flag register is illustrated in Fig. 7C-1.

<sup>c</sup> The S command performed on either IFF1 or IFF2 sets both.

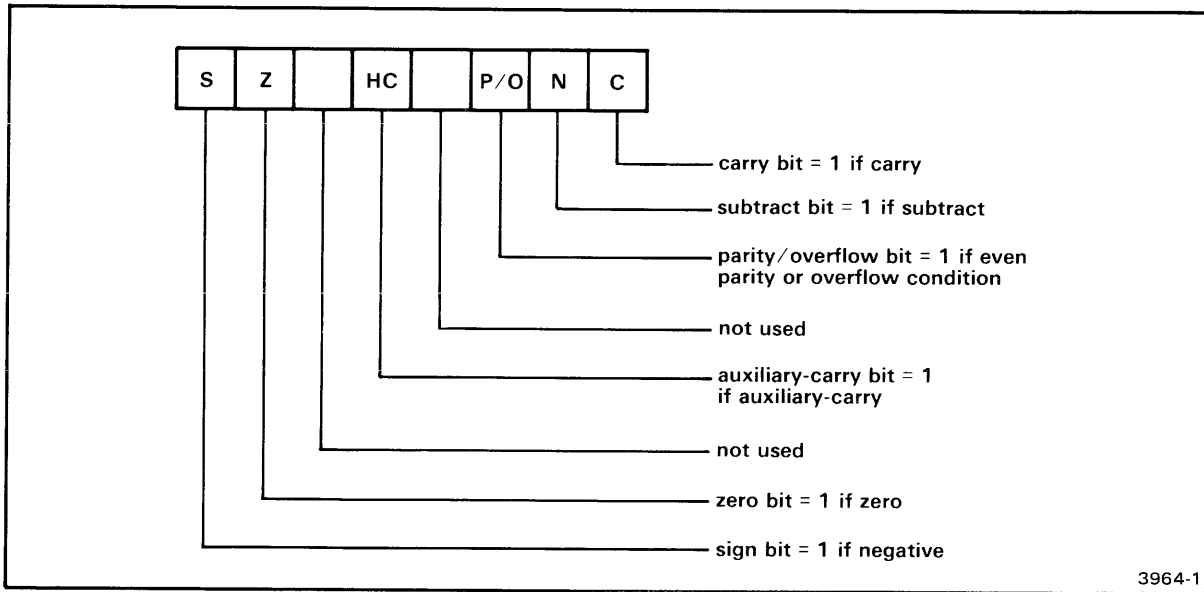


Fig. 7C-1. Flag register bit configuration in the Z80 emulator.

### BUS and EVE—Bus Operation Designators

Table 7C-2 lists the Z80 bus operation designators recognized by the Trigger Trace Analyzer's BUS command, and for the B parameter of the EVE command.

Table 7C-2  
Z80 Bus Operation Designators

Symbol	Bus Operation Type
CLR	All types
F	Instruction Fetches
NF	Non-fetches
M	Memory accesses
RD	Reads
WT	Writes
I	I/O operations



## DS—Sample Z80 Emulator Status Display

The DS (Display Status) command displays the status and register contents of the Z80. All numbers in the DS display line are hexadecimal.

Here is an example of a DS display line for the Z80 emulator:

```
> DS
PC=0000 SP=5645      F=43  A=C3  B=02  C=04  D=04  E=24  H=01  L=32
IX=1111 IY=2222     AF=00  AA=00  AB=20  AC=30  AD=40  AE=50  AH=60  AL=70
IFF1=0  IFF2=0  IM=0  I=00  R=00
```

Table 7C-1 explains the symbols displayed by the DS command.

For the Z80 emulator, the short and long forms of the DS display are the same: DS gives the same display as DS -L.

## RESET—Resetting Z80 Emulator Status

The RESET command produces a hardware reset signal to the Z80 microprocessor. The Z80 registers are reset to the values indicated in Table 7C-1.

**Example.** Suppose the DS command returns the following emulation status:

```
> DS
PC=0100 SP=5645      F=43  A=C3  B=02  C=04  D=04  E=24  H=01  L=32
IX=1111 IY=2222     AF=00  AA=00  AB=20  AC=30  AD=40  AE=50  AH=60  AL=70
IFF1=1  IFF2=1  IM=1  I=01  R=01
```

Enter the RESET command. Then check the status again with the DS command:

```
> RESET
> DS
↓
PC=0000 SP=5645      F=43  A=C3  B=02  C=04  D=04  E=24  H=01  L=32
IX=1111 IY=2222     AF=00  AA=00  AB=20  AC=30  AD=40  AE=50  AH=60  AL=70
IFF1=0  IFF2=0  IM=0  I=00  R=00
↑      ↑      ↑      ↑      ↑
```

The arrows show the changed registers.

## DI—Sample Z80 Disassembled Code

The DI (Disassemble) command translates object code in memory into assembly language instructions. DI displays object code, assembly language mnemonics, and operands. Use the DI command to verify that the values in memory correspond to the assembly language instructions of your program.

Here is an example of Z80 DI command output:

```
> DI 100 10E
LOC      INST      MNEM  OPER
000100  210005  LD    HL,0500
000103  0605    LD    B,05
000105  AF      XOR   A
000106  86      ADD  A,(HL)
000107  23      INC  HL
000108  05      DEC  B
000109  C20601  JP   NZ,0106
00010C  D3F7    OUT  (F7),A
00010E  00      NOP
```

Diagram labels and their corresponding columns:

- address of the instruction → LOC
- machine language instruction → INST
- instruction mnemonic → MNEM
- operand (s): address, register, or data being operated on → OPER

## TRA—Sample Z80 TRAcE Display

The TRA (TRAcE) command selects the range and type of instructions to be displayed as your program executes. With the Z80 emulator, the TRA -N format is the same as the TRA -L format.

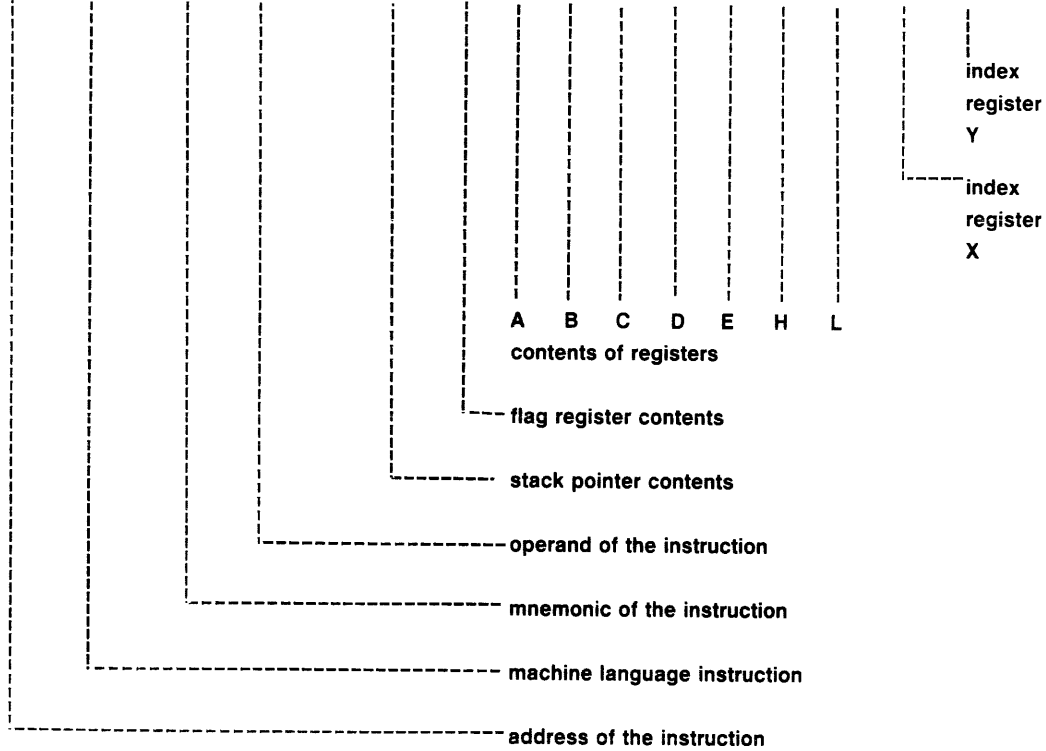
### NOTE

*When TRAcE conditions have been set, the emulator runs at slower than normal processing speeds and RTPA breakpoints are suppressed.*

Here is an example of Z80 TRA command output:

```
> TRA ALL
> G 100
```

LOC	INST	MNEM	OPER	SP	F	A	B	C	D	E	H	L	IX	IY
000100	210005	LD	HL,0500	0000	06	0F	01	00	00	00	05	00	0000	0000
000103	0605	LD	B,05	0000	06	0F	05	00	00	00	05	00	0000	0000
000105	AF	XOR	A	0000	46	00	05	00	00	00	05	00	0000	0000
000106	86	ADD	A,(HL)	0000	02	01	05	00	00	00	05	00	0000	0000
000107	23	INC	HL	0000	02	01	05	00	00	00	05	01	0000	0000
000108	05	DEC	B	0000	12	01	04	00	00	00	05	01	0000	0000
000109	C20601	JP	NZ,0106	0000	12	01	04	00	00	00	05	01	0000	0000
000106	86	ADD	A,(HL)	0000	06	03	04	00	00	00	05	01	0000	0000
000107	23	INC	HL	0000	06	03	04	00	00	00	05	02	0000	0000
000108	05	DEC	B	0000	16	03	03	00	00	00	05	02	0000	0000
000109	C20601	JP	NZ,0106	0000	16	03	03	00	00	00	05	02	0000	0000



## SERVICE CALLS

Service calls (SVCs) enable your program to use many system capabilities of your 8540, 8550, or 8560.

An SVC is invoked with a Z80 OUT instruction. The operand of the OUT instruction directs the system to a specified memory address called the SRB pointer (which points to the SRB—the Service Request Block). The SRB pointer tells the system where to find the data (stored in the SRB) that informs the system which function to perform. Refer to the Service Calls section of your System Users Manual for an explanation of service calls, service request blocks, and SRB pointers.

Your program can point to eight SRBs at any one time. As your program executes, it can store new addresses in the SRB vector. Table 7C-3 shows the default addresses for the eight SRB pointers. These addresses and their associated port numbers can be altered with the SVC command to suit your program requirements. See the Command Dictionary section of your System Users Manual for syntax and use of the SVC command.

### SVCs in Modes 1 and 2

The Z80 emulator supports SVCs in Emulation Modes 1 and 2, as described in the Service Calls section of your System Users Manual. In Mode 2, all parts of the SVC must reside in prototype memory.

#### NOTE

*In Mode 0 and 1, use one NOP instruction immediately following the OUT instruction. In Mode 2, use two NOP instructions immediately following the OUT instruction; this allows time for the SVC to occur.*

### SRB Format

The Z80 emulator uses the SAS (Small Address Space) format for SRBs and the SRB vector. This format is described in the Service Calls section of your System Users Manual.

### SVC Demonstration

Figure 7C-2 lists a Z80 program that uses four SVC functions: Assign Channel, Read ASCII, Write ASCII, and Abort. The program's algorithm is explained in the Service Calls section of your System Users Manual, which demonstrates a version of the program written in 8085A assembly language. You can perform a parallel demonstration with the Z80 emulator and Z80 A Series Assembler using the program in Fig. 7C-2.

**Table 7C-3  
Z80 Service Calls**

SVC Number	Z80 Service Calls		Default Address of SRB pointer
	mnemonic <sup>a</sup>	hexadecimal	
1	OUT (0F7H),A NOP	D3F7 00	40, 41
2	OUT (0F6H),A NOP	D3F6 00	42, 43
3	OUT (0F5H),A NOP	D3F5 00	44, 45
4	OUT (0F4H),A NOP	D3F4 00	46, 47
5	OUT (0F3H),A NOP	D3F3 00	48, 49
6	OUT (0F2H),A NOP	D3F2 00	4A, 4B
7	OUT (0F1H),A NOP	D3F1 00	4C, 4D
8	OUT (0F0H),A NOP	D3F0 00	4E, 4F

<sup>a</sup>You can use an IN instruction (opcode DB) in place of each OUT instruction given in Table 7C-3.

**NOTE**

*The program shown in Fig. 7C-2 is written for an A Series assembler (as provided for the 8550). To make this acceptable for a B Series assembler (as provided for the 8560), change each double quote (") to a single quote ('). This program shows the use of four service calls. The program's algorithm is explained in the Service Calls section of your System Users Manual. The program accepts a line of ASCII characters from the system terminal; then, when it receives a RETURN character, the program writes the line to the line printer and accepts another line. (On the 8550, output to the line printer is buffered. No text is printed until the line printer buffer in the 8501 becomes full or the program ends.) To terminate the program, enter a CTRL-Z while the program is waiting for input.*

```

; SSSSS V V CCCCC
; S V V C
; SSSSS V V C DEMONSTRATION. Z80 EMULATOR
; S V V C
; SSSSS V CCCCC

ORG 40H ; BEGINNING OF SRB VECTOR
BYTE HI(SRB1FN),LO(SRB1FN)
BYTE HI(SRB2FN),LO(SRB2FN)
BYTE HI(SRB3FN),LO(SRB3FN)
BYTE HI(SRB4FN),LO(SRB4FN)
BYTE HI(SRB5FN),LO(SRB5FN)
; END OF SRB VECTOR
ORG 100H ; SET UP SRB AREAS
; SRB1 = ASSIGN "CONI" TO CHANNEL 0
SRB1FN BYTE 10H ; ASSIGN
BYTE 00H ; TO CHANNEL 0
SRB1ST BLOCK 01H ; STATUS RETURNED HERE
BLOCK 02H ; BYTES 4 AND 5 NOT USED
BYTE 05H ; LENGTH OF "CONI"+<CR>
BYTE HI(CONI) ; POINTER TO
BYTE LO(CONI) ; "CONI"+<CR>
; END OF SRB1
; SRB2 = ASSIGN "LPT" TO CHANNEL 1
SRB2FN BYTE 10H ; ASSIGN
BYTE 01H ; TO CHANNEL 1
SRB2ST BLOCK 01H ; STATUS RETURNED HERE
BLOCK 02H ; BYTES 4 AND 5 NOT USED
BYTE 04H ; LENGTH OF "LPT"+<CR>
BYTE HI(LPT) ; POINTER TO
BYTE LO(LPT) ; "LPT"+<CR>
; END OF SRB2
; SRB3 = READ ASCII LINE FROM CONI (CHANNEL 0)
SRB3FN BYTE 01H ; READ ASCII
BYTE 00H ; FROM CHANNEL 0
SRB3ST BLOCK 01H ; STATUS RETURNED HERE
BLOCK 01H ; BYTE 4 NOT USED
BLOCK 01H ; BYTE COUNT RETURNED HERE
BYTE 00H ; 256 BYTES IN OUR BUFFER
BYTE HI(BUFFER) ; POINTER TO
BYTE LO(BUFFER) ; OUR BUFFER
; END OF SRB3
; SRB4 = WRITE ASCII LINE TO LPT (CHANNEL 1)
SRB4FN BYTE 02H ; WRITE ASCII
BYTE 01H ; TO CHANNEL 1
SRB4ST BLOCK 01H ; STATUS RETURNED HERE
BLOCK 01H ; BYTE 4 NOT USED
BLOCK 01H ; BYTE COUNT RETURNED HERE
BYTE 00H ; 256 BYTES IN OUR BUFFER
BYTE HI(BUFFER) ; POINTER TO
BYTE LO(BUFFER) ; OUR BUFFER
; END OF SRB4

```

Fig. 7C-2. Z80 SVC demonstration program listing (part 1 of 2).

```

;      SRB5 = ABORT (CLOSE ALL CHANNELS AND TERMINATE)
SRB5FN BYTE    1FH          ; ABORT
      BLOCK    07H          ; BYTES 2 THROUGH 8 NOT USED
;
;      END OF SRB5
;
;
BUFFER BLOCK    100H          ; OUR I/O AREA
CONI   ASCII   "CONI"        ; ASCII OF "CONI"
      BYTE     0DH          ; + <CR>
LPT    ASCII   "LPT"         ; ASCII OF "LPT"
      BYTE     0DH          ; + <CR>
;
;      END OF DATA DEFINITIONS
;
;
;      BEGINNING OF EXECUTABLE CODE
START  ORG      1000H          ; ENTRY POINT INTO PROGRAM
      OUT      (OF7H),A       ; CALL SVC1
      NOP
      LD       A,(SRB1ST)     ; CHECK THE STATUS TO SEE
      CP      00H            ; IF ALL WENT WELL
      JP      NZ,ABORT        ; NO? STOP EVERYTHING
      OUT      (OF6H),A       ; YES? CALL SVC2
      NOP
      LD       A,(SRB2ST)     ; CHECK THE STATUS TO SEE
      CP      00H            ; IF ALL WENT WELL
      JP      NZ,ABORT        ; NO? STOP EVERYTHING
LOOP   OUT      (OF5H),A       ; CALL SVC3
      NOP
      LD       A,(SRB3ST)     ; TO READ A "CONI" LINE
      CP      00H            ; INTO "BUFFER"
      JP      NZ,ABORT        ; ALL OK?
      OUT      (OF4H),A       ; NO? STOP EVERYTHING
      NOP
      LD       A,(SRB4ST)     ; CALL SVC4
      CP      00H            ; TO WRITE TO "LPT"
      JP      Z,LOOP          ; CHECK TO SEE IF
      NOP
      LD       A,(SRB4ST)     ; ALL IS OK
      CP      00H            ; YES? BACK TO READ ANOTHER LINE
      JP      Z,LOOP          ; NO? FALL THROUGH TO TERMINATION
;
;      ABORT
ABORT  OUT      (OF3H),A       ; CALL SVC5
      NOP
      HALT
      END      START

```

Fig. 7C-2. Z80 SVC demonstration program listing (part 2 of 2).

## Z80 SPECIAL CONSIDERATIONS

The Z80 emulator behaves like the Z80 microprocessor, with the following exceptions

- Interrupts are detected only when user code is being executed.
- When TRAcE is enabled, there is a maximum 153 ns delay on the IORQ signal.
- During TRAcE sequences, the emulator performs the auxiliary memory refresh operations between execution of user code instructions.

## Z80A JUMPERS

### Z80A Emulator Board

The Z80A Emulator board contains two jumpers, J1 and J3. Jumper J1 selects between 2 MHz and 4 MHz as the system clock speed for Emulation Mode 0. Jumper J3 is used to delete wait states in Emulation Mode 1.

### Z80A Driver/Receiver Board

The Z80A Driver/Receiver contains two jumpers, J1041 and J3051. In Emulation Mode 1, MREQ is unavailable to the prototype when jumper J1041 is in the right-most position. When J1041 is in the left-most position, MREQ is available to the prototype whenever HOLDA is not asserted.

When jumper J3051 is in the right-most position, data fetched from program memory (in Mode 1) does not appear at the probe tip. When jumper J3051 is in the left-most position, data from program memory is driven to the prototype. If jumper J3051 is in the left-most position, jumper J1041 must also be in the left-most position.

#### NOTE

*With jumper J3051 in the left-most position, prototype bus contentions may occur.*

Both jumpers J1041 and J3051 are shipped in the right-most position.



## Z80B JUMPERS

### Z80B Emulator Board

The Z80B emulator board contains four jumpers: J3003, J1059, J1061, and J1081.

Jumper J3003 selects a clock speed of either 4 MHz or 6 MHz for Emulation Mode 0. A clock speed of 6 MHz must not be used with a 8002A system. J1059 is placed in the normal position when the Z80B processor is in interrupt modes 1 and 2 or mode 0 single-byte vectors.

J1059 is placed in the IM 0 MULTI position when the processor is in interrupt Mode 0 and there is a possibility of multi-byte instruction vectors. In this position, the interrupt data is gated in from the probe tip by the INTA line, which is asserted and disabled by the stack write associated with the interrupt or the next fetch. The INTA line is asserted during a maskable interrupt shortly after MI and IOREQ are asserted by the emulator processor. In Emulation Mode 1, and with the stack pointer mapped to program memory, it is possible to get an additional short MEMREQ pulse at the probe tip during the first part of the stack write.

J1061 controls the number of wait states. (The function selected with J1061 may interact with the function selected by J1082. See the discussion, "4 MHz and Below", later in this section.) J1061 has three positions:

- The WAITS position selects no wait state and should be used only with 8540 and 8550 systems at 4 MHz and below.
- The 85XX position selects one wait state and should be used only with 8540 or 8550 systems.
- The 800X position selects two wait states and should be used only with 8001 or 8002A systems.

J1081 controls the modes of operation under which wait states are inserted. The positions required for various configurations of the systems are described in the following text.

### 8001/8002A Systems Jumper Considerations

- When J1081 is in the SLOW position, J1061 should be in the 800X position and J3003 should be in the 4 MHz position.
- Two wait states are inserted each time the program memory is accessed, or when running in emulation Mode 1.
- No wait states are inserted in Mode 2 except when operating in debug mode, or memswitch mode during the jump sequence. Two wait states are inserted in these modes.

### 8500-Series Systems Jumper Considerations

- **4 MHz and Below.** With J1081 placed in the SLOW position and J1061 placed in the WAITS position, no wait states are inserted in Emulation Modes 0, 1, or 2. One wait state is inserted during a forced jump sequence.
- **4 MHz to 6 MHz.** When operating between 4 and 6 MHz, the following wait states may be selected (J1061 in 85XX position). (If operating is Emulation Mode 1 and all memory is mapped to program memory, J1061 can be placed in WAITS position with no wait states in Modes 0, 1, or 2.)

#### NOTE

*You can generate additional wait states in Emulation Mode 1 or 2 when memory is mapped to the prototype.*

1. *J1081 in SLOW Position (used with 670-6542-00 and up memory boards).*
  - a. *One wait state inserted in Emulation Modes 0 and 1.*
  - b. *No wait states inserted in Emulation Mode 2 except during memswitch and forced jump operations when one wait state is inserted.*
2. *J1081 in FAST Position (not used with 670-6542-00 and up memory boards).*
  - a. *No wait states inserted in Emulation Modes 0 and 2.*
  - b. *One wait state is added during forced jump, memswitch operations, and Mode 1.*

### Z80B Driver/Receiver Board

The Z80B Driver/Receiver contains two jumpers: J1041 and J3051. When memory is mapped to the system in Emulation Mode 1, MREQ is unavailable to the prototype when J1041 is in the right-most position. When J1041 is in the left-most position, MREQ is available to the prototype whenever HOLDA is not asserted.

When jumper J3051 is in the right-most position, data fetched from program memory (Mode 1 only) does not appear at the probe tip. When J3051 is in the left-most position, data from program memory is driven to the prototype. If jumper J3051 is in the left-hand position, jumper J1041 must also be in the left-most position.

#### NOTE

*With jumper J3051 in the left-most position, prototype bus contentions may occur.*

Both jumpers J1041 and J3051 are shipped in the right-most position.

## Z80A EMULATOR TIMING

In Emulation Modes 1 and 2, the emulating microprocessor resides in the Prototype Control Probe, and the signals between the prototype and the emulating microprocessor are buffered. Therefore, some timing differences exist between the Z80A emulator and a Z80A microprocessor that has been inserted directly into the prototype. Table 7C-4 lists these differences. Figure 7C-4 is a timing diagram corresponding to the signals present on the Z80A emulator.

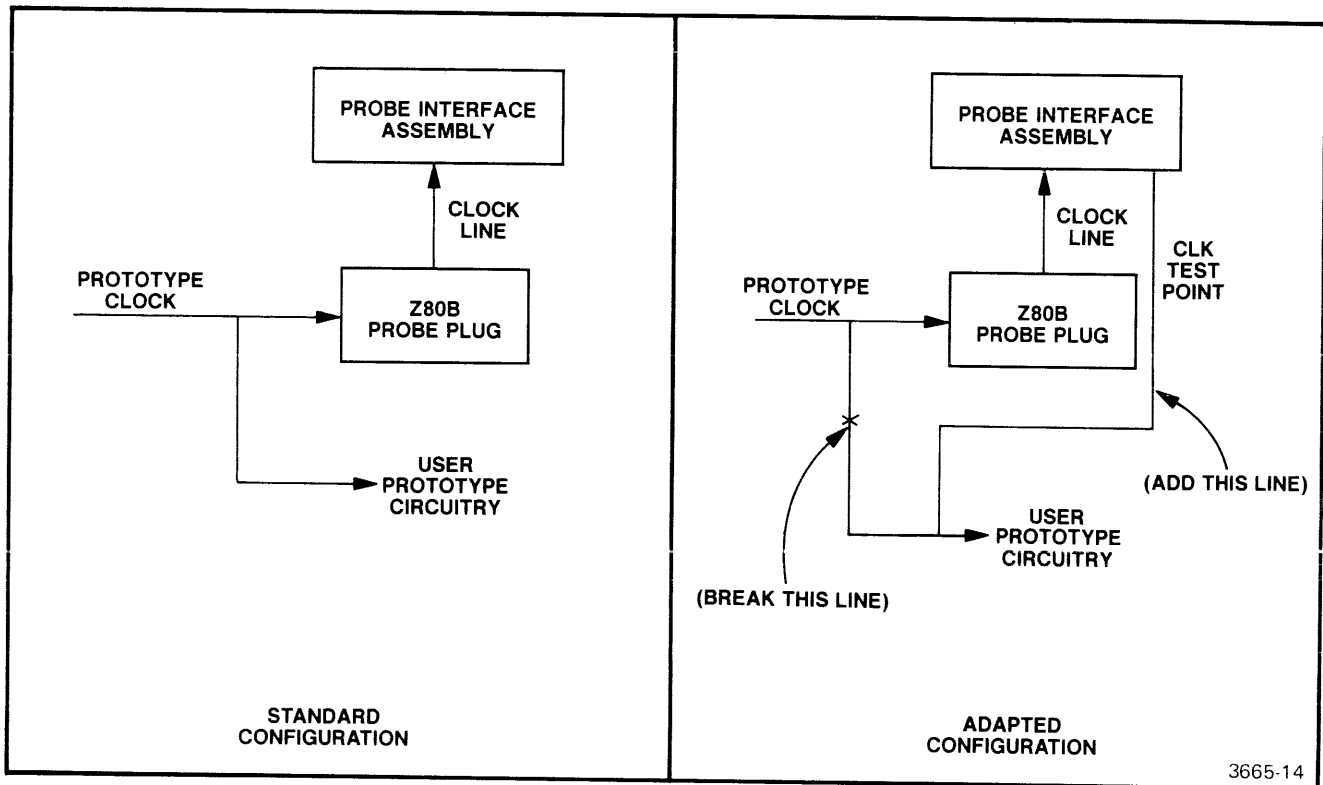


Fig. 7C-3. Connecting the prototype clock input directly to the Prototype Control Probe.

## REDUCING DELAY THROUGH THE PROTOTYPE CONTROL PROBE (Z80B ONLY)

The clock test point (on the Driver/Receiver board) can be used to obtain more accurate emulator timing under worst-case conditions above 4 MHz operation. In order to use the clock test point, you must disconnect the prototype clock input pin from the prototype logic, and reconnect it directly to the clock test point in the Prototype Control Probe using a plug-on connector.

The clock test point is located on the output of U1050 (pin 18 through a 68  $\Omega$  resistor in series), and the output is within 10 ns of the actual CPU clock. When the prototype clock is connected directly to the clock source in the Prototype Control Probe, the 20 ns delay through probe circuitry is circumvented. Figure 7C-3 illustrates the standard configuration and the adapted configuration for the prototype circuitry when implementing this clock test point user adaptation.

**Table 7C-4**  
**Representative Z80A Emulator/Z80A Microprocessor Timing Differences**

Symbol	Parameter	Processor		Emulator		Units
		Min.	Max.	Min.	Max.	
t(c)	Clock period	250	<sup>a</sup>	250	<sup>a</sup>	ns
t(w( $\phi$ H))	Clock pulse width, clock high	110	<sup>b</sup>	110	<sup>b</sup>	ns
t(w( $\phi$ L))	Clock pulse width, clock low	110	2000	110	2000	ns
t(r,f)	Clock rise and fall time		30	<sup>c</sup>	30	ns
t(D(AD))	Address output delay		110		130	ns
t(F(AD))	Delay to float				85 <sup>d</sup>	ns
t(D(D))	Data output delay		150		170	ns
t(F(F))	Delay to float during write cycle		90		110 <sup>e</sup>	ns
t(S $\phi$ (D))	Data setup time to rising edge of clock during M1 cycle	50		70 <sup>f</sup>		ns
t(S $\phi$ (D))	Data setup time of falling edge of clock during M2 to M5	60		80 <sup>f</sup>		ns
t(DL $\phi$ (MR))	MREQ delay from falling edge of clock, MREQ low	20	85	35 <sup>g</sup>	100 <sup>g</sup>	ns
t(DH $\phi$ (MR))	MREQ delay from rising edge of clock, MREQ high		85		100 <sup>g</sup>	ns
t(DH $\phi$ (MR))	MREQ delay from falling edge of clock, MREQ high		85		100 <sup>g</sup>	ns
t(DL $\phi$ (IR))	IORQ delay from rising edge of clock, IORQ low		75		90 <sup>g</sup>	ns
t(DL $\phi$ (IR))	IORQ delay from falling edge of clock, IORQ low		85		100 <sup>g,h</sup>	ns
t(DH $\phi$ (IR))	IORQ delay from rising edge of clock, IORQ high		85		100 <sup>g</sup>	ns
t(DH $\phi$ (IR))	IORQ delay from falling edge of clock, IORQ high		85		100 <sup>g</sup>	ns
t(DL $\phi$ (RD))	RD delay from rising edge of clock, RD low		85		95 <sup>g</sup>	ns
t(DL $\phi$ (RD))	RD delay from falling edge of clock, RD low		95		105 <sup>g</sup>	ns
t(DH $\phi$ (RD))	RD delay from rising edge of clock, RD high		85		95 <sup>g</sup>	ns
t(DH $\phi$ (RD))	RD delay from falling edge of clock, RD high		85		95 <sup>g</sup>	ns
t(DL $\phi$ (WR))	WR delay from rising edge of clock, WR low		65		75 <sup>g</sup>	ns

Table 7C-4 (cont)  
Representative Z80A Emulator/Z80A Microprocessor Timing Differences

Symbol	Parameter	Processor		Emulator		Units
		Min.	Max.	Min.	Max.	
t(DL $\phi$ (WR))	WR delay from falling edge of clock, WR low		80		90 <sup>g</sup>	ns
t(DH $\phi$ (WR))	WR delay from falling edge of clock, WR high		80		90 <sup>g</sup>	ns
t(DL(M1))	M1 delay from rising edge of clock, M1 low		100		110 <sup>g</sup>	ns
t(DH(M1))	M1 delay from rising edge of clock, M1 high		100		110 <sup>g</sup>	ns
t(DL(RF))	RFSH delay from rising edge of clock, RFSH low		130		140 <sup>g</sup>	ns
t(DH(RF))	RFSH delay from rising edge of clock, RFSH high		120		130 <sup>g</sup>	ns
t(S(WT))	WAIT setup time to falling edge of clock	70		90 <sup>f</sup>		ns
t(D(HT))	HALT delay time from falling edge of clock		300		315 <sup>g</sup>	ns
t(s(IT))	INT setup time to rising edge of clock	80		125 <sup>f</sup>		ns
t(s(BQ))	BUSRQ setup time to rising edge of clock	50		85 <sup>f</sup>		ns
t(DL(BA))	BUSAK delay from rising edge of clock, BUSAK low		100		110 <sup>g</sup>	ns
t(DH(BA))	BUSAK delay from falling edge of clock, BUSAK high		100		110 <sup>g</sup>	ns
t(s(RS))	RESET setup time to rising edge of clock	60		105 <sup>f</sup>		ns
t(F(C))	Delay to/from float (MREQ, IORQ, RD, and WR)		80		115 <sup>g</sup>	ns
t(mr)	M1 stable prior to IORQ Interrupt Ack.)		<sup>i</sup>		<sup>h,j</sup>	ns

$${}^a t(c) = t(w(\phi H)) + t(w(\phi L)) + t(r) + t(f)$$

<sup>b</sup>Although static by design, testing guarantees a t(w( $\phi$ H)) of 200  $\mu$ s maximum.

<sup>c</sup>Clock delay from the prototype to emulator CPU is 20 ns maximum.

<sup>d</sup>Delay measured from BUSAK asserted at CPU.

<sup>e</sup>Data will go to an indeterminate state, but will not tri-state unless BUSREQ is asserted.

<sup>f</sup>Timing reference is to CPU clock. To reference prototype clock, subtract propagation delay through driver/receiver buffers (10 ns maximum).

<sup>g</sup>Timing reference to CPU clock. To reference prototype clock, add propagation delay through driver/receiver buffers (20 ns maximum.)

<sup>h</sup>IORQ can be delayed to maximum of 153 ns during INTA cycles while the Debug TRA MODE is active.

$${}^i t(mr) = 2 * t(c) + t(w(\phi H)) + t(f) - 65$$

CPU timing reference: *Mostek Microcomputer Z80 Data Book*; Mostek Corporation. (1978).

## Z80B EMULATOR TIMING

In Emulation Modes 1 and 2, the emulating microprocessor resides in the Prototype Control Probe, and the signals between the prototype and the emulating microprocessor are buffered. Therefore, some timing differences exist between the Z80B emulator and a Z80B microprocessor which has been inserted directly into the prototype. Table 7C-5 lists these differences. Figure 7C-4 is a timing diagram corresponding to the signals present on the Z80B emulator.

**Table 7C-5**  
**Representative Z80B Emulator/Z80B Microprocessor Timing Differences**

Symbol	Parameter	Processor		Emulator		Units
		Min.	Max.	Min.	Max.	
t(c)	Clock period	165	<sup>a</sup>	165	<sup>a</sup>	ns
t(w( $\phi$ H))	Clock pulse width, clock high	70	<sup>b</sup>	70	<sup>b</sup>	ns
t(w( $\phi$ L))	Clock pulse width, clock low	70	2000	70	2000	ns
t(r,f)	Clock rise and fall time		30	<sup>c</sup>	30	ns
t(D(AD))	Address output delay		80		100	ns
t(F(AD))	Delay to float				33 <sup>d</sup>	ns
t(D(D))	Data output delay		120		140	ns
t(F(F))	Delay to float during write cycle		60		80 <sup>e</sup>	ns
t(S $\phi$ (D))	Data setup time to rising edge of clock during M1 cycle	25		4.5 <sup>d</sup>		ns
t(S $\phi$ (D))	Data setup time to falling edge of clock during M2 to M5	30		5.0 <sup>f</sup>		ns
t(DL $\phi$ (MR))	MREQ delay from falling edge of clock, MREQ low	20	60	35 <sup>g</sup>	75 <sup>g,h</sup>	ns
t(DH $\phi$ (MR))	MREQ delay from rising edge of clock, MREQ high		60		75 <sup>g</sup>	ns
t(DH $\phi$ (MR))	MREQ delay from falling edge of clock, MREQ high		60		75 <sup>g</sup>	ns
t(DL $\phi$ (IR))	IORQ delay from rising edge of clock, IORQ low		60		75 <sup>g,h</sup>	ns
t(DL $\phi$ (IR))	IORQ delay from falling edge of clock, IORQ low		60		75 <sup>g</sup>	ns
t(DH $\phi$ (IR))	IORQ delay from rising edge of clock, IORQ high		60		75 <sup>g</sup>	ns
t(DH $\phi$ (IR))	IORQ delay from falling edge of clock, IORQ high		60		75 <sup>g</sup>	ns
t(DL $\phi$ (RD))	RD delay from rising edge of clock, RD low		60		70 <sup>g</sup>	ns
t(DL $\phi$ (RD))	RD delay from falling edge of clock, RD low		70		80 <sup>g</sup>	ns

**Table 7C-5 (cont)**  
**Representative Z80B Emulator/Z80B Microprocessor Timing Differences**

Symbol	Parameter	Processor		Emulator		Units
		Min.	Max.	Min.	Max.	
t(DH $\phi$ (RD))	RD delay from rising edge of clock, RD high		60		70 <sup>g</sup>	ns
t(DH $\phi$ (RD))	RD delay from falling edge of clock, RD high		70		80 <sup>g</sup>	ns
t(DL $\phi$ (WR))	WR delay from rising edge of clock, WR low		60		70 <sup>g</sup>	ns
t(DL $\phi$ (WR))	WR delay from falling edge of clock, WR low		55		65 <sup>g</sup>	ns
t(DH $\phi$ (WR))	WR delay from falling edge of clock, WR high		55		65 <sup>g</sup>	ns
t(DL(M1))	M1 delay from rising edge of clock, M1 low		80		90 <sup>g</sup>	ns
t(DH(M1))	M1 delay from rising edge of clock, M1 high		80		90 <sup>g</sup>	ns
t(DL(RF))	RFSH delay from rising edge of clock, RFSH low		100		110 <sup>g</sup>	ns
t(DH(RF))	RFSH delay from rising edge of clock, RFSH high		100		110 <sup>g</sup>	ns
t(S(WT))	WAIT setup time to falling edge of clock	40		60 <sup>f</sup>		ns
t(D(HT))	HALT delay time from falling edge of clock		200		215 <sup>g</sup>	ns
t(s(IT))	INT setup time to rising edge of clock	45		90 <sup>f</sup>		ns
t(s(BQ))	BUSRQ setup time to rising edge of clock	40		75 <sup>f</sup>		ns
t(DL(BA))	BUSAK delay from rising edge of clock, BUSAK low		65		75 <sup>g</sup>	ns
t(DH(BA))	BUSAK delay from falling edge of clock, BUSAK high		55		65 <sup>g</sup>	ns
t(s(RS))	RESET setup time to rising edge of clock	45		90 <sup>f</sup>		ns
t(F(C))	Delay to/from float (MREQ, IORQ, RD, and WR)		60		95 <sup>g</sup>	ns
t(mr)	M1 stable prior to IORQ (Interrupt Ack.)	i		ij		ns

<sup>a</sup>t(c) = t(w( $\phi$ H)) + t(w( $\phi$ L)) + t(r) + t(f).

<sup>b</sup>Although static by design, testing guarantees a t(w( $\phi$ H)) of 200  $\mu$ s maximum.

<sup>c</sup>Clock delay from prototype to emulator CPU is 20 ns maximum.

<sup>d</sup>Delay measured from BUSAK asserted at CPU.

<sup>e</sup>Data will go to an indeterminate state, but will not tri-state unless BUSREQ is asserted.

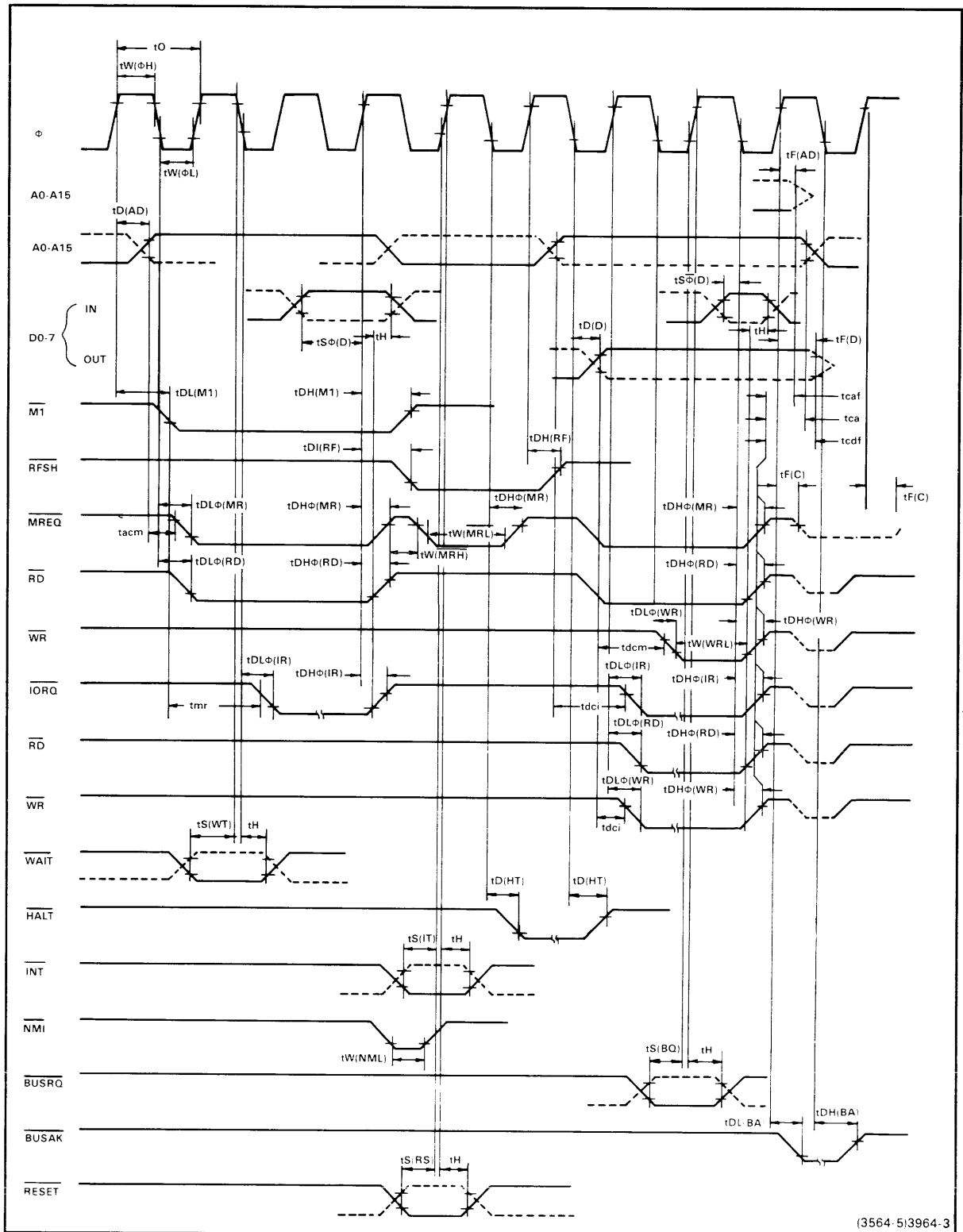
<sup>f</sup>Timing reference is to CPU clock. To reference prototype clock, subtract propagation delay through Driver/Receiver Buffers (10 ns maximum).

<sup>g</sup>Timing reference to CPU clock. To reference prototype clock, add propagation delay through Driver/Receiver Buffers (20 ns maximum).

<sup>h</sup>Emulation Mode 1, on a memory change from system to prototype, MREQ or IORQ would be delayed up to 184 ns.

<sup>i</sup>t(mr) = 2 \* t(c) + t(w( $\phi$ H)) + t(f) - 50.

<sup>j</sup>IORQ can be delayed a maximum of 153 ns during INTA cycles while the Debug TRA MODE is active.



(3564-5)3964-3

Fig. 7C-4. Z80 microprocessor bus timing.



## Z80A PROBE/PROTOTYPE INTERFACE DIAGRAM

Figure 7C-5 is a block diagram of the interface between the prototype and the emulating microprocessor in the Prototype Control Probe. This figure provides a functional overview of emulator buffering. Signal buffers labeled with a generic chip type (i.e., LS244) represent single level buffering. Non-labeled blocks represent possible multi-level buffering. A more detailed circuit description can be found in the Z80 Emulator Processor Service Manual.

## Z80B PROBE/PROTOTYPE INTERFACE DIAGRAM

Figure 7C-6 is a block diagram of the interface between the prototype and the emulating microprocessor in the Prototype Control Probe. This figure provides a functional overview of emulator buffering. Signal buffers labeled with a generic chip type (i.e., LS241) represent single level buffering. Non-labeled blocks represent possible multi-level buffering. A more detailed circuit description can be found in the Z80 Emulator Processor Service Manual.

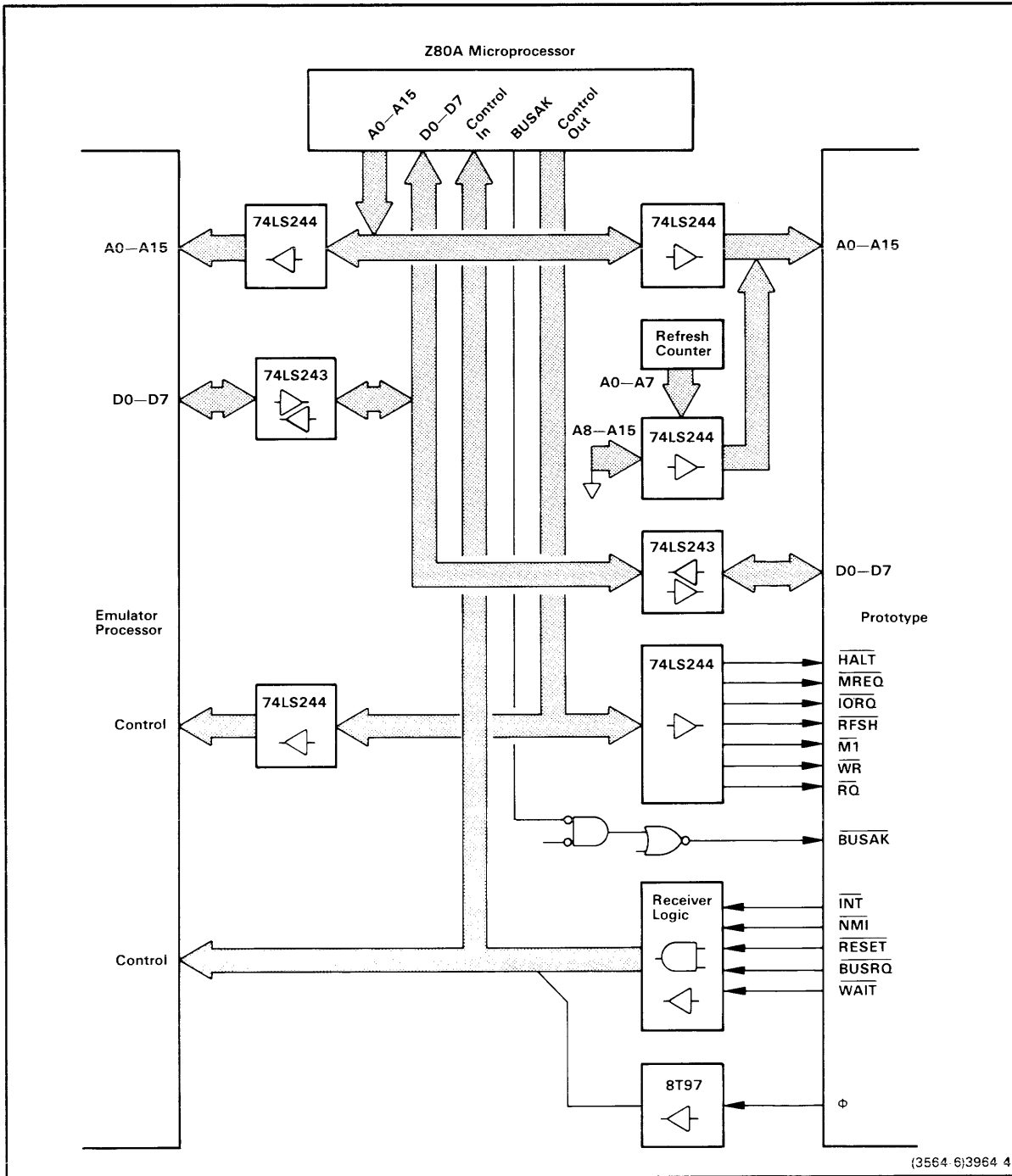


Fig. 7C-5. Block diagram of Z80A emulator/prototype interface.

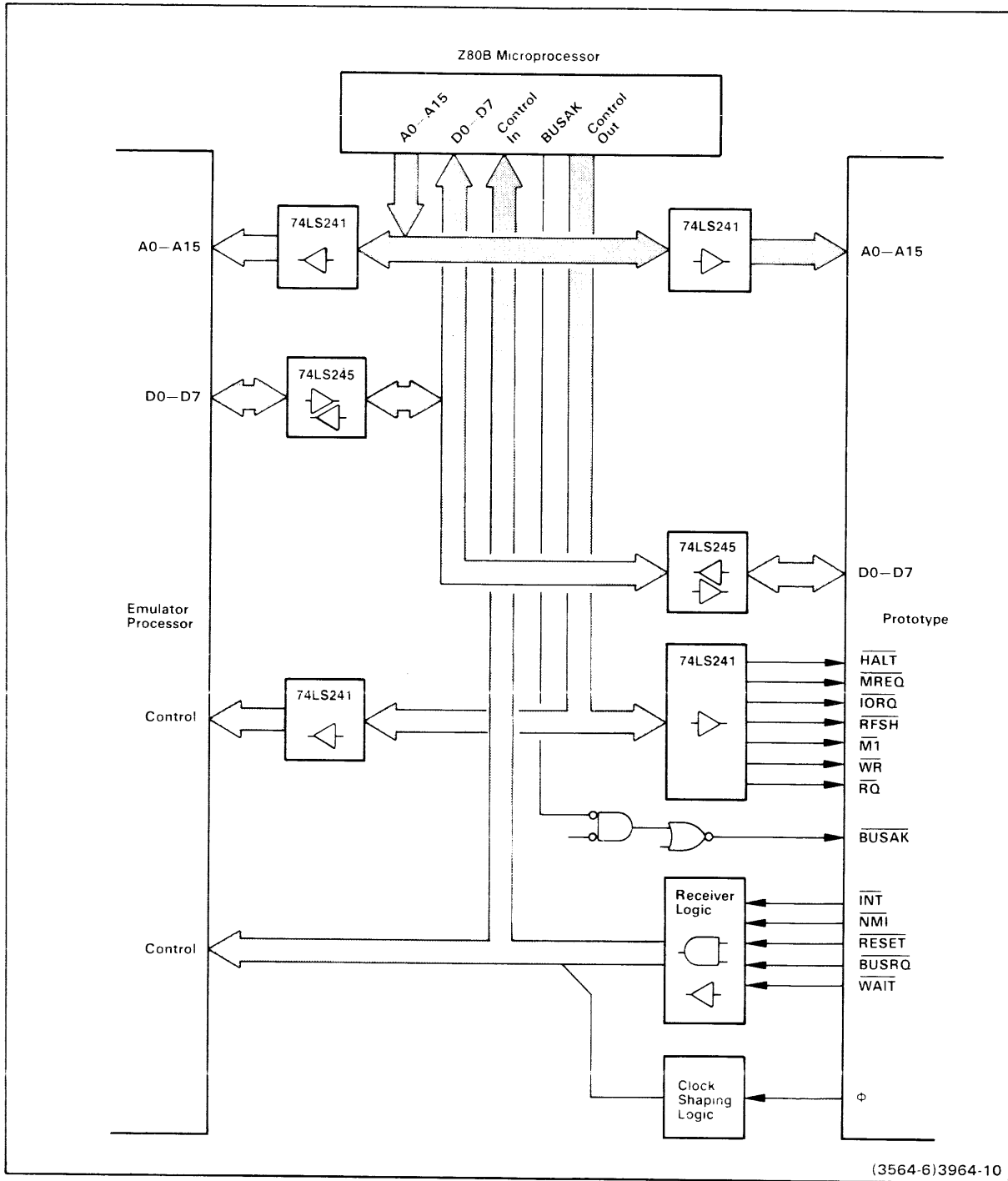


Fig. 7C-6. Block diagram of Z80B emulator/prototype interface.

## INSTALLING YOUR Z80 EMULATOR SOFTWARE

### 8540 FIRMWARE INSTALLATION PROCEDURE

The ROM devices that contain the control software for your Z80 emulator must be installed in your 8540's System ROM board. Refer to your Emulator Installation Manual for instructions on installing these ROM.

### 8550 SOFTWARE INSTALLATION PROCEDURE

Your emulator installation software consists of two disks:

- a disk that contains emulator control software, which you install onto your DOS/50 system disk so that DOS/50 can control your emulator hardware.
- a disk that contains Z80 emulator diagnostic software. For a Z80A emulator, you must install this disk onto your 8550 system diagnostic disk so that diagnostic tests can be run on your emulator as well as other 8550 system hardware. For a Z80B emulator, the diagnostic disk can be used directly.

This subsection describes how to install the emulator control software for a Z80 emulator.

To complete this installation procedure you need the following items:

- an 8550 system (with or without a Z80 emulator);
- a DOS/50 system disk with a write-enable tab over the write-protect slot;
- a Z80 emulator software installation disk with no write-enable tab; and
- (for installation of Z80A diagnostic software) an 8550 system diagnostic disk with a write-enable tab over the write-protect slot.

This installation procedure takes about five minutes.

#### Start Up and Set the Date

Turn on your 8550 system. (For start-up instructions, refer to the Learning Guide of your 8550 System Users Manual.) Place your system disk in drive 0 and shut the drive 0 door. When you see the > prompt on your system terminal, place your installation disk in drive 1 and shut the drive 1 door.

Use the DAT command to set the date and time. For example, if it is 11:05 am on October 30, 1983, type:

```
> DAT 30-OCT-83/11:05 <CR>
```

The system uses this information when it sets the CREATION time attribute of each file copied from your installation disk.

## Install the Emulator Control Software

The command file INSTALL2, which installs the emulator control software, resides on the installation disk.

### NOTE

*If your system disk contains DOS/50 Version 1, use the command file INSTALL instead of INSTALL2.*

To execute the command file, simply type its filespec:

```
> /VOL/EMU.Z80/INSTALL2 <CR>
```

DOS/50 responds with the following message:

```
* During this installation procedure, one or more of the
* following messages may appear.  IGNORE THESE MESSAGES:
*
*      Error 6E—Directory alteration invalid
*      Error 7E—Error in command execution
*      Error 1D—File not found
*
* If any OTHER error message appears, see your
* Users Manual for further instructions.
*
* If no other error message appears, you'll receive a
* message when the installation procedure is complete.
*
T, OFF
```

In this installation procedure, you may disregard error messages 6E, 7E, and 1D; these messages have no bearing on the success of the installation. However, if a message **other** than 6E, 7E, or 1D appears, take the following steps:

1. Make sure you are using the right disks.
2. Make sure your system disk has a write-enable tab.
3. Make sure there are at least three files and 20 free blocks on your system disk.
4. Begin the installation procedure again.

If the installation procedure fails again, copy down the error message and contact your Tektronix service representative.

The T, OFF command suppresses subsequent output to your system terminal (except error messages) until INSTALL2 finishes executing. Within about five minutes, INSTALL2 will finish and your system terminal will display the following message:

```
*
* Your installation has been successfully completed.
>
```

## Install the Emulator Diagnostic Software

If you are using a Z80A emulator, you can now install the emulator diagnostic software. If you are using a Z80B emulator, you can use the emulator diagnostic software disk as provided.

**Note the Name of Your Diagnostic Disk.** In order to install the emulator diagnostic software, you must know the name of your 8550 system diagnostic disk. Remove your emulator installation disk from drive 1 and insert the diagnostic disk. Enter the following command to list the names of the two disks mounted in your 8550:

```
> ATT /VOL/* WHERE <CR>
sysvol          WHERE=FLX0 ← DOS/50 system disk
8550DIAGx.x     WHERE=FLX1 ← 8550 system diagnostic disk
```

Note the name of your diagnostic disk. (It should be something like 8550DIAG2.0.)

**Insert Your Emulator Installation Disk into Drive 1.** INSTALLDIAGS, the command file that installs the diagnostics, resides on the installation disk. Remove your diagnostic disk from drive 1 and insert your installation disk. Invoke the INSTALLDIAGS command file and pass it the name of your diagnostic disk, which you just noted:

```
> /VOL/EMU.Z80/INSTALLDIAGS 8550DIAGx.x <CR>
```

DOS/50 responds with the following messages:

```
*
*****
*          DIAGNOSTIC INSTALLATION PROCEDURE          *
*****
*
*   During this installation procedure, the following error
*   message will appear once.  IGNORE THIS MESSAGE:
*
*           Error 2A Parameter required
*
*   If any OTHER error message appears or this appears more
*   than once, see your Users Manual for further instructions.
*
*   If no other error message appears, you'll receive a message
*   when the installation is complete.
*
T,OFF
COP:          Error 2A Parameter required
*
* → Remove the DOS/50 System Disc
* → Insert the 8550 System Diagnostic Disc
* → Type C0 -A
*
SUSP, -A
>>
```

**Insert Your Diagnostic Disk into Drive 0.** Remove your 8550 system disk from drive 0 and insert your 8550 system diagnostic disk. Then enter the command C0 -A to continue execution of the command file:

```
> C0 -A <CR>
```

After a few minutes, the following message is displayed:

```
COP, -BN,/VOL/EMU.Z80/DIAGS/Z80.TST,/VOL/8550DIAGx.x/Z80.TST
* -> Remove 8550 System Diagnostic Disc
* -> Insert DOS/50 System Disc
* -> Press CTRL-C
* -> Type C0 -A
*
SUSP, -A
```

**Insert Your DOS/50 System Disk into Drive 0.** Remove your diagnostic disk from drive 0 and insert your DOS/50 system disk. Then enter the C0 -A command again:

```
> C0 -A <CR>
```

The command file finishes with the following message:

```
USER, ,NO.NAME
*****
*      DIAGNOSTIC INSTALLATION COMPLETE      *
*****
>
```

In this installation procedure, error message 2A should appear once. If any other error message appears, check your disks and begin the diagnostic installation procedure again. If the installation procedure fails again, copy down the error message and contact your Tektronix service representative.

Once your software is installed, you can:

- remove your disks and turn off your 8550 system, or
- install more software, or
- continue with the Z80 Emulator Demonstration Run that follows in this section. If you do this, you do not have to restart the system or reset the date and time.

#### NOTE

*At this point NO.NAME is the current user. To change the current user back to yourname, enter USER,,yourname.*

## Z80 DEMONSTRATION RUN

### INTRODUCTION

This demonstration run shows you how to load, execute, and monitor a simple Z80 assembly language program on your 8540 or 8550. In order to perform this demonstration, your Z80 emulator hardware and control software must be installed in your 8540 or 8550.

Figure 7C-8 shows the source and object code for the demonstration program.

If you have an 8550 (as in Fig. 7C-7, Case 1), the source code and object code for the demonstration program are provided on the installation disk that contains your Z80 emulator control software. This demonstration shows you how to assemble the program on your 8550. (If your system disk does not contain a Z80 assembler, you will have to skip that part of the demonstration.)

If you have an 8540/8560 system (See Fig. 7C-7, Case 2), and your 8560 has a Z80 assembler installed, you can create and assemble the program on the 8560 and then download it to the 8540. This demonstration shows how.

If you have an 8540 (Fig. 7C-7, Case 3) that is connected to a host computer other than an 8560, we can't give you a specific list of commands for creating and assembling the program on your host (since we don't know what host you're using). However, Fig. 7C-9 gives the object code for the program in Extended Tekhex format. You can create the Tekhex file using your host's assembler or text editor, and then download the file to the 8540 via the 8540's optional COM interface.

If none of these cases applies to you, you can patch the program into memory by using

the P command. This demonstration shows how.

Once the program is loaded or patched into memory, you can execute the program on your emulator.

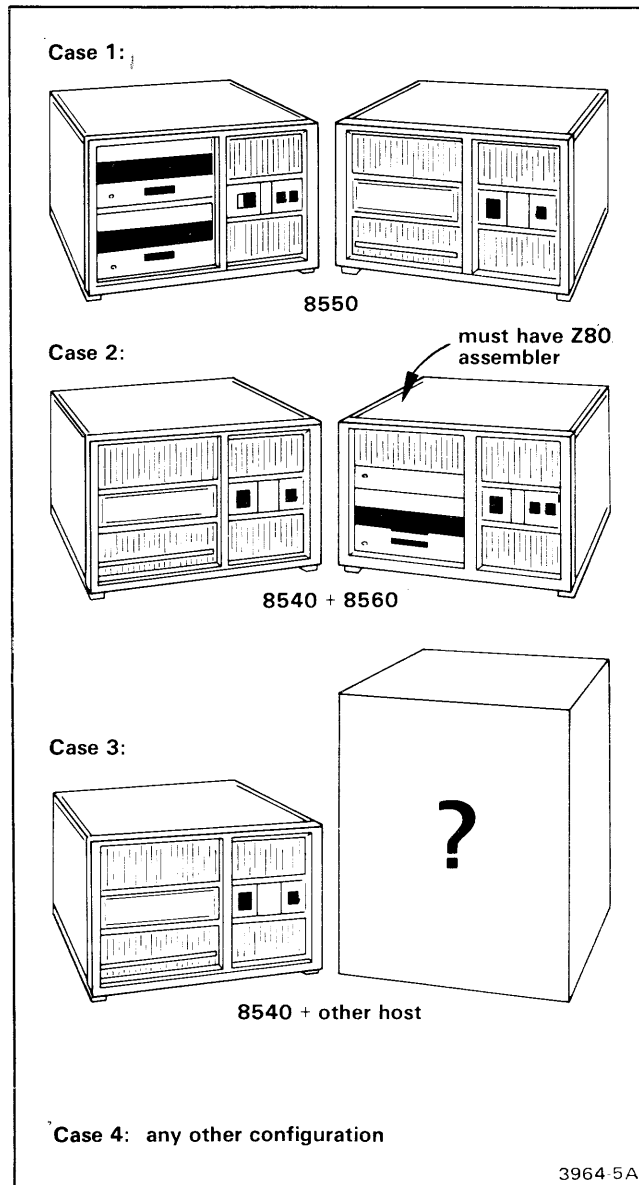


Fig. 7C-7. System configurations.



## NOTE

The 8540 commands shown in this demonstration can also be used for an 8550 that is connected to an 8560 or other host computer.

```

01             ;Z80 DEMONSTRATION RUN PROGRAM
02             SECTION DEMO
03             ORG 100H           ;START PROGRAM CODE AT ADDRESS 100
04 000100 210005  START LD HL,TABLE ;SET TABLE POINTER
05 000103 0605    LD B,TSIZE  ;SET PASS COUNTER
06 000105 AF     XOR A         ;CLEAR ACCUMULATOR
07 000106 86     LOOP  ADD A,(HL) ;ADD BYTE FROM TABLE
08 000107 23     INC HL       ;POINT TO NEXT BYTE
09 000108 05     DEC B        ;DECREMENT PASS COUNTER
10 000109 C20601 JP NZ,LOOP   ;LOOP IF NOT FIVE PASSES YET
11 00010C D3F7   OUT (OF7H),A ;OTHERWISE CALL EXIT SVC
12 00010E 00     NOP          ; TO END PROGRAM EXECUTION
13             ;SRB POINTER
14             ORG 40H          ;STORE SRB POINTER AT ADDRESS 40
15 000040 000042 BYTE 00,42H  ;POINT TO SRB FOR EXIT SVC
16             ;SRB FOR EXIT SVC
17 000042 1A     BYTE 1AH     ;1AH = FUNCTION CODE FOR EXIT SVC
18             ;TABLE OF NUMBERS TO BE ADDED
19             TSIZE EQU 5     ;TABLE SIZE = 5
20             ORG 500H        ;SET UP TABLE AT ADDRESS 500
21             TABLE BLOCK TSIZE
22             LIST DBG
23             END START
=====
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
+--- object code
+----- address
+----- source code line number
          source code            comments

```

Fig. 7C-8. Demonstration program.

```

(A)

%2769231002100050605AF862305C20601D3F700
%0E62B24000421A
%3A3494DEM0010350514LOOP310615START310015TABLE350025TSIZE15
%098153100

(B)

FIRST DATA BLOCK: object code for addresses 100--10E

header
  | load address  object code
  |-----|
%2769231002100050605AF862305C20601D3F700

SECOND DATA BLOCK: object code for addresses 40--42

header
  | load object
  | address code
  |-----|
%0E62B24000421A

SYMBOL BLOCK

header
  | section      section
  | name        definition
  |-----|
%3A3494DEM0010350514LOOP310615START310015TABLE350025TSIZE15

TERMINATION BLOCK

header
  | transfer
  | address
  |-----|
%098153100
    
```

Fig. 7C-9. Demonstration program: Extended Tekhex format.

Figure 7C-9A shows an Extended Tekhex load module that contains the object code and program symbols for the demonstration program. Figure 7C-9B labels the different fields in the message blocks. If you have a host computer other than an 8560, you can create this load module and download it to your 8540 or 8550.

## EXAMINE THE DEMONSTRATION PROGRAM

The demonstration program adds five numbers from a table stored in locations 500 to 504 in program memory and leaves the sum in register A. (You will place values in the table later in this demonstration.) The 8085A emulator demonstration run in the Learning Guide of your System Users Manual contains a flowchart that illustrates the steps of the Program.

The source code contains two kinds of statements: assembler directives (such as ORG and BYTE) and Z80 assembly language instructions. The assembler directives are microprocessor-independent and are explained in the 8085A emulator demonstration run. The Z80 assembly language instructions are discussed in the following paragraphs.

**Set Table Pointer.** The LD HL, TABLE instruction loads the address of the table (500) into the H-L register pair. As a result, the H-L pair points to the first element of the table. The label START is used by the END directive to specify that the LD HL, TABLE instruction is the first to be executed.

**Set Pass Counter.** Register B is used as the pass counter. The LD B, TSIZE instruction loads the value 5 into register B. This step sets the number of passes to 5.

**Clear Accumulator.** The XOR A instruction zeros the accumulator (register A) so you can start adding numbers from the table.

**Add Byte from Table.** The ADD A, (HL) instruction adds the byte addressed by the H-L register pair into the accumulator. The label LOOP represents the address of this instruction; this label is used by the JP NZ instruction.

**Point to Next Byte.** The INC HL instruction increments the address contained by the H-L register pair; the H-L register pair then points to the next byte in the table. For example, the H-L register pair is initialized to contain 500. After the INC HL instruction is first executed, the H-L register pair will contain 501, the address of the second byte in the table.

**Decrement Pass Counter.** The DEC B instruction decrements register B, the pass counter. In this program, B is decremented each time a number is added to the accumulator.

**Loop If Not Five Passes Yet.** The JP NZ, LOOP instruction checks the contents of register B and jumps to the LOOP label if B does not contain zero. If B contains zero, the program proceeds to the next instruction, OUT (OF7H),A.

**Exit.** The OUT (OF7H),A and NOP instructions constitute a service call (SVC) that causes an exit from the program. For more information on SVCs, refer to the Service Calls section of your System Users Manual.

## ASSEMBLE AND LOAD THE DEMONSTRATION PROGRAM

Now it's time to create the program so you can run it on your emulator. One of the following discussions describes the set of steps that is appropriate for your hardware configuration:

- **For 8550 users**—Case 1: Assemble and Load on the 8550
- **For 8540/8560 users**—Case 2: Assemble on the 8560; Download to the 8540
- **For 8540 users with a host computer other than the 8560**—Case 3: Download from Your Host to the 8540
- **For other hardware configurations**—Case 4: Patch the Program into Memory

Go ahead and work through the discussion that's appropriate for you. Once you've put the program into program memory, turn to the heading Run the Demonstration Program, later in this section.

### CASE 1: ASSEMBLE AND LOAD ON THE 8550

This discussion shows you how to copy the demonstration program from your Z80 emulator software installation disk, assemble the program, and load it into 8550 program memory.

#### Start Up and Log On

Turn on your 8550 system. (For start-up instructions, refer to the paragraph Start Up the 8550 and Its Peripherals in the Learning Guide of your System Users Manual.) Place your system disk in drive 0 and shut the drive 0 door. When your system displays the > prompt, place your Z80 emulator software installation disk in drive 1 and shut the drive 1 door.

Use the DAT command to set the current date and time. For example, if it is 2:30 pm on October 31, 1981, enter the following command line:

```
> DAT 31-OCT-81/2:30 PM <CR>
```

Use the SEL command to tell DOS/50 to use the assembler and emulator software designed for the Z80:

```
> SEL Z80 <CR>
```

The SEL command automatically sets the emulation mode to 0.

#### Copy the Demonstration Run Program from the Installation Disk

Enter the following command lines to create an empty directory called DEMO on your system disk and make DEMO the current directory. The BR command creates a brief name, ROOT, to mark the old current directory. At the end of this demonstration, you will return to this ROOT directory and delete the DEMO directory and its contents.

```
> BR ROOT/USR <CR>  
> CREATE DEMO <CR>  
> USER DEMO <CR>
```

Now use the COP command to copy all the files in the DEMO2 directory on the installation disk to the DEMO directory you just created:

```
> COP /VOL/EMU.Z80/DEMO2/* * <CR>
```

Remove your installation disk from drive 1 and put it away.

Now list the files you have just copied to the current directory:

```
> L <CR>
```

```
FILENAME
```

```
ASM
```

```
LOAD
```

```
Files used      124
Free files      132
Free blocks     821
Bad blocks      0
```

The file named ASM contains the assembly language source code for this demonstration program, and the file named LOAD contains the executable object code. This copy of LOAD will be used in the demonstration only if you do not have a Z80 assembler (and thus cannot create your own object file and load file from the source file.)

### Examine the Demonstration Program

Enter the following command line to display the source file ASM on the system terminal:

```
> CON ASM <CR>
;Z80 DEMONSTRATION RUN PROGRAM
SECTION DEMO
ORG 100H ;START PROGRAM CODE AT ADDRESS 100
START LD HL,TABLE ;SET TABLE POINTER
LD B,TSIZE ;SET PASS COUNTER
XOR A ;CLEAR ACCUMULATOR
LOOP ADD A,(HL) ;ADD BYTE FROM TABLE
INC HL ;POINT TO NEXT BYTE
DEC B ;DECREMENT PASS COUNTER
JP NZ,LOOP ;LOOP IF NOT FIVE PASSES YET
OUT (OF7H),A ;OTHERWISE CALL EXIT SVC
NOP ;TO END PROGRAM EXECUTION
;SRB POINTER
ORG 40H ;STORE SRB POINTER AT ADDRESS 40
BYTE 00,42H ;POINT TO SRB FOR EXIT SVC
;SRB FOR EXIT SVC
BYTE 1AH ;1AH = FUNCTION CODE FOR EXIT SVC
;TABLE OF NUMBERS TO BE ADDED
TSIZE EQU 5 ;TABLE SIZE = 5
ORG 500H ;SET UP TABLE AT ADDRESS 500
TABLE BLOCK TSIZE
LIST DBG
END START
```

### Assemble the Source Code

If you do not have a Z80 assembler on your system disk, you cannot perform this step, so skip the next four commands (ASM, COP, LINK, and L).

The ASM (AsSeMble) command translates assembly language (source code) into binary machine language (object code). The ASM command also creates an assembler listing that can be used to correlate the object code with the source code. Enter the following command line to assemble the source code in the file ASM and create the listing the object files ASML and OBJ:

```
> ASM OBJ ASML ASM <CR>
```

----- source file  
 ----- assembler listing file  
 ----- object file

```
Tektronix      Z80 ASM Vx.x
**** Pass 2
   23 Source Lines  23 Assembled Lines  xxxxx Bytes Available
>>> No assembly errors detected <<<<
```

Make sure your line printer is turned on and properly connected, then enter the following command to copy the assembler listing onto the printer:

```
> COP ASML LPT <CR>
```

Refer to Fig. 7C-8 for an explanation of the different fields in your assembler listing. For a more detailed explanation, consult your Assembler Users Manual.

**Link the Object Code.** The linker creates an executable load file from one or more object files. Enter the LINK command to invoke the linker:

```
> LINK <CR>
8550 LINKER Vx.x
*
```

Now enter the following linker commands to create a load file called LOAD from your object file, OBJ:

```
*LINK OBJ <CR>
*LOAD LOAD <CR>
*DEBUG <CR>
*END <CR>
```

The linker commands LINK and LOAD specify the object file and load file, respectively. The DEBUG command causes the linker to pass the program symbols from the object file to the load file, for use in program debugging. After you type the END command, the linker executes the commands you have entered, and the following information is displayed:

```
NO ERRORS          NO UNDEFINED SYMBOLS
1 MODULE           1 SECTIONS
TRANSFER ADDRESS IS 0100
```

The files generated by the ASM and LINK commands should now be on your disk. Enter the following command to list the files in your current directory:

```
> L <CR>
```

```
FILENAME
```

```
ASM
LOAD
OBJ
ASML
```

```
Files used      126
Free files      130
Free blocks     811
Bad blocks      0
```

Notice that there are now four files listed in your directory. OBJ and ASML were created by the assembler, and LOAD was created by the linker.

### Load the Program into Memory

Now it's time to load the object code from the load file LOAD into program memory. Once you've loaded the object code, you execute the program.

**Zero Out Memory.** Before you load any code, use the F (Fill) command to fill program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no effect on how the program is loaded.) Enter the following command line to fill memory addresses 40 to 11F with zeros:

```
> F 40 11F 00 <CR>
```

**Check That Memory Was Filled with Zeros.** Check the contents of memory with the D (Dump) command. The D command's display shows the data (in hexadecimal format) and also shows the corresponding ASCII characters. Display the contents of memory addresses 40 to 11F with the following command line:

```
> D 40 11F <CR>
```

```

  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

**Load the Object Code into Memory.** Enter the following command line to load the object code for the demonstration program into program memory:

```
> LO <LOAD <CR>
    == ==
    ↑
  load file
```

**Load the Program Symbols.** The source code for the demonstration program contained the directive LIST DBG. Because of this directive, the object file contains a list of the symbols that appeared in the source code, and the values associated with those symbols. Because you included the DEBUG command when you invoked the linker, those symbols were passed to the load file. Use the SYMLO command to load those symbols into the symbol table in 8550 system memory.

```
> SYMLO -S <LOAD <CR>
```

The -S option means that both addresses and scalars are loaded. If you omit the -S, only addresses are loaded. (A scalar is a number that is not an address — for example, TSIZE, the length of the table.)

Later in this demonstration, whenever you use a symbol in a command line, DOS/50 refers to the symbol table to find the value that the symbol represents.

You've assembled and linked the demonstration program and loaded it into memory. Now skip ahead to the heading Run the Demonstration Program.

## CASE 2: ASSEMBLE ON THE 8560; DOWNLOAD TO THE 8540

This discussion shows you how to create the demonstration program source code and assemble it on the 8560, then download it to 8540 (or 8550) program memory. If your 8560 does not have a Z80 assembler, you cannot complete this part of the demonstration, so skip ahead to the heading CASE 4: Patch the Program into Memory.

### Start Up and Log In

Start up your 8540, make sure it's in TERM mode, and log in to the 8560 operating system, TNIX. See your 8560 System Users Manual for details.

Since you're logged in to TNIX, your system prompt is \$. (Later in the demonstration, we'll show the system prompt as >, in deference to people using 8540s and 8550s in LOCAL mode.) Every command you enter is processed by TNIX. If you enter an OS/40 command, TNIX passes it to the 8540.

Enter the following commands to select the Z80 assembler on the 8560 and the Z80 emulator on the 8540:

```
$ uP =z80; export UP <CR >

$ sel Z80 <CR >
```

The *sel* command automatically sets the emulation mode to 0.



## Create the Demonstration Program

Enter the following TNIX command lines to create an empty directory called *demo* and make *demo* the working directory. You'll create your source file and related files in this *demo* directory.

```
$ mkdir demo <CR>
$ cd demo <CR>
```

Now use the TNIX editor *ed*, to create the demonstration program source file. The following command line invokes the editor and specifies that you want to create a file called *asm*:

```
$ ed asm <CR>
?asm
```

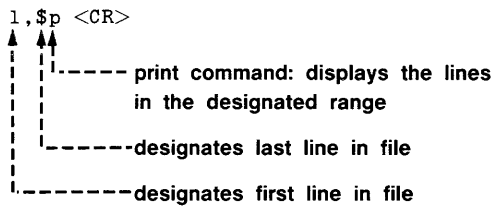
The editor responds ?asm to remind you that *asm* does not already exist. Notice that the editor does not give a prompt to let you know that it's ready for input.

**Enter the Text.** Now enter the editor command *a* (append text) and type in the program. Use the BACK-SPACE key to erase any typing mistakes.

```
a <CR>
      column  column  column
      8       6       24
      ↓       ↓       ↓
;Z80 DEMONSTRATION RUN PROGRAM <CR>
SECTION DEMO <CR>
ORG 100H ;START PROGRAM CODE AT ADDRESS 100 <CR>
START LD HL, TABLE ;SET TABLE POINTER <CR>
LD B, TSIZE ;SET PASS COUNTER <CR>
XOR A ;CLEAR ACCUMULATOR <CR>
LOOP ADD A, (HL) ;ADD BYTE FROM TABLE <CR>
INC HL ;POINT TO NEXT BYTE <CR>
DEC B ;DECREMENT PASS COUNTER <CR>
JP NZ, LOOP ;LOOP IF NOT FIVE PASSES YET <CR>
OUT (OF7H), A ;OTHERWISE CALL EXIT SVC <CR>
NOP ; TO END PROGRAM EXECUTION <CR>
;SRB POINTER <CR>
ORG 40H ;STORE SRB POINTER AT ADDRESS 40 <CR>
BYTE 00, 42H ;POINT TO SRB FOR EXIT SVC <CR>
;SRB FOR EXIT SVC <CR>
BYTE 1AH ;1AH = FUNCTION CODE FOR EXIT SVC <CR>
;TABLE OF NUMBERS TO BE ADDED <CR>
TSIZE EQU 5 ;TABLE SIZE = 5 <CR>
ORG 500H ;SET UP TABLE AT ADDRESS 500 <CR>
TABLE BLOCK TSIZE <CR>
LIST DBG <CR>
END START <CR>
.<CR>
```

At the end of your text, enter a period on a line by itself. The editor will now accept new commands.

**Check for Errors.** Type the following editor command to display the text you have entered. Check for typing mistakes.



If you made any mistakes, fix them now. In case you're not familiar with the editor, Table 7C-6 lists the commands you need in order to add, delete, or replace a line. For more information on the TNIX editor, refer to your 8560 System Users Manual.

**Table 7C-6  
Basic 8560 Editing Commands**

Command	Function
mm,nnp <CR>	Displays lines mm through nn
nn <CR>	Makes line nn the current line
d <CR>	Deletes the current line
a <CR> <line(s) of text> . <CR>	Adds text below the current line
c <CR> <line(s) of text> . <CR>	Replaces the current line with the text you type in

Once your text is correct, enter the *w* command to write the text to the source file, *asm*:

```

w <CR>
902
  
```

The editor responds with the number of characters written to the file.

Finally, enter the *q* command to quit the editor and return to TNIX:

```

q <CR>
$ ←----TNIX prompt
  
```

### Assemble the Source Code.

The TNIX *asm* (*assemble*) command translates assembly language (source code) into binary machine language (object code). The *asm* command also creates an assembler listing that you can use to correlate the object code with the source code. Enter the following command line to assemble the source code in the file *asm* and create the listing and object files *asm1* and *obj*:

```
$ asm obj asm1 asm <CR>
```

↑           ↑           ↑  
 --- source file  
 --- assembler listing file  
 --- object file

```
Tektronix ASM Z80
Vxx.xx-xx (8560)
*****Pass 2
```

```
23 Lines Read
23 Lines Processed
0 Errors
```

Enter the following command to print the assembler listing on the 8560's line printer:

```
$ lplr asm1 <CR>
```

Examine page 1 of your listing. Did the assembler issue any error messages? There should be none. If your source code contains errors, take the following steps.

1. Refer to your Assembler Users Manual to find out what the error messages mean.
2. Enter the command *ed asm* to get back into the editor and fix the mistakes in your source code. Exit the editor with the *w* and *q* commands, as before.
3. Enter the command *asm obj asm1 asm* to re-assemble your source code.

### Link the Object Code

The linker creates an executable load file from one or more object files. Enter the following command to create a load file called *load* from your object file, *obj*. Be sure to enter all parameters exactly as shown.

```
$ link -d -o obj -o load <CR>
```

The *-d* option causes the linker to pass the program symbols from the object file to the load file, for use in program debugging.

The files generated by the *asm* and *link* commands should now be in your working directory, *demo*. Enter the following command to list the files in your working directory:

```
$ ls <CR>
asm
asm1
load
obj
```

Notice that there are now four files listed in your directory: *obj* and *asm1* were created by the assembler, and *load* was created by the linker.

### Download the Program to the 8540

Now it's time to download the object code produced by the 8560's linker into 8540 program memory.

**Zero Out Memory.** Before you download any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no effect on how the program is loaded.) Enter the following command line to fill memory addresses 40 to 11F with zeros:

```
$ f 40 11f 00 <CR>
```

**Check That Memory Was Filled with Zeros.** Check the contents of memory with the OS/40 D (dump) command. The D command's display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40 to 11F with the following command line:

```
$ d 40 11f <CR>
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

**Download the Object Code.** Enter the following command line to download the object code from the 8560 file *load* to 8540 program memory:

```
$ l o <load> <LCR>
      ↑
    load file
```

**Download the Program Symbols.** The source code for the demonstration program contains the directive LIST DBG. Because of this directive, the object file contains a list of the symbols that appear in the source code and the values associated with those symbols. Because you included the -d option in the link command line, those symbols were passed to the load file. Use the OS/40 SYMLO command to download those symbols into the symbol table in 8540 system memory.

```
$ symlo -s <load> <CR>
```

The -s option means that both addresses and scalars are downloaded. If you omit the -S, only addresses are downloaded. (A scalar is a number that is not an address — for example, TSIZE, the length of the table.)

Later in this demonstration, whenever you use a symbol in an OS/40 command line, OS/40 refers to the symbol table to find the value that the symbol represents.

You've assembled and linked the demonstration program and downloaded it into memory. Now skip ahead to the heading Run the Demonstraion Program.

## CASE 3: DOWNLOAD FROM YOUR HOST TO THE 8540

This discussion gives some general instructions for downloading the demonstration program from a host computer other than the 8550 or 8560 to 8540 (or 8550) program memory. If your 8540 is not equipped with the optional COM Interface Package, you cannot complete this part of the demonstration, so skip ahead to the heading Case 4: Patch the Program into Memory for instructions. COM Interface software is standard on the 8550.

Since we don't know what host computer you're using, we can only provide a general outline for creating the demonstration program and downloading it to the 8540. Once you have determined the command sequence that is appropriate for your host, record this information in the space provided in Fig. 7C-10.

### Create the Extended Tekhex Load Module

In order for object code to be downloaded to the 8540, it must be in Extended Tekhex format, as shown in Fig. 7C-9. You can create the load module in one of two ways:

1. using your host computer's text editor, key the load module in by hand; or
2. using your host computer's Z80 assembler to
  - a. translate the demonstration program into the language of your host's Z80 assembler;
  - b. create and assemble the source file;
  - c. link the object code, if necessary; and
  - d. translate the object code produced by the assembler or linker into Extended Tekhex format. The Intersystem Communication section of your System Users Manual provides a general algorithm for conversion to Extended Tekhex format.

### Prepare the 8540

Start up your 8540 and enter the following command to select the Z80 emulator:

```
> SEL Z80 <CR>
```

The SEL command automatically sets the emulation mode to 0.

Create the Extended Tekhex Load Module

Prepare the 8540

(Start up the 8540.)

```
> SEL Z80 <CR>  
> F 40 11F 00 <CR>  
> D 40 11F <CR>
```

Establish Communication

Download the Load Module

Terminate Communication

Fig. 7C-10. Host computer commands for preparing demonstration program.

**Zero Out Memory.** Before you download any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no affect on how the program is loaded.) Enter the following command line to fill memory addresses 40-11F with zeros:

```
> F 40 11F 00 <CR>
```

**Check That Memory Was Filled with Zeros.** Check the contents of memory with the OS/40 D (Dump) command. The D command's display shows the data (in hexadecimal format) and the corresponding ASCII characters. Display the contents of memory addresses 40 to 11F with the following command line:

```
> D 40 11F <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

### Download the Load Module to the 8540

Be sure that your 8540 and your host computer are connected via an RS-232-C compatible communication link. Refer to the Intersystem Communication section of your System Users Manual to determine the commands and parameters that are appropriate for your host computer. Then perform the following steps to download the Tekhex load module to 8540 program memory.

1. Enter the 8540 COM command to establish communication. (The parameters of the COM command are host-specific.) Log on to your host and execute any necessary host initialization commands.
2. Enter the command line that downloads the Tekhex load module to the 8540. This command line consists of the host computer command that performs the download, followed by a null character (CTRL-@ on most terminals) and a carriage return. COM places the object code in 8540 program memory, and puts the program symbols into the symbol table in 8540 system memory.
3. Log off from your host, and then terminate COM command execution by entering the null character, then pressing the ESC key.

Once you've downloaded the program to the 8540, skip ahead to the heading Run the Demonstration Program.

## CASE 4: PATCH THE PROGRAM INTO MEMORY

This discussion shows you how to patch the demonstration program into 8540 (or 8550) program memory using the P command, and then add the program symbols into the symbol table using the ADDS command.

Ordinarily, you would load the object code and symbols from a binary or hexadecimal load file, as illustrated for Cases 1, 2, and 3. The procedure presented here is **not** normally used for preparing a program for execution. Use this procedure only if you have no standard means for preparing the program, but would still like to try out your emulator.

**Start Up the 8540**

Start up your 8540 and enter the following command to select the Z80 emulator:

```
> SEL Z80 <CR>
```

The SEL command automatically sets the emulation mode to 0.

**Zero Out Memory**

Before you patch in any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. Enter the following command line to fill memory from addresses 40 to 11F with zeros:

```
> F 40 11F 00 <CR>
```

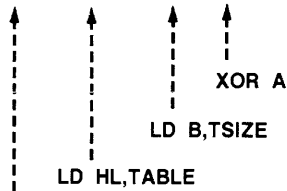
**Check That Memory Was Filled with Zeros.** Check the contents of memory with the OS/40 D (Dump) command. The D command's display shows the data (in hexadecimal format) and the corresponding ASCII characters. Display the contents of memory addresses 40 to 11F with the following command line:

```
> D 40 11F <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

**Patch the Object Code into Memory**

The OS/40 P (Patch) command stores a sequence of bytes into memory, replacing the previous memory contents. Enter the following command to store the object code for the first three instructions in the program (LD HL, LD B, and XOR A) starting at location 100:

```
> P 100 210005 0605 AF <CR>
```



patch address

Now patch in the next four instructions (ADD A, INC HL, DEC B, and JP NZ, LOOP)...

```
> P 106 86 23 05 C20601 <CR>
```



... and now the last two instructions (OUT and NOP):

```
> P 10C D3F7 00 <CR>
```

Finally, patch in the Exit SVC information at address 40:

```
> P 40 00421A <CR>
```

You'll examine the contents of memory later in this demonstration.

### Put Symbols into the Symbol Table

Later in this demonstration, you will use symbols from the demonstration program (START, LOOP, TSIZE, AND TABLE) when communicating with OS/40. Whenever you use a symbol in a command, OS/40 consults a symbol table in 8540 system memory to find the value the symbol represents. Enter the following command line to add the program symbols to the symbol table, along with their values:

```
> ADDS START=100 LOOP=106 -S TSIZE=5 TABLE=500 <CR>
```

The ADDS command cannot provide all the symbol-related information that is provided by the SYMLO command (as in Cases 1 and 2) or the COM command (as in Case 3). Because this information is missing, some of the displays you produce later in this demonstration will not match the symbolic displays shown in this manual. For more information on the ADDS command, refer to the Command Dictionary of your System Users Manual.

You've patched the demonstration program into program memory and placed the program symbols in the symbol table. Now it's time to run the program.

## RUN THE DEMONSTRATION PROGRAM

From now until the end of the demonstration, the commands you are to enter are shown in lowercase. If you are not logged into an 8560, you may enter commands in either lowercase or uppercase. If you **are** using an 8560, you must enter the name of every command in lowercase (and your system prompt is \$, not > ).

Now that you've loaded the program into memory, you need to:

- verify that the program was loaded correctly; and
- put values into the table in memory, for the program to add.

**Check Memory Contents Again.** Before you loaded the program, you filled memory locations 40 to 11F with zeros. Look at the same memory area again with the following command line:

```
> d 40 11f <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

The object code is loaded in two different blocks:

- The Z80 machine instructions are loaded at address 100 (specified by the first ORG directive in the source code).
- The information for the Exit SVC is loaded at address 40 (specified by the second ORG directive).

The contents of the table at address 500 are still undefined, but you'll put some values into the table in just a few minutes.

**Turn On Symbolic Debug.** Enter the following command to turn on symbolic debug:

```
> symd on <CR>
```

**Disassemble the Object Code.** The DI (Disassemble) command displays memory contents both in hexadecimal notation and in assembly language mnemonics. You can use the DI command to verify that the object code in memory corresponds to your source code. Enter the following command to disassemble the area of memory occupied by the executable part of your program:

```
> di 100 10e <CR>
LOC      INST      MNEM OPER
SECTION (DEMO)
START    210005     LD   HL,0500
+000103  0605     LD   B,05
+000105  AF        XOR  A
LOOP     86        ADD  A,(HL)
+000107  23        INC  HL
+000108  05        DEC  B
+000109  C20601     JP   NZ,0106
+00010C  D3F7     OUT  (F7),A
+00010E  00        NOP
```

Compare the DI display with the assembler listing you generated earlier, or refer back to Fig. 7C-8.

The line SECTION (DEMO) in the DI display indicates that the object code being disassembled comes from the program section called DEMO. In fact, the entire memory area used by your program (location 0 through the end of the table — location 504) belongs to section DEMO. This section was declared by the SECTION directive in the source code.

The LOC (location) column of the DI display contains information that enables you to correlate the display with your assembler listing. The symbols START and LOOP in the DI display correspond to the labels START and LOOP in the source code. In the display, when a location does not correspond to a label in the symbol table, DI substitutes the address of the instruction **relative to the beginning of the section**, as shown in the address field of your assembler listing. If you haven't loaded the pertinent symbols and related information into the symbol table (using a command such as SYMLO), the DI command supplies absolute (actual) addresses in the LOC column. (Since section DEMO begins at address 0, the relative address, or **offset**, is the same as the absolute address in this display. This offset feature is much more useful for sections that **don't** start at address 0.)

Now you've seen that your system can use the symbol table to translate numbers into symbols to make a display easier to read. Your system can also translate a symbol in a command line into an address. For example, since your system knows that the symbol START is equivalent to the address 100, you could have entered the DI command in any of the following ways:

```
di 100 10E
di START 10E
di start start+0e
di 100 START+0E
```

Notice that a symbol can be entered in either lowercase or uppercase.

The feature that enables DOS/50 and OS/40 to correlate symbols from your program with the numbers they represent is termed *symbolic debug*.

**Put Values into the Table in Memory.** The demonstration program sums five numbers from a table in memory. Use the P (Patch) command to place the numbers 1, 2, 3, 4, and 5 in the table. Do you remember what the address of the table is? It doesn't matter, as long as you remember that the symbol TABLE represents that address.

```
> p table 0102030405 <CR>
=====
  ↑           ↑
address of   string of bytes to be stored
table: 500   at addresses 500 to 504
```

**Check the Contents of the Table.** Use the D command to display the contents of the table. (When you don't specify an upper boundary for the area to be dumped, the D command dumps 16 bytes.)

```
----- lower address: 500
  ↓
----- upper address: omitted
      (defaults to lower address + 0F)
  ↓
> d table == <CR>
  0 1 2 3 4 5 6 7 8 9 A B C D E F
000500 01 02 03 04 05 27 EB CF C3 BC EB B6 9D AB AF DB .....
```

Notice that bytes 500 to 504 (the table) contain the values you patched in. Bytes 505 to 50F contain random data left over from previous system operations.

The following command dumps only the contents of the table:

```
> d table table+tsize-1 <CR>
      0 1 2 3 4 5 6 7 8 9 A B C D E F
000500 01 02 03 04 05                .....
```

### Start Program Execution

Now enter the G (Go) command to start program execution at location 100, the transfer address specified by the END directive in the source code. (If you followed Case 4: Patch the Program into Memory, you must enter G START instead.)

```
> g <CR>
LOC   INST      MNEM OPER          SP   F  A  B  C  D  E  H  L  IX  IY
00010F 00      NOP          0000 42 0F 00 00 00 00 05 05 0000 0000
00010F 00 <BREAK  >
```

The program executes, and when the Exit SVC occurs, the program breaks (stops), and the contents of the emulator registers are displayed. The accumulator contains the sum of the numbers in the memory table:  $1+2+3+4+5=0F$ .

## MONITOR PROGRAM EXECUTION

You have assembled, loaded, and executed the demonstration program. The rest of this demonstration shows you some commands for monitoring program execution. You can watch the changes in the emulator's registers and observe the effect of each instruction as the program proceeds.

**Trace All Instructions.** The TRA (TRAcE) command lets you observe the changes in the Z80 registers as the program proceeds. When you enter a TRA command and then start execution with the G command, display lines are sent to the system terminal. As each instruction executes, the display line shows the instruction (as in the Disassemble display) and the contents of the registers after that instruction has executed. Enter the following command to trace all of the program's instructions:

```
> tra all <CR>
```

Enter the command G START (or G 100) to resume program execution at the beginning of the program:

```
> g start <CR>
```

As the program executes, the following trace is displayed. Remember that you can type CTRL-S to suspend the display and CTRL-Q to resume the display.

LOC	INST	MNEM	OPER	SP	F	A	B	C	D	E	H	L	IX	IY
SECTION (DEMO)														
START	210005	LD	HL,0500	FFFF	42	0F	00	00	00	00	05	00	0000	0000
+000103	0605	LD	B,05	FFFF	42	0F	05	00	00	00	05	00	0000	0000
+000105	AF	XOR	A	FFFF	44	00	05	00	00	00	05	00	0000	0000
LOOP	86	ADD	A,(HL)	FFFF	00	01	05	00	00	00	05	00	0000	0000
+000107	23	INC	HL	FFFF	00	01	05	00	00	00	05	01	0000	0000
+000108	05	DEC	B	FFFF	02	01	04	00	00	00	05	01	0000	0000
+000109	C20601	JP	NZ,0106	FFFF	02	01	04	00	00	00	05	01	0000	0000
LOOP	86	ADD	A,(HL)	FFFF	00	03	04	00	00	00	05	01	0000	0000
+000107	23	INC	HL	FFFF	00	03	04	00	00	00	05	02	0000	0000
+000108	05	DEC	B	FFFF	02	03	03	00	00	00	05	02	0000	0000
+000109	C20601	JP	NZ,0106	FFFF	02	03	03	00	00	00	05	02	0000	0000
LOOP	86	ADD	A,(HL)	FFFF	00	06	03	00	00	00	05	02	0000	0000
+000107	23	INC	HL	FFFF	00	06	03	00	00	00	05	03	0000	0000
+000108	05	DEC	B	FFFF	02	06	02	00	00	00	05	03	0000	0000
+000109	C20601	JP	NZ,0106	FFFF	02	06	02	00	00	00	05	03	0000	0000
LOOP	86	ADD	A,(HL)	FFFF	08	0A	02	00	00	00	05	03	0000	0000
+000107	23	INC	HL	FFFF	08	0A	02	00	00	00	05	04	0000	0000
+000108	05	DEC	B	FFFF	02	0A	01	00	00	00	05	04	0000	0000
+000109	C20601	JP	NZ,0106	FFFF	02	0A	01	00	00	00	05	04	0000	0000
LOOP	86	ADD	A,(HL)	FFFF	08	0F	01	00	00	00	05	04	0000	0000
+000107	23	INC	HL	FFFF	08	0F	01	00	00	00	05	05	0000	0000
+000108	05	DEC	B	FFFF	42	0F	00	00	00	00	05	05	0000	0000
+000109	C20601	JP	NZ,0106	FFFF	42	0F	00	00	00	00	05	05	0000	0000
LOC	INST	MNEM	OPER	SP	F	A	B	C	D	E	H	L	IX	IY
+00010C	D3F7	OUT	(F7),A	FFFF	42	0F	00	00	00	00	05	05	0000	0000
+00010C	D3F	<BREAK	TRACE>											

After the accumulator is cleared, it begins to store the sum of the numbers being added. The ADD A instruction adds a number from the table into the accumulator. At the end of the program, the accumulator contains the sum of the numbers you put into the table.

Register B, the pass counter, is set to contain 5 (TSIZE) at the beginning of the program. It decreases by one (because of the DEC B instruction) each time a number is added into the accumulator. The program ends after register B reaches zero.

The H-L register pair, set to contain 500 (TABLE) at the start of the program, increases by one each time a number is added to the accumulator. At the end of the program, the H-L register pair has been incremented five times and contains 505.

**Trace to the Line Printer.** By adding the parameter >LPT to a command, you can direct that command's output to the line printer instead of to the system terminal. First, verify that your line printer is properly connected and powered up. Then enter the following command to execute the program with trace output directed to the line printer:

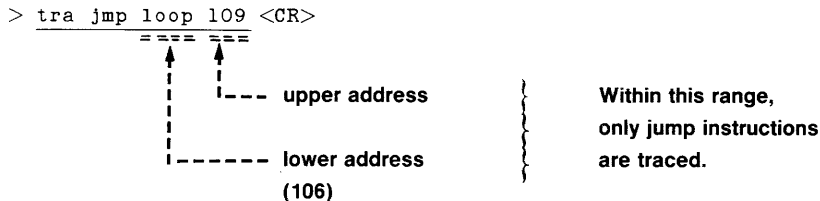
```
> g start >LPT <CR>
```

#### NOTE

If you're operating in TERM mode with an 8560, use one of the following commands in place of the command shown:

- `g start |lplr` sends the display to the 8560 line printer.
- `g start \>LPT` sends the display to the line printer on the 8540 or 8550.

**Trace Jump Instructions Only.** Another way to monitor the program's execution is to look only at the jump instructions. By tracing the jump instructions, you can still observe the changes in the registers, but you save time and space by not tracing the instructions within the loop. Enter the following command to trace only the jump instructions when the loop is being executed:



Again, start your program with the G command. The following trace is displayed:

```
> g start <CR>
```

LOC	INST	MNEM	OPER	SP	F	A	B	C	D	E	H	L	IX	IY
SECTION (DEM0)														
START	210005	LD	HL,0500	FFFF	42	0F	00	00	00	00	05	00	0000	0000
+000103	0605	LD	B,05	FFFF	42	0F	05	00	00	00	05	00	0000	0000
+000105	AF	XOR	A	FFFF	44	00	05	00	00	00	05	00	0000	0000
+000109	C20601	JP	NZ,0106	FFFF	02	01	04	00	00	00	05	01	0000	0000
+000109	C20601	JP	NZ,0106	FFFF	02	03	03	00	00	00	05	02	0000	0000
+000109	C20601	JP	NZ,0106	FFFF	02	06	02	00	00	00	05	03	0000	0000
+000109	C20601	JP	NZ,0106	FFFF	02	0A	01	00	00	00	05	04	0000	0000
+00010C	D3F7	OUT	(F7),A	FFFF	42	0F	00	00	00	00	05	05	0000	0000
+00010C	D3F	<BREAK	TRACE >											

As with the TRA ALL display, observe that register B (the pass counter) is decremented, the H-L register pair (the table pointer) is incremented, and the accumulator stores the sum of the numbers from the table. With the TRA JMP selection in effect, the instructions within the loop are not displayed.

**Check the Status of the Trace.** The TRA command without any parameters displays the trace conditions that are currently set. Because you can have up to three trace selections in effect at the same time, it is useful to be able to see which selections are active. Check your trace status with the following command line:

```
> tra <CR>
```

```
TRACE ALL,000000,00FFFF
```

```
TRACE JMP,LOOP,000109
```

As you've specified, TRA ALL is in effect for addresses 0 to 105, TRA JMP is in effect for addresses 106 to 109, and TRA ALL is again in effect for addresses 10A to FFFF.

**Set a Breakpoint after a Specific Instruction.** Now that you've seen how the program adds the numbers together, here's a new task: to add only the third and fourth numbers from the table. To perform this task, you want the pass counter to contain 2, and the table pointer to contain 502 (the address of the third number in the table). You can accomplish these changes without altering the object code in memory. First, stop program execution after the pass counter and the table pointer have been set. Next, while the program is stopped, enter new values for the pass counter and table pointer. When execution resumes, the program treats the new values as if they were the original programmed value.

Enter the following command line to trace all of the instructions as the program executes:

```
> tra all <CR>
```

Check the status of the trace with the following command line:

```
> tra <CR>
TRACE    ALL,000000,00FFFF
```

The TRA ALL command just entered makes the earlier TRA selections obsolete.

Now you set a breakpoint so that the program stops after the table pointer and pass counter have been set. The following command causes the program to stop after it executes the LD B instruction at address 103:

```
> bk 1 103 <CR>
  =  =  =  =
  ↑   ↑
  |   |
  |   |----- breakpoint address
  |   |
  |   |----- breakpoint number
  |   |
  |   |----- (can be 1 or 2)
```

Use the G command to start program execution:

```
> g start <CR>
LOC   INST      MNEM OPER          SP   F   A   B   C   D   E   H   L   IX   IY
SECTION (DEMO)
START 210005    LD   HL,0500    FFFF 42 0F 00 00 00 00 05 00 0000 0000
+000103 0605    LD   B,05      FFFF 42 0F 05 00 00 00 05 00 0000 0000
+000103 060 <BREAK  TRACE, BKPT1 >
```

The TRA ALL command enabled display of all instructions up to and including the instruction at the breakpoint.

**Set New Values in Pass Counter and Table Pointer; Check Results.** Now that you've reached the breakpoint, you can change the contents of the registers while execution is stopped. The break display shows that register B (the pass counter) contains 5, and the H-L register pair (the table pointer) contains the address 500. Use the S (Set) command to set the number of passes to two and set the table pointer to 502:

```
> s B=2 L=2 <CR>
```

The S command does not produce a display, but you can use the DS (Display Status) command to check the values in the registers you changed. DS displays the contents of each emulator register and status flag. Check the result of the previous S command with the following command line:

```
> ds <CR>
PC=0105 SP=FFFF      F=42  A=0F  B=02  C=00  D=00  E=00  H=05  L=02
IX=0000 IY=0000     AF=00 AA=00 AB=00 AC=00 AD=00 AE=00 AH=00 AL=00
IFF1=0  IFF2=0  IM=0  I=00  R=37
```

The DS display shows that the pass counter and table pointer now contain the new values.

**Resume Program Execution.** If you enter the G command with no parameters, program execution starts where it left off. Resume program execution after the breakpoint with the following command:

```
> g <CR>
LOC      INST      MNEM OPER          SP   F   A   B   C   D   E   H   L   IX   IY
SECTION (DEMO)
+000105 AF        XOR  A           FFFF 44 00 02 00 00 00 05 02 0000 0000
LOOP    86        ADD  A,(HL)      FFFF 00 03 02 00 00 00 05 02 0000 0000
+000107 23        INC  HL          FFFF 00 03 02 00 00 00 05 03 0000 0000
+000108 05        DEC  B           FFFF 02 03 01 00 00 00 05 03 0000 0000
+000109 C20601   JP   NZ,0106     FFFF 02 03 01 00 00 00 05 03 0000 0000
LOOP    86        ADD  A,(HL)      FFFF 00 07 01 00 00 00 05 03 0000 0000
+000107 23        INC  HL          FFFF 00 07 01 00 00 00 05 04 0000 0000
+000108 05        DEC  B           FFFF 42 07 00 00 00 00 05 04 0000 0000
+000109 C20601   JP   NZ,0106     FFFF 42 07 00 00 00 00 05 04 0000 0000
+00010C D3F7      OUT  (F7),A      FFFF 42 07 00 00 00 00 05 04 0000 0000
+00010C D3F <BREAK  TRACE>
```

Notice that the program performed two passes through the loop, and that the program added the third and fourth numbers in the table:  $3+4=7$ .

### Delete the Demonstration Run Files

Now that you've finished the demonstration run, you can delete the source file, object file, listing file, and load file. If you're using an 8550, the source and load files are still available to you on the Z80 emulator installation disk. If you're using an 8560, remember that once you delete the source file (*asm*), there is no way of recovering it.

**Delete 8550 Files.** If your files are on the 8550, use the following procedure to delete them. First use the USER command to move from the DEMO directory back in to the directory you were in at the start of the demonstration. Recall that you marked that directory with the brief name /ROOT.

```
> USER /ROOT <CR>
```

Now enter the following command to delete the DEMO directory and the files it contains:

```
> DEL DEMO / * DEMO <CR>
Delete ASM   ?  Y <CR>
Delete LOAD  ?  Y <CR>
Delete OBJ   ?  Y <CR>
Delete ASML  ?  Y <CR>
Delete DEMO  ?  Y <CR>
```



Before deleting each file, DOS/50 asks you whether you really want to delete it. You type Y for yes.

**Delete 8560 Files.** If your files are on the 8560, use the following procedure to delete them. Enter the following command to remove all files in the working directory, including the source file:

```
$ rm * <CR>
```

Now move from the *demo* directory back into the parent directory and remove the *demo* directory itself:

```
$ cd .. <CR>  
$ rmdir demo <CR>
```

### Turn Off Your System

For instructions on turning off your 8540 or 8550, refer to the Learning Guide of your System Users Manual.

## SUMMARY OF Z80 EMULATOR DEMONSTRATION RUN

You have assembled, loaded, executed, and monitored the demonstration run program. You have used the following commands:

- SEL — selects the Z80 assembler and emulator
- ASM — creates object code from an assembly language program
- LINK — links object code into a load module
- F — fills an area of memory with a specified value
- D — displays memory contents in ASCII and hexadecimal format
- LO — loads object code into memory
- SYMLO — loads program symbols for use in symbolic debug
- DI — translates memory contents into assembly language mnemonics
- P — patches a string of bytes into memory
- SYMD — turns on symbolic debug displays
- G — begins or resumes program execution
- TRA — selects instructions to be traced during program execution
- BK — sets a breakpoint
- S — modifies emulator registers
- DS — displays emulator registers



This manual supports the following TEKTRONIX products:

8550 Option	Products	8540 Option	Products
2H	8300E10	2H	8300E10 Option 01
3K	8300P10	3K	8300P10
3L	8300P11	3L	8300P11
3M	8300P12	3M	8300P12
3N	8300P13	3N	8300P13

This manual supports a software/firmware module that is compatible with:

DOS/50 Version 2 (8550)  
OS/40 Version 1 (8540)

**PLEASE CHECK FOR CHANGE INFORMATION  
AT THE REAR OF THIS MANUAL.**

**8500**  
**MODULAR MDL SERIES**  
**8048/8021/8041A/8022**  
**EMULATOR SPECIFICS**  
**USERS MANUAL**

**Tektronix, Inc.**  
**P.O. Box 500**  
**Beaverton, Oregon 97077**

070-3967-01  
Product Group 61

Serial Number \_\_\_\_\_

First Printing APR 1982  
Revised SEP 1983

## LIMITED RIGHTS LEGEND

Software License No. \_\_\_\_\_

Contractor: Tektronix, Inc.

Explanation of Limited Rights Data Identification Method

Used: Entire document subject to limited rights.


Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

## RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Copyright © 1982 Tektronix, Inc. All rights reserved. Contents of this publication may not be reproduced in any form without the written permission of Tektronix, Inc.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and  are registered trademarks of Tektronix, Inc. TELEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Printed in U.S.A. Specification and price change privileges are reserved.

-----  
Section 7H

8048/8021/8041A/8022 EMULATOR SPECIFICS

	Page
<u>Introduction</u> .....	7H-1
<u>General Information</u> .....	7H-1
8048/8021/8041A/8022 Emulator Hardware Configurations .....	7H-1
Terminology .....	7H-2
<u>Emulator-Specific Commands, Parameters, and Displays</u> .....	7H-2
SEL --- Selecting an Emulator .....	7H-2
Byte/Word Parameter .....	7H-2
Symbolic Debug .....	7H-2
MAP --- Mapping Memory .....	7H-3
EVT Address Parameter .....	7H-3
TTA Bus Operation Designators .....	7H-3
Setting Breakpoints .....	7H-4
Memory Allocation Commands .....	7H-4
Register Designators .....	7H-4
DS --- Sample Status Display .....	7H-6
RESET --- Resetting Emulator Status .....	7H-7
DI --- Sample Disassembled Code .....	7H-8
TRA --- Sample TRAcE Display .....	7H-8
<u>Service Calls</u> .....	7H-10
SVCs in Modes 1 and 2 .....	7H-10
SVC Demonstration .....	7H-11
SVC Design Features .....	7H-14
<u>Special Considerations</u> .....	7H-16
Clock Rate .....	7H-16
Memory Mapping .....	7H-17
Emulation Modes .....	7H-21
NOP--MOVX Instruction Sequence .....	7H-21
TRA and the Timer/Counter .....	7H-21
8048/8049 Special Considerations .....	7H-22
8021 Special Considerations .....	7H-23
8041A Special Considerations .....	7H-23
8022 Special Considerations .....	7H-24
<u>Emulator Timing</u> .....	7H-25
Probe/Prototype Interface Diagrams .....	7H-31
<u>Installing Your 8048/8021/8041A/8022 Emulator Software</u> .....	7H-34
8540 Firmware Installation Procedure .....	7H-34
8550 Installation Procedure .....	7H-34
<u>8048/8021/8041A/8022 Demonstration Run</u> .....	7H-36
Introduction .....	7H-36
Examine the Demonstration Run Program .....	7H-39
Assemble and Load the Demonstration Program .....	7H-40
Case 1: Assemble and Load on the 8550 .....	7H-41
Case 2: Assemble on the 8560; Download to the 8540 .....	7H-47

Case 3: Download from Your Host to the 8540 .....	7H-53
Case 4: Patch the Program into Memory .....	7H-56
Run the Demonstration Program .....	7H-58
Monitor Program Execution .....	7H-61
Summary of 8048/8021/8041A/8022 Emulator Demonstration Run .....	7H-66
Delete the Demonstration Run Files .....	7H-67

### TABLES

#### Table

##### No.

7H-1 Microcomputers Supported by the 8048/8021/8041A/8022 Emulator	7H-1
7H-2 8048/8021/8041A/8022 Bus Operation Designators .....	7H-3
7H-3 8048/8021/8041A/8022 Registers and Flags .....	7H-5
7H-4 8048/8021/8041A/8022 Service Calls .....	7H-10
7H-5 Maximum Allowable Clock Frequencies .....	7H-16
7H-6 Probe/Prototype Interface Delays for the 8048/8021 .....	7H-25
7H-7 Probe/Prototype Interface Delays for the 8041A .....	7H-27
7H-8 Representative 8041A Probe/Microcomputer Timing Differences .	7H-27
7H-9 Probe/Prototype Interface Delays for the 8022 .....	7H-29
7H-10 Basic 8560 Editing Commands .....	7H-49

### ILLUSTRATIONS

#### Fig.

##### No.

7H-1 8048/8021/8041A/8022 program status word .....	7H-6
7H-2 8048 SVC demonstration program listing .....	7H-12
7H-3 8048/8021/8041A/8022 memory layout .....	7H-18
7H-4 8048/8021 timing diagrams .....	7H-26
7H-5 8041A timing diagrams .....	7H-28
7H-6 8022 timing diagrams .....	7H-30
7H-7 Block diagram of 8048/8021 probe/prototype interface .....	7H-31
7H-8 Block diagram of 8041A probe/prototype interface .....	7H-32
7H-9 Block diagram of 8022 probe/prototype interface .....	7H-33
7H-10 Demonstration program .....	7H-37
7H-11 Demonstration program: Extended Tekhex format .....	7H-38
7H-12 Host computer commands for preparing demonstration program .	7H-54

-----  
Section 7H

8048/8021/8041A/8022 EMULATOR SPECIFICS

INTRODUCTION

This section is designed to be inserted into Section 7 of the 8550 System Users Manual (DOS/50 Version 2) or the 8540 System Users Manual. This Emulator Specifics section explains the features of the 8550 and 8540 systems that are unique to the 8048/8021/8041A/8022 emulator. Throughout the section, "your System Users Manual" refers to the 8550 System Users Manual or the 8540 System Users Manual, and "the operating system" refers to DOS/50 Version 2 or OS/40. The 8048/8021/8041A/8022 Demonstration Run is designed to be used with Section 1, the Learning Guide of your System Users Manual; the rest of this section contains reference material.

GENERAL INFORMATION

8048/8021/8041A/8022 EMULATOR HARDWARE CONFIGURATIONS

In order for the 8048/8021/8041A/8022 emulator to function correctly, it must be connected to the prototype control probe that is appropriate for the microcomputer being emulated. Table 7H-1 lists the microcomputers supported by the 8048/8021/8041A/8022 emulator and gives the corresponding hardware configurations.

Table 7H-1  
Microcomputers Supported by the 8048/8021/8041A/8022 Emulator

Microcomputer	Hardware Configuration
8041A/8741A	8041A Prototype Control Probe
8022	8022 Prototype Control Probe
8021	8048/8021 Prototype Control Probe with 8021 Prototype Control Probe Adapter (*a)
8048/8648/8748/8035	8048/8021 Prototype Control Probe (*a)
8049/8039/8039-6	8048/8021 Prototype Control Probe (*a)

(\*a) DIP switches on the 8048/8021 Prototype Control Probe select the microcomputer to be emulated, the prototype clock rate, and the presence or absence of external program memory. These switches are described in the 8048/8021/8041A/8022 Emulator Processor Installation Manual.

---

TERMINOLOGY

The term 8048 refers collectively to the 8048 and to those microcomputers that differ from the 8048 only in the form of on-board ROM (the 8648, 8748, and 8035). The term 8048/8049 refers collectively to those microcomputers that support the 8048 instruction set (the 8048, 8648, 8748, 8035, 8049, 8039, and 8039-6). The terms 8048 family and 8048/8021/8041A/8022 refer collectively to all microcomputers supported by the 8048/8021/8041A/8022 emulator, as listed in Table 7H-1. The term 8041A refers to the 8741A as well as to the 8041A.

In this Emulator Specifics section, the term program memory refers to the on-board ROM and/or external memory in which 8048/8021/8041A/8022 program instructions are stored. The term 8550/8540 program memory refers to the RAM in your 8550 or 8540 that may be used as a substitute for memory in the microcomputer and/or prototype. Throughout the other sections of your System Users Manual, the term program memory refers to 8550 or 8540 program memory.

EMULATOR-SPECIFIC COMMANDS, PARAMETERS, AND DISPLAYSSEL --- Selecting an Emulator

The SEL (SElect) command allows you to select the emulator you want to use with your 8550 or 8540. The following command line selects the 8048/8021/8041A/8022 emulator and assembler:

```
> SEL 8048 <CR>
```

Circuitry in the prototype control probe tells the operating system which microcomputer in the 8048 family is being emulated. For more information, refer to the "8048/8021/8041A/8022 Emulator Configurations" discussion at the beginning of this section.

Enter the following command line to select the 8048 assembler on the 8560:

```
$ uP=8048; export uP <CR>
```

Byte/Word Parameter

Several commands offer you the choice of operating on memory on a byte-oriented or word-oriented basis. In affected commands, this choice is represented by the -B or -W parameter. For the 8048/8021/8041A/8022 emulator, the default value is -B (Byte).

Symbolic Debug

The 8048/8021/8041A/8022 emulator supports the use of symbolic debug. Some of the displays in this document include symbolic debug information.

MAP --- Mapping Memory

The MAP command assigns 128-byte blocks of memory either to 8550/8540 program memory or to prototype memory. For the 8048/8021/8041A/8022 emulator, only blocks in the following address ranges can be mapped:

0000--0FFF (8048/8049 program memory)

2000--20FF (8048/8049 external data memory)

NOTE

No memory mapping is possible for the 8021, 8041A, or 8022: these microcomputers do not support external memory.

For more information on memory considerations, refer to the "Special Considerations" discussion later in this section.

EVT Address Parameter

If you are using the Real-Time Prototype Analyzer (RTPA) option with your 8550, the addresses used in the EVT command line must reflect the memory mapping conventions defined in the "Special Considerations" subsection later in this section. (The RTPA option is not supported on the 8540.)

TTA Bus Operation Designators

Table 7H-2 lists the 8048/8021/8041A/8022 bus operation designators recognized by the Trigger Trace Analyzer's BUS command.

Table 7H-2  
8048/8021/8041A/8022 Bus Operation Designators

Symbol	Bus Operation Type
CLR	All types
F	Instruction fetches
I	I/O operations (SVCs only)
NF	Non-fetches
M	Memory accesses
RD	Reads
WT	Writes



---

Setting Breakpoints

The 8048/8021/8041A/8022 emulator allows you to specify up to two breakpoints with the BK (BreAkpoinT) command. Breakpoints should be restricted to the ranges 0--0FFF (program memory) and 2000--20FF (8048/8049 external data memory). The 8048/8021/8041A/8022 emulator cannot monitor accesses to internal data memory. Refer to the "Special Considerations" discussion later in this section for information on the memory limitations of your microcomputer.

Memory Allocation Commands

The Memory Allocation Controller (MAC) option cannot be used with the 8048/8021/8041A/8022 emulator. The 8048/8021/8041A/8022 emulator does not use the MEMSP command, and does not support memory space qualifiers or expressions. The 8048 family emulator supports the AL (ALlocate) command, as described in the Command Dictionary of your System Users Manual. The DEAL, MEM, and NOMEM commands are not valid with the 8048/8021/8041A/8022 emulator.

NOTE

You should avoid allocating memory in the range 2000--20FFH and in the range 4000--407FH. These memory ranges are used to emulate External Data Memory and Internal Data Memory, respectively. Allocation of memory in these ranges may yield unexpected results.

Register Designators

Table 7H-3 alphabetically lists the symbols used by DOS/50 and OS/40 to designate the registers and flags used by the microcomputers in the 8048 family. The table provides the following information for each symbol:

- the microcomputers for which that symbol is used
- a description of the register or flag that the symbol represents
- the size of the register or flag
- the value assigned to the register or flag by the RESET command
- whether the register or flag can be assigned a value by the S (Set) command

Figure 7H-1 shows the contents of the 8048/8021/8041A/8022 program status word.

Table 7H-3  
8048/8021/8041A/8022 Registers and Flags

DOS/50 or OS/40 Symbol	8 8 8 8   0 0 0 0   4 2 4 2   8 1 1 2	Description	Size in Bits (*a)	Value After RESET (*b)	Altered by S Command?
A	x x x x	Accumulator	8	NC	yes
AO--A7	x x	Alternate registers 0--7	8 each	NC	yes
AC	x x x x	Auxiliary carry flag: bit 6 of PSW	1	0	yes
AN	x	Analog input pin	1	NC	yes
CHIP	x x x x	Microcomputer name	NA	NC	no
CY	x x x x	Carry flag: bit 7 of PSW	1	0	yes
DMA	x	EN DMA instruction exe- cuted since last RESET? 1=yes; 0=no	1	0	no
EI	x x x	External interrupt flag: 1=enabled; 0=disabled	1	0	yes
EPM	x	Switch set to enable external program memory? Y=yes; N=no	NA	NC	no
FO	x x	Flag 0: bit 5 of PSW	1	0	yes
F1	x x	Flag 1	1	0	yes
FLG	x	EN FLAGS instruction exe- cuted since last RESET? 1=yes; 0=no	1	NC	no
IBF	x	Input buffer full flag	1	NC	no
IIP	x x x	Interrupt in progress flag: Y=yes; N=no	NA	N	no
MB	x	Memory bank	1	0	yes
OBF	x	Output buffer full flag	1	NC	no
PC	x x x x	Program counter	12	0	no
PSW	x x x x	Program status word (*c)	8	0 or 8	yes
RO--R7	x x x x	Registers 0--7	8 each	NC	yes
RB	x x	Register bank: bit 4 of PSW	1	0	yes
RETURN	x x x x	Value of last word pushed	16	(*d)	no (*d)
STACK	x x x x	Stack pointer: 08 to 16, derived from bits 0--2 of PSW	NA	08	no (*d)
STF	x	STS flags	4	0	no
TC	x x x x	Timer/counter selector: 0=stop; 1=timer; 2=counter	2	0	yes
TF	x x x x	Timer overflow flag	1	0	yes
TI	x x x	Timer/counter interrupt flag: 1=enabled; 0=disabled	1	0	yes
TR	x x x x	Timer/counter	8	NC	yes

(\*a) NA refers to information not maintained by the microcomputer itself.

(\*b) NC means not changed by RESET.

(\*c) Figure 7H-1 shows the contents of the program status word.

(\*d) You cannot use the S command to alter STACK or RETURN directly;  
however, these values depend on bits 0--2 of the PSW, which can be  
directly altered using S.

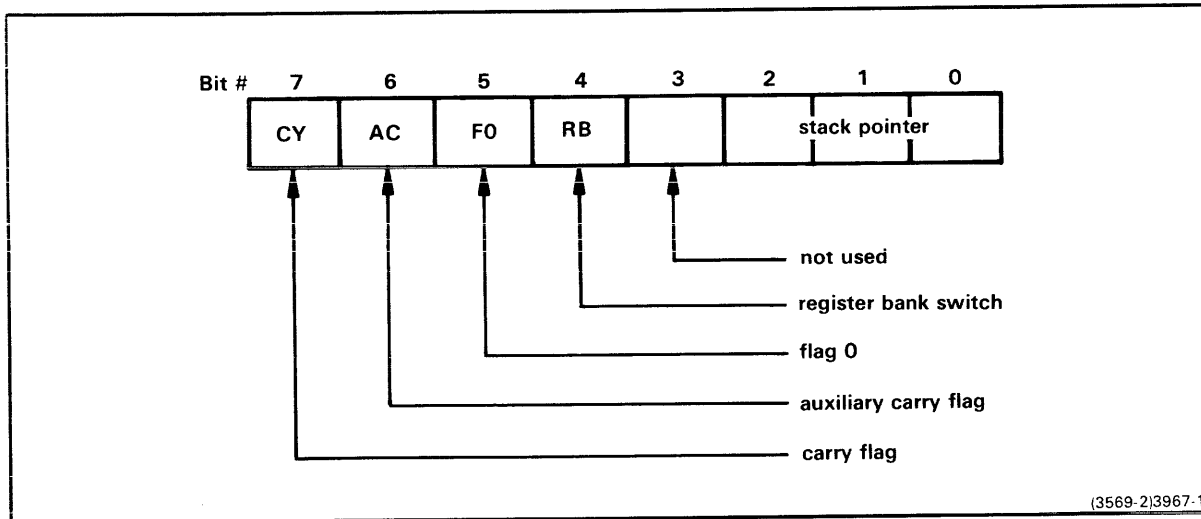


Fig. 7H-1. 8048/8021/8041A/8022 program status word.

### DS --- Sample Status Display

The DS (Display Status) command displays the status and register contents of the emulator. All numbers in the DS display are hexadecimal. Here is an example of a DS display line produced by the 8048/8021/8041A/8022 emulator when it is used to emulate an 8048:

> DS <CR>

PC	CHIP	EPM	A	PSW	TR	RB		REGISTERS
010A	8048	Y	11	00	00	0		RO-R7= 19 36 07 A6 FD 00 00 00
								A0-A7= 86 00 2F 40 49 09 7D 00

IIP	EI	TI	TC	MB	STACK	RETURN	TF	FO	F1	AN	STF	DMA	IBF	OBF	FLG
N	0	0	0	0	08	1F06	0	0	0						

The DS display differs slightly for the 8021, 8022, and 8041A. Table 7H-3 explains the symbols displayed by DS.

For the 8048/8021/8041A/8022 emulator, the short and long forms of the DS display are the same: DS -L gives the same display as DS.

---

RESET --- Resetting Emulator Status

The RESET command sends a hardware reset signal to the emulating microcomputer. The "Value After RESET" column of Table 7H-3 indicates which registers are affected by the RESET command.

Example. Suppose the DS command returns the following status:

> DS <CR>

PC	CHIP	EPM	A	PSW	TR	RB	REGISTERS								
090A	8048	Y	02	FF	40	1	R0-R7=	86	00	2F	40	49	09	7D	00
							A0-A7=	19	36	07	A6	FD	00	FF	00

IIP	EI	TI	TC	MB	STACK	RETURN	TF	FO	F1	AN	STF	DMA	IBF	OBF	FLG
N	1	1	2	1	16	8208	1	1	1						

Enter the RESET command; then use DS to check the results.

> RESET <CR>

> DS <CR>

PC	CHIP	EPM	A	PSW	TR	RB	REGISTERS								
0000	8048	Y	02	08	40	0	R0-R7=	19	36	07	A6	FD	00	FF	00
							A0-A7=	86	00	2F	40	49	09	7D	00

IIP	EI	TI	TC	MB	STACK	RETURN	TF	FO	F1	AN	STF	DMA	IBF	OBF	FLG
N	0	0	0	0	08	1F06	0	0	0						
^	^	^	^	^	^	^	^	^	^						

The arrows show the changed registers.

DI --- Sample Disassembled Code

The DI (DISassemble) command translates object code in memory into assembly language instructions. DI displays object code, assembly language mnemonics, and operands. Here is an example of 8048 DI output:

> DI 100 10E <CR>

```

LOC      INST MNEM  OPER
000100 B932 MOV   R1,#32
000102 BF05 MOV   R7,#05
000104 27  CLR   A
000105 61  ADD   A,@R1
000106 19  INC   R1
000107 EF05 DJNZ  R7,#05 0105
000109 AA  MOV   R2,A     =====
00010A 23F7 MOV   A,#F7     |
00010C 00  NOP                       effective address
00010D A3  MOVP  A,@A     of branch instruction
00010E 00  NOP

```

```

=====
|      |      |      |
|      |      |      |
|      |      |      |      +-- operand(s): address, registers, or data
|      |      |      |      |      being operated on
|      |      |      |
|      |      |      |      +----- instruction mnemonic
|      |      |      |
|      |      |      |      +----- machine language instruction
|      |      |      |
+----- address of instruction

```

TRA --- Sample TRAcE Display

The TRAcE command establishes the conditions for displaying trace lines during program execution. For the 8021 and 8022, the TRA -L display is the same as the default (TRA -S) display. For the 8048/8049 and the 8041A, the TRA -L display includes the contents of the alternate registers (A0--A7) in addition to the TRA -S information.

NOTE

When TRAcE conditions have been set, the emulator runs at slower than normal processing speeds and RTPA breakpoints are suppressed.

Here is an example of 8048 TRAcE output:



SERVICE CALLS

Service calls (SVCs) enable your program to use many system capabilities of your 8540, 8550, or 8560 while your program is running on the emulator processor.

An SVC is invoked with the following 8048/8021/8041A/8022 instruction sequence:

```

NOP
MOVP A,@A
NOP

```

The value in the accumulator at the time of the SVC directs the system to a specified memory address called the SRB pointer (which points to the SRB -- the Service Request Block). The SRB pointer tells the system where to find the data (stored in the SRB) that informs the system which function to perform. Refer to the Service Calls section of your System Users Manual for an explanation of service calls, service request blocks, and SRB pointers.

Your program can point to eight SRBs at any one time. (Under certain circumstances, your program can store new addresses in the SRB pointers as it executes. See the following discussion, "SVC Design Features.") Table 7H-4 shows the default addresses for the eight SRB pointers. These addresses and their associated accumulator values can be altered with the SVC command to suit your program requirements. See the Command Dictionary section of your System Users Manual for syntax and use of the SVC command.

SVCs in Modes 1 and 2

The 8048/8021/8041A/8022 emulator does not support SVCs in emulation modes 1 or 2.

Table 7H-4  
8048/8021/8041A/8022 Service Calls

SVC Number	Value in Accumulator	Default SRB Pointer Location
1	F7	40,41
2	F6	42,43
3	F5	44,45
4	F4	46,47
5	F3	48,49
6	F2	4A,4B
7	F1	4C,4D
8	F0	4E,4F

---

SVC Demonstration

Figure 7H-2 lists an 8048/8021/8041A/8022 program that uses four SVC functions: Assign Channel, Read ASCII, Write ASCII, and Abort. The program's algorithm is explained in the Service Calls section of your System Users Manual, which demonstrates a version of the program written in 8085A assembly language. You can perform a parallel demonstration with the 8048/8021/8041A/8022 emulator and A Series Assembler using the program in Fig. 7H-2.

NOTE

The program shown in Fig. 7H-2 is written for an A Series assembler. To make this acceptable for a B Series assembler (as required by an 8560), change each double quote (") to a single quote (').



```

; SSSSS V V CCCCC
; S V V C
; SSSSS V V C DEMONSTRATION. 8048 EMULATOR
; S V V C
; SSSSS V CCCCC
; ORG 40H ; BEGINNING OF SRB VECTOR
; BYTE HI(SRB1FN),LO(SRB1FN)
; BYTE HI(SRB2FN),LO(SRB2FN)
; BYTE HI(SRB3FN),LO(SRB3FN)
; BYTE HI(SRB4FN),LO(SRB4FN)
; BYTE HI(SRB5FN),LO(SRB5FN)
; END OF SRB VECTOR
; ORG 50H ; SET UP SRB AREAS
; SRB1 = ASSIGN "CONI" TO CHANNEL 0
SRB1FN BYTE 10H ; ASSIGN
; TO CHANNEL 0
SRB1ST BLOCK 01H ; STATUS RETURNED HERE
; BLOCK 02H ; BYTES 4 AND 5 NOT USED
; BYTE 05H ; LENGTH OF "CONI"+<CR>
; BYTE HI(CONI) ; POINTER TO
; BYTE LO(CONI) ; "CONI"+<CR>
; END OF SRB1
; SRB2 = ASSIGN "LPT" TO CHANNEL 1
SRB2FN BYTE 10H ; ASSIGN
; TO CHANNEL 1
SRB2ST BLOCK 01H ; STATUS RETURNED HERE
; BLOCK 02H ; BYTES 4 AND 5 NOT USED
; BYTE 04H ; LENGTH OF "LPT"+<CR>
; BYTE HI(LPT) ; POINTER TO
; BYTE LO(LPT) ; "LPT"+<CR>
; END OF SRB2
; SRB3 = READ ASCII LINE FROM CONI (CHANNEL 0)
SRB3FN BYTE 01H ; READ ASCII
; FROM CHANNEL 0
SRB3ST BLOCK 01H ; STATUS RETURNED HERE
; BLOCK 01H ; BYTE 4 NOT USED
; BLOCK 01H ; BYTE COUNT RETURNED HERE
; BYTE 00H ; 256 BYTES IN OUR BUFFER
; BYTE HI(BUFFER) ; POINTER TO
; BYTE LO(BUFFER) ; OUR BUFFER
; END OF SRB3
; SRB4 = WRITE ASCII LINE TO LPT (CHANNEL 1)
SRB4FN BYTE 02H ; WRITE ASCII
; TO CHANNEL 1
SRB4ST BLOCK 01H ; STATUS RETURNED HERE
; BLOCK 01H ; BYTE 4 NOT USED
; BLOCK 01H ; BYTE COUNT RETURNED HERE
; BYTE 00H ; 256 BYTES IN OUR BUFFER
; BYTE HI(BUFFER) ; POINTER TO
; BYTE LO(BUFFER) ; OUR BUFFER
; END OF SRB4
; SRB5 = ABORT (CLOSE ALL CHANNELS AND TERMINATE)
SRB5FN BYTE 1FH ; ABORT
; BLOCK 07H ; BYTES 2 THROUGH 8 NOT USED
; END OF SRB5

```

3967-10

Fig. 7H-2. 8048 SVC demonstration program listing (part 1 of 2).

```

BUFFER  BLOCK  100H          ; OUR I/O AREA
CONI    ASCII  "CONI"       ; ASCII OF "CONI"
        BYTE   ODH          ; + <CR>
LPT     ASCII  "LPT"        ; ASCII OF "LPT"
        BYTE   ODH          ; + <CR>
;      END OF DATA DEFINITIONS
;
;      BEGINNING OF EXECUTABLE CODE
START   ORG    10H          ; ENTRY POINT INTO PROGRAM
        MOV    A,#0F7H      ; CALL SVC1
        NOP    ; TO ASSIGN "CONI"
        MOVP   A,@A        ; TO CHANNEL 0
        NOP    ;
        MOV    A,#LO(SRB1ST) ; CHECK THE STATUS
        MOVP   A,@A        ; TO SEE IF ALL WENT WELL
        JNZ    ABORT       ; NO? STOP EVERYTHING
        MOV    A,#0F6H      ; CALL SVC2
        NOP    ; TO ASSIGN "LPT"
        MOVP   A,@A        ; TO CHANNEL 1
        NOP    ;
        MOV    A,#LO(SRB2ST) ; CHECK THE STATUS
        MOVP   A,@A        ; TO SEE IF ALL WENT WELL
        JNZ    ABORT       ; NO? STOP EVERYTHING
LOOP    MOV    A,#0F5H      ; CALL SVC3
        NOP    ; TO READ A LINE
        MOVP   A,@A        ; FROM "CONI"
        NOP    ; INTO THE BUFFER
        MOV    A,#LO(SRB3ST) ; CHECK THE STATUS
        MOVP   A,@A        ; TO SEE IF ALL WENT WELL
        JNZ    ABORT       ; NO? STOP EVERYTHING
        MOV    A,#0F4H      ; CALL SVC4
        NOP    ; TO WRITE THE LINE
        MOVP   A,@A        ; FROM THE BUFFER
        NOP    ; TO "LPT"
        MOV    A,#LO(SRB4ST) ; CHECK THE STATUS
        MOVP   A,@A        ; TO SEE IF ALL WENT WELL
        JZ     LOOP        ; YES? BACK TO READ ANOTHER LINE
        ; NO? FALL THROUGH TO TERMINATION
ABORT   MOV    A,#0F3H      ; CALL SVC5
        NOP    ; TO DO THE ABORT
        MOVP   A,@A        ;
        NOP    ;
        END    START      ; END OF PROGRAM

```

3967-11

Fig. 7H-2. 8048 SVC demonstration program listing (part 2 of 2).

This program shows the use of four 8550 service calls. The program's algorithm is explained in the Service Calls section of your System Users Manual. The program accepts a line of ASCII characters from the system terminal; then, when it receives a RETURN character, the program writes the line to the line printer and accepts another line. (On the 8550, output to the line printer is buffered. No text is printed until the line printer buffer in the 8501 becomes full or the program ends.) To terminate the program, enter a CTRL-Z while the program is waiting for input.

-----  
SVC DESIGN FEATURES

SVC data blocks (Service Request Blocks, SRB pointers, and I/O buffers) can reside in either of the following address ranges:

0000--0FFF program memory (internal and external)  
2000--20FF external data memory

SVC data blocks cannot reside in internal data memory. If you are emulating an 8021, 8022, or 8041A, external data memory is not available, and program memory is limited to 400H bytes (for the 8021 and 8041A) or 800H bytes (for the 8022). For more information, refer to Figure 7H-3 and the discussion of memory mapping under the heading "Special Considerations".

If you are emulating an 8048/8049, programming considerations may require you to put some SVC data blocks in external data memory. For example, suppose you have a program that places information in an I/O buffer, and then uses an SVC to write out the contents of the buffer. If the information is read directly into the buffer and undergoes no further change (as in the preceding SVC demonstration program), the buffer can reside in program memory. Otherwise the buffer must reside in external data memory, because there are no 8048/8021/8041A/8022 instructions that write to program memory.

For information on allocating code to different types of memory, refer to the heading "Designating Memory Areas in Assembly Language Programs" in the Special Considerations subsection.

#### Moving Data Into and Out of Program Memory

The 8021, 8041A, and 8022 cannot access external memory. To provide a one-byte output buffer for these microcomputers, DOS/50 and OS/40 allow your program to change the contents of byte 0 of program memory. The following instruction sequence is used to move the contents of the accumulator to program memory location 0:

```
NOP
MOVX @R1,A
```

The following instruction sequence is used to move the contents of program memory location 0 to the accumulator:

```
NOP
MOVX A,@R1
```

---

NOTE

Whenever a NOP instruction is followed by a MOVX instruction, DOS/50 or OS/40 process the instruction sequence as an access to program memory location 0. The contents of register R1 and the selected register bank are ignored.

The NOP--MOVX sequences may be used when emulating any of the microcomputers in the 8048 family. However, before you execute these instruction sequences, make sure your 8550 or 8540 is in emulation mode 0 and TRACE is OFF. Avoid using these sequences unless you're accessing program memory location 0.

Example. The following assembler directives set up an output buffer in bytes 0 and 1 of program memory. Your program can write to byte 0 using the NOP--MOVX instruction sequence. Byte 1 contains a RETURN character to mark the end of the buffer.

```
                ORG    0    ; BYTE 0 OF PROGRAM MEMORY
CHARBUF        BLOCK  1    ; USED AS AN OUTPUT BUFFER.
                BYTE   0DH  ; BUFFER ENDS IN <CR>.
```

---

SPECIAL CONSIDERATIONS

The emulating microcomputer for the 8048/8021/8041A/8022 emulator is contained at all times on the prototype control probe, rather than on the emulator processor module (as with most other emulators). The 8048/8021 prototype control probe contains an 8039 microcomputer, and the 8041A and 8022 prototype control probes each contain an 8035 microcomputer. Since the emulating microcomputer is not the same as the microcomputer being emulated, certain special considerations must be noted here.

Other considerations of interest to users are also outlined in this subsection. First, considerations common to all microcomputers in the 8048 family are described. Then, considerations that apply only to certain devices are described.

#### CLOCK RATE

The frequency of the emulator clock, used in emulation mode 0, is 5.0 MHz. Table 7H-5 gives the maximum allowable frequencies for the prototype's clock, used in modes 1 and 2.

#### NOTE

For all microcomputers, the prototype clock frequency must be at least 2.0 MHz in order to fulfill the clock requirement of the emulating 8035 or 8039 microcomputer.

Table 7H-5  
Maximum Allowable Clock Frequencies

Emulation Mode	Maximum Allowable Clock Frequency			
	8049, 8039 (*a)	8048, 8648, 8748, 8035, 8039-6	8041A, 8741A	8022
Mode 1 (Mapped to 8550)	11.0 MHz (*b)	6.0 MHz	6.0 MHz	3.58 MHz
Mode 1 (Mapped to Prototype) and Mode 2	11.0 MHz	6.0 MHz	External (prototype) memory is not addressable by these microcomputers.	

(\*a) To use clock frequencies above 6.0 MHz, the configuration switch in the prototype control probe's interface assembly must be set as prescribed in the 8048/8021/8041A/8022 Emulator Processor and Prototype Control Probe Installation Service Manual.

(\*b) In emulation mode 1 with memory mapped to 8550 program memory, the clock frequency is divided by 2 to produce an effective clock rate of 5.5 MHz.

---

**MEMORY MAPPING**

Figure 7H-3 illustrates the memory layouts for the microcomputers in the 8048 family. You can access all three 8048/8021/8041A/8022 memory areas (program memory, internal data memory, and external data memory) using standard DOS/50 or OS/40 memory manipulation commands such as D, EX, F, LO, P, and SEA.

**NOTE**

In this Emulator Specifics section, the term program memory refers to the on-board ROM and/or external memory in which 8048/8021/8041A/8022 program instructions are stored. The term 8550/8540 program memory refers to the RAM in the 8550 or 8540 that may be used as a substitute for memory in the microcomputer and/or prototype. Throughout the other sections of your System Users Manual, the term program memory refers to 8550 or 8540 program memory.

Program Memory. The 8048 contains 1K of on-board ROM, and the 8049 contains 2K of on-board ROM. The 8048 and 8049 can each address 4K of external program memory.

The 8021 and 8041A each contain 1K of on-board ROM. The 8022 contains 2K of on-board ROM. The 8021, 8041A, and 8022 cannot access external memory.

The emulating microcomputer contains no on-board ROM. Program memory is emulated by bytes 0000--0FFF of 8550/8540 program memory. For example, the Dump command D 0 3F displays bytes 0--3F of program memory.

For the 8048/8049 only, program memory can be mapped to the prototype. To run a program that resides in external program memory on the prototype, three conditions must be satisfied:

- The prototype EA line (8048 input) must be asserted.
- In mode 1, the appropriate address ranges of program memory must be mapped to the prototype (MAP U).
- The switch pack on the 8048/8021 prototype control probe assembly must be set to enable external program memory. For information on how to set the switch, refer to your 8048/8021/8041A/8022 Emulator Processor Installation Manual.

**NOTE**

DOS/50 and OS/40 cannot write to program memory on the prototype. For example, you cannot use the LO or MOV command to load instructions into program memory on the prototype.

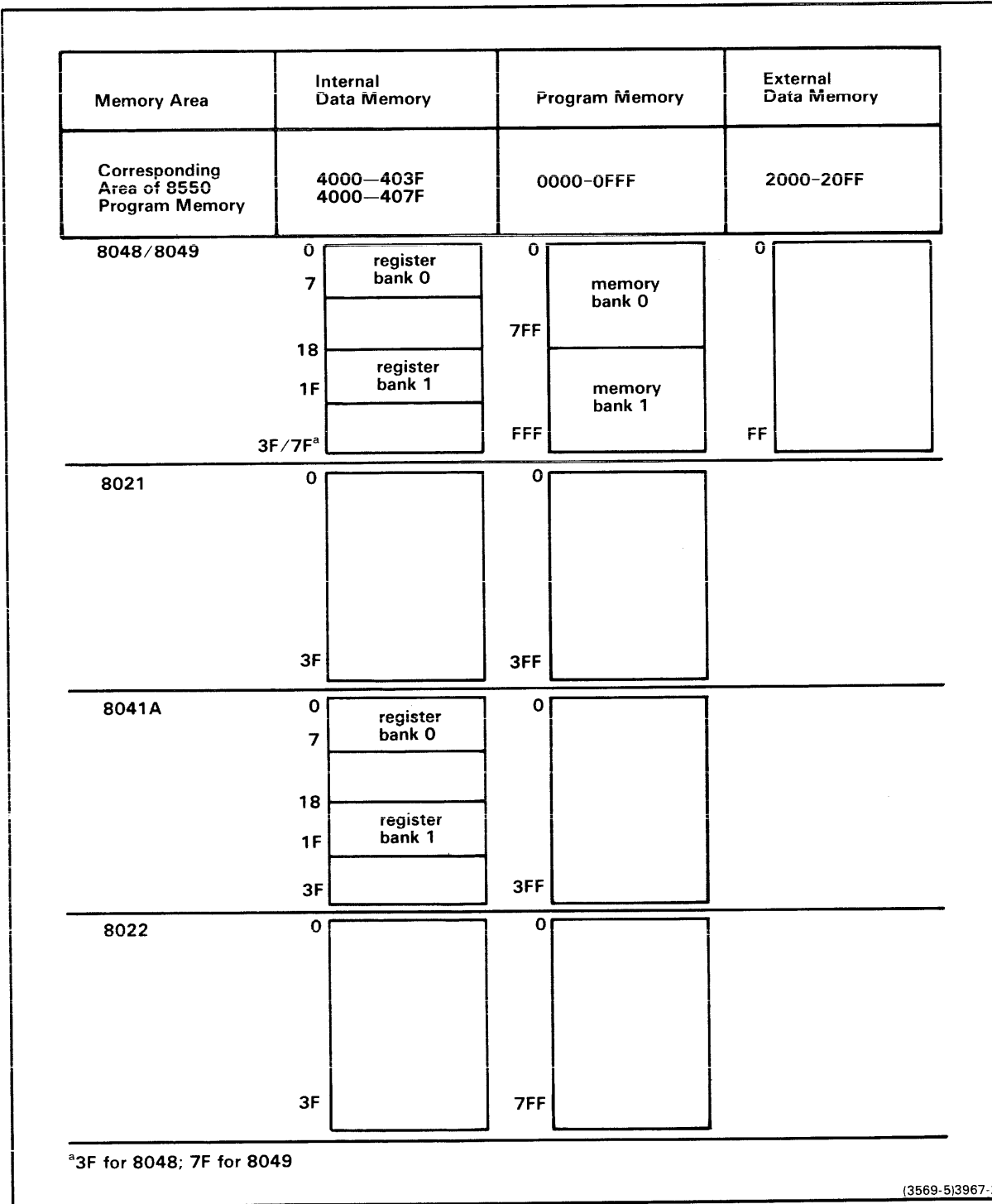


Fig. 7H-3. 8048/8021/8041A/8022 memory layout.

Internal Data Memory. The 8049 contains 128 bytes of internal RAM ("internal data memory"). Each of the other microcomputers (the 8048, 8021, 8041A, and 8022) contains 64 bytes of internal RAM.

Although this RAM actually resides in the emulating microcomputer, it can be treated as if it occupied bytes 4000--403F of 8550/8540 program memory (for the 8049, bytes 4000--407F.) For example, the command D 4000 4007 displays the contents of registers R0--R7, which occupy bytes 0--7 of internal data memory. You can access the registers in internal data memory using the S (Set) command as well as the standard DOS/50 or OS/40 memory manipulation commands.

#### NOTE

To access 8550/8540 program memory in the range 4000--407F, first transfer the contents of locations 4000-407F to another area of 8550/8540 program memory:

> MOV PP 4000 407F address <CR> (MOVE Program to Program),

then use standard operating system commands.

To transfer the contents of internal data memory to 8550/8540 program memory, enter:

> MOV UP 4000 407F address <CR> (MOVE User to Program).

#### NOTE

When any 8048-family microcomputer other than the 8049 is being emulated, bytes 4040--407F of internal data memory contain OFFH, and can only be accessed by the D command.

Internal data memory cannot be mapped to the prototype.

External Data Memory. Using the MOVX instruction, the 8048/8049 can address 256 bytes of external RAM. This external data memory is emulated by bytes 2000--20FF of 8550 or 8540 program memory. For example, the command D 2000 20FF displays the contents of external data memory. The command MAP U 2000 20FF maps all of external data memory to the prototype.

The 8021, 8041A, and 8022 cannot access external memory.

#### NOTE

To correctly emulate a microcomputer, restrict programming to within its memory limitations, as shown in Fig. 7H-4. Errors in loading or execution will result if your program attempts to access locations outside the appropriate memory areas.



---

Designating Memory Areas in Assembly Language Programs

The DOS/50 or OS/40 LO command can be used to load data into all three 8048/8021/8041A/8022 memory areas. To load a data item into internal data memory, locate the data item in the range 4000--403F (for the 8049, 4000--407F). To load a data item into external data memory, locate the data item in the range 2000--20FF.

Modules That Do Not Require Linking. To locate the data in the proper range, use ORG directives. In the following example, after the module is assembled and loaded, bytes 18--1F of internal data memory are set to zero and bytes 0--4 of external data memory contain the word "ERROR".

```

ORG      4018H          ; INTERNAL DATA MEMORY LOCATION 18
BYTE    0,0,0,0,0,0,0 ; 8 ZEROS
ORG      2000H          ; EXTERNAL DATA MEMORY LOCATION 0
ASCII   "ERROR"        ; 5 LETTERS
END

```

Modules That Must Be Linked Before Loading. Some modules must be linked before they are loaded. For example, program modules that use symbolic debug are linked in order to access symbols from the object file. In the following example, a new section is defined for each area of memory. The ORG directives are relative to the beginning of each section. Then, when you invoke either the A Series or B Series linker, include the LOCATE command option for each section. (The demonstration run program, later in this section, includes examples of linker usage.)

```

SECTION INTERNAL      ; DEFINE THE INTERNAL MEMORY SECTION
ORG      18H          ; INTERNAL DATA MEMORY LOCATION 18
FIRST   BYTE    0,0,0,0,0,0,0,0 ; 8 ZEROS
SECTION EXTERNAL      ; DEFINE THE EXTERNAL MEMORY SECTION
ORG      0H           ; EXTERNAL DATA MEMORY LOCATION 0
NEXT    ASCII   "ERROR" ; 5 LETTERS
LIST    DBG
END      FIRST

```

For more information about the linker, see your A Series or B Series Assembler Users Manual.

---

EMULATION MODES8021/8041A/8022 Emulation Modes

Because the 8021, 8041A, and 8022 have no external memory to map to the prototype, only emulation modes 0 and 1 are available for these microcomputers. In mode 0, the emulator clock is used and program I/O is handled through service calls (SVCs). In mode 1, SVCs are disabled and the prototype's clock and I/O facilities are used. In both modes, 8550/8540 program memory serves as a substitute for the microcomputer's program memory.

8048/8049 Emulation Modes

All three emulation modes are available for the 8048/8049: both program memory and external data memory can be mapped to the prototype in modes 1 and 2. If a break occurs while the microcomputer is in an interrupt service routine, the IIP (interrupt in process) flag is not reset, and the microcomputer does not return from the interrupt. When the microcomputer is restarted after the break, the only way to return from the interrupt is to complete the interrupt service routine (or enter the RESET command). Otherwise, the microcomputer will not accept additional interrupts, and the IIP flag will remain set.

When an interrupt and an instruction fetch on a breakpoint occur simultaneously, a false break line appears. The microcomputer performs the interrupt service routine, then returns and breaks.

NOP--MOVX INSTRUCTION SEQUENCE

Avoid using the NOP--MOVX command sequence except to access byte 0 of program memory. This design feature is described more fully under the heading "SVC Design Features" earlier in this section.

Normally, the MOVX instruction is illegal for the 8021, 8041A, and 8022 microcomputers. However, DOS/50 and OS/40 recognizes this MOVX sequence as valid for the specific purpose of accessing program memory location 0.

NOTE

Be sure that you have specified TRA OFF before attempting to use MOVX for this purpose.

TRA AND THE TIMER/COUNTER

Be sure that TRAcE is OFF before your program attempts to use the timer/counter. The timer/counter can interfere with the internal operations of DOS/50 or OS/40 when TRA selections are active.

---

8048/8049 SPECIAL CONSIDERATIONSPort 0 Latch

On an OUTL instruction, the 8048/8049 normally outputs data during S5 of the first cycle of the instruction. On ANL and ORL instructions, it outputs data during S4 of the second cycle. However, for all three instructions, the emulator outputs port 0 data during S1 of the second cycle of the instruction.

Port 2 Latch and External Program Memory Switch

A switch in the 8048/8021 prototype control probe enables the emulating microcomputer to access external program memory. The status of this switch affects activity on pins P20--P23, which output the four high-order bits of the program counter during instruction fetches to external program memory.

When this switch is on, pins P20--P23 behave as if external program memory is always being accessed, regardless of the value of the program counter and the state of the EA input. The address information on port 2 will always be output; the I/O information on port 2 will unlatch during every memory access, then relatch on the leading (rising) edge of the subsequent ALE. When an IN P2,A instruction is executed and the switch is on, the emulator will sample the P20--P23 pins regardless of whether they were previously programmed as "output." Any of the P20--P23 lines that are not driven by the prototype will appear as ones in the accumulator.

On an OUTL instruction, the 8048/8049 normally latches data during S5 of the first cycle of the instruction. On ANL and ORL instructions, it latches data during S4 of the second cycle. However, when the external program memory switch is off, the emulator latches P20--P23 I/O data during S3 of the second cycle of the next instruction.

8048/8021 Address Latch Enable (ALE)

The emulating microcomputer in the 8048/8021 prototype control probe is an 8039. The 8039 continuously issues the ALE signal to the prototype, even when it is executing 8550 of 8540 debug routines. The internal program addresses of these debug routines may be interspersed with your program's addresses.

Prototype Clock

The 8048/8021 prototype control probe treats the X1 clock pin as an input and the X2 pin as an output in order to drive a crystal circuit. However, Intel's revised specifications for the 8048, 8035, 8049, and 8039 microcomputers require that both pins be driven if the clock source is an external TTL clock. Therefore, if your prototype uses an external TTL clock, the prototype X2 driver should be disconnected while using the emulator.

---

8021 SPECIAL CONSIDERATIONS8048/8021 Address Latch Enable (ALE)

The emulating microcomputer in the 8048/8021 prototype control probe is an 8039. The 8039 continuously issues the ALE signal to the prototype, even when it is executing 8550 or 8540 debug routines. The internal program addresses of these debug routines may be interspersed with your program's addresses.

JMP and CALL Instructions

The 8021 is emulated by an 8039 microcomputer. The 8039 can address 2K of program memory, while the 8021 has only 1K. Thus, whenever you use a JMP or CALL instruction with the 8021, the op code must be in the range 04, 14, ..., 64, 74. For the 8021, the following op codes are illegal: 84, 94, A4, B4, C4, D4, E4, and F4. If the emulator attempts to execute one of these illegal codes, no error indication will appear in your TRAcE display; the program will simply halt. The only way to determine the cause of the halt is to inspect the op codes in your listing.

8041A SPECIAL CONSIDERATIONSJMP and CALL Instructions

The 8041A is emulated by an 8035 microcomputer. The 8035 can address 2K of program memory, while the 8041A has only 1K. Thus, whenever you use a JMP or CALL instruction with the 8041A, the op code must be in the range 04, 14, ..., 64, 74. For the 8041A, the following op codes are illegal: 84, 94, A4, B4, C4, D4, E4, and F4. If the emulator attempts to execute one of these illegal codes, no error indication will appear in your TRAcE display; the program will simply halt. The only way to determine the cause of the halt is to inspect the op codes in your listing.

Input/Output Buffers

The 8041A has internal input/output buffers. On the 8041A prototype control probe, these buffers are located external to the emulating 8035 microcomputer. Thus, the buffer contents can be altered even when the emulator is not active. In such cases, the buffer flags would be affected, and the DS display would not show the correct emulator status.

---

RETR Instruction

In the 8041A prototype control probe, the F0 flag is placed on the F0 flag stack by an interrupt, but not by a subroutine call. Therefore, use a RET instruction to return from a subroutine. Use a RETR instruction only to return from an interrupt service routine.

## 8022 SPECIAL CONSIDERATIONS

RETI Instruction

The op code for the 8022 RETI instruction is 93H. However, the emulating 8035 microcomputer recognizes that op code as a RETR instruction. The 8035 RETR instruction performs three functions:

- allows interrupts (if interrupts are enabled);
- restores the program counter from the stack;
- restores the program status word (PSW) from the stack.

However, the 8022 RETI instruction only performs the first two functions. Thus, whenever your program includes a RETI instruction, the 8035 will automatically restore the PSW. However, your prototype 8022 will not restore the PSW.

Also note that the 8022 RETI instruction increments the program counter, while the 8035 RETR instruction does not. Thus, the possibility exists that your program could execute the same instruction twice upon return from an interrupt.

A-to-D Converter

The 8022 requires that the analog input be maintained at a constant voltage during the sample time. A SEL AN0 or SEL AN1 instruction changes the input voltage. Thus, the first RAD instruction after any SEL instruction will be inaccurate. Each succeeding RAD will be accurate.

Pullup Options

Two pullup options are available with the 8022. With the emulating 8035 microcomputer, these options are selected by a cuttable run (T1) and by switch settings (Port 0) on the 8022 prototype control probe. Refer to your 8048/8021/8041A/8022 Emulator Processor Installation Manual for information on how to set these pullup options.

---

EMULATOR TIMING

The emulating microcomputer resides in the prototype control probe, and the signals between the prototype and the emulating microcomputer are buffered. Therefore, some timing differences exist between the 8048/8021/8041A/8022 emulator and a microcomputer inserted directly into the prototype.

Table 7H-6 lists the probe/microcomputer timing differences for the 8048/8021 prototype control probe. Figure 7H-4 contains timing diagrams corresponding to the signals listed in that table.

Similarly, Tables 7H-7 and 7H-8 and Fig. 7H-5 give timing differences for the 8041A prototype control probe. Table 7H-9 and Fig. 7H-6 give timing differences for the 8022 prototype control probe.

Table 7H-6  
Probe/Prototype Interface Delays for the 8048/8021

Signal		t(PHL) (ns) Maximum	t(PLH) (ns) Maximum
ALE		20	20
PSEN		32	32
RD, WR		26	22
PROG		20	20
DB0--DB7 (*a) (P00-P07)	t(1)---fetch cycle t(2)---execute cycle	90 38	90 38
Prototype to CPU			
DB0--DB7 (P00-P07)	t(3)---address out t(4)---external data out	38 38	38 38
CPU to Prototype	t(5)---OUTL, ANL, ORL data out	(*b)	(*b)
P10--P17, P24--P27		2	2
P20--P23		(*c)	(*c)
T0 (*d) out/in		15	15
T1			182
INT		32	32
RST (t(PHL) for 8021, t(PLH) for 8048)		284	284
SS		32	32
CLK		79	79

(\*a)  $t(RD) = t(1,2) + t(\text{prototype memory access})$

(\*b) OUTL, ANL, and ORL bus I/O information is latched during S1 of the second cycle of the instruction.

(\*c) When external program memory is enabled (by a DIP switch in the prototype control probe), the maximum address delay is 32 ns. When external program memory is disabled, OUTL, ANL, and ORL information is latched during the S3 cycle following the next instruction fetch.

(\*d) If the prototype clock frequency is greater than 6 MHz and memory is mapped to the 8550 or 8540, T0 out is divided by 2.

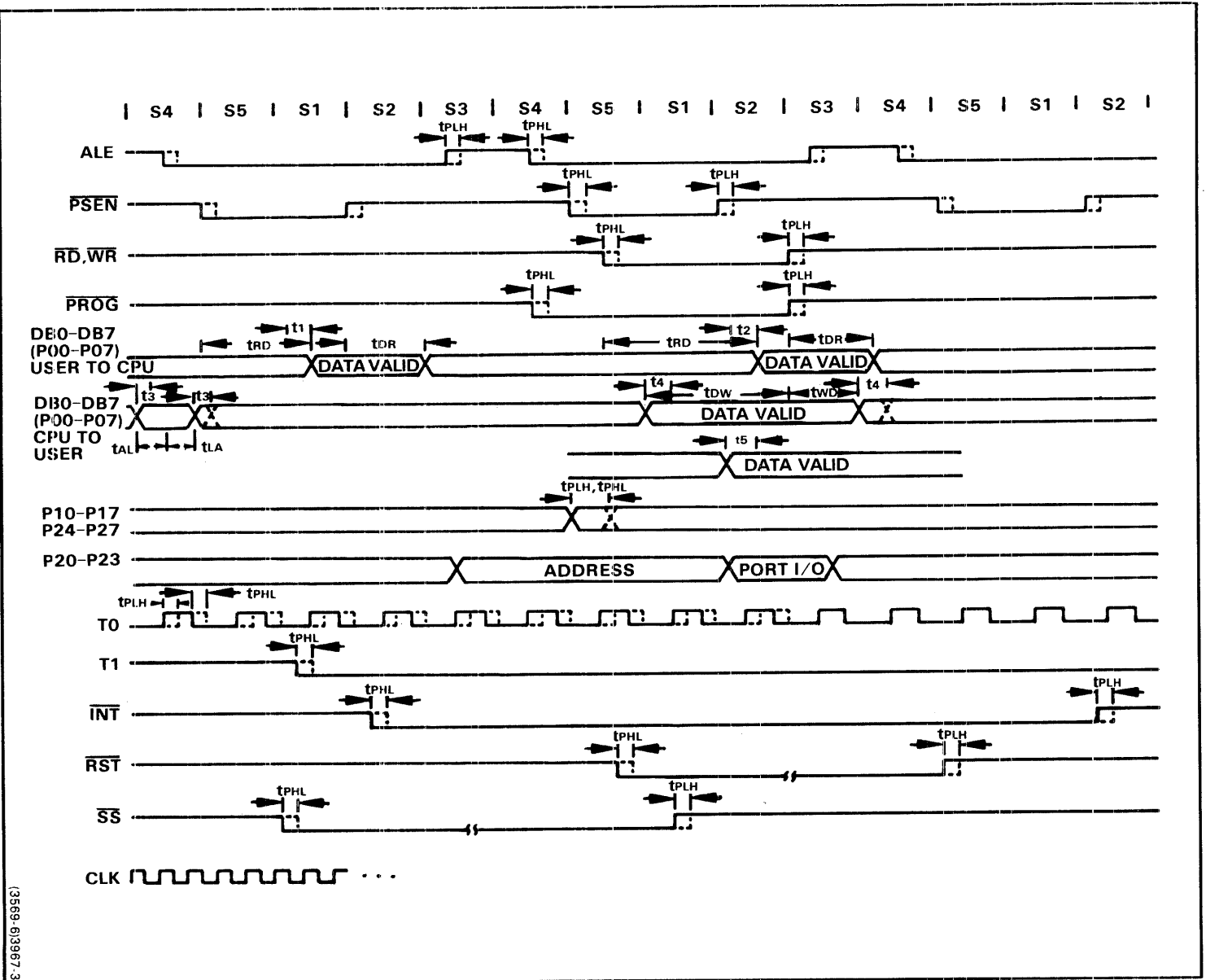


Fig. 7H-4. 8048/8021 timing diagrams.

These timing diagrams illustrate the signals described in Table 7H-6. For signals sent from the CPU to the prototype, solid lines represent timing at the prototype. For signals sent from the prototype to the CPU, solid lines represent timing at the prototype and dashed lines represent timing seen by the CPU.

Table 7H-7  
Probe/Prototype Interface Delays for the 8041A

Signal	t(PHL) (ns) Maximum	t(PLH) (ns) Maximum
SYNC	20	20
PROG	20	20
T1		39
P10--P17	2	2
T0	45	34

Table 7H-8  
Representative 8041A Probe/8041A Microcomputer Timing Differences

Symbol	Parameter	Microcomputer		Probe		Units
		Min.	Max.	Min.	Max.	
t(ACC)	DACK fall to WR or RD	0		54		ns
t(CAC)	RD or WR to DACK rise	0		71		ns
t(ACD)	DACK fall to data valid		225		225	ns
t(CRQ)	RD or WR to DRQ cleared		200		200	ns
t(AW)	CS, A0 setup to WR fall	0		0		ns
t(WA)	CS, A0 hold after WR fall	0		24		ns
t(WW)	WR pulse width	250		250		ns
t(DW)	Data setup to WR rise	150		150		ns
t(WD)	Data hold after WR rise	0		70		ns

Timing assumptions: CPU timing reference is Intel Peripheral Design Handbook, published by Intel Corp., 1979.



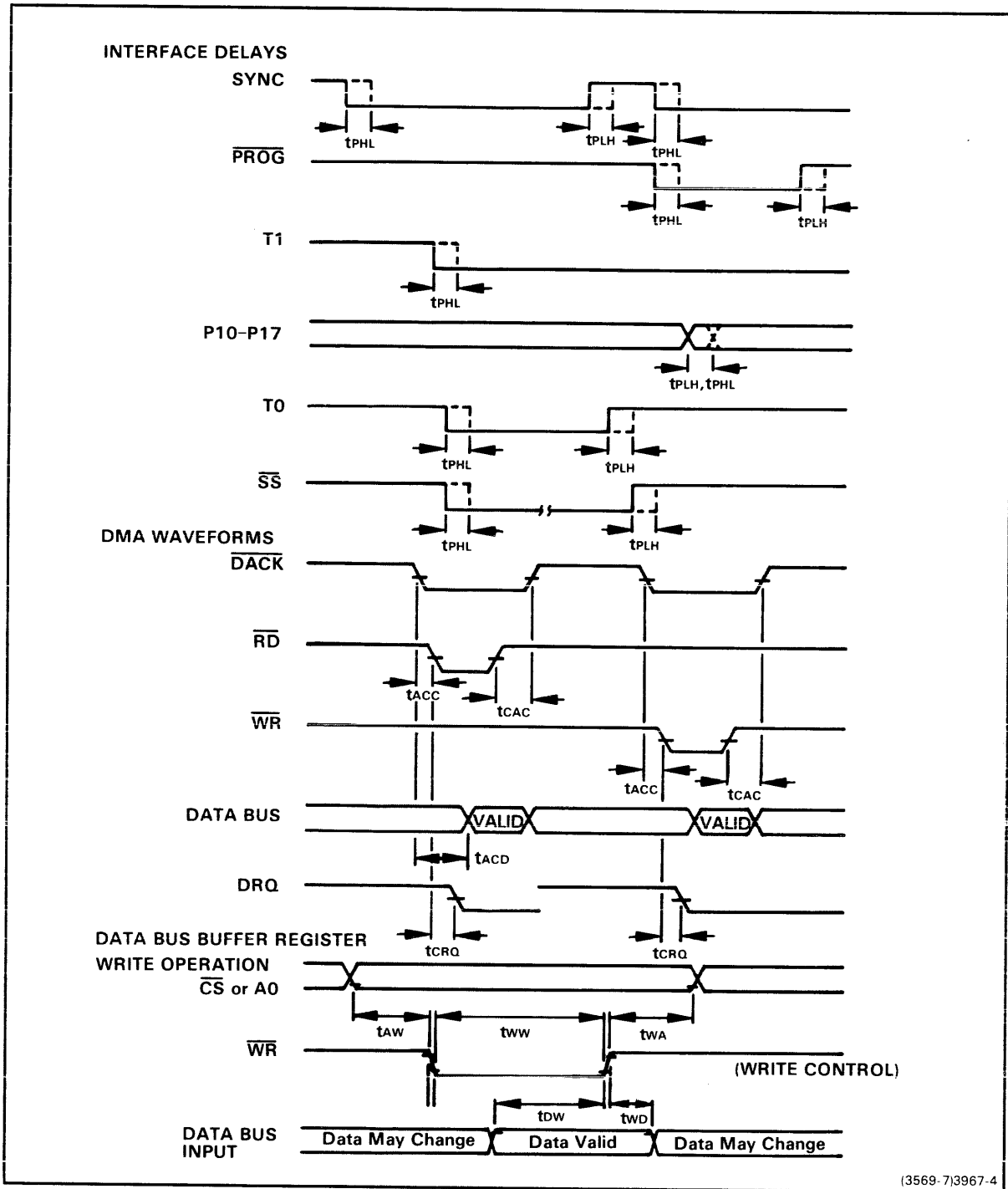


Fig. 7H-5. 8041A timing diagrams.

These timing diagrams illustrate the signals described in Tables 7H-7 and 7H-8. Dashed lines represent signal delays between the CPU and the prototype.

Table 7H-9  
Probe/Prototype Interface Delays for the 8022

Signal	t(PLH) Maximum	t(PHL) Maximum	Units
ALE	34	46	ns
P00--P07 CPU to prototype	87	91	ns
P00--P07 prototype to CPU	1.3	1.3	us
P10--P17   t(1)---CPU to   prototype	2	2	ns
t(2)---prototype   to CPU	2	2	ns
P20--P23   t(3)---A to P2	18	24	ns
(*a, *b)   t(4)---P2 to A	18	24	ns
PROG	18	24	ns
T0	24	24	ns
T1	182	182	ns
ANO, AN1 (*c)	444	444	ns
XTAL1	33	45	ns
RESET	229	229	ns

- (\*a) Inputs must be present until read by an input instruction (Intel Specification).  
 (\*b) For OUTL, ORL, and ANL instructions, data will be valid before ALE following the next instruction fetch.  
 (\*c) Input capacitance: 37 pf maximum.

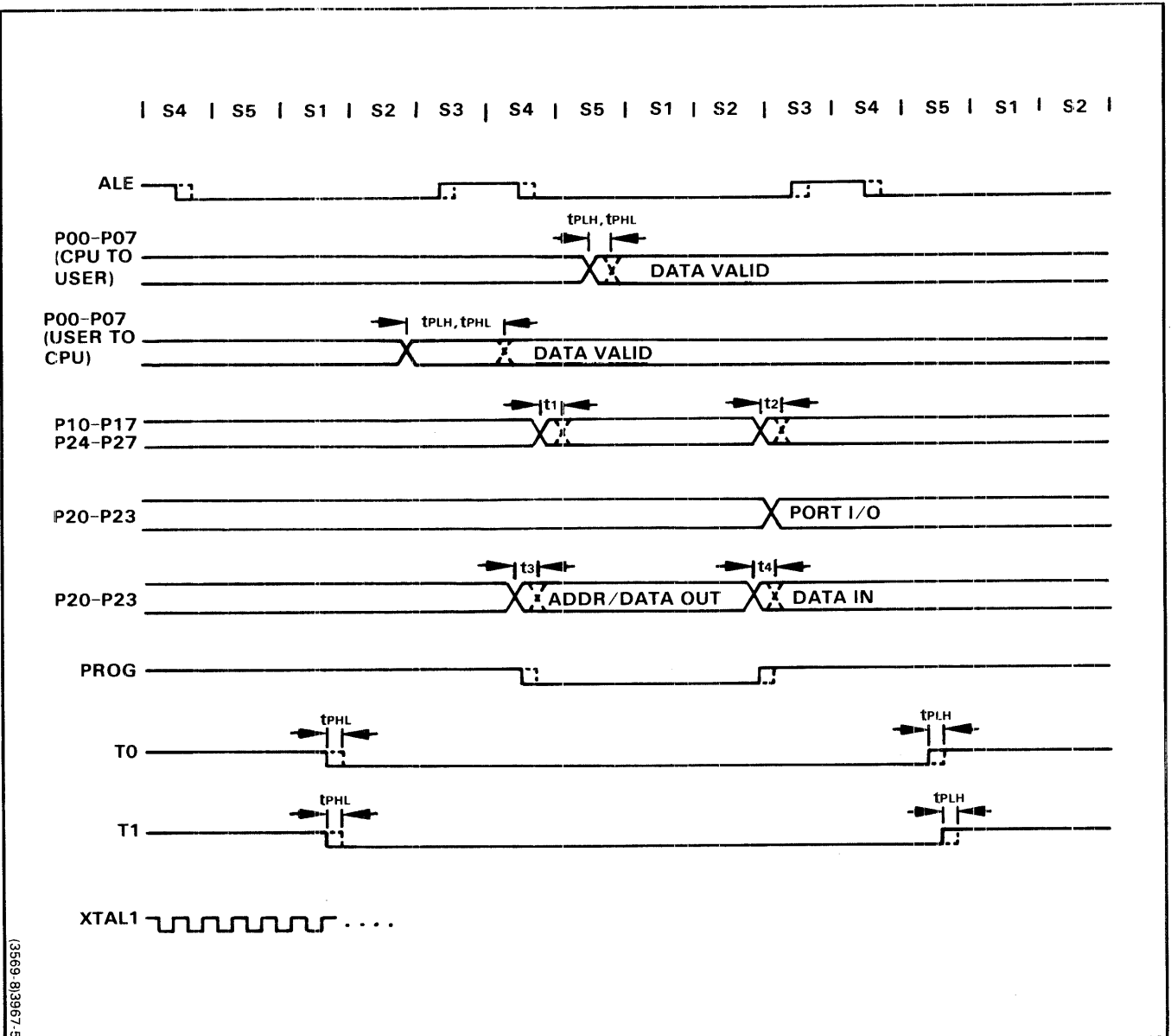
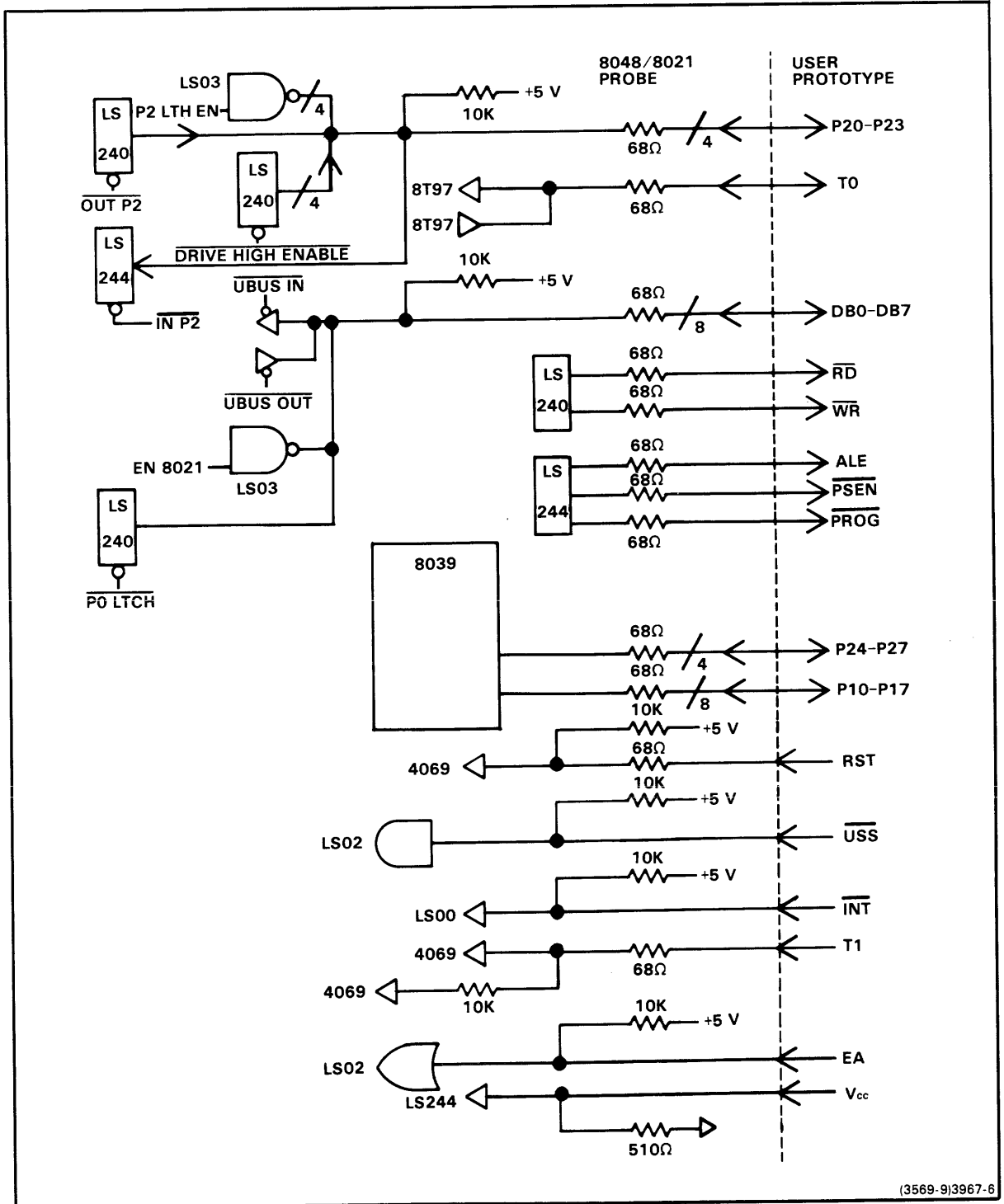


Fig. 7H-6. 8022 timing diagrams.

These timing diagrams illustrate the signals described in Table 7H-9. Dashed lines represent signal delays between the CPU and the prototype.

PROBE/PROTOTYPE INTERFACE DIAGRAMS

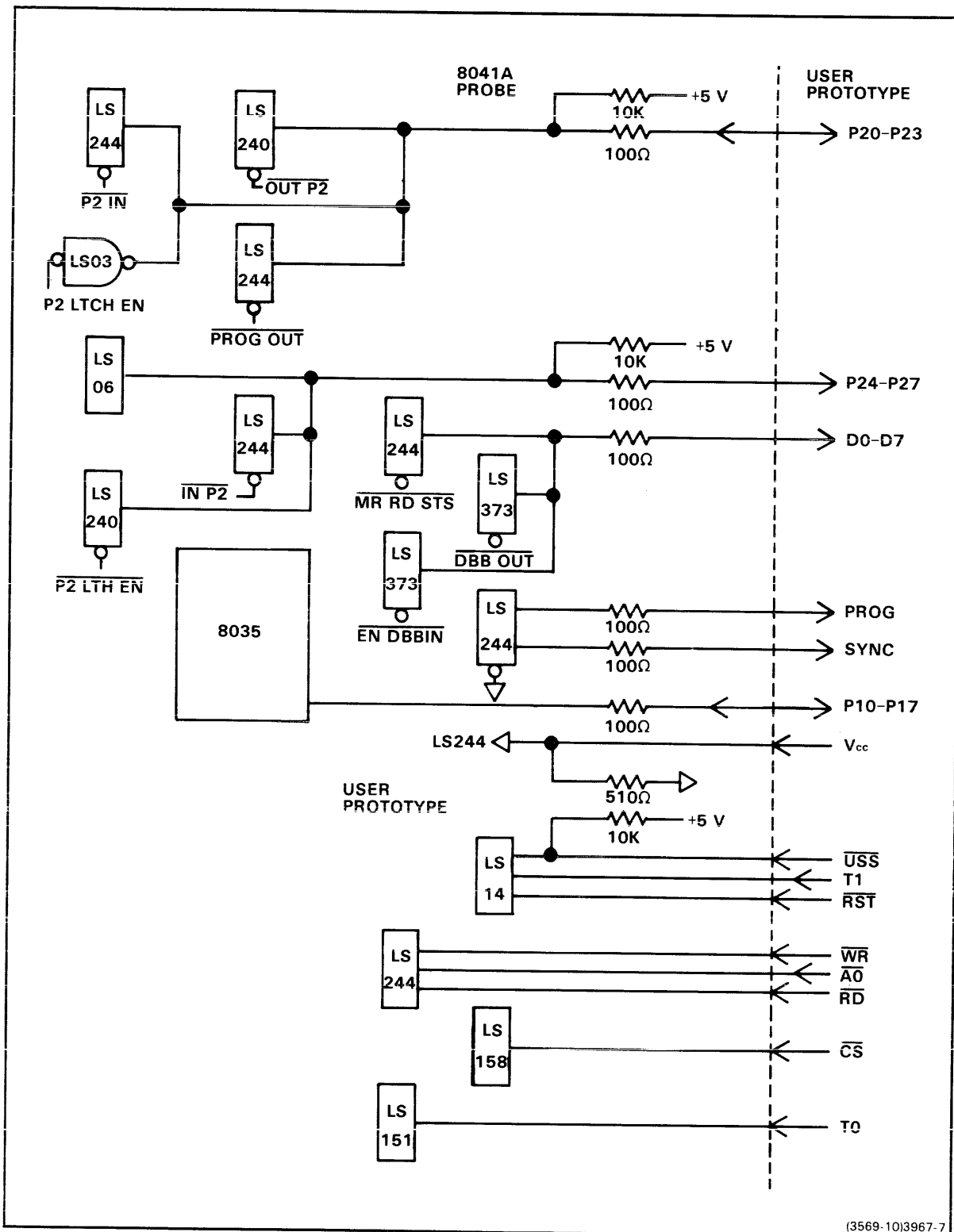
Figures 7H-7, 7H-8, and 7H-9 are block diagrams of the probe/prototype interfaces for the 8048/8021, 8041A, and 8022 prototype control probes, respectively.



(3569-9)3967-6

Fig. 7H-7. Block diagram of 8048/8021 probe/prototype interface.

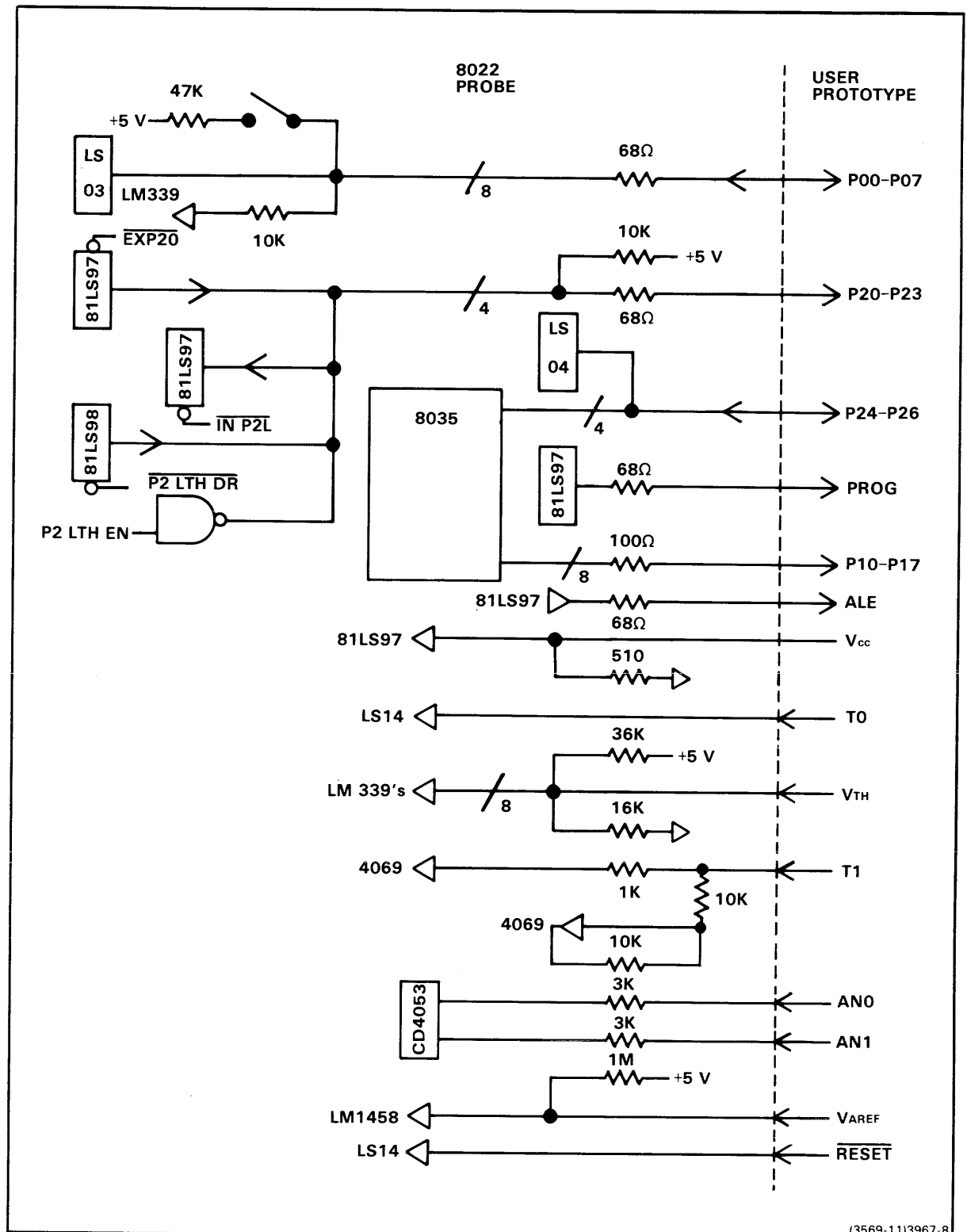
This figure provides a functional overview of signal buffering between the prototype and the emulating microcomputer on the 8048/8021 prototype control probe. A more detailed circuit description can be found in the 8048/8021/8041A/8022 Emulator Service Manual.



(3569-10)3967-7

Fig. 7H-8. Block diagram of 8041A probe/prototype interface.

This figure provides a functional overview of signal buffering between the prototype and the emulating microcomputer on the 8041A prototype control probe. A more detailed circuit description can be found in the 8048/8021/8041A/8022 Emulator Service Manual.



(3569-11)3967-8

Fig. 7H-9. Block diagram of 8022 probe/prototype interface.

This figure provides a functional overview of signal buffering between the prototype and the emulating microcomputer on the 8022 prototype control probe. A more detailed circuit description can be found in the 8048/8021/8041A/8022 Emulator Service Manual.

---

INSTALLING YOUR 8048/8021/8041A/8022 EMULATOR SOFTWARE8540 FIRMWARE INSTALLATION PROCEDURE

The ROMs that contain the control software for your 8048/8021/8041A/8022 emulator must be installed in your 8540's System ROM Board. Refer to your Emulator Installation Manual for instructions on installing these ROMs.

8550 INSTALLATION PROCEDURE

This subsection describes how to install the control software for your 8048/8021/8041A/8022 emulator. To complete this installation procedure, you need the following items:

- an 8550 system (with or without an 8048/8021/8041A/8022 emulator)
- a DOS/50 system disk with a write-enable tab over the write-protect slot
- an 8048/8021/8041A/8022 emulator software installation disk with no write-enable tab.

This procedure takes about five minutes.

Start up and Set the Date

Turn on your 8550 system. (For start-up instructions, refer to the paragraph "Start Up the 8550 and Its Peripherals" in the Learning Guide section of your System Users Manual.) Place your system disk in disk drive 0 and shut the drive 0 door. When you see the > prompt on your system terminal, place your installation disk in disk drive 1 and shut the drive 1 door.

Use the DAT command to set the date and time. For example, if it is 11:05 am on April 1, 1983, type:

```
> DAT 01-APR-83/11:05 <CR>
```

The system will use this information when it sets the Creation Time attribute of each file copied from your installation disk.

Install the Software

The command file INSTALL2, which is used to install the software, resides on the installation disk.

NOTE

If your system disk contains DOS/50 Version 1, use the command file INSTALL instead of INSTALL2.

To execute this command file, type its filespec:

```
> /VOL/EMU.8048/INSTALL2 <CR>
```

-----  
DOS/50 responds with the following message:

```
* During this installation procedure, one or more of the
* following messages may appear.  IGNORE THESE MESSAGES:
*
*      Error 6E - Directory alteration invalid
*      Error 7E - Error in command execution
*      Error 1D - File not found
*
* If any OTHER error message appears, see your
* Users Manual for further instructions.
*
* If no other error message appears, you'll receive a
* message when the installation procedure is complete.
*
T,OFF
```

In the installation process, you may disregard error messages 6E, 7E, and 1D; these messages have no bearing on the success of the installation. However, if a message other than 6E, 7E, or 1D appears, take the following steps:

1. Make sure you are using the right disks.
2. Make sure your system disk has a write-enable tab.
3. Make sure there are at least 3 free files and 20 free blocks on your system disk.
4. Begin the installation procedure again.

If the installation procedure fails again, copy down the error message and contact your Tektronix service representative.

The "T,OFF" command suppresses subsequent output to your system terminal (except error messages) until INSTALL2 finishes executing. Within about five minutes, INSTALL2 will finish and your system terminal will display the following message:

```
*
* Your installation has been successfully completed.
>
```

Once your software is installed, you can:

- remove your disks and turn off your 8550 system, or
- install more software, or
- continue with the 8048/8021/8041A/8022 Emulator Demonstration Run that follows in this section. If you do this, you do not have to restart the system or reset the date and time.

#### NOTE

At this point, "NO.NAME" is the current user. To change the current user back to "yourname", enter USER,,yourname.



---

 8048/8021/8041A/8022 DEMONSTRATION RUN
 

---

## INTRODUCTION

This Demonstration Run shows you how to assemble, load, execute, and monitor a simple 8048 assembly language program on your 8540 or 8550. This program can be executed using any prototype control probe that is compatible with the 8048/8021/8041A/8022 emulator. In order to perform this demonstration, your 8048/8021/8041A/8022 emulator, prototype control probe, and control software must be installed in your 8540 or 8550.

Figure 7H-10 shows the source and object code for the demonstration program. If you have an 8550, the source code and object code for the demonstration program are provided on the installation disk that contains your 8048/8021/8041A/8022 emulator control software. This demonstration shows you how to assemble the program on your 8550. (If your system disk does not contain a 8048/8021/8041A/8022 assembler, you will have to skip that part of the demonstration.)

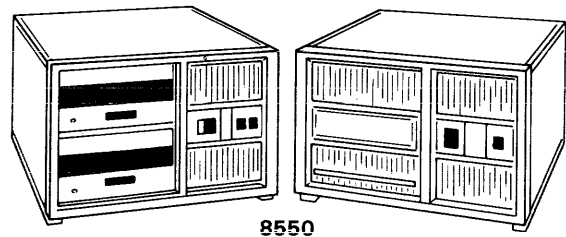
If you have an 8540/8560 system and your 8560 has a 8048/8021/8041A/8022 assembler installed, you can create and assemble the program on the 8560 and then download it to the 8540. This demonstration shows how.

If you have an 8540 that is connected to a host computer other than an 8560, we can't give you a specific list of commands for creating and assembling the program on your host (since we don't know what host you're using). However, Fig. 7H-11 gives the object code for the program in Extended Tekhex format. You can create the Tekhex file using your host's assembler or text editor, and then download the file to the 8540 via the 8540's optional COM interface.

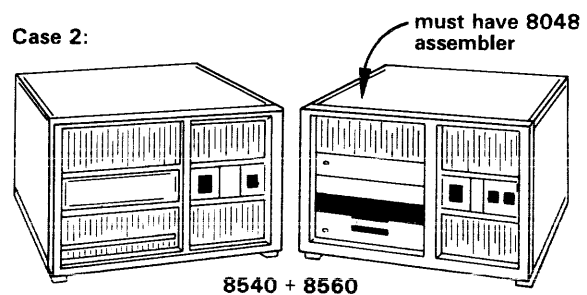
If none of these cases applies to you, you can patch the program into memory using the P command. This demonstration shows how.

Once the program is loaded or patched into memory, you can execute the program on your emulator.

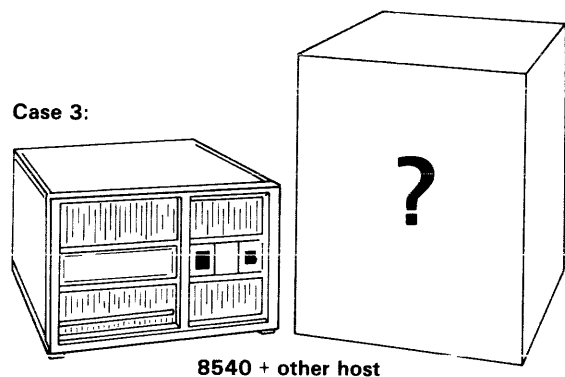
Case 1:



Case 2:



Case 3:



Case 4:

any other configuration

(3964-5)3967-9

NOTE

The 8540 commands shown in this demonstration can also be used for an 8550 that is connected to an 8560 or other host computer.

01	;8048 DEMONSTRATION RUN PROGRAM
02	SECTION DEMO
03 000100	ORG 100H ;START PROGRAM CODE AT ADDRESS 100
04 000100 B932	START MOV R1,#TABLE ;SET TABLE POINTER
05 000102 BF05	MOV R7,#TSIZE ;SET PASS COUNTER
06 000104 27	CLR A ;CLEAR ACCUMULATOR
07 000105 61	LOOP ADD A,@R1 ;ADD BYTE FROM TABLE
08 000106 19	INC R1 ;POINT TO NEXT BYTE
09 000107 EF05	DJNZ R7,LOOP ;DECREMENT PASS COUNTER AND
10	; LOOP IF NOT FIVE PASSES YET
11 000109 AA	MOV R2,A ;OTHERWISE SAVE SUM IN R2
12 00010A 23F7	MOV A,#0F7H ; AND CALL EXIT SVC
13 00010C 00	NOP ; TO END PROGRAM EXECUTION
14 00010D A3	MOVP A,@A
15 00010E 00	NOP
16	;SRB POINTER
17 00040	ORG 40H ;STORE SRB POINTER AT ADDRESS 40
18 000040 0042	BYTE 00,42H ;POINT TO SRB FOR EXIT SVC
19	;SRB FOR EXIT SVC
20 000042 1A	BYTE 1AH ;1AH = FUNCTION CODE FOR EXIT SVC
21	;TABLE OF NUMBERS TO BE ADDED
22	TSIZE EQU 5 ;TABLE SIZE = 5
23	SECTION INTERNAL ;SECTION FOR INTERNAL DATA MEMORY
24	ORG 32H ;SET UP TABLE AT ADDRESS 32
25	TABLE BLOCK TSIZE
26	LIST DBG
27	END START
=====	
	source code
	comments
	+--- object code
	+----- address
	+----- source code line number

3967-12

Fig. 7H-10. Demonstration program.

```
(A) %276C33100B932BF05276119EF05AA23F700A300
    %0E62B24000421A
    %393464DEMO010350514LOOP310515START310015TABLE23225TSIZE15
    %233318INTERNAL04400023715TABLE44032
    %098153100
```

(B) FIRST DATA BLOCK: object code for addresses 100--10E

```
header load address  object code
  |      |             |
  =====
  %276C33100B932BF05276119EF05AA23F700A300
```

SECOND DATA BLOCK: object code for addresses 40--42

```
header load object
  | address code
  |      |
  =====
  %0E62B24000421A
```

FIRST SYMBOL BLOCK

```
header section      section
  | name  field      definition
  |      |             |
  =====
  %393464DEMO010350514LOOP310515START310015TABLE23225TSIZE15
```

SECOND SYMBOL BLOCK

```
header section      section      symbol
  | name  field      definition  definition
  |      |             |             |
  =====
  %233318INTERNAL04400023715TABLE44032
```

TERMINATION BLOCK

```
header transfer
  | address
  |
  =====
  %098153100
```

3967-13

Fig. 7H-11. Demonstration program: Extended Tekhex format.

Figure 7H-11A shows an Extended Tekhex load module that contains the object code and program symbols for the demonstration program. Figure 7H-11B labels the different fields in the message blocks. If you have a host computer

-----  
other than an 8560, you can create this load module and download it to your 8540 or 8550.

#### EXAMINE THE DEMONSTRATION RUN PROGRAM

The demonstration run program adds five numbers from a table stored in 8048 internal data memory locations 32--36 and leaves the sum in register R2. (You will place values in the table later in this demonstration.) The 8085A emulator demonstration run in the Learning Guide section of your System Users Manual contains a flowchart that illustrates the steps of the program.

The source code contains two kinds of statements: assembler directives (such as ORG and BYTE) and 8048 assembly language instructions. The assembler directives are microprocessor-independent and are explained in the 8085A emulator demonstration run. The 8048 assembly language instructions are discussed in the following paragraphs.

Set Table Pointer. The MOV R1,#TABLE instruction loads the lower byte of the address of the table (4032) from internal data memory into register R1. As a result, R1 points to the first element of the table. The label START is used by the END directive to specify that the MOV R1,#TABLE instruction is the first to be executed.

Set Pass Counter. Register R7 is used as the pass counter. The MOV R7,#TSIZE instruction moves the value of TSIZE into R7. This step sets the number of passes to 5. Each time a number is taken from the table and added into the accumulator, R7 is decremented.

Clear Accumulator. The CLR A instruction zeros the accumulator so that you can start adding numbers from the table.

Add Byte From Table. The ADD A,@R1 instruction adds the byte addressed by R1 into the accumulator. The label LOOP represents the address of this instruction; this label is used by the DJNZ instruction.

Point to Next Byte. The INC R1 instruction increments R1; this register then points to the next byte in the table. For example, R1 is initialized to contain 32. After the INC R1 instruction is first executed, R1 will contain 33, the address of the second element of the table.

Decrement Pass Counter and Loop If Not Five Passes Yet. The DJNZ R7,LOOP instruction decrements R7, the pass counter. In this program, R7 is decremented each time a number is added to the accumulator. If R7 does not contain zero after being decremented, the program jumps back to the LOOP label. If R7 contains zero, the program proceeds to the next instruction, MOV R2,A.

-----  
Save Sum in R2. After all five numbers have been added together, the MOV R2,A instruction moves the sum from the accumulator into register R2. The sum must be saved in R2 because the accumulator is used by the service call that follows.

Exit. The last four instructions of the program are:

```
MOV A,#0F7H
NOP
MOVP A,@A
NOP
```

These instructions constitute a service call (SVC) that causes an exit from the program. For more information on SVCs, refer to the Service Calls discussion in this Emulator Specifics section.

#### ASSEMBLE AND LOAD THE DEMONSTRATION PROGRAM

Now it's time to create the program so you can run it on your emulator. One of the following discussions describes the set of steps that is appropriate for your hardware configuration:

- For 8550 users -- Case 1: Assemble and Load on the 8550
- For 8540/8560 users -- Case 2: Assemble on the 8560; Download to the 8540
- For 8540 users with a host computer other than the 8560 -- Case 3: Download from Your Host to the 8540
- For other hardware configurations -- Case 4: Patch the Program into Memory

Go ahead and work through the discussion that's appropriate for you. Once you've put the program into 8550/8540 program memory, turn to the heading "Run the Demonstration Program," later in this section.

---

**CASE 1: ASSEMBLE AND LOAD ON THE 8550**

This discussion shows you how to copy the demonstration program from your 8048/8021/8041A/8022 emulator software installation disk, assemble the program, and load it into 8550 program memory.

Start Up and Log On

Turn on your 8550 system. (For start-up instructions, refer to the paragraph "Start Up the 8550 and Its Peripherals" in the Learning Guide section of your System Users Manual.) Place your system disk in disk drive 0 and shut the drive 0 door. When you see the > prompt on your system terminal, place your installation disk in disk drive 1 and shut the drive 1 door.

Use the DAT command to set the date and time. For example, if it is 2:35 pm on April 1, 1983, type:

```
> DAT 01-APR-83/2:35 pm <CR>
```

Use the SEL command to tell DOS/50 to use the assembler and emulator software designed for the 8048 family:

```
> SEL 8048 <CR>
```

The SEL command automatically sets the emulation mode to 0.

Copy the Demonstration Run Program from the Installation Disk

Enter the following command lines to create an empty directory called DEMO on your system disk and make DEMO the current directory. The BR command creates a brief name, ROOT, to mark the old current directory. At the end of this demonstration, you will return to this ROOT directory and delete the DEMO directory and its contents.

```
> BR ROOT /USR <CR>
```

```
> CREATE DEMO <CR>
```

```
> USER DEMO <CR>
```

Now use the COP command to copy all the files in the DEMO2 directory on the installation disk to the DEMO directory you just created:

```
> COP /VOL/EMU.8048/DEMO2/* * <CR>
```

Remove your installation disk from drive 1 and put it away.

Now list the files you have just copied to the current directory:

---

```
> L <CR>
```

```
Filename
```

```
ASM
LOAD
```

```
Files used      96
Free files      160
Free blocks     738
Bad blocks      0
```

The file named ASM contains the assembly language source code for the demonstration program, and the file named LOAD contains the executable object code. This copy of LOAD will be used in the demonstration only if you do not have an 8048/8021/8041A/8022 assembler (and thus cannot create your own object file and load file from the source file.)

### Examine the Demonstration Program

Enter the following command line to display the source file ASM on the system terminal:

```
> CON ASM <CR>
;8048 DEMONSTRATION RUN PROGRAM
SECTION DEMO
ORG      100H      ;START PROGRAM CODE AT ADDRESS 100
START   MOV      R1,#TABLE ;SET TABLE POINTER
        MOV      R7,#TSIZE ;SET PASS COUNTER
        CLR      A      ;CLEAR ACCUMULATOR
LOOP    ADD      A,@R1    ;ADD BYTE FROM TABLE
        INC      R1      ;POINT TO NEXT BYTE
        DJNZ     R7,LOOP  ;DECREMENT PASS COUNTER AND
                        ; LOOP IF NOT FIVE PASSES YET
        MOV      R2,A      ;OTHERWISE SAVE SUM IN R2
        MOV      A,#0F7H  ; AND CALL EXIT SVC
        NOP      ; TO END PROGRAM EXECUTION
        MOVP     A,@A
        NOP
;SRB POINTER
ORG      40H      ;STORE SRB POINTER AT ADDRESS 40
        BYTE    00,42H    ;POINT TO SRB FOR EXIT SVC
;SRB FOR EXIT SVC
        BYTE    1AH      ;1AH = FUNCTION CODE FOR EXIT SVC
;TABLE OF NUMBERS TO BE ADDED
TSIZE   EQU      5      ;TABLE SIZE = 5
SECTION INTERNAL
ORG      32H      ;SET UP TABLE AT ADDRESS 32
TABLE   BLOCK    TSIZE
        LIST    DBG
        END     START   ;END OF SOURCE CODE
```

---

Assemble the Source Code

If you do not have a 8048/8021/8041A/8022 assembler on your system disk, you cannot perform this step, so skip the next four commands (ASM, COP, LINK, and L.)

The ASM (assemble) command translates assembly language (source code) into binary machine language (object code). The ASM command also creates an assembler listing that can be used to correlate the object code with the source code. Enter the following command line to assemble the source code in the file ASM and create the listing and object files ASML and OBJ:

```
> ASM OBJ ASML ASM <CR>
      |   |   |
      |   |   +--- source file
      |   |   +----- assembler listing file
      |   |   +----- object file
Tektronix   8048 ASM Vx.x
**** Pass 2
    26 Source Lines  26 Assembled Lines  xxxxx Bytes Available
      >>> No assembly errors detected <<<
```

Enter the following command to copy the assembler listing onto the line printer. (First, make sure the printer is turned on and properly connected.)

```
> COP ASML LPT <CR>
```

The different fields of your source listing are presented in Fig. 7H-10, earlier in this demonstration. For a detailed explanation of assembler listings, consult your Assembler Users Manual.

Link the Object Code. The linker creates an executable load file from one or more object files. Enter the LINK command to invoke the linker:

```
> LINK <CR>
8550 LINKER Vx.x
*
```

Now enter the following linker commands to create a load file called LOAD from your object file, OBJ:

```
*LINK OBJ <CR>
*LOAD LOAD <CR>
*LOCATE INTERNAL,BASE(4000H) <CR>
*DEBUG <CR>
*END <CR>
```



The linker commands LINK and LO specify the object file and load file, respectively. The LOCATE command establishes the base address for section INTERNAL. The DEBUG command causes the linker to pass the program symbols from the object file along to the load file, for use in program debugging. After you enter the END command, the linker executes the commands you have entered, and the following information is displayed:

TRUNCATION ERROR AT 0101 IN MODULE \*NONAME\*  
FILE OBJ

2 ERRORS NO UNDEFINED SYMBOLS  
1 MODULE 2 SECTIONS  
TRANSFER ADDRESS IS 0100

NOTE

Because the instruction MOV R1,#TABLE is a two-byte instruction, the linker truncates 4032 to 32, and displays an error message. Ignore this truncation error message.

The files generated by the ASM and LINK commands should now be on your disk. Enter the following command to list the files in your current directory:

> L <CR>

FILENAME

ASM  
LOAD  
OBJ  
ASML

Files used	126
Free files	130
Free blocks	811
Bad blocks	0

Notice that there are now four files listed in your directory. OBJ and ASML were created by the assembler, and LOAD was created by the linker.

---

Load the Program into Memory

Now it's time to load the object code from the load file LOAD into program memory.

Zero Out Memory. Before you load any code, use the F (Fill) command to fill 8550/8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no effect on how the program is loaded.) Enter the following command line to fill memory from address 40 through address 11F with zeros:

```
> F 40 11F 00 <CR>
```

Check That Memory Was Filled with Zeros. Check the contents of memory with the D (Dump) command. The D command's display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40--11F with the following command line:

```
> D 40 11F <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Load the Object Code into Memory. Enter the following command line to load the object code for the demonstration program into program memory:

```
> LO <LOAD <CR>
      ===
      |
      load file
```

-----  
Load the Program Symbols. Recall that the source code for the demonstration program contained the directive LIST DBG. Because of this directive, the object file contains a list of the symbols that appeared in the source code, and the values associated with those symbols. Because you included the DEBUG command when you invoked the linker, those symbols were passed to the load file. Use the SYMLO command to load those symbols into the symbol table in 8550 system memory.

> SYMLO -S <LOAD <CR>

The -S option means that both addresses and scalars are loaded. If you omit the -S, only addresses are loaded. (A scalar is a number that is not an address --- for example, TSIZE, the length of the table.)

Later in this demonstration, whenever you use a symbol in a command line, the operating system refers to the symbol table to find the value that the symbol represents.

You've assembled and linked the demonstration program and loaded it into memory. Now skip ahead to the heading "Run the Demonstration Program."

---

**CASE 2: ASSEMBLE ON THE 8560; DOWNLOAD TO THE 8540**

This discussion shows you how to create the demonstration program source code and assemble it on the 8560, then download it to 8540 (or 8550) program memory. If your 8560 does not have an 8048/8021/8041A/8022 assembler, you cannot complete this part of the demonstration, so skip ahead to the heading "Case 4: Patch the Program into Memory" for instructions.

Start Up and Log In

Start up your 8540, make sure it's in TERM mode, and log in to the 8560 operating system, TNIX. See your 8560 System Users Manual for details.

Since you're logged in to TNIX, your system prompt is "\$". (Later in the demonstration, we'll show the system prompt as ">", in deference to people using 8540s and 8550s in LOCAL mode.) Every command you enter is processed by TNIX. If you enter an OS/40 command, TNIX passes it to the 8540.

Enter the following command to select the 8048 assembler on the 8560 and the 8048 emulator on the 8540:

```
$ sel 8048 <CR>
$ uP=8048; export uP <CR>
```

The sel command automatically sets the emulation mode to 0.

Create the Demonstration Program

Enter the following TNIX command lines to create an empty directory called demo and make demo the working directory. You'll create your source file and related files in this demo directory.

```
$ mkdir demo <CR>
$ cd demo <CR>
```

Now use the TNIX editor, ed, to create the demonstration program source file. The following command line invokes the editor and specifies that you want to create a file called asm:

```
$ ed asm <CR>
?asm
```

The editor responds "?asm" to remind you that asm does not already exist. Notice that the editor does not give a prompt to let you know it's ready for input.

-----  
 Enter the Text. Now enter the editor command a (append text) and type in the program. Use the BACKSPACE key to erase typing mistakes.

```

a <CR>
      column  column  column
      8      16      26
      |      |      |
      v      v      v
;8048 DEMONSTRATION RUN PROGRAM <CR>
  SECTION DEMO <CR>
  ORG 100H ;START PROGRAM CODE AT ADDRESS 100 <CR>
START MOV R1,#TABLE ;SET TABLE POINTER <CR>
  MOV R7,#TSIZE ;SET PASS COUNTER <CR>
  CLR A ;CLEAR ACCUMULATOR <CR>
LOOP  ADD A,@R1 ;ADD BYTE FROM TABLE <CR>
  INC R1 ;POINT TO NEXT BYTE <CR>
  DJNZ R7,LOOP ;DECREMENT PASS COUNTER AND <CR>
  ; LOOP IF NOT FIVE PASSES YET <CR>
  MOV R2,A ;OTHERWISE SAVE SUM IN R2 <CR>
  MOV A,#0F7H ; AND CALL EXIT SVC <CR>
  NOP ; TO END PROGRAM EXECUTION <CR>
  MOVP A,@A <CR>
  NOP <CR>
;SRB POINTER <CR>
  ORG 40H ;STORE SRB POINTER AT ADDRESS 40 <CR>
  BYTE 00,42H ;POINT TO SRB FOR EXIT SVC <CR>
;SRB FOR EXIT SVC <CR>
  BYTE 1AH ;1AH = FUNCTION CODE FOR EXIT SVC <CR>
;TABLE OF NUMBERS TO BE ADDED <CR>
TSIZE EQU 5 ;TABLE SIZE = 5 <CR>
  SECTION INTERNAL <CR>
  ORG 32H ;SET UP TABLE AT ADDRESS 32 <CR>
TABLE BLOCK TSIZE <CR>
  LIST DBG <CR>
  END START ;END OF SOURCE CODE <CR>
. <CR>

```

At the end of your text, enter a period on a line by itself. The editor will go back to accepting commands.

Check for Errors. Enter the following editor command to display the text you have entered. Check for typing mistakes.

```

1,$p <CR>
| |
| |+- print command: displays the lines
| | in the designated range
| |
| |
| +--- designates last line in file
|
+----- designates first line in file

```

If you made any mistakes, go ahead and fix them. In case you're not familiar with ed, Table 7H-10 lists the commands you need in order to add, delete, or replace any line. For more information on ed, refer to your 8560 System Users Manual.

-----  
 Table 7H-10  
 Basic 8560 Editing Commands  
 -----

Command	Function
mm,nnp <CR>	Displays lines mm through nn
nn <CR>	Makes line nn the current line
d <CR>	Deletes the current line
a <CR> <line(s) of text> . <CR>	Adds text after the current line
c <CR> <line(s) of text> . <CR>	Replaces the current line with the text you type in

Once your text is correct, enter the w command to write the text to the source file, asm:

```
w <CR>
778
```

The editor responds with the number of characters it wrote to the file.

Finally, enter the q command to quit the editor and return to TNIX:

```
q <CR>
$ <--- TNIX prompt
```

#### Assemble the Source Code.

The TNIX asm (assemble) command translates assembly language (source code) into binary machine language (object code). The asm command also creates an assembler listing which can be used to correlate the object code with the source code. Enter the following command line to assemble the source code in the file asm and create the listing and object files asml and obj:

```
$ asm obj asml asm <CR>
  |   |   |
  |   |   +-- source file
  |   |
  |   +----- assembler listing file
  |
  +----- object file
ASM MCS 8048 Vxx.xx-xx Copyright 198x Tektronix, Inc.
*****Pass 2
```

```
27 Lines Read
27 Lines Processed
0 Errors
```

-----  
 Enter the following command to print the assembler listing on the 8560's line printer:

```
$ lp1r asml <CR>
```

Check page 1 of your listing. Did the assembler issue any error messages? There should be none. If your source code contains errors, take the following steps:

1. Refer to your Assembler Users Manual to find out what the error messages mean.
2. Enter the command ed asm to get back into the editor and fix the mistakes in your source code. Exit the editor with the w and q commands, as before.
3. Enter the command asm obj asml asm to re-assemble your source code.

### Link the Object Code

The linker creates an executable load file from one or more object files. Enter the following command to create a load file called load from your object file, obj. Be sure to capitalize all parameters as shown.

```
$ link -d -O obj -o load -m prog=0-120h -m int=4000-4037h -L sec=DEMO range prog -L sec=INTERNAL range int <CR>
```

### NOTE

To simplify your task, you may want to create a linker command file to hold the command options. To do so, invoke ed, and place each command option on a separate line. Then, to execute the command file, you would enter:  
\$ link -c commandfilename.

The -d option causes the linker to pass the program symbols from the object file to the load file, for use in program debugging. The -m option defines the memory map. In this case, the areas mapped are in program memory and internal data memory. The -L option locates the two sections DEMO and INTERNAL.

The linker responds with the following truncation error message. The truncation errors are caused by the MOV R1,#TABLE and DJNZ R7,LOOP instructions. These instructions each expect an 8-bit operand. You may ignore these messages.

```
link:115 (E) Truncation error at      101
link:115 (E) Truncation error at      108
```

-----  
 The files generated by the asm and link commands should now be in your working directory, demo. Enter the following command to list the files in your working directory:

```
$ ls <CR>
asm
asml
load
obj
```

Notice that there are now four files listed in your directory. obj and asml were created by the assembler, and load was created by the linker.

#### Download the Program to the 8540

Now it's time to download the object code produced by the 8560's linker into 8540 program memory.

Zero Out Memory. Before you download any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no effect on how the program is loaded.) Enter the following command line to fill memory from address 40 through address 11F with zeros:

```
$ f 40 11f 00 <CR>
```

Check That Memory Was Filled with Zeros. Check the contents of memory with the OS/40 D (Dump) command. The D command's display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40--11F with the following command line:

```
$ d 40 11f <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```



-----  
Download the Object Code. Enter the following command line to download the object code from the 8560 file load to 8540 program memory:

```
$ lo <load <CR>
    ====
    |
    load file
```

Download the Program Symbols. Recall that the source code for the demonstration program contains the directive LIST DBG. Because of this directive, the object file contains a list of the symbols that appear in the source code, and the values associated with those symbols. Because you included the -d option in the link command line, those symbols were passed to the load file. Use the OS/40 SYMLO command to download those symbols into the symbol table in 8540 system memory.

```
$ symlo -s <load <CR>
```

The -S option means that both addresses and scalars are downloaded. If you omit the -S, only addresses are downloaded. (A scalar is a number that is not an address --- for example, TSIZE, the length of the table.)

Later in this demonstration, whenever you use a symbol in an OS/40 command line, OS/40 refers to the symbol table to find the value that the symbol stands for.

You've assembled and linked the demonstration program and downloaded it into memory. Now skip ahead to the heading "Run the Demonstration Program."

---

**CASE 3: DOWNLOAD FROM YOUR HOST TO THE 8540**

This discussion gives some general instructions for downloading the demonstration program from an unspecified host computer to 8540 (or 8550) program memory. If your 8540 is not equipped with the optional COM Interface Package, you cannot complete this part of the demonstration, so skip ahead to the heading "Case 4: Patch the Program into Memory" for instructions. COM Interface software is standard on the 8550.

Since we don't know what host computer you're using, we can only provide a general outline for creating the demonstration program and downloading it to the 8540. Once you have determined the command sequence that is appropriate for your host, record this information in the space provided in Fig. 7H-12.

Create the Extended Tekhex Load Module

In order for the object code to be downloaded to the 8540, it must be in Extended Tekhex format, as shown in Fig. 7H-11, earlier in this demonstration. You can create the load module in one of two ways:

1. Use your host computer's text editor, and key the load module in by hand.
2. Use your host computer's 8048 assembler:
  - a. Translate the demonstration program into the language of your host's 8048 assembler.
  - b. Create and assemble the source file.
  - c. Link the object code, if necessary.
  - d. Translate the object code produced by the assembler or linker into Extended Tekhex format. The Intersystem Communication section of your System Users Manual provides a general algorithm for conversion to Extended Tekhex format.

Prepare the 8540

Start up your 8540 and enter the following command to select the 8048 emulator:

```
> SEL 8048 <CR>
```

The SEL command automatically sets the emulation mode to 0.

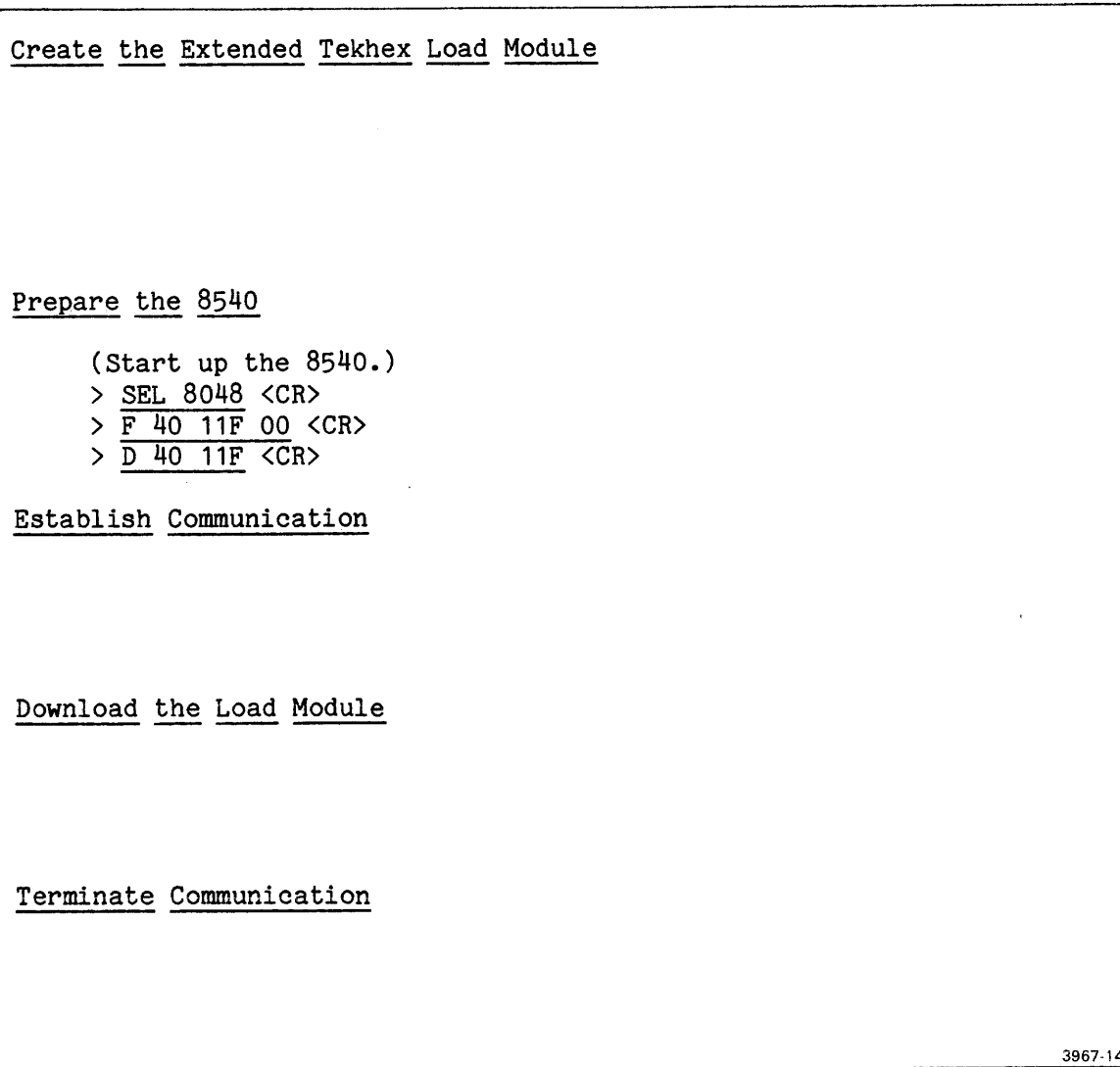


Fig. 7H-12 Host computer commands for preparing demonstration program

Zero Out Memory. Before you download any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no effect on how the program is loaded.) Enter the following command line to fill memory from address 40 through address 11F with zeros:

```
> F 40 11F 00 <CR>
```

-----  
Check That Memory Was Filled with Zeros. Check the contents of memory with the OS/40 D (Dump) command. The D command's display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40--11F with the following command line:

```
> D 40 11F <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

#### Download the Load Module to the 8540

Be sure that your 8540 and your host computer are connected via an RS-232-C-compatible communications link. Then perform the following steps to download the Tekhex load module to 8540 program memory. (Refer to the Intersystem Communication section of your System Users Manual to determine the commands and parameters that are appropriate for your host computer.)

- a. Enter the 8540 COM command to establish communication. (The parameters of the COM command are host-specific.) Log on to your host and execute any necessary host initialization commands.
- b. Enter the command line that downloads the Tekhex load module to the 8540. This command line consists of a host computer command that performs the download, followed by a null character (CTRL-@ on most terminals) and a carriage return. COM places the object code in 8540 program memory, and puts the program symbols into the symbol table in 8540 system memory.
- c. Log off from your host, and then terminate COM command execution by entering the null character, then pressing the ESC key.

Once you've downloaded the program to the 8540, skip ahead to the heading "Run the Demonstration Program."

-----  
CASE 4: PATCH THE PROGRAM INTO MEMORY

This discussion shows you how to patch the demonstration program into 8540 (or 8550) program memory using the P command, and then add the program symbols into the symbol table using the ADDS command.

Ordinarily, you would load the object code and symbols from a binary or hexadecimal load file, as illustrated for Cases 1, 2, and 3. The procedure presented here is not normally used for preparing a program for execution. Use this procedure only if you have no standard means for preparing the program, but would still like to try out your emulator.

Start Up the 8540

Start up your 8540 and enter the following command to select the 8048 emulator:

```
> SEL 8048 <CR>
```

The SEL command automatically sets the emulation mode to 0.

Zero Out Memory

Before you patch in any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. Enter the following command line to fill memory from address 40 through address 11F with zeros:

```
> F 40 11F 00 <CR>
```

Check That Memory Was Filled with Zeros. Check the contents of memory with the OS/40 D (Dump) command. The D command's display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40--11F with the following command line:

```
> D 40 11F <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Patch the Object Code into Memory

The OS/40 P (Patch) command stores a sequence of bytes into memory, replacing the previous memory contents. Enter the following command to store the object code for the first three instructions in the program (MOV R1, MOV R7, and CLR A) starting at location 100:

```
> P 100 B932 BF05 27 <CR>
    === ===== ===== ==
    |           |           |           |
    |           |           |           | CLR A
    |           |           |           |
    |           |           |           | MOV R7,#TSIZE
    |           |           |           |
    |           |           |           | MOV R1,#TABLE
    |           |           |           |
patch address
```

Now patch in the next four instructions (ADD, INC, DJNZ, and MOV R2)...

```
> P 105 61 19 EF05 AA <CR>
```

... and now the last four instructions (MOV A, NOP, MOVP A, and NOP).

```
> P 10A 23F7 00 A3 00 <CR>
```

Finally, patch in the Exit SVC information at address 40:

```
> P 40 00421A <CR>
```

You'll check the contents of memory later in this demonstration.

Put Symbols into the Symbol Table

Later in this demonstration, you will use symbols from the demonstration program (START, LOOP, TSIZE, and TABLE) when communicating with OS/40. Whenever you use a symbol in a command line, OS/40 consults a symbol table in 8540 system memory to find the values that the symbol stands for. Enter the following command line to add the program symbols to the symbol table, along with their values:

```
> ADDS START=100 LOOP=106 -S TSIZE=5 TABLE=4032 <CR>
```

The ADDS command cannot provide all the symbol-related information that is provided by the SYMLO command (as in Cases 1 and 2) or the COM command (as in Case 3). Because this information is missing, some of the displays you produce later in this demonstration will not match the symbolic displays shown in this manual. For more information on the ADDS command, refer to the Command Dictionary of your System Users Manual.

You've patched the demonstration program into program memory and placed the program symbols in the symbol table. Now it's time to run the program.

-----  
 RUN THE DEMONSTRATION PROGRAM

From now until the end of the demonstration, the commands you are to enter are shown in lowercase. If you are not logged in to an 8560, you may enter commands in either lowercase or uppercase. If you are using an 8560, you must enter the name of every command in lowercase (and your system prompt is "\$", not ">").

Now that you've loaded the program into memory, you need to:

- verify that the program was loaded correctly; and
- put values into the table in memory, for the program to add.

Check Memory Contents Again. Before you loaded the program, you filled memory locations 40--11F with zeros. Look at the same memory area again with the following command line:

```
> d 40 11f <CR>
   0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0040 00 42 1A 00 00 00 00 00 00 00 00 00 00 00 00 .B.....
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0100 B9 32 BF 05 27 61 19 EF 05 AA 23 F7 00 A3 00 00 .2..'a....#.....
0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

The object code is loaded in two different blocks:

- The 8048 machine instructions are loaded at address 100 (specified by the first ORG directive in the source code).
- The information for the Exit SVC is loaded at address 40 (specified by the second ORG directive).

The contents of the table at address 32 are still undefined, but you'll put some values into the table in just a few minutes.

Turn On Symbolic Display. Enter the following command to tell the system to modify its displays by replacing hexadecimal numbers with symbols from your program, where appropriate.

```
> synd on <CR>
```

-----  
Disassemble the Object Code. The DI (DISassemble) command displays memory contents both in hexadecimal notation and in assembly language mnemonics. You can use the DI command to verify that the object code in memory corresponds to your source code. Enter the following command to disassemble the area of memory occupied by the executable part of your program:

```
> di 100 10e <CR>
```

LOC	INST	MNEM	OPER
SECTION (DEMO)			
START	B932	MOV	R1,#32
+000102	BF05	MOV	R7,#05
+000104	27	CLR	A
LOOP	61	ADD	A,@R1
+000106	19	INC	R1
+000107	EF05	DJNZ	R7,#05 0105
+000109	AA	MOV	R2,A
+00010A	23F7	MOV	A,#F7
+00010C	00	NOP	
+00010D	A3	MOVP	A,@A
+00010E	00	NOP	
+00010E	12	NOP	

Compare the DI display with the assembler listing you generated earlier, or refer back to Fig. 7H-10.

The line "SECTION (DEMO)" in the DI display indicates that the object code being disassembled comes from the program section called DEMO. In fact, the entire memory area used by your program (location 0 through the end of the program --- location 10E) belongs to section DEMO. This section was declared by the SECTION directive in the source code. (If you used the ADDS command to create your symbols, as in Case 4, the section name shown in the DI display is NO.SECTION.)

The LOC (location) column of the DI display contains information that enables you to correlate the display with your assembler listing. The symbols START and LOOP in the DI display correspond to the labels START and LOOP in the source code. For those lines of the display where the location does not correspond to a label in the symbol table, DI substitutes the address of the instruction relative to the beginning of the section, as shown in the address field of your assembler listing. If you haven't loaded the pertinent symbols and related information into the symbol table (using a command such as SYML0), the DI command supplies absolute (actual) addresses in the LOC column. (Since section DEMO begins at address 0, the relative address, or offset, is the same as the absolute address in this display. This offset feature is much more useful for sections that don't start at address 0.)

Now you've seen that your system can use the symbol table to translate numbers into symbols to make a display easier to read. Your system can also translate a symbol in a command line into an address. For example, since your system knows that the symbol START is equivalent to the address 100, you could have entered the DI command in any of the following ways:



```

di 100 10E
di START 10E
di start start+0e
di 100 START+0E

```

Notice that a symbol can be entered in either lowercase or uppercase.

The feature that enables DOS/50 and OS/40 to correlate symbols from your program with the numbers they represent is termed symbolic debug.

Put Values into the Table in Memory. The demonstration program sums five numbers from a table in memory. Use the P (Patch) command to store the numbers 1, 2, 3, 4, and 5 in the table. Do you remember what the address of the table is? It doesn't matter, as long as you remember that the symbol TABLE represents that address.

```

> p table 0102030405 <CR>
=====
|           |
address of   string of bytes to be stored
table: 4032  at addresses 4032--4036

```

Check the Contents of the Table. Use the D command to display the contents of the table. (When you don't specify an upper boundary for the area to be dumped, the D command dumps 16 bytes.)

```

+----- lower address: 4032
|
|   +-- upper address: omitted
|   |   (defaults to lower address + 0F)
|   |
|   |
===== =
> d table <CR>
  2 3 4 5 6 7 8 9 A B C D E F 0 1
004032 01 02 03 04 05 27 EB 8F C3 3C EB B6 9D 2B 00 42 ...'...<...+...B

```

Notice that bytes 4032--4036 (the table) contain the values you patched in. Bytes 4037--4041 contain random data left over from previous system operations.

The following command dumps only the contents of the table:

```

> d table table+tsize-1 <CR>
  2 3 4 5 6 7 8 9 A B C D E F 0 1
004032 01 02 03 04 05
.....

```

---

Start Program Execution

Enter the G (Go) command to start program execution at location 100, the transfer address specified by the END directive in the source code. (If you followed "Case 4: Patch the Program into Memory," you must enter G START instead.)

> g <CR>

				accumulator		table pointer		sum of table					pass counter				
				v	v	v	v	v	v	v	v	v	v	v			
LOC	INST	MNEM	OPERAND	EADD	A	PSW	FLAGS	TR	RB	R0	R1	R2	R3	R4	R5	R6	R7
010D	A3	MOVP	A,@A		00	08	00	00	0	19	37	0F	A6	FD	00	BE	00
010D	<BREAK		TRACE>														

The program executes, and when the Exit SVC occurs, the program breaks (stops), and the contents of the emulator registers are displayed. The Exit SVC uses the accumulator, so the sum of the numbers in the table is transferred into R2: 1+2+3+4+5=0F.

#### MONITOR PROGRAM EXECUTION

You have assembled, loaded, and executed the demonstration program. The rest of this demonstration shows you some commands for monitoring program execution. You can watch the changes in the emulator's registers and observe the effect of each instruction as the program proceeds.

Trace All Instructions. The TRA (TRAcE) command lets you observe the changes in the 8048 registers as the program proceeds. When you enter a TRA command and then start execution with the G command, display lines are sent to the system terminal. As each instruction executes, the display line shows the instruction (as in the DISassemble display) and the contents of the registers after that instruction has executed. Enter the following command line to trace all of the program's instructions:

> tra all <CR>

-----  
 Enter the command G START (or G 100) to resume program execution at the beginning of the program:

> g start <CR>

As the program executes, the following trace is displayed. Remember that you can type CTRL-S to suspend the display and CTRL-Q to resume the display.

LOC	INST	MNEM	OPERAND	EADD	A	PSW	FLAGS	TR	RB	RO	R1	R2	R3	R4	R5	R6	R7	
SECTION (DEMO)																		
START	B932	MOV	R1,#32H		07	08	00	00	00	0	19	32	07	A6	FD	00	BE	00
+000102	BF05	MOV	R7,#05H		07	08	00	00	00	0	19	32	07	A6	FD	00	BE	05
+000104	27	CLR	A		00	08	00	00	00	0	19	32	07	A6	FD	00	BE	05
LOOP	61	ADD	A,@R1		01	08	00	00	00	0	19	32	07	A6	FD	00	BE	05
+000106	19	INC	R1		01	08	00	00	00	0	19	33	07	A6	FD	00	BE	05
+000107	EF05	DJNZ	R7,#05H	0105	01	08	00	00	00	0	19	33	07	A6	FD	00	BE	04
LOOP	61	ADD	A,@R1		03	08	00	00	00	0	19	33	07	A6	FD	00	BE	04
+000106	19	INC	R1		03	08	00	00	00	0	19	34	07	A6	FD	00	BE	04
+000107	EF05	DJNZ	R7,#05H	0105	03	08	00	00	00	0	19	34	07	A6	FD	00	BE	03
LOOP	61	ADD	A,@R1		06	08	00	00	00	0	19	34	07	A6	FD	00	BE	03
+000106	19	INC	R1		06	08	00	00	00	0	19	35	07	A6	FD	00	BE	03
+000107	EF05	DJNZ	R7,#05H	0105	06	08	00	00	00	0	19	35	07	A6	FD	00	BE	02
LOOP	61	ADD	A,@R1		0A	08	00	00	00	0	19	35	07	A6	FD	00	BE	02
+000106	19	INC	R1		0A	08	00	00	00	0	19	36	07	A6	FD	00	BE	02
+000107	EF05	DJNZ	R7,#05H	0105	0A	08	00	00	00	0	19	36	07	A6	FD	00	BE	01
LOOP	61	ADD	A,@R1		0F	08	00	00	00	0	19	36	07	A6	FD	00	BE	01
+000106	19	INC	R1		0F	08	00	00	00	0	19	37	07	A6	FD	00	BE	01
+000107	EF05	DJNZ	R7,#05H	0105	0F	08	00	00	00	0	19	37	07	A6	FD	00	BE	00
+000109	AA	MOV	R2,A		0F	08	00	00	00	0	19	37	0F	A6	FD	00	BE	00
+00010A	23F7	MOV	A,#F7H		F7	08	00	00	00	0	19	37	0F	A6	FD	00	BE	00
+00010C	00	NOP			F7	08	00	00	00	0	19	37	0F	A6	FD	00	BE	00
+00010D	A3	MOV	A,@A		00	08	00	00	00	0	19	37	0F	A6	FD	00	BE	0C
+00010D	<BREAK		TRACE>															

After the accumulator is cleared, the program begins to store the sum of the numbers being added. The 8048 ADD A,@R1 instruction adds a number from the table into the accumulator. Because the accumulator is also used for the Exit SVC, the program transfers the sum of numbers (1+2+3+4+5=0F) from the accumulator into R2.

Register R7, the pass counter, is set to contain 5 at the beginning of the program. It decreases by one (because of the DJNZ instruction) each time a number is added into the accumulator. The program ends after register R7 reaches zero.

The R1 register, set to contain 32 at the start of the program, increments (because of the INC R1 instruction) each time a number is added to the accumulator. At the end of the program, the R1 register has been incremented five times and contains 37.

Trace to the Line Printer. By adding the parameter >LPT to a command, you can direct that command's output to the line printer instead of to the system terminal. First, verify that your line printer is properly connected and powered up. Then enter the following command to execute the program with trace output directed to the line printer:

```
> g start >LPT <CR>
```

#### NOTE

If you're operating in TERM mode with an 8560, use one of the following commands in place of the command shown:

- g start | lp1r sends the display to the 8560 line printer.
- g start \>LPT sends the display to the line printer on the 8540 or 8550.

Trace Jump Instructions Only. Another way to monitor the program's execution is to look only at the jump instructions. By tracing the jump instructions, you can still observe the changes in the registers, but you save time and space by not tracing the instructions within the loop. Enter the following command line to trace only the jump instructions when the loop is being executed:

```
> tra jmp loop 108 <CR>
     |
     |
     | +--- upper address } Within this range,
     |                       } only jump instructions
     |
     | +----- lower address } are traced.
     |
     | (105)
```

Again, enter the G command line to start program execution. The following trace is displayed:

```
> g start <CR>

LOC      INST MNEM  OPERAND EADD A PSW FLAGS TR RB RO R1 R2 R3 R4 R5 R6 R7
SECTION (DEMO)
START   B932 MOV   R1,#32H      07 08 00 00 00  0 19 32 07 A6 FD 00 BE 00
+000103 61  ADD   A,@R1      01 08 00 00 00  0 19 32 07 A6 FD 00 BE 05
+000105 19  INC   R1          01 08 00 00 00  0 19 33 07 A6 FD 00 BE 05
+000107 EF05 DJNZ  R7,#05H 0105 01 08 00 00 00  0 19 33 07 A6 FD 00 BE 04
+000107 EF05 DJNZ  R7,#05H 0105 03 08 00 00 00  0 19 34 07 A6 FD 00 BE 03
+000107 EF05 DJNZ  R7,#05H 0105 06 08 00 00 00  0 19 35 07 A6 FD 00 BE 02
+000107 EF05 DJNZ  R7,#05H 0105 0A 08 00 00 00  0 19 36 07 A6 FD 00 BE 01
+000107 EF05 DJNZ  R7,#05H 0105 0F 08 00 00 00  0 19 37 07 A6 FD 00 BE 00
+000109 AA  MOV   R2,A          0F 08 00 00 00  0 19 37 0F A6 FD 00 BE 00
+000109 <BREAK  TRACE>
```

As with the TRA ALL display, observe that register R7 (the pass counter)

is decremented. The R1 register (the table pointer) is incremented, and the accumulator stores the sum of the numbers from the table. With the TRA JMP command in effect, the instructions within the loop are not displayed.

Check the Status of the Trace. The TRA command without any parameters displays the trace conditions that are presently set. Because you can have up to three trace selections in effect at the same time, it is useful to be able to see which trace selections are active. Check your trace status with the following command line:

```
> tra <CR>

TRACE   ALL,000000,00FFFF
TRACE   JMP,LOOP,0108
```

As you've specified, TRA ALL is in effect for addresses 0--104, TRA JMP is in effect for addresses 105--108, and TRA ALL is again in effect for addresses 109--FFFF.

Set a Breakpoint after a Specific Instruction. Now that you've seen how the program adds the numbers together, here's a new task: to add only the third and fourth numbers from the table. To perform this task, you want the pass counter to contain 2, and the table pointer to contain 34 (the address of the third number in the table). You can accomplish these changes without altering the object code in program memory. First, stop program execution after the pass counter and the table pointer have been set. While the program is stopped, enter new values for the pass counter and the table pointer. When execution resumes, the program treats the new values as if they were the original programmed values.

Enter the following command line to trace all of the instructions as the program executes:

```
> tra all <CR>
```

Check the trace status with the following command line:

```
> tra <CR>

TRACE   ALL,0000,FFFF
```

The TRA selections we set earlier are made obsolete by the TRA ALL command just entered.

Now set a breakpoint so that the program stops after the table pointer and pass counter have been set. The following command causes the program to stop after it executes the MOV R7 instruction at address 102:

```
> bk 1 102 <CR>
   = ==
   |  |
   |  +--- breakpoint address
   |
   +----- breakpoint number
```

(can be 1 or 2)

Use the G command to start program execution:

```
> g start <CR>
```

```
LOC      INST MNEM  OPERAND EADD A PSW FLAGS TR RB R0 R1 R2 R3 R4 R5 R6 R7
SECTION (DEMO)
START   B932 MOV   R1,#32H      07 08 00 00 00 0 19 32 07 A6 FD 00 BE 00
+000102 BF05 MOV   R7,#05H      07 08 00 00 00 0 19 32 07 A6 FD 00 BE 05
+000102 <BREAK  TRACE, BKPT1>
```

The TRA ALL command enabled display of all instructions up to and including the instruction at the breakpoint.

Set New Values in Pass Counter and Table Pointer; Check Results. Now that you've reached the breakpoint, you can change the contents of the registers while execution is stopped. The break display shows that R7 (the pass counter) contains 5, and that the R1 register (the table pointer) points to address 32. Use the S (Set) command to set the number of passes to two and set the table pointer to 34:

```
> s r1=34 r7=02 <CR>
      |      |
      |      +--- contents of register R7
      |
      +----- contents of register R1
```

The S command does not produce a display, but you can use the DS (Display Status) command to check the values in the registers you changed. DS displays the contents of each emulator register and status flag. Check the result of the previous S command with the following command line:

```
> ds <CR>
```

```
PC  CHIP EPM  A PSW TR RB          REGISTERS
0104 8048 Y  OF  08 00 0    R0-R7= 19 34 OF A6 FD 00 BE 02
                                A0-A7= 86 00 2F 40 49 09 7D 00

IIP EI TI TC MB STACK RETURN TF FO F1 AN STF DMA IBF OBF FLG
N  0  0  0  0  0  08  1F06  0  0  0
```

The DS display shows that the pass counter and the table pointer now contain the new values.

#### NOTE

The DS display varies slightly for different microcomputers in the 8048 family.

-----  
Resume Program Execution. If you enter the G command with no parameters, program execution starts where it left off. Resume program execution after the breakpoint with the following command line:

> g <CR>

LOC	INST	MNEM	OPERAND	EADD	A	PSW	FLAGS	TR	RB	RO	R1	R2	R3	R4	R5	R6	R7		
SECTION (DEMO)																			
+000104	27	CLR	A			00	08	00	00	00	0	19	34	0F	A6	FD	00	BE	02
LOOP	61	ADD	A,@R1			01	08	00	00	00	0	19	34	0F	A6	FD	00	BE	02
+000106	19	INC	R1			01	08	00	00	00	0	19	35	0F	A6	FD	00	BE	02
+000107	EF05	DJNZ	R7,#05H	0105		01	08	00	00	00	0	19	35	0F	A6	FD	00	BE	01
LOOP	61	ADD	A,@R1			03	08	00	00	00	0	19	35	0F	A6	FD	00	BE	01
+000106	19	INC	R1			03	08	00	00	00	0	19	36	0F	A6	FD	00	BE	01
+000107	EF05	DJNZ	R7,#05H	0105		03	08	00	00	00	0	19	36	0F	A6	FD	00	BE	00
+000109	AA	MOV	R2,A			0F	08	00	00	00	0	19	36	07	A6	FD	00	BE	00
+00010A	23F7	MOV	A,#F7H			F7	08	00	00	00	0	19	36	F7	A6	FD	00	BE	00
+00010C	00	NOP				F7	08	00	00	00	0	19	36	07	A6	FD	00	BE	00
+00010D	A3	MOVP	A,@A			00	08	00	00	00	0	19	36	07	A6	FD	00	BE	00
+00010D	<BREAK		TRACE>																

Notice that the program performed two passes through the loop, and that the program added the third and fourth numbers in the table (3 + 4 = 7).

#### SUMMARY OF 8048/8021/8041A/8022 EMULATOR DEMONSTRATION RUN

You have assembled, loaded, executed, and monitored the demonstration run program. You have used the following commands:

- SEL -- selects the 8048 assembler and emulator
- ASM -- creates object code from an assembly language program
- LINK -- links object code into a load module
- F -- fills an area of memory with a specified value
- D -- displays memory contents in ASCII and hexadecimal format
- LO -- loads object code into memory
- DI -- translates memory contents into assembly language mnemonics
- P -- patches a string of bytes into memory
- SYMD ON -- turns on symbolic debug feature
- G -- begins or resumes program execution
- TRA -- selects instructions to be traced during program execution
- BK -- sets a breakpoint
- S -- modifies emulator processor registers
- DS -- displays the status of the emulator processor

---

### Delete the Demonstration Run Files

Now that you've finished the demonstration run, you can delete the source file, object file, listing file, and load file. If you're using an 8550, the source and object files are still available to you on the 8048/8021/8041A/8022 emulator installation disk. If you're using an 8560, remember that once you delete the source file (asm), there is no way of recovering it.

Delete 8550 Files. If your files are on the 8550, use the following procedure to delete them. First use the USER command to move from the DEMO directory back into the directory you were in at the start of the demonstration. Recall that you marked that directory with the brief name /ROOT.

```
> USER /ROOT <CR>
```

Now enter the following command to delete the DEMO directory and the files it contains:

```
> DEL /DEMO/* DEMO <CR>
```

```
Delete ASM          ? Y <CR>
Delete LOAD         ? Y <CR>
Delete OBJ          ? Y <CR>
Delete ASML         ? Y <CR>
Delete DEMO         ? Y <CR>
```

Before deleting each file, DOS/50 asks you whether you really want to delete it. You type "Y" for yes.

Delete 8560 Files. If your files are on the 8560, use the following procedure to delete them. Enter the following command to remove all files in the working directory, including the source file:

```
$ rm * <CR>
```

Now move from the demo directory back into the parent directory and remove the demo directory itself:

```
$ cd .. <CR>
$ rmdir demo <CR>
```

### Turn Off Your System

For instructions on turning off your 8550 or 8540, refer to the Learning Guide of your System Users Manual.



-----  
Resume Program Execution. If you enter the G command with no parameters, program execution starts where it left off. Resume program execution after the breakpoint with the following command line:

> g <CR>

LOC	INST	MNEM	OPERAND	EADD	A	PSW	FLAGS	TR	RB	RO	R1	R2	R3	R4	R5	R6	R7
SECTION (DEMO)																	
+000104	27	CLR	A		00	08	00 00 00	00	0	19	34	0F	A6	FD	00	BE	02
LOOP	61	ADD	A,@R1		01	08	00 00 00	00	0	19	34	0F	A6	FD	00	BE	02
+000106	19	INC	R1		01	08	00 00 00	00	0	19	35	0F	A6	FD	00	BE	02
+000107	EF05	DJNZ	R7,#05H	0105	01	08	00 00 00	00	0	19	35	0F	A6	FD	00	BE	01
LOOP	61	ADD	A,@R1		03	08	00 00 00	00	0	19	35	0F	A6	FD	00	BE	01
+000106	19	INC	R1		03	08	00 00 00	00	0	19	36	0F	A6	FD	00	BE	01
+000107	EF05	DJNZ	R7,#05H	0105	03	08	00 00 00	00	0	19	36	0F	A6	FD	00	BE	00
+000109	AA	MOV	R2,A		0F	08	00 00 00	00	0	19	36	07	A6	FD	00	BE	00
+00010A	23F7	MOV	A,#F7H		F7	08	00 00 00	00	0	19	36	F7	A6	FD	00	BE	00
+00010C	00	NOP			F7	08	00 00 00	00	0	19	36	07	A6	FD	00	BE	00
+00010D	A3	MOVP	A,@A		00	08	00 00 00	00	0	19	36	07	A6	FD	00	BE	00
+00010D	<BREAK		TRACE>														

Notice that the program performed two passes through the loop, and that the program added the third and fourth numbers in the table (3 + 4 = 7).

#### SUMMARY OF 8048/8021/8041A/8022 EMULATOR DEMONSTRATION RUN

You have assembled, loaded, executed, and monitored the demonstration run program. You have used the following commands:

- SEL -- selects the 8048 assembler and emulator
- ASM -- creates object code from an assembly language program
- LINK -- links object code into a load module
- F -- fills an area of memory with a specified value
- D -- displays memory contents in ASCII and hexadecimal format
- LO -- loads object code into memory
- DI -- translates memory contents into assembly language mnemonics
- P -- patches a string of bytes into memory
- SYMD ON -- turns on symbolic debug feature
- G -- begins or resumes program execution
- TRA -- selects instructions to be traced during program execution
- BK -- sets a breakpoint
- S -- modifies emulator processor registers
- DS -- displays the status of the emulator processor

---

Delete the Demonstration Run Files

Now that you've finished the demonstration run, you can delete the source file, object file, listing file, and load file. If you're using an 8550, the source and object files are still available to you on the 8048/8021/8041A/8022 emulator installation disk. If you're using an 8560, remember that once you delete the source file (asm), there is no way of recovering it.

Delete 8550 Files. If your files are on the 8550, use the following procedure to delete them. First use the USER command to move from the DEMO directory back into the directory you were in at the start of the demonstration. Recall that you marked that directory with the brief name /ROOT.

```
> USER /ROOT <CR>
```

Now enter the following command to delete the DEMO directory and the files it contains:

```
> DEL /DEMO/* DEMO <CR>
```

```
Delete ASM           ? Y <CR>
Delete LOAD          ? Y <CR>
Delete OBJ           ? Y <CR>
Delete ASML          ? Y <CR>
Delete DEMO          ? Y <CR>
```

Before deleting each file, DOS/50 asks you whether you really want to delete it. You type "Y" for yes.

Delete 8560 Files. If your files are on the 8560, use the following procedure to delete them. Enter the following command to remove all files in the working directory, including the source file:

```
$ rm * <CR>
```

Now move from the demo directory back into the parent directory and remove the demo directory itself:

```
$ cd .. <CR>
$ rmdir demo <CR>
```

Turn Off Your System

For instructions on turning off your 8550 or 8540, refer to the Learning Guide of your System Users Manual.

## Section 8

# PROM PROGRAMMER SPECIFICS

This section describes the general use of the PROM Programmer, and provides a place to insert your PROM Programmer Specifics supplements.

### USING THE PROM PROGRAMMER

The 8500 MDL Series PROM Programmer consists of three assemblies:

1. The **controller** is a circuit board that is installed in the 8540 mainframe.
2. The **front panel assembly** is installed in the 8540 front panel.
3. The **characteristic module** is a circuit card that you insert into the front panel assembly in order to configure your PROM Programmer for a particular set of PROM devices.

Following are step-by-step procedures for inserting and using a characteristic module. It is assumed that the PROM Programmer Controller, front panel assembly, and operating system ROM have been installed according to the procedures in the PROM Programmer Controller Installation Manual. The steps within each procedure must be performed in the specified order.

#### CAUTION

*Do not start up or turn off the 8540 when you have a PROM device inserted in the zero-insertion-force (ZIF) socket of the characteristic module. Voltage transients may damage your PROM.*

#### NOTE

*If you are using a multi-board emulator, refer to your Emulator Specifics supplement for more information on using your PROM Programmer.*

### Inserting a Characteristic Module

Slide the characteristic module upward into the front panel assembly until the front panel of the characteristic module fits snugly against the front panel assembly. See Fig. 8-1 for the correct angle of insertion. When the characteristic module is properly inserted, the ZIF socket on the characteristic module protrudes from the front panel assembly.

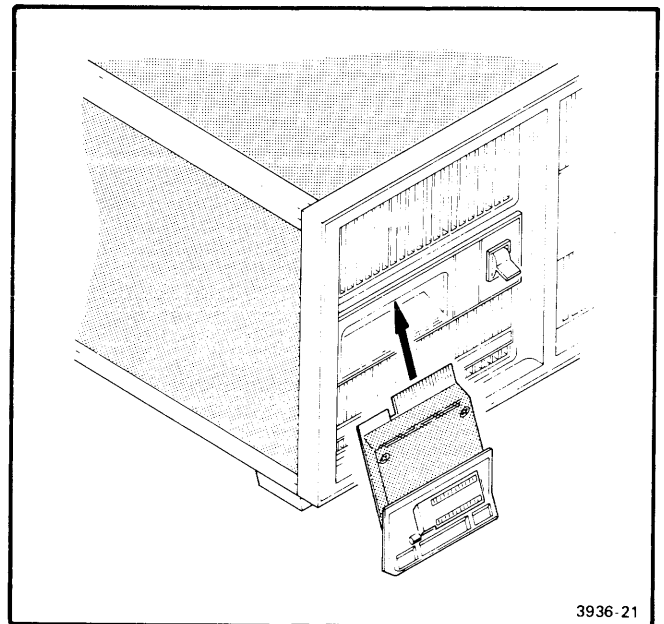


Fig. 8-1. Inserting a characteristic module.

## Starting Up the 8540

1. First, be certain that you do not have a PROM device in the ZIF socket of the characteristic module. Then, start up the 8540. (For start-up instructions, refer to the Learning Guide section of this manual.) The SELF TEST indicator on the front panel assembly of the PROM Programmer should light briefly (5 seconds maximum), and then go out.
2. Turn on the PROM POWER switch, located on the PROM Programmer front panel assembly. The READY indicator will light if a characteristic module is present in the front panel assembly **and** the PROM Programmer has powered up correctly.

If the SELF TEST indicator stays on, or if the READY indicator does not light, enter the PSTAT command on your system terminal. The PSTAT display should indicate the problem. For PSTAT display examples and error message explanations, see the PROM Programmer Specifics supplement for the characteristic module you are using. If the problem persists, refer to the PROM Programmer Controller Installation Manual.

## Inserting a PROM into the Characteristic Module



*To prevent potential damage to your PROM device, do not insert or remove a PROM when the SELF TEST light on the front panel assembly is ON.*

You can insert or remove a PROM when any of the following conditions are true:

- the characteristic module has not been inserted into the front panel assembly, or
- the 8540 is not powered up, or
- the PROM POWER switch is OFF, or
- the READY light on the front panel assembly is ON.

At the time of this writing, all supported PROMs are inserted according to the following procedure:

1. Raise the lever on the ZIF socket of the characteristic module.
2. Insert the PROM into the ZIF socket so that pin 1 of the PROM is next to the lever.
3. Depress the lever so that the socket grips the pins of the PROM.



*Be certain that pin 1 of the PROM is at the end of the ZIF socket next to the lever. The PROM Programmer cannot always detect an incorrectly socketed device; damage to the PROM can result from an attempt to program an incorrectly socketed PROM.*

*Most PROM devices have an indentation near pin 1; other PROMs may have a small dot painted on that end. Refer to the manufacturer's specifications if you are in doubt.*

## PROM Programmer Commands

The system commands that control the PROM Programmer are described in detail in each PROM Programmer Specifics supplement.

## Turning Off the 8540

Before you turn off the 8540, be certain to remove the PROM from the characteristic module.

# PROM PROGRAMMER SPECIFICS SUPPLEMENTS

Each characteristic module is accompanied by a PROM Programmer Specifics supplement that provides information specific to the characteristic module. Each supplement is designed to be inserted into this manual.

These PROM Programmer Specifics supplements are numbered as if they were separate sections of this manual. For example, the supplement that documents the 2716/2732 PROM Programmer Module is labeled "Section 8A," and the first illustration in that supplement is numbered "Fig. 8A-1." Similarly, other supplements are labeled Sections 8B, 8C, etc. Figures, pages, and tables are numbered accordingly.

Each supplement presents the following information:

- A list of PROMs supported by the characteristic module.
- System commands, parameters, and displays that are specific to the characteristic module.
- One or more examples of each command supported by the characteristic module.
- Any irregularities that should be noted.
- Features designed to support PROMs with special programming needs.



This manual supports the following TEKTRONIX products:

8550 Option	8540 Option	Product
31	31	8550F31

This manual supports a software/firmware module that is compatible with

DOS/50 V02.xx-xx (8550)  
OS/40 V01.xx-xx (8540)

**PLEASE CHECK FOR CHANGE INFORMATION  
AT THE REAR OF THIS MANUAL.**

**8500**  
MODULAR MDL SERIES  
**2716/2732**  
PROM PROGRAMMER  
SPECIFICS  
USERS MANUAL

Tektronix, Inc.  
P.O. Box 500  
Beaverton, Oregon 97077  
070-3868-00  
Product Group 61

Serial Number \_\_\_\_\_

First Printing OCT 1981  
Revised NOV. 1981

## LIMITED RIGHTS LEGEND

Software License No. \_\_\_\_\_

Contractor: Tektronix, Inc.  
Explanation of Limited Rights Data Identification Method  
Used: Entire document subject to limited rights.


Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

## RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Copyright © 1981 Tektronix, Inc. All rights reserved. Contents of this publication may not be reproduced in any form without the written permission of Tektronix, Inc.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and  are registered trademarks of Tektronix, Inc. TELEEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Printed in U.S.A. Specification and price change privileges are reserved.

## Section 8A

# 2716/2732 PROM PROGRAMMER SPECIFICS

	Page
<b>Introduction</b> .....	8A-1
<b>Devices Supported</b> .....	8A-1
<b>PROM Programmer Commands</b> .....	8A-2
Command Parameters .....	8A-2
Command Modifiers .....	8A-3
PROM Programmer Command Dictionary .....	8A-4
CPR—Compares PROM With Memory .....	8A-4
PSTAT—Reports PP Status .....	8A-6
PTYPE—Displays PROM Types .....	8A-7
RPR—Reads PROM .....	8A-8
WPR—Writes to PROM .....	8A-10
<b>Error Messages</b> .....	8A-12

## Section 8A

# 2716/2732 PROM PROGRAMMER SPECIFICS

## INTRODUCTION

This supplement is designed to be inserted into one of the following sections:

- Section 8 of the 8540 System Users Manual,
- Section 8 of the 8550 System Users Manual (DOS/50 Version 2), or
- Section 9 of the 8550 System Users Manual (DOS/50 Version 1).

Throughout this supplement, "your System Users Manual" refers to the 8550 System Users Manual or 8540 System Users Manual. "Module 1" refers to the 2716/2732 PROM Programmer Module.

### CAUTION

*The PROM Programmer module is static-sensitive. To avoid damage to the module, handle it as little as possible in high-static situations. Pick up the module by the body, not by the edge connectors. Avoid sliding the module over any surface.*

## DEVICES SUPPORTED

Module 1 enables the PROM Programmer to program a set of similar devices. Each of the PROMs supported by Module 1 is a MOS, single supply, byte-wide chip. Table 8A-1 lists the device types supported by Module 1.

**Table 8A-1**  
**Programmable Devices Supported by Module 1**

Manufacturer	Device	PROM type <sup>a</sup>	Description
Texas Instruments	TMS 2508	2508/TI	1K x 8 bit EPROM
Intel	2758	2758/I	1K x 8 bit EPROM
Intel	2758 S1865	2758S1865/I	1K x 8 bit EPROM
Texas Instruments	TMS 2516	2516/TI	2K x 8 bit EPROM
Intel	2716	2716/I	2K x 8 bit EPROM
Intel	2816	2816/I	2K x 8 bit EEPROM
Texas Instruments	TMS 2532	2532/TI	4K x 8 bit EPROM
Intel	2732	2732/I	4K x 8 bit EPROM
Intel	2732A	2732A/I	4K x 8 bit EPROM
Motorola	MCM 68764	68764/M	8K x 8 bit EPROM

<sup>a</sup> The names that specify the devices supported by Module 1.



PROMs with programming characteristics similar to a PROM type may be available from other manufacturers. The PROM Programmer with Module 1 programs any PROM that has exactly the same specifications as one of those listed above. Refer to the manufacturer's data sheets, and use the PROM type identification of the device in the table whose specifications match those of the one you have.

If you attempt to use a PROM which is not supported by this characteristic module, you will receive an error message:

```
> CPR 0 3232/Z 0 40      (specifying a Zilog 3232 PROM)
CPR:      Error 80 PROM type not supported
```

### CAUTION

*The PROM Programmer does not verify that the PROM inserted matches the PROM type specification. The voltage levels necessary to successfully program a PROM vary from one type to another. If the wrong PROM type is specified, your PROM may be destroyed.*

## PROM PROGRAMMER COMMANDS

There are five system commands for use with the PROM Programmer:

- **CPR** (Compare PROM) compares the contents of the specified PROM addresses with the corresponding data in program memory, and displays any differences.
- **RPR** (Read PROM) reads the contents of the specified PROM addresses into program memory.
- **WPR** (Write PROM) writes data from program memory into the PROM.
- **PSTAT** (PROM STATus) displays the current status of the PROM Programmer, including any error conditions.
- **PTYPE** (PROM TYPE) displays information about the characteristic module currently inserted in the front panel of the PROM Programmer, informing the user what PROMs it supports and what their characteristics are.

The PSTAT and PTYPE commands are issued without parameters. The following subsections describe the parameters that are used with the CPR, RPR, and WPR commands.

### Command Parameters

The following are **necessary** parameters for any invocation of the CPR, RPR, or WPR commands.

- **Memlo** specifies the starting address from which data is transferred to the PROM or to which data is placed after being read from the PROM.

- **Promtype** is the alphanumeric name specifying the PROM with which you are working. See Table 8A-1 for the **promtype** associated with your chip, or use the PTYPE command described below. The **promtype** must be entered exactly as shown; letters must be uppercase.
- **Promlo** specifies the (hexadecimal) PROM address which you wish to be the starting point for the specified command.
- **Promhi** specifies the ending PROM address.

### Command Modifiers

For illustrations of the use of the command modifiers, see the following subsection, PROM Programmer Command Dictionary.

#### NOTE

*Syntax for this section has been updated for DOS/50 Version 2 and OS/40. If you are using DOS/50 Version 1, all "dash modifiers" (such as -L, -M) should be "colon modifiers" (:L, :M). The Technical Notes section of the 8550 System Users Manual describes the differences between Version 1 and Version 2.*

#### Modifier Meaning

- a Provides inverted addressing (hi=0, low=1). Use -A in applications where the levels applied to the PROM's address pins are inverted.
- d Data is stored in the PROM in inverted form (hi=0, low=1). For a read operation (RPR) this means that the data from the PROM is inverted before being stored in program memory. During a write (WPR) data is inverted before being written into PROM.
- l For the 8-bit PROMs supported by Module 1, this modifier indicates that data being read from (or written into) the PROM represents the least significant bytes of 16-bit words in program memory. See also -R.
- m For 8-bit PROMs, this modifier indicates that data being read from (or written into) PROM represents the most significant bytes of 16-bit words in program memory. See also -R.
- n Specifies that no erase check is performed before a write operation. See the WPR command for examples.
- r The -R modifier is used when the data in program memory is arranged in 16-bit words. -R indicates that the bytes are arranged in reverse order in program memory, that is, least significant byte first, followed by most significant byte.  
  
-R is used with -M or -L. For example, if modifiers -M and -R are entered as part of a RPR command when reading an 8-bit PROM, the bytes read from the PROM will be placed in the second byte of each 16-bit word. See the RPR command for examples.

## PROM Programmer Command Dictionary

This subsection gives the syntax and use of each PROM Programmer command as it pertains to Module 1. The format is the same as that of the Command Dictionary of your System Users Manual.

### SYNTAX

```
cpr [-a] [-d] [-m] [-l] [-r] memlo promtype promlo promhi
```

### EXPLANATION

Compares the contents of the specified PROM addresses with the contents of program memory, starting at the given memory address. The locations of any differences are displayed. If the contents match, no message is displayed.

The CPR command is useful when you wish to verify that the data in the PROM is the same as the data in a specified region of program memory.

### EXAMPLES

Enter the following command to read the contents of locations 0-40 of an Intel 2716 PROM into locations 0-40 of program memory.

```
> RPR 0 2716/I 0 40
```

Now dump program memory to see what's been read:

```
> D 0 40
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0010 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0020 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0030 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0040 54
T
```

If the data was read correctly, a CPR command at this point will return no message.

```
> CPR 0 2716/I 0 40
```

```
>
```

To illustrate the response you receive when an error is detected, use the P command to patch new data into two locations in program memory.

```
> P 05 00
```

```
> P 07 00
```

Now compare the contents again:

```
> CPR 0 2716/I 0 40
```

```
*** COMPARE ERROR AT PROM ADDRESS=00005H
```

```
*** COMPARE ERROR AT PROM ADDRESS=00007H
```

The discrepancies introduced into locations 5 and 7 of program memory were identified as errors by the CPR command.

If you use the -A modifier in a CPR command, the address given in a "COMPARE ERROR" message is actually the complement of the value that was applied at the PROM's address pins.

**SYNTAX**

**pstat**

**EXPLANATION**

Displays the current condition of the PROM Programmer. Normally, PSTAT will report "no errors." Conditions which prompt an error message include power failure, the absence of a characteristic module, or PROM Programmer power off.

Error messages are discussed elsewhere in this supplement.

**EXAMPLES**

When no errors are present:

```
> PSTAT  
... NO PROM PROGRAMMER ERRORS DETECTED ...
```

Reporting an error condition:

```
> PSTAT  
PP ERROR-- POWER FAIL
```

**SYNTAX****ptype****EXPLANATION**

Displays information about PROMs supported by Module 1. The display consists of four columns:

- **PROM TYPE:** the name that specifies each PROM supported by the characteristic module. Each name is the model number of the primary source manufacturer, followed by a slash and one or more letters specifying the manufacturer. For example, Intel's 2716 PROM is identified as 2716/I. (Note that there are no spaces in the PROM TYPE designation.)
- **MAX ADDRESS:** the maximum addressable location of each PROM listed. For example, 3FFH indicates a total of 400H addressable bytes (0 through 3FF).
- **ERASED STATE:** the state of the bits (high or low) when the PROM device has been erased. For the 8-bit PROMs supported by Module 1, HI is FF and LOW is 00.
- **WORD LENGTH:** the number of bits per word for each PROM. Since Module 1 supports only 1-byte (8-bit) wide PROMs, word length for all PROMs listed is eight.

**EXAMPLES**> PTYPE

PROM TYPE	MAX ADDRESS	ERASED STATE	WORD LENGTH
2508/TI	3FFH	HI	8
2758/I	3FFH	HI	8
2758S1865/I	3FFH	HI	8
2516/TI	7FFH	HI	8
2716/I	7FFH	HI	8
2816/I	7FFH	HI	8
2532/TI	FFFH	HI	8
2732/I	FFFH	HI	8
2732A/I	FFFH	HI	8
68764/M	1FFFH	HI	8

**SYNTAX**

**rpr [-a] [-d] [-m] [-l] [-r] memlo promtype promlo promhi**

**EXPLANATION**

Reads the contents of the specified PROM addresses into the specified location in program memory. No message is displayed unless an error condition exists.

**EXAMPLES**

First use the F command to fill program memory with EE. Later, when you examine memory, the EEs make it easy to see that new data was read in.

```
> F 0 40 EE

> D 0 40
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0010 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0020 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0030 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0040 EE
```

Now use the RPR command to read the contents of memory locations 0-40 of a Texas Instruments 2532 PROM into program memory, beginning at address 0. Then dump the contents of program memory, to verify that it now contains new data from the PROM.

```
> RPR 0 2532/TI 0 40

> D 0 40
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0010 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0020 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0030 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0040 54 T
```

In the example above, no optional modifiers were needed. Next, assume that our 8-bit PROM is to be used as the most significant half of a 16-bit memory system. Again, fill program memory with EE's, read the PROM, and dump the results. This time, however, the -M modifier is used.

```
> F 0 40 EE

> RPR -M 0 2532/TI 0 40

> D 0 40
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 EE T.H.I.S. .I.S. .
0010 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E EE A. .T.E.S.T.....
0020 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 EE T.H.I.S. .I.S. .
0030 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E EE A. .T.E.S.T.....
0040 54 T
```

The results of the D command show that the 8-bit bytes received from the PROM are now stored in the most significant byte positions of a 16-bit data base in program memory.

Compare those results with the example below, in which memory is filled with EE and the RPR command line includes the -L modifier.

```
> F 0 40 EE
> RPR -L 0 2532/TI 0 40
> D 0 40
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 EE 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 .T.H.I.S. .I.S.
0010 EE 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E .A. .T.E.S.T....
0020 EE 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 .T.H.I.S. .I.S.
0030 EE 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E .A. .T.E.S.T....
0040 EE
```

Notice that this time the PROM bytes have been placed in the least significant positions of the 16-bit data base.

Now assume that the PROM contains most significant bytes, and that the data base in program memory should consist of 16-bit data arranged in reverse order (least significant byte followed by most significant byte). Note that -R should be used only with either a -L or a -M modifier.

```
> F 0 40 EE
> RPR -MR 0 2532/TI 0 40
> D 0 40
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 EE 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 .T.H.I.S. .I.S.
0010 EE 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E .A. .T.E.S.T....
0020 EE 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 .T.H.I.S. .I.S.
0030 EE 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E .A. .T.E.S.T....
0040 EE
```

The combined -M and -R modifiers produce the same results in program memory as does the -L modifier.

If the PROM contained the least significant bytes of a 16-bit memory system, and the program memory data were arranged in reverse order, the -L and -R modifiers would be used. The output would look exactly like that produced by the example above using -M.



**SYNTAX**

`wpr [-a] [-d] [-l] [-m] [-r] [-n] memlo promtype promlo promhi`

**EXPLANATION**

Writes the contents of program memory beginning at the address **memlo** into the section of a **promtype** PROM between **promlo** and **promhi**. The resulting PROM data is checked for accuracy. If the data matches, no message is reported; otherwise, an error message is printed. For more detailed descriptions of these parameters, see the Command Parameters subsection earlier in this supplement.

Any byte into which you wish new data written must be erased before it is written to.

With one exception, the PROMs supported by Module 1 are EPROMs—that is, they may be erased by exposure to ultra-violet light. The PROM Programmer does not perform this erasure; it must be done by a unit designed for that purpose.

The exception is the Intel 2816 EEPROM (promtype 2816/I). This is an electrically-erasable PROM. If you specify the entire EEPROM address space in the WPR command, the PROM Programmer erases the entire chip. Otherwise, it will erase only those portions of the EEPROM which it has been ordered to program. To erase individual bytes of the 2816, write FF into their memory locations. The -N modifier **must** be used when programming a 2816/I type PROM.

If asked to write to an EPROM, the PROM Programmer checks the designated memory space to ensure that it is erased and ready to accept new information. You may disable this erase check by using the -N command modifier. This does **not** allow you to reprogram an unerased portion of a chip.

**EXAMPLES**

When a write command is properly executed, no message is returned.

```
> WPR 20 2732/I 0 80
```

If you do not use the -N modifier, an attempt to write to a PROM which has not been fully erased will produce an error message.

```
> WPR 20 2732/I 0 80  
PP ERROR-- DEV NOT ERASED
```

Suppose, however, that you have programmed a PROM, discovered an error in your code, and wish to change only a few bytes of memory. If the bits you wish to change are currently in an erased state, you may disable the erase check by using modifier -N, and write over the section of the device that contains the code to be changed.

The following example first writes data from program memory into the PROM, and then reads back from the device:

```
> WPR 00 2716/I 00 10
> RPR 00 2716/I 00 10
> D 0 10
   0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 54 48 4D 53 20 49 53 20 49 20 54 45 53 54 2E 2E THMS IS I TEST..
0010 54                                     T
```

Next new data is patched into locations 02 and 08 of program memory, and the new contents of program memory are displayed:

```
> P 02 49
> P 08 41
> D 0 10
   0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0010 54                                     T
```

Now you can write the corrected data to the PROM. Normally, an EPROM would need to be erased (with ultra-violet light) before the new write operation. In this case, however, since both changes required changing bits which were still in an erased state (1), the programming can be done "over the top" of the old data. "M" (01001011) is changed to "I" (01001001), and "I" is changed to "A" (01000001).

```
> WPR 00 2716/I 0 10
  PP ERROR-- DEV NOT ERASED
```

An error occurs because the -N modifier was omitted. Try again:

```
> WPR -N 00 2716/I 00 10
>
```

The write operation was successful, so you are prompted for your next command.

If you use modifier -N and attempt to write into a region of the PROM that is not reprogrammable, you will be informed that the write operation was not successful (that the resulting data in the PROM does not match program memory).

```
> WPR -N 0 2716/I 24 OFF
  PP ERROR-- DEV NOT REPROG      (device not reprogrammable)
```

Since the PROM Programmer has changed bits which were in an erased state, and has been unsuccessful at changing others, this may leave you with a garbled mess in your PROM.

## ERROR MESSAGES

PROM Programmer error messages are identified by the keywords "PP ERROR." They are divided into two categories.

The first group of messages may appear in response to a PROM Programmer command (CPR, PSTAT, PTYPE, RPR, or WPR) during normal operation. They indicate that the PROM Programmer is unable to execute the command. This condition may be easily fixed (by turning on PROM Programmer power, for example), or it may require a call to your Tektronix service representative.

The second group of messages announces problems discovered by the PROM Programmer self-test procedure, executed automatically during power up.

If you see an error message without the words "PP ERROR," refer to the section on Error Messages in your System Users manual.

### Execution Errors

The following error messages may be received in response to a PROM Programmer command (CPR, PSTAT, PTYPE, RPR, or WPR).

**PP ERROR—CHAR MOD ABSENT.** There is no characteristic module inserted in the front panel of the PROM Programmer.

**PP ERROR—CONFIG MEM CKSUM.** The characteristic module memory shows an error. Call your service representative. Since the PROM Programmer Controller is not malfunctioning, you should still be able to use a different characteristic module.

**PP ERROR—DEV NOT ERASED.** A WPR (write) command was attempted, but the PROM is not erased for correct programming. See the discussion of the WPR command elsewhere in this supplement for more information on erasing a PROM.

**PP ERROR—DEV NOT REPROG.** After writing to a programmable device, the PROM Programmer compares the data in the PROM with program memory to ensure that the data was correctly transmitted. The message "DEV NOT REPROG" indicates that an attempt was made to write to a device, and the results in the PROM do not match program memory. This may indicate a bad part, or an attempt to use the -N modifier with the WPR command on an unerased section of device memory. See the discussion of the WPR command elsewhere in this supplement.

**PP ERROR—POWER FAIL.** The operation was aborted due to an internal power failure. Try the operation again. If the POWER FAIL message appears frequently, call your service representative.

**PP ERROR—PROM POWER OFF.** The power switch on the PROM Programmer front panel assembly must be turned on before PROM Programmer commands can execute.

---

**Error 80 PROM type not supported.** The **promtype** parameter entered with a CPR, RPR, or WPR command is not recognized by this characteristic module. For a list of PROMs supported and their promtype designations, see Table 8A-1 or enter the PTYPE command. Note that the promtype name must be entered exactly as shown. For example, 2732/i or 2732/INTEL are not accepted for the 2732/I promtype.

### Power Up Errors

Assuming you have powered up the PROM Programmer following the instructions in the introduction to this section, the READY indicator on the PROM Programmer front panel should light. If the READY indicator does not light, type in the PSTAT command. You may receive one of the responses listed above, or one of the following messages:

```
PP ERROR-- CPU
PP ERROR-- POWER CONTROL
PP ERROR-- RAM
PP ERROR-- ROM
PP ERROR-- SUPPLY MONITOR
PP ERROR-- SYST INTERFACE
```

If you receive the message NO PROM PROGRAMMER ERRORS DETECTED, the PROM Programmer should be functioning correctly. The problem may be with the READY indicator. Try executing a command.

The other error messages indicate a problem in the hardware or firmware of the PROM Programmer Controller. Call your service representative for assistance.



This manual supports the following TEKTRONIX products:

8550 Option	8540 Option	Product
32	32	8550F32

This manual supports a software/firmware module that is compatible with

DOS/50 V02.xx-xx (8550)  
OS/40 V01.xx-xx (8540)

**PLEASE CHECK FOR CHANGE INFORMATION  
AT THE REAR OF THIS MANUAL.**

**8500**  
MODULAR MDL SERIES  
**8748/8741A**  
**8749/8755A**  
**PROM PROGRAMMER**  
**SPECIFICS**  
**USERS MANUAL**

**Tektronix, Inc.**  
**P.O. Box 500**  
**Beaverton, Oregon 97077**  
  
070-3869-00  
Product Group 61

Serial Number \_\_\_\_\_

First Printing OCT 1981  
Revised DEC 1981

## LIMITED RIGHTS LEGEND

Software License No. \_\_\_\_\_

Contractor: Tektronix, Inc.

Explanation of Limited Rights Data Identification Method

Used: Entire document subject to limited rights.


Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

## RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Copyright © 1981 Tektronix, Inc. All rights reserved. Contents of this publication may not be reproduced in any form without the written permission of Tektronix, Inc.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and  are registered trademarks of Tektronix, Inc. TELEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Printed in U.S.A. Specification and price change privileges are reserved.

## Section 8B

# 8748/8741A/8749/8755A PROM PROGRAMMER SPECIFICS

	Page
<b>Introduction</b> .....	8B-1
<b>Devices Supported</b> .....	8B-1
<b>PROM Programmer Commands</b> .....	8B-2
Command Parameters .....	8B-2
Command Modifiers .....	8B-3
<b>PROM Programmer Command Dictionary</b> .....	8B-4
CPR—Compares Device Memory With Program Memory .....	8B-4
PSTAT—Reports PP Status .....	8B-6
PTYPE—Displays Device Types .....	8B-7
RPR—Reads Device Memory .....	8B-8
WPR—Writes to Device Memory .....	8B-10
<b>Error Messages</b> .....	8B-12

## Section 8B

# 8748/8741A/8749/8755A PROM PROGRAMMER SPECIFICS

### INTRODUCTION

This supplement is designed to be inserted into one of the following sections:

- Section 8 of the 8540 System Users Manual,
- Section 8 of the 8550 System Users Manual (DOS/50 Version 2), or
- Section 9 of the 8550 System Users Manual (DOS/50 Version 1).

Throughout this supplement, "your System Users Manual" refers to the 8550 System Users Manual or 8540 System Users Manual. "Module 2" refers to the 8748/8741A/8749/8755A PROM Programmer Module.

**CAUTION**

*The PROM Programmer module is static-sensitive. To avoid damage to the module, handle it as little as possible in high-static situations. Pick up the module by the body, not by the edge connectors. Avoid sliding the module over any surface.*

### DEVICES SUPPORTED

Module 2 enables the PROM Programmer to program a set of similar devices. Table 8B-1 lists the device types supported by Module 2.

**Table 8B-1**  
**Programmable Devices Supported by Module 2**

Manufacturer	Device	PROM Type <sup>a</sup>	Description
Intel	8755A	8755A/I	2K x 8 bit MCS-85 Peripheral (EPROM with I/O)
Intel	8748	8748/I	1K x 8 bit Microcomputer
Intel	8749	8749/I	2K x 8 bit Microcomputer
Intel	8741A	8741A/I	1K x 8 bit Peripheral Interface Microcomputer

<sup>a</sup> The names that specify the devices supported by Module 2.



Devices with programming characteristics similar to a PROM type may be available from other manufacturers. The PROM Programmer with Module 2 programs any device that has exactly the same specifications as one of those listed above. Refer to the manufacturer's data sheets, and use the PROM type identification of the device in the table whose specifications match those of the one you have.

If you attempt to use a device which is not supported by this characteristic module, you will receive an error message:

```
> CPR 0 3232/Z 0 40      (specifying a Zilog 3232 PROM)
CPR: Error 80 PROM type not supported
```

**CAUTION**

*The PROM Programmer does not verify that the device inserted matches the PROM type specification. The voltage levels necessary to successfully program a device vary from one type to another. If the wrong PROM type is specified, your device may be destroyed.*

## PROM PROGRAMMER COMMANDS

There are five system commands for use with the PROM Programmer:

- **CPR** (Compare PROM) compares the contents of the specified PROM addresses with the corresponding data in program memory, and displays any differences.
- **RPR** (Read PROM) reads the contents of the specified PROM addresses into program memory.
- **WPR** (Write PROM) writes data from program memory into the device.
- **PSTAT** (PROM STATus) displays the current status of the PROM Programmer, including any error conditions.
- **PTYPE** (PROM TYPE) displays the characteristic module currently inserted in the front panel of the PROM Programmer, informing the user what devices it supports and what their characteristics are.

The PSTAT and PTYPE commands are issued without parameters. The following subsections describe the parameters that are used with the CPR, RPR, and WPR commands.

### Command Parameters

The following are **necessary** parameters for any invocation of the CPR, RPR, or WPR commands.

- **Memlo** specifies the starting address in program memory from which data will be transferred to the device or to which data is to be placed after being read from the device.
- **Promtype** is the alphanumeric name specifying the device with which you are working. See Table 8B-1 for the **promtype** associated with your device, or use the PTYPE command described below. The **promtype** must be entered exactly as shown; letters must be uppercase.

- **Promlo** specifies the (hexadecimal) PROM address which you wish to be the starting point for the specified command.
- **Promhi** specifies the ending PROM address.

## Command Modifiers

For illustrations of the use of the command modifiers, see the following subsection, PROM Programmer Command Dictionary.

### NOTE

*Syntax for this section has been updated for DOS/50 Version 2 and OS/40. If you are using DOS/50 Version 1, all "dash modifiers" (such as -L, -M) should be "colon modifiers" (:L, :M). The Technical Notes section of the 8550 System Users Manual describes the differences between Version 1 and Version 2.*

### Modifier Meaning

- a Provides inverted addressing (high=0, low=1). Use -A in applications where the levels applied to the device's address pins are inverted.
- d Data is stored in the device in inverted form (high=0, low=1). For a read operation (RPR) this means that the data from the device is inverted before being stored in program memory. During a write (WPR) data is inverted before being written into PROM.
- l For the 8-bit devices supported by Module 2, this modifier indicates that data being read from (or written into) the device represents the least significant bytes of 16-bit words in program memory. See also -R.
- m For 8-bit devices, this modifier indicates that data being read from (or written into) the device memory represents the most significant bytes of 16-bit words in program memory. See also -R.
- n Specifies that no erase check is performed before a write operation. See the WPR command for examples.
- r The -R modifier is used when the data in program memory is arranged in 16-bit words. -R indicates that the bytes are arranged in reverse order in program memory, that is, least significant byte first, followed by most significant byte.  
-R is used with -M or -L. For example, if modifiers -M and -R are entered as part of a RPR command when reading an 8-bit device, the bytes read from the device will be placed in the second byte of each 16-bit word. See the RPR command for examples.

## PROM Programmer Command Dictionary

This subsection gives the syntax and use of each PROM Programmer command as it pertains to Module 2. The format is the same as that of the Command Dictionary of your System Users Manual.

### SYNTAX

```
cpr [-a] [-d] [-m] [-l] [-r] memlo promtype promlo promhi
```

### EXPLANATION

Compares the contents of the specified device addresses with the contents of program memory, starting at the given memory address. The locations of any differences are displayed. If the contents match, no message is displayed.

The CPR command is useful when you wish to verify that the data in the device is the same as the data in a specified region of program memory.

### EXAMPLES

Enter the following command to read the contents of locations 0-40 of an Intel 8749 device into locations 0-40 of program memory.

```
> RPR 0 8749/I 0 40
```

Now dump program memory to see what's been read:

```
> D 0 40
   0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0010 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0020 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0030 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0040 54                                     T
```

If the data was read correctly, a CPR command at this point will return no message.

```
> CPR 0 8749/I 0 40
```

```
>
```

To illustrate the response you receive when an error is detected, use the P command to patch new data into two locations in program memory:

```
> P 05 00  
> P 07 00
```

Now compare the contents again:

```
> CPR 0 8749/I 0 40  
*** COMPARE ERROR AT PROM ADDRESS=00005H  
*** COMPARE ERROR AT PROM ADDRESS=00007H
```

The discrepancies introduced into locations 5 and 7 of program memory were identified as errors by the CPR command.

If you use the -A modifier in a CPR command, the address given in a "COMPARE ERROR" message is actually the complement of the value that was applied to the device's address pins.

**SYNTAX**

**pstat**

**EXPLANATION**

Displays the current condition of the PROM Programmer. Normally, PSTAT will report "no errors." Conditions which prompt an error message include power failure, the absence of a characteristic module, or PROM Programmer power off.

Error messages are discussed elsewhere in this supplement.

**EXAMPLES**

When no error conditions are present:

```
> PSTAT  
... NO PROM PROGRAMMER ERRORS DETECTED ...
```

Reporting an error condition:

```
> PSTAT  
PP ERROR-- CHAR MOD ABSENT
```

**SYNTAX****ptype****EXPLANATION**

Displays information about devices supported by Module 2. The display consists of four columns:

- **PROM TYPE:** the name that specifies each device supported by the characteristic module. Each name is the model number of the primary source manufacturer, followed by a slash and one or more letters specifying the manufacturer. For example, Intel's 8755A device is identified as 8755A/I. (Note that there are no spaces in the PROM TYPE designation.)
- **MAX ADDRESS:** the maximum addressable location of each device listed. For example, 3FFH indicates a total of 400H addressable bytes (0 through 3FF).
- **ERASED STATE:** the state of the bits (high or low) when the device has been erased. For the 8-bit devices supported by Module 2, HI is FF and LOW is 00.
- **WORD LENGTH:** the number of bits per word for each device. Since Module 2 supports only 1-byte (8-bit) wide devices, word length for all devices listed is eight.

**EXAMPLES**> PTYPE

PROM TYPE	MAX ADDRESS	ERASED STATE	WORD LENGTH
8755A/I	7FFH	HI	8
8748/I	3FFH	LO	8
8749/I	7FFH	LO	8
8741A/I	3FFH	LO	8

**SYNTAX**

**rpr [-a] [-d][<sup>-m</sup>[-l]] [-r] memlo promtype promlo promhi**

**EXPLANATION**

Reads the contents of the specified device addresses into the specified location in program memory. No message is displayed unless an error condition exists.

**EXAMPLES**

First use the F command to fill program memory with EE. Later, when you examine memory, the EEs make it easy to see that new data was read in.

```
> F 0 40 EE

> D 0 40
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0010 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0020 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0030 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0040 EE
```

Now use the RPR command to read the contents of memory locations 0-40 of an Intel 8755A device into program memory, beginning at address 0. Then dump the contents of program memory, to verify that it now contains new data from the device.

```
> RPR 0 8755A/I 0 40

> D 0 40
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0010 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0020 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0030 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0040 54 T
```

In the example above, no optional modifiers were needed. Next, assume that our 8-bit device is to be used as the most significant half of a 16-bit memory system. Again, fill program memory with EE's, read the device, and dump the results. This time, however, the -M modifier is used.

```
> F 0 40 EE

> RPR -M 0 8755A/I 0 40

> D 0 40
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 EE T.H.I.S. .I.S. .
0010 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E EE A. .T.E.S.T.....
0020 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 EE T.H.I.S. .I.S. .
0030 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E EE A. .T.E.S.T.....
0040 54 T
```

The results of the D command show that the 8-bit bytes received from the device are now stored in the most significant byte positions of a 16-bit data base in program memory.

Compare those results with the example below, in which memory is filled with EE and the RPR command line includes the -L modifier.

```
> F 0 40 EE
> RPR -L 0 8755A/I 0 40
> D 0 40
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 EE 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 .T.H.I.S. .I.S.
0010 EE 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E .A. .T.E.S.T....
0020 EE 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 .T.H.I.S. .I.S.
0030 EE 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E .A. .T.E.S.T....
0040 EE
```

Notice that this time the bytes of data from the device have been placed in the least significant positions of the 16-bit data base.

Now assume that the device contains most significant bytes, and that the data base in program memory should consist of 16-bit data arranged in reverse order (least significant byte followed by most significant byte). Note that -R should be used only with either a -L or a -M modifier.

```
> F 0 40 EE
> RPR -MR 0 8755A/I 0 40
> D 0 40
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 EE 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 .T.H.I.S. .I.S.
0010 EE 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E .A. .T.E.S.T....
0020 EE 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 .T.H.I.S. .I.S.
0030 EE 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E .A. .T.E.S.T....
0040 EE
```

The combined -M and -R modifiers produce the same results in program memory as does the -L modifier.

If the device contained the least significant bytes of a 16-bit memory system, and the program memory data were arranged in reverse order, the -L and -R modifiers would be used. The output would look exactly like that produced by the example above using -M.



### SYNTAX

```
wpr [-a] [-d] [-l] [-m] [-r] [-n] memlo promtype promlo promhi
```

### EXPLANATION

Writes the contents of program memory beginning at the address **memlo** into the section of a **promtype** device between **promlo** and **promhi**. The resulting data in the device memory is checked for accuracy. If the data matches, no message is reported; otherwise, an error message is printed. For more detailed descriptions of these parameters, see the Command Parameters subsection earlier in this supplement.

Any byte into which you wish new data written must be erased before it is written to.

The devices supported by Module 2 may be erased by exposure to ultra-violet light. The PROM Programmer is not designed to do this erasure. It must be done by a unit designed for that purpose.

When a WPR command is issued, the PROM Programmer examines the designated memory area to ensure that it is erased and ready to accept new information. You may disable this erase check by using command modifier **-N**. This does **not** allow you to reprogram an unerased portion of a device.

### EXAMPLES

When a write command is properly executed, no message is returned.

```
> WPR 20 8748/I 0 80
```

If you do not use the **-N** modifier, an attempt to write to a device which has not been fully erased will produce an error message.

```
> WPR 20 8748/I 0 80  
PP ERROR-- DEV NOT ERASED
```

Suppose, however, that you have programmed a device, discovered an error in your code, and wish to change only a few bytes of memory. **If** the bits you wish to change are currently in an erased state, you may disable the erase check by using modifier **-N**, and write over the section of the device that contains the code to be changed.

The following example writes data from program memory into the device, and then reads back from the device:

```
> WPR 00 8755A/I 00 10
> RPR 00 8755A/I 00 10
> D 0 10
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 54 48 4D 53 20 49 53 20 49 20 54 45 53 54 2E 2E THMS IS I TEST..
0010 54                                     T
```

Next new data is patched into locations 02 and 08 of program memory, and the new contents of program memory are displayed:

```
> P 02 49
> P 08 41
> D 0 10
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0010 54                                     T
```

Now you can write the corrected data to the device. Normally, the device would need to be erased (with ultra-violet light) before the new write operation. In this case, however, since both changes required changing bits which were still in an erased state (1), the programming can be done "over the top" of the old data. "M" (01001011) is changed to "I" (01001001), and "I" is changed to "A" (01000001).

```
> WPR 00 8755A/I 0 10
  PP ERROR-- DEV NOT ERASED
```

An error occurs because the -N modifier was omitted. Try again:

```
> WPR -N 00 8755A/I 00 10
>
```

The write operation was successful, so you are prompted for your next command.

If you use modifier -N and attempt to write into a region of the device that is not reprogrammable, you will be informed that the write operation was not successful (that the resulting data in the device does not match program memory).

```
> WPR -N 0 8748/I 24 OFF
  PP ERROR-- DEV NOT REPROG          (device not reprogrammable)
```

Since the PROM Programmer has changed bits which were in an erased state, and has been unsuccessful at changing others, this may leave you with a garbled mess in your device.

## ERROR MESSAGES

PROM Programmer error messages are identified by the keywords "PP ERROR." They are divided into two categories.

The first group of messages may appear in response to a PROM Programmer command (CPR, PSTAT, PTYPE, RPR, or WPR) during normal operation. They indicate that the PROM Programmer is unable to execute the command. This condition may be easily fixed (by turning on PROM Programmer power, for example), or it may require a call to your Tektronix service representative.

The second group of messages announces problems discovered by the PROM Programmer self-test procedure, executed automatically during power up.

If you see an error message without the words "PP ERROR," refer to the section on Error Messages in your System Users manual.

### Execution Errors

The following error messages may be received in response to a PROM Programmer command (CPR, PSTAT, PTYPE, RPR, or WPR).

**PP ERROR—CHAR MOD ABSENT.** There is no characteristic module inserted in the front panel of the PROM Programmer.

**PP ERROR—CONFIG MEM CKSUM.** The characteristic module memory shows an error. Call your service representative. Since the PROM Programmer Controller is not malfunctioning, you should still be able to use a different characteristic module.

**PP ERROR—DEV NOT ERASED.** A WPR (write) command was attempted, but the device is not erased for correct programming. See the discussion of the WPR command elsewhere in this supplement for more information on the erasing of a device.

**PP ERROR—DEV NOT REPROG.** After writing to a programmable device, the PROM Programmer compares the data in the device with program memory to ensure that the data was correctly transmitted. The message "DEV NOT REPROG" indicates that an attempt was made to write to a device, and the results in the device memory do not match program memory. This may indicate a bad part, or an attempt to use the -N modifier with the WPR command on an unerased section of device memory. See the discussion of the WPR command elsewhere in this supplement.

**PP ERROR—POWER FAIL.** The operation was aborted due to an internal power failure. Try the operation again. If the POWER FAIL message appears frequently, call your service representative.

**PP ERROR—PROM POWER OFF.** The power switch on the PROM Programmer front panel assembly must be turned on before PROM Programmer commands can execute.

**Error 80 PROM type not supported.** The **promtype** parameter entered with a CPR, RPR, or WPR command is not recognized by this characteristic module. For a list of devices supported and their promtype designations, see Table 8B-1 or enter the PTYPE command. Note that the promtype name must be entered exactly as shown. For example, 8748/i or 8748/INTEL are not accepted for the 8748/l promtype.

## Power Up Errors

Assuming you have powered up the PROM Programmer following the instructions in the introduction to this section, the READY indicator on the PROM Programmer front panel should light. If the READY indicator does not light, type in the PSTAT command. You may receive one of the responses listed above, or one of the following messages:

```
PP ERROR-- CPU
PP ERROR-- POWER CONTROL
PP ERROR-- RAM
PP ERROR-- ROM
PP ERROR-- SUPPLY MONITOR
PP ERROR-- SYST INTERFACE
```

If you receive the message NO PROM PROGRAMMER ERRORS DETECTED, the PROM Programmer should be functioning correctly. The problem may be with the READY indicator. Try executing a command.

The other error messages indicate a problem in the hardware or firmware of the PROM Programmer Controller. Call your service representative for assistance.



This manual supports the following TEKTRONIX product:

8550F33

This manual supports a software/firmware module that is compatible with:

DOS/50 Version 1 (8550)  
DOS/50 Version 2 (8550)  
OS/40 Version 1 (8540)

**PLEASE CHECK FOR CHANGE INFORMATION  
AT THE REAR OF THIS MANUAL.**

**8500**  
MODULAR MDL SERIES  
**2764**  
**PROM PROGRAMMER  
SPECIFICS  
USERS MANUAL**

Tektronix, Inc.  
P.O. Box 500  
Beaverton, Oregon 97077

Serial Number \_\_\_\_\_

## LIMITED RIGHTS LEGEND

Software License No. \_\_\_\_\_

Contractor: Tektronix, Inc.

Explanation of Limited Rights Data Identification Method

Used: Entire document subject to limited rights.


Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

## RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Copyright © 1982 Tektronix, Inc. All rights reserved. Contents of this publication may not be reproduced in any form without the written permission of Tektronix, Inc.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and  are registered trademarks of Tektronix, Inc. TELEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Printed in U.S.A. Specification and price change privileges are reserved.

---

## SECTION 8C

### 2764 PROM PROGRAMMER SPECIFICS

	Page
<b>Introduction</b> .....	8C-1
<b>Devices Supported</b> .....	8C-2
<b>Demonstration Run</b> .....	8C-3
Creating the Test Data .....	8C-3
Programming the Target Device .....	8C-4
Reading Back the Test Data .....	8C-4
Summary .....	8C-5
<b>PROM Programmer Commands</b> .....	8C-6
CPR—Compares PROM With Memory .....	8C-7
PSTAT—Reports PP Status .....	8C-9
PTYPE—Displays PROM Types .....	8C-10
RPR—Reads PROM .....	8C-11
WPR—Writes PROM .....	8C-14
<b>Error Messages</b> .....	8C-17
Execution Errors .....	8C-17
Errors Detected by the PROM Programmer .....	8C-17
Errors Detected by the Operating System .....	8C-18
Power Up Errors .....	8C-18

## Section 8C

# 2764 PROM PROGRAMMER SPECIFICS

### INTRODUCTION

This supplement is designed to be inserted into one of the following sections:

- Section 8 of the 8540 System Users Manual,
- Section 8 of the 8550 System Users Manual (DOS/50 Version 2), or
- Section 9 of the 8550 System Users Manual (DOS/50 Version 1).

Throughout this supplement, the phrase “your System Users Manual” refers to the 8550 System Users Manual or 8540 System Users Manual. The 2764 PROM Programmer Module is referred to as simply “the 2764 Module”.



*The 2764 Module is static-sensitive. To avoid damage to the module, handle it as little as possible in high-static situations. Pick up the module by the body, not by the edge connectors. Avoid sliding the module over any surface.*

This supplement discusses the following topics:

- The devices supported by the 2764 Module.
- A demonstration run for the 2764 Module.
- PROM Programmer commands, as they apply to the 2764 Module.
- Error messages.



## DEVICES SUPPORTED

The 2764 Module enables the PROM Programmer to program a set of similar devices. Each of the PROMs supported by the 2764 Module is a MOS, single supply, byte-wide chip. Table 8C-1 lists the device types supported by 2764 Module.

**Table 8C-1**  
**Programmable Devices Supported by the 2764 Module**

Manufacturer	Device	PROM Type <sup>a</sup>	Description
Texas Instruments	TMS 2564	2564/TI	8K x 8 bit EPROM
Intel	2764	2764/I	8K x 8 bit EPROM
Intel	27128	27128/I	16K x 8 bit EPROM
Intel	2817	2817/I	2K x 8 bit EEPROM (This device includes programming control logic.)

<sup>a</sup> The name used in a PROM Programmer command to specify the device.

Other PROMs, similar to the devices listed in Table 8C-1, may be available from other manufacturers. If you are using such a device, verify that its programming specifications are identical to those of a listed device. If the specifications are identical, the PROM Programmer, with the 2764 Module installed, will properly program the device. Use the appropriate PROM type (from Table 8C-1) in your programming commands.

If you attempt to use a PROM that is not supported by this PROM Programmer module, you will receive an error message:

```
> CFR 0 3232/Z 0 40 (specifying a Zilog 3232 PROM)
CFR: Error 80 PROM type not supported
```



*The PROM Programmer does not verify that the inserted device matches the PROM type that you specified. The voltage levels necessary to successfully program a PROM vary from one type to another. If you specify the wrong PROM type, your PROM may be destroyed.*

## DEMONSTRATION RUN

This demonstration run provides you with an introduction to PROM programming, using the 2764 Module. This demonstration may also be used to verify the module's operation.

The demonstration run consists of three steps. First, you'll load some test data into program memory. Then, you'll direct the PROM Programmer to program this test data into your target device. Finally, you'll read back the data to verify that it was correctly programmed into the PROM.

This demonstration run uses the following conventions:

1. Underlined—Underlined characters in a command line must be entered from your system terminal. Those characters not underlined are system output.
2. <CR>—Each command line ends with a carriage return. When a carriage return is to be entered, the symbol <CR> is used.

## Creating the Test Data

The test data for this demonstration run is in the form of an ASCII string. First, clear part of the development system memory by filling it with zeros:

```
> F 00 00FF 00 <CR>
```

Now enter the test data, using the following command (omit the -A modifier if you're using DOS/50 Version 1):

```
> F 00 00FF -A "TIME FOR COFFEE!" <CR>
```

Using the D (Dump) command, examine the contents of memory and verify that the test data was correctly entered:

```
> D 00 00FF <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0010  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0020  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0030  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0040  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0050  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0060  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0070  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0080  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0090  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00A0  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00B0  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00C0  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00D0  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00E0  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00F0  54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
```

## Programming the Target Device

Plug the 2764 Module into the opening on your development system's front panel. Next, raise the 2764 Module's ZIF socket handle, and install your target PROM into the ZIF socket. Align the PROM so that pin 1 is next to the ZIF socket handle.



*Be careful not to touch the pins of either your target device or the ZIF socket. Damage from static discharge may result.*

Now, turn the PROM POWER switch on. The READY indicator on the PROM Programmer front panel assembly should now be lit. (If it is not lit, a problem exists in your PROM Programmer Controller; contact a Tektronix field service representative.)

Now you can program the test data into your target PROM by entering one of the following programming commands. Select the command that is appropriate for your device type.

- > WPR 0 2764/I 00 00FF <CR> (for 2764 PROMs)
- > WPR 0 27128/I 00 00FF <CR> (for 27128 PROMs)
- > WPR 0 2564/II 00 00FF <CR> (for 2564 PROMs)
- > WPR -N 0 2817/I 00 00FF <CR> (for 2817 PROMs)

Remember to include the -N modifier in the command if your target device is a 2817. Otherwise, the PROM Programmer may think that the device is an unerasable EPROM.

Your target device should now contain the test data.

## Reading Back the Test Data

Now it's time to verify that the test data was accurately programmed in your PROM. First, fill the development system memory again with zeros:

```
> F 00 00FF 00 <CR>
```

Next, enter one of the following commands to read the contents of the PROM into memory. Select the command that is appropriate for your device type.

- > RPR 0 2764/I 00 00FF <CR> (for 2764 PROMs)
- > RPR 0 27128/I 00 00FF <CR> (for 27128 PROMs)
- > RPR 0 2564/I 00 00FF <CR> (for 2564 PROMs)
- > RPR 0 2817/I 00 00FF <CR> (for 2817 PROMs)

Locations 00—FF of memory should now contain your test data again. To find out, enter the following command:

```
> D 00 00FF <CR>
  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
0010 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
0020 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
0030 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
0040 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
0050 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
0060 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
0070 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
0080 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
0090 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
00A0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
00B0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
00C0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
00D0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
00E0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
00F0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21 TIME FOR COFFEE!
```

If this display appears, the PROM was programmed correctly, and the 2764 Module is operational.

If the display does not appear as shown here, either your target PROM or the 2764 Module is defective. Try the demonstration again, using a different target PROM to isolate the problem. If the module is defective, contact a Tektronix field service representative.

## Summary

This demonstration introduced you to the basic commands necessary to program PROMs with the 2764 Module. The following discussion explains these and other PROM Programmer commands in more detail.

The demonstration run also verified the 2764 Module's operation. If you suspect that the module may not be operating properly, contact a Tektronix field service representative.

## PROM PROGRAMMER COMMANDS

Five operating system commands are used with the PROM Programmer:

- **CPR** (Compare PROM) compares the contents of the specified PROM addresses with the corresponding data in program memory, and displays any differences.
- **RPR** (Read PROM) reads the contents of the specified PROM addresses into program memory.
- **WPR** (Write PROM) writes data from program memory into the PROM.
- **PSTAT** (PROM STATus) displays the current status of the PROM Programmer, including any error conditions.
- **PTYPE** (PROM TYPE) displays information about the PROM Programmer module currently installed in the front panel of the PROM Programmer: what PROMs are supported by the module, and what their characteristics are.

This subsection gives the syntax and use of each PROM Programmer command as it pertains to the 2764 Module. The format is the same as that of the Command Dictionary of your 8550 or 8540 System Users Manual.

### NOTE

*Syntax for this section corresponds to DOS/50 Version 2 and OS/40. If you are using DOS/50 Version 1, all "dash modifiers" (such as -L, -M, etc.) should be "colon modifiers" (:L, :M, etc.). The Technical Notes section of the 8550 System Users Manual describes the differences between DOS/50 Version 1 and Version 2.*

### SYNTAX

```
cpr [-a] [-d] [-m] [-l] [-r] memlo promtype promlo promhi
```

### PARAMETERS

- a            Provides inverted addressing (hi=0, low=1). Use -A in applications where the levels applied to the PROM's address pins are inverted.
- d            Data is stored in the PROM in inverted form (hi=0, low=1). For a compare operation (CPR), this means that the data from the PROM is inverted before being compared with program memory.
- l            For the 8-bit PROMs supported by the 2764 Module, this modifier indicates that data being read from the PROM represents the least significant bytes of 16-bit words in program memory. See also -R.
- m            For 8-bit PROMs, this modifier indicates that data being read from the PROM represents the most significant bytes of 16-bit words in program memory. See also -R.
- r            When the data in program memory is arranged in 16-bit words, indicates that the bytes are arranged in reverse order in program memory; that is, the least significant byte is first, followed by the most significant byte.  
  
The -R modifier is used with -M or -L. For example, if modifiers -M and -R are entered as part of a CPR command when reading an 8-bit PROM, the bytes read from the PROM will be compared with the second byte of each 16-bit word.
- memlo        Specifies the starting address to which data is placed after being read from the PROM.
- promtype     The alphanumeric name specifying the PROM with which you are working. To determine the PROM type associated with your chip, see Table 8C-1 (earlier in this section), or use the PTYPE command. The **promtype** parameter must be **entered** exactly as shown; letters must be uppercase.
- promlo        Specifies the PROM address that you wish to be the starting point for the specified command.
- promhi        Specifies the ending PROM address.

## EXPLANATION

The CPR command compares the contents of the specified PROM addresses with the contents of program memory, starting at the given memory address. The locations of any differences are displayed. If the contents match, no message is displayed.

The CPR command is useful when you want to verify that the data in the PROM is the same as the data in a specified region of program memory.

## EXAMPLES

Enter the following command to read the contents of locations 0—40 of an Intel 2764 PROM into locations 0—40 of program memory.

```
> RPR 0 2764/I 0 40
```

Now dump program memory to see what's been read:

```
> D 0 40
  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0010 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0020 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0030 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0040 54                                     T
```

If the data was read correctly, a CPR command at this point will not return a message:

```
> CPR 0 2764/I 0 40
```

Let's see what happens when an error is detected. Use the P command to patch new data into two locations in program memory.

```
> P 05 00
> P 07 00
```

Now compare the contents again:

```
> CPR 0 2764/I 0 40
*** COMPARE ERROR AT PROM ADDRESS 00005H
*** COMPARE ERROR AT PROM ADDRESS 00007H
```

The CPR command identified the discrepancies introduced into locations 5 and 7 of program memory.

## NOTE

*If you include the -A modifier in the CPR command line, the address given in a "COMPARE ERROR" message is actually the complement of the value that was applied at the PROM's address pins.*

<b>SYNTAX</b>
<b>pstat</b>

### EXPLANATION

The PSTAT command displays the current condition of the PROM Programmer. Normally, PSTAT will report "no errors". Conditions that prompt an error message include power failure, the absence of a PROM Programmer module, or PROM Programmer power off.

Error messages are discussed later in this section.

### EXAMPLES

When no errors are present:

```
> PSTAT
... NO PROM PROGRAMMER ERRORS DETECTED ...
```

Reporting an error condition:

```
> PSTAT
PP ERROR-- POWER FAIL
```



**SYNTAX****ptype****EXPLANATION**

The PTYPE command displays information about PROMs supported by the 2764 Module. The display consists of four columns:

- **PROM TYPE:** the name that specifies each PROM supported by the PROM Programmer module. Each name is the model number of the primary source manufacturer, followed by a slash and one or more letters specifying the manufacturer. For example, the Intel 2764 PROM is identified as 2764/I. (Note that there are no spaces in the PROM TYPE designation.)
- **MAX ADDRESS:** the maximum addressable location of each PROM listed. For example, 1FFFH indicates a total of 2000H addressable bytes (00—1FFF).
- **ERASED STATE:** the state of the bits (high or low) when the PROM device has been erased. For all PROMs supported by the 2764 Module, HI is FF and LOW is 00.
- **WORD LENGTH:** the number of bits per word for each PROM. Since the 2764 Module supports only byte-wide (8-bit) PROMs, the word length for all PROMs listed is 8.

**EXAMPLES**

```
> PTYPE
PROM TYPE      MAX ADDRESS    ERASED STATE    WORD LENGTH
2764/I          1FFFH          HI              8
27128/I         3FFFH          HI              8
2817/I          7FFFH          HI              8
2564/II         1FFFH          HI              8
```

## SYNTAX

```
rpr [-a] [-d] [-m] [-l] [-r] memlo promtype promlo promhi
```

## PARAMETERS

- a** Provides inverted addressing (hi=0, low=1). Use **-A** in applications where the levels applied to the PROM's address pins are inverted.
- d** Data is stored in the PROM in inverted form (hi=0, low=1). For a read operation (RPR), this means that the data from the PROM is inverted before being stored in program memory.
- l** For the 8-bit PROMs supported by the 2764 Module, this modifier indicates that data being read from the PROM represents the least significant bytes of 16-bit words in program memory. See also **-R**.
- m** For 8-bit PROMs, this modifier indicates that data being read from PROM represents the most significant bytes of 16-bit words in program memory. See also **-R**.
- r** When the data in program memory is arranged in 16-bit words, indicates that the bytes are arranged in reverse order in program memory; that is, the least significant byte is first, followed by the most significant byte.
- The **-R** modifier is used with **-M** or **-L**. For example, if modifiers **-M** and **-R** are entered as part of a RPR command when reading an 8-bit PROM, the bytes read from the PROM will be placed in the second byte of each 16-bit word.
- memlo** Specifies the starting address to which data is placed after being read from the PROM.
- promtype** The alphanumeric name specifying the PROM with which you are working. To determine the PROM type associated with your chip, see Table 8C-1 (earlier in this section), or use the PTYPE command. The **promtype** parameter must be entered exactly as shown; letters must be uppercase.
- promlo** Specifies the PROM address that you wish to be the starting point for the specified command.
- promhi** Specifies the ending PROM address.

**EXPLANATION**

The RPR command reads the contents of the specified PROM addresses into the specified location in program memory. No message is displayed unless an error is detected.

**EXAMPLES**

First, use the F command to fill program memory with the data byte EE. Later, when you examine memory, the EEs will make it easy to see that new data was read in.

```
> F 0 40 EE
> D 0 40
  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0010 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0020 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0030 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0040 EE
```

Now use the RPR command to read the contents of memory locations 0—40 of a Texas Instruments 2564 PROM into program memory, beginning at address 0. Then dump the contents of program memory, to verify that the memory now contains new data from the PROM.

```
> RPR 0 2564/TI 0 40
> D 0 40
  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0010 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0020 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0030 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0040 54
```

Let's assume that your 8-bit PROM is to be used as the most significant half of a 16-bit memory system. Again, fill program memory with EEs, read the PROM, and dump the results. This time, however, include the -M modifier in the RPR command line.

```
> F 0 40 EE
> RPR -M 0 2564/TI 0 40
> D 0 40
  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 EE T.H.I.S. .I.S. .
0010 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E EE A. .T.E.S.T.....
0020 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 EE T.H.I.S. .I.S. .
0030 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E EE A. .T.E.S.T.....
0040 54
```

The D command display shows that the 8-bit bytes received from the PROM are now stored in program memory in the most significant byte positions of 16-bit data words.

Now see what happens when the RPR command line includes the -L modifier:

```
> F O 40 EE
> RPR -L O 2564/TI O 40
> D O 40
  O 1 2 3 4 5 6 7 8 9 A B C D E F
0000 EE 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 .T.H.I.S. .I.S.
0010 EE 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E .A. .T.E.S.T....
0020 EE 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 .T.H.I.S. .I.S.
0030 EE 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E .A. .T.E.S.T....
0040 EE
```

Notice that this time, bytes from the PROM are placed in the least significant byte positions of 16-bit words.

In the next example, assume that program memory is organized as 16-bit words, but now the least significant byte of a word **precedes** the most significant byte. You want the data in the PROM to go into program memory as the most significant byte of each word. Note that -R should be used only with either a -L or -M modifier.

```
> F O 40 EE
> RPR -MR O 2564/TI O 40
> D O 40
  O 1 2 3 4 5 6 7 8 9 A B C D E F
0000 EE 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 .T.H.I.S. .I.S.
0010 EE 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E .A. .T.E.S.T....
0020 EE 54 EE 48 EE 49 EE 53 EE 20 EE 49 EE 53 EE 20 .T.H.I.S. .I.S.
0030 EE 41 EE 20 EE 54 EE 45 EE 53 EE 54 EE 2E EE 2E .A. .T.E.S.T....
0040 EE
```

If you wanted the PROM data in this example to go into the least significant byte of program memory, you would have entered "-LR" in the RPR command line.

### SYNTAX

```
wpr [-a] [-d] [-m] [-i] [-r] [-n] memlo promtype promlo promhi
```

### PARAMETERS

- a Provides inverted addressing (hi=0, low=1). Use -A in applications where the levels applied to the PROM's address pins are inverted.
- d Data is stored in the PROM in inverted form (hi=0, low=1). During a write (WPR), data in program memory is inverted before being written into PROM.
- i For the 8-bit PROMs supported by the 2764 Module, this modifier indicates that data being written into the PROM represents the least significant bytes of 16-bit words in program memory. See also -R.
- m For 8-bit PROMs, this modifier indicates that data being written into the PROM represents the most significant bytes of 16-bit words in program memory. See also -R.
- n Specifies that no erase check is performed before a write operation.
- r When the data in program memory is arranged in 16-bit words, indicates that the bytes are arranged in reverse order in program memory; that is, the least significant byte is first, followed by the most significant byte.  
  
The -R modifier is used with -M or -L. For example, if modifiers -M and -R are entered as part of a WPR command when programming an 8-bit PROM, the bytes written to the PROM will be taken from the second byte of each 16-bit word in program memory.
- memlo Specifies the starting address from which data is transferred to the PROM.
- promtype The alphanumeric name specifying the PROM with which you are working. To determine the PROM type associated with your chip, see Table 8C-1 (earlier in this section), or use the PTYPE command. The **promtype** parameter must be **entered** exactly as shown; letters must be uppercase.
- promlo Specifies the PROM address that you wish to be the starting point for the specified command.
- promhi Specifies the ending PROM address.

## EXPLANATION

The WPR command writes the contents of the specified locations in program memory into the specified PROM addresses. The programming routine verifies that each byte is programmed correctly. No message is displayed unless an error is detected. A location must be erased before it can be programmed with new data.

The Intel 2817 EEPROM (promtype 2817/I) is electrically erasable. If you specify the entire EEPROM address space in the WPR command, then the entire chip is automatically erased. Otherwise, only those portions of the EEPROM between **promlo** and **promhi** are automatically erased. To manually erase individual bytes of the 2817, write FF into the appropriate memory locations. You **must** use the -N modifier when programming a 2817/I type PROM.

All other PROMs supported by the 2764 Module are EPROMs—that is, they must be erased by exposure to ultraviolet light. Before writing to an EPROM, the PROM Programmer checks the designated addresses in the PROM to ensure that the device has been erased and is ready to accept new information. You may disable this erase check by using the -N command modifier.

### NOTE

*The PROM Programmer has no capability for erasing EPROMS. Erasure must be performed by a unit designed for this purpose.*

## EXAMPLES

Assume that you have 512 bytes of code for an 8-bit microprocessor in program memory, starting at address 00. To program this code into the bottom addresses of an Intel 2764 device, enter the following command:

```
>WPR 00 2764/I 0 1FF
```

Suppose now that you have the following application. The memory system for your 16-bit micro-computer uses inverting buffers for both address and data lines. You want to program 12K words of code into your PROMs. Within your development system, the most significant byte of each 16-bit word is located at an even address.

To burn your code into PROMs, you'll need two PROM devices: one for the most significant byte of each word, and one for the least significant byte. First, program the least-significant-byte PROM:

```
>WPR -ADL 00 27128/I 0 2FFF
```

Now, program the most-significant-byte PROM:

```
>WPR -ADM 00 27128/I 0 2FFF
```

Notice that in both examples, **memlo** was specified as 00. When programming 16-bit data into 8-bit PROMs, you only need to specify the starting address of the 16-bit word. The -M and -L modifiers automatically take care of any byte displacements.

Sometimes you may discover an error in your code after you have burned it into a PROM. If you're using an electrically-erasable device, such an error would be easy to correct. But if you're using an EPROM, it may take 30 minutes or longer to erase and reprogram the device; such delays are annoying when you're working under tight schedules. In many cases, though, these delays can be eliminated.

Suppose you read the contents of your recently-programmed PROM back into program memory, and see the following dump:

```
> RPR 00 2764/I 00 10
> D 0 10
   0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 54 48 4D 53 20 49 53 20 49 20 54 45 53 54 2E 2E THMS IS A TEST..
0010 54                                     T
```

At first glance, it appears that you'll have to erase and reprogram the PROM. However, a little more thought reveals a quick solution.

Let's look at the binary representation of the ASCII characters "M" and "I". They differ only by one bit:

```
M = 0100 1101
I = 0100 1001
```

Although you can't change any bit in an EPROM from a 0 to a 1 without erasing the entire device, you **can** change a 1 to a 0. In this case, changing a 1 to a 0 can turn the "M" (at address 02) into an "I":

```
> P 02 49
> WPR -N 00 2764/I 0 10
```

Notice that the -N modifier is used to inhibit the erase check of the device.

Obviously, not all errors in a PROM's data can be corrected using this procedure, but the technique may be useful when a small error is detected.

## ERROR MESSAGES

PROM Programmer error messages are identified by the keywords "PP ERROR." They are divided into two categories: execution errors and power-up errors.

**Execution errors** may appear in response to a PROM Programmer command (CPR, PSTAT, PTYPE, RPR, or WPR) during normal operation. They indicate that the PROM Programmer is unable to execute the command. This condition may be easily fixed (by turning on PROM Programmer power, for example), or it may require a call to your Tektronix service representative.

**Power-up errors** describe problems discovered by the PROM Programmer self-test procedure, executed automatically during power up.

If you see an error message without the words "PP ERROR," refer to the section on Error Messages in your System Users Manual.

### Execution Errors

The following error messages may be received in response to a PROM Programmer command (CPR, PSTAT, PTYPE, RPR, or WPR). Some errors are detected by the PROM Programmer, and are reported by the operating system. Other errors are detected by the operating system directly.

#### Errors Detected by the PROM Programmer

**PP ERROR—CHAR MOD ABSENT.** There is no PROM Programmer module inserted in the front panel of the PROM Programmer.

**PP ERROR—CONFIG MEM CKSUM.** The PROM Programmer module ROM is defective. Contact your Tektronix field service representative. Since the PROM Programmer Controller is not malfunctioning, try using a different PROM Programmer module.

**PP ERROR—DEV NOT ERASED.** A WPR (write) command was attempted, but the PROM is not erased for correct programming. See the discussion of the WPR command elsewhere in this section for more information.

**PP ERROR—DEV NOT REPROG.** An attempt was made to write to a device, and the results in the PROM do not match program memory. This may indicate a bad component, or an attempt to use the -N modifier with the WPR command on an unerased section of device memory. See the discussion of the WPR command elsewhere in this section for more information.

**PP ERROR—POWER FAIL.** The operation was aborted due to an internal power failure. Try the operation again. If the POWER FAIL message persists, contact your Tektronix field service representative.

**PP ERROR—PROM POWER OFF.** The power switch on the PROM Programmer front panel assembly must be turned on before PROM Programmer commands can execute.



## Errors Detected by the Operating System

**Error 80 PROM type not supported.** The **promtype** parameter entered with a CPR, RPR, or WPR command is not recognized by this PROM Programmer module. For a list of PROMs supported and their **promtype** designations, see Table 8C-1 (earlier in this section) or enter the PTYPE command. Note that the **promtype** name must be entered exactly as shown. For example, 2764/i or 2764/INTEL are not accepted for the 2764/I promtype.

**Error 81 Maximum PROM address exceeded.** Either the **memlo** or **memhi** parameter you entered in a CPR, RPR, or WPR command exceeded the specified device's address range.

**Error 82 PROM Programmer PTYPE data error.** The data specifying the PROM Programmer module's device support was corrupted when transferred from the PROM Programmer to the development system's operating system. This error indicates some hardware fault in the PROM Programmer. Contact your Tektronix field service representative.

**Error 84 PROM Programmer hardware I/O error.** The interface between the development system and the PROM Programmer malfunctioned. If this error persists, contact your Tektronix field service representative.

## Power Up Errors

If, after you power up the PROM Programmer according to the instructions in Section 8 of this manual, the READY indicator on the PROM Programmer front panel is not illuminated, or if the SELF TEST indicator is illuminated, a power-up error occurred. When this condition exists, the PSTAT command will report one of the following errors:

```
PP ERROR-- CPU
PP ERROR-- POWER CONTROL
PP ERROR-- RAM
PP ERROR-- ROM
PP ERROR-- SUPPLY MONITOR
PP ERROR-- SYST INTERFACE
```

These errors are hardware failures in the PROM Programmer Controller. Contact your Tektronix field service representative to have the PROM Programmer repaired.

# Tektronix®

COMMITTED TO EXCELLENCE

This manual supports the following TEKTRONIX products:

8550 Option	8540 Option	Product
35	35	8550F35

This manual supports a software/firmware module that is compatible with:

DOS/50 Version 1 (8550)  
DOS/50 Version 2 (8550)  
OS/40 Version 1 (8540)

**PLEASE CHECK FOR CHANGE INFORMATION  
AT THE REAR OF THIS MANUAL**

**8500**  
MODULAR MDL SERIES  
**8751**  
PROM PROGRAMMER  
SPECIFICS  
USERS MANUAL

Tektronix, Inc.  
P.O. Box 500  
Beaverton, Oregon 97077

070-4414-00  
Product Group 61

Serial Number \_\_\_\_\_

First Printing AUG 1982

## LIMITED RIGHTS LEGEND

Software License No. \_\_\_\_\_

Contractor: Tektronix, Inc.

Explanation of Limited Rights Data Identification Method

Used: Entire document subject to limited rights.


Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

## RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Copyright © 1982 Tektronix, Inc. All rights reserved. Contents of this publication may not be reproduced in any form without the written permission of Tektronix, Inc.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and  are registered trademarks of Tektronix, Inc. TELEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Printed in U.S.A. Specification and price change privileges are reserved.

## Section 8D

# 8751 PROM PROGRAMMER SPECIFICS

	Page
<b>Introduction</b> .....	8D-1
<b>Devices Supported</b> .....	8D-2
<b>Demonstration Run</b> .....	8D-2
Creating the Test Data .....	8D-2
Programming the Target Device .....	8D-3
Reading Back the Test Data .....	8D-4
Summary .....	8D-5
<b>PROM Programmer Commands</b> .....	8D-5
CPR—Compares PROM With Memory .....	8D-6
PSTAT—Reports PP Status .....	8D-8
PTYPE—Displays PROM Types .....	8D-9
RPR—Reads PROM .....	8D-10
WPR—Writes to PROM .....	8D-12
<b>Error Messages</b> .....	8D-14
Execution Errors .....	8D-14
Errors Detected by the PROM Programmer .....	8D-15
Errors Detected by the Operating System .....	8D-15
Power-Up Errors .....	8D-16

## Section 8D

# 8751 PROM PROGRAMMER SPECIFICS

### INTRODUCTION

This supplement provides information that enables you to use the 8751 PROM Programmer Module. The following topics are discussed:

- Devices supported by the 8751 PROM Programmer Module.
- A demonstration run for the 8751 PROM Programmer Module.
- PROM Programmer commands, as they apply to the 8751 PROM Programmer Module.
- Error messages.

This supplement is designed to be inserted into one of the following sections:

- Section 8 of the 8540 System Users Manual,
- Section 8 of the 8550 System Users Manual (DOS/50 Version 2), or
- Section 9 of the 8550 System Users Manual (DOS/50 Version 1).

Throughout this supplement, the term "your System Users Manual" refers to either the 8550 System Users Manual or 8540 System Users Manual. The 8751 PROM Programmer Module is referred to as simply the "8751 module".



**CAUTION**

*The 8751 module is static-sensitive. To avoid damage to the module, handle it as little as possible in high-static situations. Pick up the module by the body, not by the edge connectors. Avoid sliding the module over any surface.*

## DEVICES SUPPORTED

The 8751 module enables the PROM Programmer to program the Intel 8751 microcomputer. The 8751 contains a 4K x 8 EPROM for program memory.

Other microcomputers, similar to the Intel 8751, may be available from other manufacturers. If you are using such a device, verify that its programming specifications are identical to those of the Intel 8751. If the specifications are identical, the PROM Programmer, with the 8751 module installed, will properly program the device.

## DEMONSTRATION RUN

This demonstration run provides you with an introduction to PROM programming, using the 8751 module. This demonstration may also be used to verify the module's operation.

The demonstration run consists of three steps. First, you'll load some test data into program memory. Then, you'll direct the PROM Programmer to program this test data into the target device. Finally, you'll read back the data to verify that it was correctly programmed into the target device.

This demonstration run uses the following conventions:

1. Underlined—Underlined characters in a command line must be entered from your system terminal. Those characters not underlined are system output.
2. <CR>—Each command line ends with a carriage return. When a carriage return is to be entered, the symbol <CR> is used.

## Creating the Test Data

The test data for this demonstration run is in the form of an ASCII string. First, clear part of the development system memory by filling it with zeros:

```
> F 00 00FF 00 <CR>
```

Now enter the test data, using the following command (omit the -A modifier if you're using DOS/50 Version 1):

```
> F 00 00FF -A "TIME FOR COFFEE!" <CR>
```

Using the D (Dump) command, examine the contents of memory and verify that the test data was correctly entered:

```
> D 00 00FF <CR>
```

```

      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0010 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0020 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0030 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0040 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0050 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0060 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0070 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0080 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0090 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00A0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00B0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00C0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00D0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00E0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00F0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!

```

## Programming the Target Device

Plug the 8751 module into the PROM Programmer opening on your development system's front panel. Next, raise the 8751 module's ZIF socket handle, and install your target microcomputer into the ZIF socket. Align the device so that pin 1 is next to the ZIF socket handle.

### CAUTION

*Be careful not to touch the pins of either your target device or the ZIF socket. Damage from static discharge may result.*

Now, turn the PROM POWER switch on. The READY indicator on the PROM Programmer front panel assembly should now be lit. (If it is not lit, a problem exists in your PROM Programmer Controller; contact a Tektronix field service representative.)

Now you can program the test data into your target device by entering the following programming command:

```
> WPR 0 8751/I 00 00FF <CR>
```

Your target device should now contain the test data.

## Reading Back the Test Data

Now it's time to verify that the test data was accurately programmed in your target device. First, fill the development system memory again with zeros:

```
> F 00 00FF 00 <CR>
```

Next, enter the following command to read the contents of the microcomputer's PROM into memory:

```
> RPR 0 8751/I 00 00FF <CR>
```

Locations 00–FF of memory should now contain your test data again. To find out, enter the following command:

```
> D 00 00FF <CR>
```

```

      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0010 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0020 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0030 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0040 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0050 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0060 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0070 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0080 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
0090 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00A0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00B0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00C0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00D0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00E0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!
00F0 54 49 4D 45 20 46 4F 52 20 43 4F 46 46 45 45 21  TIME FOR COFFEE!

```

If this display appears, the PROM in the 8751 microcomputer was programmed correctly, and the 8751 module is operational.

If the display does not appear as shown here, either your target device or the 8751 module is defective. Try the demonstration again, using a different target device to isolate the problem. If the module is defective, contact a Tektronix field service representative.



## Summary

This demonstration introduced you to the basic commands necessary to program 8751 micro-computers with the 8751 module. The following discussion explains these and other PROM Programmer commands in more detail.

The demonstration run also verified the 8751 module's operation. If you suspect that the module may not be operating properly, contact a Tektronix field service representative.

## PROM PROGRAMMER COMMANDS

Five operating system commands are used with the PROM Programmer:

- **CPR** (Compare PROM) compares the contents of the specified PROM addresses with the corresponding data in program memory, and displays any differences.
- **RPR** (Read PROM) reads the contents of the specified PROM addresses into program memory.
- **WPR** (Write PROM) writes data from program memory into the PROM.
- **PSTAT** (PROM STATus) displays the current status of the PROM Programmer, including any error conditions.
- **PTYPE** (PROM TYPE) displays information about the PROM Programmer module currently installed in the front panel of the PROM Programmer: what PROMs or devices are supported by the module, and what their characteristics are.

The following syntax descriptions give the syntax and use of each PROM Programmer command as it pertains to the 8751 module. The format is the same as that of the Command Dictionary of your 8550 or 8540 System Users Manual.

### NOTE

*Syntax for this section corresponds to DOS/50 Version 2 and OS/40. If you are using DOS/50 Version 1, all "dash modifiers" (such as -L, -M, etc.) should be "colon modifiers" (:L, :M, etc.). The Technical Notes section of the 8550 System Users Manual (DOS/50 Version 2) describes the differences between DOS/50 Version 1 and Version 2.*

**SYNTAX**

`cpr [-a] [-d] [  $\begin{matrix} -m \\ -l \end{matrix}$ ] [-r] memlo promtype promlo promhi`

**PARAMETERS**

- a Provides inverted addressing (hi=0, low=1). Not used with the 8751 module.
- d Data is stored in the PROM in inverted form (hi=0, low=1). For a compare operation (CPR), this means that the data from the PROM is inverted before being compared with program memory.
- l The data being read from the PROM represents the least significant bytes of 16-bit words in program memory. Not used with the 8751 module.
- m The data being read from the PROM represents the most significant bytes of 16-bit words in program memory. Not used with the 8751 module.
- r When the data in program memory is arranged in 16-bit words, indicates that the bytes are arranged in reverse order in program memory. Not used with the 8751 module.
- memlo Specifies the starting address to which data is placed after being read from the PROM.
- promtype The alphanumeric name specifying the PROM with which you are working. The 8751 module only supports the **promtype** 8751/I, specifying the Intel 8751 micro-computer. The **promtype** parameter must be **entered** exactly as shown; the letter I must be uppercase.
- promlo Specifies the PROM address that you want to be the starting point for the CPR command.
- promhi Specifies the ending PROM address.

**EXPLANATION**

The CPR command compares the contents of the specified PROM addresses with the contents of program memory, starting at the given memory address. The locations of any differences are displayed. If the contents match, no message is displayed.

The CPR command is useful when you want to verify that the data in the PROM is the same as the data in a specified region of program memory.

### EXAMPLES

Enter the following command to read the contents of locations 0–40 of an Intel 8751's EPROM into locations 0–40 of program memory:

```
> RPR 0 8751/I 0 40 <CR>
```

Now dump program memory to see what's been read:

```
> D 0 40 <CR>
```

```

      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0010 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0020 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0030 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0040 54                                     T

```

If the data was read correctly, a CPR command at this point will not return a message:

```
> CPR 0 8751/I 0 40 <CR>
```

Let's see what happens when an error is detected. Use the P command to patch new data into two locations in program memory.

```
> P 05 00 <CR>
```

```
> P 07 00 <CR>
```

Now compare the contents again:

```
> CPR 0 8751/I 0 40 <CR>
*** COMPARE ERROR AT PROM ADDRESS 00005H
*** COMPARE ERROR AT PROM ADDRESS 00007H

```

The CPR command identified the discrepancies introduced into locations 5 and 7 of program memory.

**SYNTAX**

**pstat**

**EXPLANATION**

The PSTAT command displays the current condition of the PROM Programmer. Normally, PSTAT will report "no errors". Conditions that prompt an error message include power failure, the absence of a PROM Programmer module, or PROM Programmer power off.

Error messages are discussed later in this section.

**EXAMPLES**

When no errors are present:

```
> PSTAT <CR>  
... NO PROM PROGRAMMER ERRORS DETECTED ...
```

Reporting an error condition:

```
> PSTAT <CR>  
PP ERROR-- POWER FAIL
```

**SYNTAX****ptype****EXPLANATION**

The PTYPE command displays information about devices supported by the 8751 module. The display consists of four columns:

- **PROM TYPE:** the name that specifies each device supported by the PROM Programmer module. Each name is the model number of the primary source manufacturer, followed by a slash and one or more letters specifying the manufacturer. The 8751 module supports only the Intel 8751 microcomputer. The PROM TYPE is 8751/I. (Note that there are no spaces in the PROM TYPE designation.)
- **MAX ADDRESS:** the maximum addressable location of the 8751's internal EPROM.
- **ERASED STATE:** the state of the bits (high or low) when the PROM device has been erased.
- **WORD LENGTH:** the number of bits per word for each PROM.

**EXAMPLES**

```
> PTYPE <CR>
```

PROM TYPE	MAX ADDRESS	ERASED STATE	WORD LENGTH
8751/I	OFFFH	HI	8
160-1831-00			

### SYNTAX

`rpr [-a ] [-d ] [  $\begin{matrix} -m \\ -l \end{matrix}$  ] [-r ] memlo promtype promlo promhi`

### PARAMETERS

- a Provides inverted addressing (hi=0, low=1). Not used with the 8751 module.
- d Data is stored in the PROM in inverted form (hi=0, low=1). For a read operation (RPR), this means that the data from the PROM is inverted before being stored in program memory.
- l The data being read from the PROM represents the least significant bytes of 16-bit words in program memory. Not used with the 8751 module.
- m The data being read from the PROM represents the most significant bytes of 16-bit words in program memory. Not used with the 8751 module.
- r When the data in program memory is arranged in 16-bit words, indicates that the bytes are arranged in reverse order in program memory. Not used with the 8751 module.
- memlo Specifies the starting address to which data is placed after being read from the PROM.
- promtype The alphanumeric name specifying the PROM with which you are working. The 8751 module only supports the **promtype** 8751/l, specifying the Intel 8751 micro-computer. The **promtype** parameter must be **entered** exactly as shown; the letter l must be uppercase.
- promlo Specifies the PROM address that you want to be the starting point for the RPR command.
- promhi Specifies the ending PROM address.

## EXPLANATION

The RPR command reads the contents of the specified PROM addresses into the specified program memory locations. No message is displayed unless an error is detected.

## EXAMPLES

First, use the F command to fill program memory with the data byte EE. Later, when you examine memory, the EEs will make it easy to see that new data was read in.

```
> F 0 40 EE <CR>
```

```
> D 0 40 <CR>
```

```

      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0010 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0020 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0030 EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE .....
0040 EE

```

Now use the RPR command to read the contents of memory locations 0–40 of an 8751's EPROM into program memory, beginning at address 0. Then dump the contents of program memory, to verify that the memory now contains new data from the PROM.

```
> RPR 0 8751/I 0 40 <CR>
```

```
> D 0 40 <CR>
```

```

      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0010 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0020 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0030 54 48 49 53 20 49 53 20 41 20 54 45 53 54 2E 2E THIS IS A TEST..
0040 54

```

**SYNTAX**

**wpr** [-a ] [-d ]  $\left[ \begin{array}{l} -m \\ -l \end{array} \right]$  [-r ] [-n ] memlo promtype promlo promhi

**PARAMETERS**

- a Provides inverted addressing (hi=0, low=1). Not used with the 8751 module.
- d Data is stored in the PROM in inverted form (hi=0, low=1).
- l The data being written to the PROM represents the least significant bytes of 16-bit words in program memory. Not used with the 8751 module.
- m The data being written to the PROM represents the most significant bytes of 16-bit words in program memory. Not used with the 8751 module.
- n Specifies that no erase check is performed before a write operation.
- r When the data in program memory is arranged in 16-bit words, indicates that the bytes are arranged in reverse order in program memory. Not used with the 8751 module.
- memlo Specifies the starting address from which data is written to the PROM.
- promtype The alphanumeric name specifying the PROM with which you are working. The 8751 module only supports the **promtype** 8751/I, specifying the Intel 8751 micro-computer. The **promtype** parameter must be **entered** exactly as shown; the letter I must be uppercase.
- promlo Specifies the PROM address that you want to be the starting point for the WRP command.
- promhi Specifies the ending PROM address.

**EXPLANATION**

The WPR command writes the contents of the specified program memory locations into the specified PROM addresses. The programming routine verifies that each byte is programmed correctly. No message is displayed unless an error is detected.

A location within the 8751's EPROM must be erased by exposure to ultraviolet light before it can be programmed with new data. Before programming the EPROM, the PROM Programmer checks the designated addresses in the EPROM to ensure that the device has been erased and is ready to accept new information. You may disable this erase check by using the -N command modifier.



*NOTE*

*The PROM Programmer has no capability for erasing EPROMs. Erasure must be performed by a unit designed for this purpose.*

**EXAMPLES**

Assume that you have 512 bytes of 8751 code in program memory, starting at address 00. To program this code into the bottom addresses of an Intel 8751 device, enter the following command:

```
> WPR 00 8751/I 0 1FF <CR>
```

Suppose now that you have the following design problem. A peripheral output device connected to your 8751 microcomputer has logically-inverted inputs. Your program contains a 256-byte data table, starting at address 100 in program memory, that may be output to this device. The data table will also start at location 100 within the 8751.

Your original design had the microcomputer invert the data before sending it to the peripheral device, but this inversion degraded your prototype's performance. The -D modifier in the WPR command offers a solution to this problem: each data byte is logically inverted before the byte is programmed into a PROM.

Using the -D modifier in your WPR command line, you can store the data table in inverted form. Thus, the microcomputer is relieved of the inversion task, and prototype performance improves to an acceptable level. The command looks like this:

```
> WPR -D 100 8751/I 100 1FF <CR>
```

Sometimes you may discover an error in your code after you have burned it into the 8751's EPROM. Normally, you would spend 30 minutes or longer to erase and reprogram the device; such delays are annoying when you're working under tight schedules. In many cases, though, these delays can be eliminated.

Suppose you read the contents of the recently-programmed 8751 EPROM back into program memory, and see the following dump:

```
> RPR 00 8751/I 00 10 <CR>
```

```
> D 0 10 <CR>
```

```

  0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 54 48 4D 53 20 49 53 20 49 20 54 45 53 54 2E 2E  THMS IS A TEST..
0010 54
                                     T
```

At first glance, it appears that you'll have to erase and reprogram the EPROM. However, a little more thought reveals a quick solution.

Let's look at the binary representation of the ASCII characters "M" and "I". They differ only by one bit:

```
M = 0100 1101
I = 0100 1001
```

Although you can't change any bit in an EPROM from a 0 to a 1 without erasing the entire device, you **can** change a 1 to a 0. In this case, changing a 1 to a 0 can turn the "M" (at address 02) into an "I":

```
> P 02 49 <CR>
```

```
> WPR -N 00 8751/I 0 10 <CR>
```

Notice that the -N modifier is used to inhibit the erase check of the device. Obviously, not all errors in the EPROM's data can be corrected using this procedure, but the technique may be useful when a small error is detected.

## ERROR MESSAGES

PROM Programmer error messages are identified by the keywords "PP ERROR." They are divided into two categories: execution errors and power-up errors.

**Execution errors** may appear in response to a PROM Programmer command (CPR, PSTAT, PTYPE, RPR, or WPR) during normal operation. They indicate that the PROM Programmer is unable to execute the command. This condition may be easily fixed (by turning on PROM Programmer power, for example), or it may require a call to your Tektronix service representative.

**Power-up errors** describe problems discovered by the PROM Programmer self-test procedure, executed automatically during power-up.

If you see an error message without the words "PP ERROR," refer to the section on Error Messages in your System Users Manual.

### Execution Errors

The following error messages may be received in response to a PROM Programmer command (CPR, PSTAT, PTYPE, RPR, or WPR). Some errors are detected by the PROM Programmer, and are reported by the operating system. Other errors are detected by the operating system directly.

### Errors Detected by the PROM Programmer

**PP ERROR—CHAR MOD ABSENT.** There is no PROM Programmer module inserted in the front panel of the PROM Programmer.

**PP ERROR—CONFIG MEM CKSUM.** The PROM Programmer module ROM is defective. Contact your Tektronix field service representative. Since the PROM Programmer Controller is not malfunctioning, try using a different PROM Programmer module.

**PP ERROR—DEV NOT ERASED.** A WPR (write) command was attempted, but the PROM is not erased for correct programming. See the discussion of the WPR command elsewhere in this section for more information.

**PP ERROR—DEV NOT REPROG.** An attempt was made to write to a device, and the results in the PROM do not match program memory. This may indicate a bad component, or an attempt to use the -N modifier with the WPR command on an unerased section of device memory. See the discussion of the WPR command elsewhere in this section for more information.

**PP ERROR—POWER FAIL.** The operation was aborted due to an internal power failure. Try the operation again. If the POWER FAIL message persists, contact your Tektronix field service representative.

**PP ERROR—PROM POWER OFF.** The power switch on the PROM Programmer front panel assembly must be turned on before PROM Programmer commands can execute.

### Errors Detected by the Operating System

**Error 80 PROM type not supported.** The **promtype** parameter entered with a CPR, RPR, or WPR command is not recognized by this PROM Programmer module. Note that the **promtype** name must be entered exactly as shown. For example, 8751/i or 8751/INTEL are not accepted for the 8751/I promtype.

**Error 81 Maximum PROM address exceeded.** Either the **memlo** or **memhi** parameter you entered in a CPR, RPR, or WPR command exceeded the specified device's address range.

**Error 82 PROM Programmer PTYPE data error.** The data specifying the PROM Programmer module's device support was corrupted when transferred from the PROM Programmer to the development system's operating system. This error indicates some hardware fault in the PROM Programmer. Contact your Tektronix field service representative.

**Error 84 PROM Programmer hardware I/O error.** The interface between the development system and the PROM Programmer malfunctioned. If this error persists, contact your Tektronix field service representative.

## Power-Up Errors

If, after you power up the PROM Programmer according to the instructions in Section 8 of this manual, the READY indicator on the PROM Programmer front panel is not illuminated, or if the SELF TEST indicator is illuminated, a power-up error has occurred. When this condition exists, the PSTAT command will report one of the following errors:

```
PP ERROR-- CPU
PP ERROR-- POWER CONTROL
PP ERROR-- RAM
PP ERROR-- ROM
PP ERROR-- SUPPLY MONITOR
PP ERROR-- SYST INTERFACE
```

These error messages indicate hardware failures in the PROM Programmer Controller. Contact your Tektronix field service representative to have the PROM Programmer repaired.

## Section 9

# TECHNICAL NOTES

### NOTE 1. PRACTICAL LIMITS OF EMULATION

No available emulator exactly emulates the target microprocessor. Connecting test circuitry to a device always has some loading effect on the device you are testing. Engineers have accepted this fact with regard to voltmeters, oscilloscopes, and other test instruments. Similarly, when you connect a control probe to a prototype circuit, you are adding circuitry that allows you to monitor and control the prototype. This added circuitry has an effect, if only a slight one, on the performance of the prototype. Here are some considerations to bear in mind when using an emulator:

- **Load and Drive Parameters.** We have chosen to use LSTTL buffers between the user's prototype and the emulating microprocessor. The consequence is that, where most microprocessors have MOS-type inputs and outputs, our emulator has LSTTL inputs and outputs. The designer of the target system must keep this in mind when dealing with load and drive considerations.
- **Noise.** Another parameter that is directly affected by emulation is noise. Since the user's circuitry is being driven via a cable from the prototype control probe, the signals will necessarily be noisier than when the microprocessor is in the circuit. Further, existing noise in the prototype circuit may be aggravated by the cable.
- **Timing.** Finally, because of propagation delays through cables and buffers, the timing of certain signals at the probe plug may differ slightly from those measured at the microprocessor itself.

In all cases, the variations mentioned are within the worst case specifications of the microprocessor being emulated. You should remember, though, that these variations do exist. If you encounter problems under emulation and not with the microprocessor plugged in, or vice versa, check your design for possible marginal timing, loading, or noise conditions.

### NOTE 2. COMPARISON OF OS/40 VERSION 1 WITH DOS/50 VERSION 2

This Technical Note compares the 8540 operating system (OS/40 Version 1) and the 8550 operating system (DOS/50 Version 2). The two operating systems are similar except for a few commands, functions related to file handling, and RTPA support.

#### Commands

- All 8540 commands except PERMSTR and ROMPATCH are also valid on the 8550.
- All commands that are shared by the 8540 and the 8550 are identical except for the CONFIG command.

### File Handling Considerations

The 8540 relies upon its host computer to do file handling tasks, unlike the 8550, which handles its own files. As a result, the 8540 performs differently in the following areas:

- **Command set.** The following 8550 commands are not supported by the 8540:

ACE	COP	FD	L	SYS
ASM	CRC	FL	LINK	SYSPATCH
ATT	CREATE	FORMAT	LIBGEN	T
BR	DAT	HELP	MDL	USER
CMPF	DEL	HOST	PRINT	VERIFY
CON	DUP	K	REN	

- **SVCs.** All files used by 8540 SVCs must reside on the host (for example, the 8560). All files used by 8550 SVCs must reside on the 8550.

The 8540 does not support SVCs 06, 18, 21, 48-4C, 4E, 58, and 59.

On the 8540, Overwrite SVCs are implemented as Write SVCs, and I/O-and-Proceed SVCs are implemented as I/O-and-Wait SVCs.

- **intersystem Communication.** The 8540 cannot be a host computer.
- **Software Installation.** On the 8540, software is installed via ROM. On the 8550, software is installed via disk.

### Support of TTA, RTPA

The 8540 supports **only** the Trigger Trace Analyzer (TTA). The 8550 supports both the TTA and the Real-Time Trace Analyzer (RTPA).

## NOTE 3—COMPARISON OF THE TTA AND THE RTPA

This technical note compares the Trigger Trace Analyzer and the Real-Time Prototype Analyzer. Refer to your TTA Users Manual, the Operating Procedures Section of this manual, and the Emulation Section of this manual for more information about the TTA.

**Differences**

- a. The TTA can monitor up to 24 address lines. The RTPA can monitor up to 16 address lines.
- b. The smallest time unit that the TTA can count is 200 ns. The smallest time unit the RTPA can count is 1  $\mu$ s.
- c. The TTA's Acquisition Memory can hold 255 operations. The RTPA's Real-Time Trace buffer can hold 128 operations.
- d. The TTA has 4 event/counter/trigger sets. The RTPA has 2 event/counter/trigger sets.
- e. The event/counter/trigger relationships of the TTA are much more flexible than those of the RTPA's. As a result, the TTA's TS command has no counterpart in the RTPA.
- f. Each TTA trigger can key on a range of addresses or data. The RTPA can key on a single range but requires both event/counter/triggers to do it. similarities

**Similarities**

- a. The following functions are similar on both instruments: tracing, counting, triggers, and breakpoints.
- b. Both the TTA and the RTPA can use the following input data: address bus, data bus, control lines, and data acquisition probe clips.
- c. The TTA's EVE command performs many of the same functions as the RTPA's EVT command.
- d. The TTA's AD, BUS, DATA, and PRO commands are analogous to the parameters A, B, D, and T of the RTPA's EVT command.
- e. The TTA's ACQ command performs the functions of the RTPA's RT command and has additional features.
- f. The TTA's DISP command is similar to the RTPA's DRT command.

## Section 10

# ERROR MESSAGES

This section describes error messages which are generated by OS/40 or issued by the 8540 upon request from the host. Other Tektronix software products that run on the 8560 or 8550 can generate error messages which are documented in other manuals. Table 10-1 describes the general format of each type of error message, and indicates the appropriate reference document.

### NOTE

*Any of these codes may be returned as an SRB status code after an SVC. Refer to the Service Calls section of this manual for information regarding SRB status codes.*

**Table 10-1**  
**Summary of Error Message Types**

Source	Order	Case	Format	Reference Document
8540/8550	numeric, hexadecimal	lower	CMD: Error nn Message	8540 or 8550 System Users Manual Error Messages section
8560	alphabetic by command	lower	cmd: Message cmd: Message-Detail	8560 System Users Manual Message: refer to the generating command Detail: see Error Messages section
A Series Assembler	numeric	lower	****ERROR nnn:Message	8550 Assembler Core Users Manual (A Series) nnn = 000-199, Error Messages section nnn = 200-255, Assembler Specifics
A Series Linker	alphabetic	upper	MESSAGE	8550 Assembler Core Users Manual (A Series) The Linker section
A Series Library Generator	alphabetic	upper	MESSAGE	8550 Assembler Core Users Manual (A Series) The Library Generator section
B Series Assembler	numeric	lower	ASM: nnn (s) Message *** ASM: nnn (s) Message	8550 Assembler Core Users Manual (B Series) nnn = 000-199, Error Messages section nnn = 200-255, Assembler Specifics
B Series Linker	numeric	lower	LINK:nnn (s) Message ***LINK:nnn (s) Message	8550 Assembler Core Users Manual (B Series) Error Messages section nnn = 000-099, command processing errors nnn = 100-199, errors during link
B Series Library Generator	numeric	lower	LIBGEN:nnn (s) Message *** LIBGEN:nnn (s) Message	8550 Assembler Core Users Manual (B Series) nnn = 000-199, Error Messages section nnn = 200-255, Assembler Specifics
8550 DOS/50 Editor	alphabetic	upper	MESSAGE	8550 Editor Users Manual Error Messages section
ACE	alphabetic	upper	MESSAGE	8500 Advanced CRT-Oriented Editor Users Manual
MDL/ $\mu$	alphabetic	lower	Message	8550 MDL/ $\mu$ Compiler Users Manual Error Messages section
PROM Programmer	alphabetic by function group	upper	PP ERROR—MESSAGE FATAL PP ERROR—MESSAGE	PROM Programmer Specifics supplement for the characteristic module; see the heading, Error Conditions



**02—Illegal channel number.** The channel number used was outside the range 0-9.

**03—I/O channel not open.** An I/O operation has been attempted on a channel that has not been assigned to a file or device.

**05—Illegal function for device.** An attempt has been made to perform an illegal function, such as reading from the line printer or writing to the paper tape reader.

**06—Short or unterminated read.** The number of bytes read was less than the number of bytes requested, or a carriage return was not detected. This is the normal status when the last (short) block of a file is read.

**07—Short or unterminated write.** A carriage return was not detected in the specified number of characters to be written.

**0A—Device not operational.** Power to the device is on, but for some reason it cannot function properly. For example, the line printer may be out of paper.

**0C—Device not ready.** The 8540 cannot communicate with the device you want to access (for example, the host computer). Be sure that the device is powered up and connected to the 8540.

**0D—Device in use.** An attempt has been made to assign a non-shareable device on a second channel. Only CON1 and CON0 are shareable.

**10—Error reading disk bit map.** An error in reading a disk bit map has been detected. Host-dependent SVC error.

**11—I/O error or access violation on read.** An error on REM1, CON1, or PPTR has been detected, or you do not have read access to the directory or file on the host.

**12—I/O error or access violation on write.** An error on REM0, CON0, or PPTP has been detected, or you do not have write access to the directory or file on the host.

**13—Command not found.** OS/40 does not recognize the command name. Be sure that the proper emulator software has been installed and selected.

**16—Illegal file specification.** A filespec may be an incorrect length or contain illegal characters. May be host-dependent.

**17—Illegal SVC function code.** An SVC code has been specified that does not exist or is not available to the user.

**18—Channel already open.** Self-explanatory.

**19—Volume or disk not found.** A file has been specified on a volume that is not mounted on the host. Host-dependent SVC error.

**1B—Checksum error.** A checksum error in a load file has been detected by the LO or RH command.

**1C—Command area in use.** Commands may not be executed simultaneously.

**1D—File not found.** An attempt has been made to open a file for reading or writing, but the file either does not exist, or exists in a protected directory. Host-dependent SVC error.

**1E—Invalid parameter.** OS/40 does not recognize one of the parameters entered in a command line. Check the required format and parameters in the Command Dictionary.

**1F—No header on load file.** An attempt has been made to load a file that is not in load module format.

**20—Invalid input parameter.** An invalid input parameter has been entered. Refer to the Command Dictionary for the correct syntax.

**21—Invalid output parameter.** An invalid output parameter has been entered. Refer to the Command Dictionary for the correct syntax.

**22—No transfer address.** This status code is returned when you use SVC 17 or 57 to load a program that does not have a transfer address.

**23—Command buffer overflow.** The command line entered (or the resulting line after all string and parameter substitution) must be less than or equal to 80 characters, including the carriage return.

**24—Symbol table full.** An attempt has been made to load more symbols into the symbol table (with COM, SYMLO, or ADDS) than allowed. The number of symbols possible is about 1000, depending on symbol length. Use REMS to remove symbols.

**25—Data format error.** The input data is not in the format expected by the command.

**26—No emulator in system.** The emulator hardware is not installed, or the emulator software is not SElected.

**27—Numeric parameter out of range.** Either the clock count or the number of lines to be printed has been set to a value that is outside the range 0-64K.

**28—System interface error.** Serious problems have been detected in the host interface. Check your 8560 or host cable connections and interface baud rate. If this problem continues, contact your Tektronix service representative.

**29—Seek error on file I/O.** Host-dependent file error.

- 2A—Parameter required.** A command line has been entered that requires another parameter. Check the required parameters in the Command Dictionary.
- 2B—Too many parameters.** A command line has been entered that contains more parameters than required. Check the required format in the Command Dictionary.
- 2C—Invalid address parameter.** An address parameter may contain numbers, register names, symbols, memory space designators, don't-care values, and the operators + and -. Refer to the Command Dictionary for more information regarding valid address expressions.
- 2E—System must be idle to SElect.** The system must not have any active commands or programs. SEL destroys the current program debugging status.
- 2F—Buffer overflow on HSI operation.** This error indicates serious hardware or software problems. The 8540 Installation Guide contains verification procedures.
- 30—Invalid address range.** An invalid address range has been entered. The high address must be greater than or equal to the low address.
- 32—Too many trace ranges.** An attempt has been made to set a fourth TRA command. Only three TRA commands may be active at one time. Check the Command Dictionary for correct parameters.
- 34—Command not active.** An attempt has been made to abort or suspend an inactive command.
- 35—Command not suspended.** An attempt has been made to continue a command that is executing or that has finished executing.
- 37—Invalid hexadecimal string.** The commands EX, F, and P accept a hexadecimal string. Refer to the Command Dictionary for the requirements of a hexadecimal string.
- 38—Missing close quotation mark.** A quoted string must have opening and closing quotation marks.
- 39—No emulator selected.** An attempt has been made to execute an emulator-specific command before an emulator has been selected.
- 3A—Bad acknowledge.** The valid acknowledgments are "0" (ASCII 30H) for ACK and "7" (ASCII 37H) for NAK.
- 3B—Transfer aborted.** A data transfer between the 8540 and the host was incomplete, so the transfer was aborted. Check cable connections, and power switches and connections. If the problem persists, run the system verification described in the 8540 Installation Guide and/or have a qualified service representative run diagnostics on the systems communications hardware as described in the 8540 Service Manual.
- 3C—Memory write error.** Data written to memory could not be read back. The memory may be ROM, or there may be no RAM for the addresses specified.
- 3E—Invalid memory space designator.** A memory space designator (such as SC:) has been incorrectly entered.
- 3F—Illegal use of don't-care expression.** A don't-care expression has been used where a unique value is required.
- 40—Memory space designator illegal in expr.** A memory space designator has been used in a parameter that does not allow memory space designators. For example, in a pair of parameters that represent an address range, only the first may contain a memory space designator.
- 41—Word not filled.** The string entered with the F or P command did not fill an even number of bytes.
- 42—Invalid use of multiple memory spaces.** Multiple memory spaces can only be used with the commands listed under MEMSP in the Command Dictionary.
- 43—No Trigger Trace Analyzer in system.** The TTA hardware has not been installed.
- 44—Emulator clock missing.** Make sure the prototype power switch is on and that the prototype clock is functioning correctly.
- 45—Emulator faulted.** The emulator is malfunctioning. Error code 5F may also occur under this condition. Have a qualified service representative check the hardware and fuses, and clean the board contacts. Make sure that the prototype power switch is on, that the prototype clock is functioning properly, that the entered commands are valid, and that the instructions being executed are legal.
- 46—Odd word address.** An attempt has been made to modify an area of memory that does not begin on a word boundary.
- 47—Byte not filled.** An uneven number of hexadecimal digits has been entered in an attempt to modify memory with the F or P command. You cannot modify a half byte.
- 48—Port has no carrier signal.** No carrier signal has been detected at the REMI/REMO port.
- 49—Port parity error.** A parity error has been detected at the REMI/REMO port.
- 4A—Port framing error.** A framing error has been detected at the REMI/REMO port. That is, the number of start, stop, and data bits received was not what was expected. The COM -M command can be used to set the desired framing.
- 4B—All job streams active.** The system is busy. Commands cannot be entered until one of the currently executing commands is finished.

**4C—Invalid section name.** The section does not exist, or the section name contains an illegal character or is too long.

**4D—PROM power failure.** The power to the PROM Programmer has failed.

**52—Command busy.** An attempt was made to enter a command again when it was already executing.

**53—Symbol not found.** An attempt has been made to access a symbol that does not exist or has been entered incorrectly. Valid symbols include register names and any program symbols you have placed in the symbol table. Be sure to include a leading zero on any hexadecimal number that starts with a letter, and on any number that begins with an X (don't-care).

**54—Invalid symbol.** An invalid character has been detected in a symbol, or the symbol is too long.

**55—Symbol value not alterable.** The symbol to which you are attempting to assign a value with the S command is not alterable.

**56—Truncation error.** An attempt has been made to assign too large a value to a symbol or register, or the hex string specified in the F (Fill) command does not evenly fill the specified memory area.

**57—Invalid arithmetic operator.** The only valid arithmetic operators are + and - .

**58—Invalid term in expression.** An invalid character or character string has been detected in an expression.

**59—Overflow in expression.** The resulting value of an expression (or some intermediate value obtained in evaluating it) is too large. Expressions are evaluated using 32-bit arithmetic.

**5A—Invalid dash modifier.** Dash modifiers must be letters only. Refer to the Command Dictionary for the dash modifiers accepted by each command.

**5C—Too many files open.** No more than eight channels to files can be simultaneously open by commands and user programs. Enter the command **A -A** to close all channels.

**5D—Bad character in number.** The valid digits are 0-9, A-F, and the standard suffixes: H (hexadecimal), T (decimal), O or Q (octal), and Y (binary).

**5F—Emulator halted.** The emulator halted while a program was executing. Error code 45 (Emulator faulted) may also occur under this condition. Make sure that the appropriate emulator board is installed and the instructions entered are valid.

**60—Emulator SVC synchronization error.** A serious emulator software error has been detected. Reboot and reselect the emulator. If the problem persists, contact your Tektronix service representative.

**61—Disk protected against writing.** A disk on the host is write-protected. Host-dependent SVC error.

**63—Disk not formatted.** Host-dependent SVC error.

**64—Disk CRC error.** A bad block (parity error) has been detected on a disk on the host. Host-dependent SVC error.

**65—Disk full.** There are no blocks available for allocation on this host disk, or there are no free files. Host-dependent SVC error.

**66—System synchronization error.** The 8540 and the host have serious interface problems. Reboot your system and try again. Contact your Tektronix service representative if problems continue.

**67—Exclusive access conflict.** An attempt has been made to open a host file or device that is already being used exclusively by another process. Host-dependent SVC error.

**69—Too many channels open.** Serious hardware and/or software problems have been detected. Reboot your system and try again. Contact your Tektronix service representative if problems continue.

**6A—Disk structure corrupt.** Serious problems have been detected in the file structure of your host's disk. Host-dependent SVC error.

**6B—Current user is not file owner.** An attempt has been made to access a file on the host to which you do not have access privileges. Host-dependent SVC error.

**6E—Directory alteration invalid.** An attempt has been made to create, delete, or rename a file on the host in a directory to which you do not have write access, or to create a duplicate name within a directory. Host-dependent SVC error.

**6F—Invalid file linkage attempt.** Host-dependent SVC error.

**70—File full.** The file structure of a volume on the host is full. Host-dependent SVC error.

**71—String already exists.** Using the PERMSTR command, an attempt has been made to store a permanent string in EEPROM when a string by that name is already stored.

**74—Program memory jumped incorrectly.** Using the 68000 or Z8001, the SElect command cannot set up the MAC board properly since program memory has been strapped so that addresses do not have a unique location.

**76—ASCII read to CONI terminated by CTRL-C.** CTRL-C was typed while OS/40 was performing a read operation. The data read is not valid.

**77—Emulator SVC request outstanding.** A prior SVC is still in progress. Your current request will be filled when the first SVC has finished.

**78—No more program memory available.** An attempt has been made to ALlocate program memory when there is none available. Use DEAL to deallocate memory.

**79—Program memory address not allocated.** In order for you to access the specified address range, memory must be allocated to it using the AL command.

**7A—Program memory address already allocated.** An attempt has been made to ALlocate an address that has already been allocated.

**7B—String not found.** OS/40 does not recognize the string name entered. Check that you have spelled the name correctly. Use the STR -L command to list temporary strings and/or the PERMSTR -L command to list permanent strings.

**7C—String area full.** The capacity of either the temporary or the permanent string area has been exceeded. Use the STR or the PERMSTR command to delete unused strings.

**7D—System memory parity error.** Reboot the system. If the problem persists, contact your Tektronix service representative.

**7E—Error in command execution.** A command has been executed that detected errors but continued.

**80—PROM type not supported.** The characteristic module currently installed does not support the PROM type entered.

**81—Maximum PROM address exceeded.** The address given is negative, or exceeds the maximum for the PROM type.

**82—PROM Programmer PTYPE data error.** System error. The PROM type specified in a RPR, WPR, or CPR command is not supported by the characteristic module currently installed.

**83—Modifier required.** The command entered needs a dash modifier.

**84—PROM Programmer hardware I/O error.** System hardware error. Contact your Tektronix service representative if the error persists.

**85—Allocation hardware disabled.** Program memory allocation hardware error. Contact your Tektronix service representative if the problem continues.

**87—Invalid trigger number.** There are four TTA triggers, numbered 1-4.

**88—Signals cannot occur simultaneously.** Using either the TTA, or a Z8001/Z8002, 8086, or 68000 emulator, an attempt has been made to set an event or breakpoint on bus signals that are mutually exclusive (such as a read and a write on the same line).

**8A—Invalid event linkage (wraparound).** Using CONS, an attempt has been made to completely link together all events. Thus, no trigger can occur.

**8B—Restart requires gate option.** In the COU command, the restart option must be used in conjunction with a gate option.

**8C—Restart/gate not available on trigger 1.** The trigger 1 hardware has no prior trigger channel from which to gate.

**8D—No section contains specified address.** The ADDS command requires that any address you add be contained within a previously defined program section. Use the COM or SYMLO command to download the section definition information into the symbol table from your load file.

**8E—Segmentation trap pending.** Either your program or a system program has attempted to access memory which is invalid according to the Z8000 Memory Management Unit. Refer to the Z8001/Z8002 Emulator Specifics supplement for further information.

**8F—User memory declared non-existent.** An attempt has been made to access memory which was declared non-existent with the NOMEM command. Check memory declarations with the MEM or NOMEM command. If the problem persists after checking your program, check your MAC board.

**90—Invalid arming mode.** The -A arming modifier needs two programmed breakpoints, but only one is currently programmed. This error occurs only when using an emulator such as the Z8001/Z8002, 8086, or 68000.

**91—Invalid initial value for counter.** Zero is an invalid initial value for decrementing counter modes.

**92—No such label or scalar.** An attempt has been made to remove a symbol which is non-existent or is a section name.

93—**invalid symbol specification.** In a symbol specification of the form section:label, either the section name is too long, the label name is too long, or the specification contains an illegal character.

94—**No prototype control probe attached.** An attempt was made to change from emulation mode 0 to mode 1 or 2, but the prototype control probe is not connected to the emulator and the prototype. This error occurs only when using the 8086 emulator.

95—**Prototype not ready.** When the emulating microprocessor attempted to access prototype memory, the prototype held the READY line "not ready" for too many wait states. Check the prototype. This error occurs only when using the 8086 emulator. The number of wait states allowed is jumper-selected.

96—**Prototype bus hang.** The prototype has held the bus for an inordinate length of time. This is considered a "hung" condition; detection of this condition is jumper-selectable. The timeout is on the order of one-half second. You may hold the bus for any length of time if the jumper is not used. Check the prototype. This error occurs only when using the 8086 emulator.

D4—**Internal parse error.** Serious software errors have been detected. Contact your Tektronix service representative if this problem continues.

D7—**Internal term error.** Serious software errors have been detected. Contact your Tektronix service representative if this problem continues.

E1—**Emulator double fault or odd stack pointer.** On the 68000, the emulator has halted during a user job. Possible causes are a double address or bus error, or an odd system stack pointer. Reset the registers and check the program and prototype.

E2—**Processor registers changed.** Following a 68000 processor halt, the emulator had to reset the PC, SSP, and SR registers before all the registers were saved.

E4—**Emulator system error.** Unknown emulator error. Reboot and reselect. Contact your Tektronix service representative if this problem continues.

E6—**No MAC board in system.** No Memory Allocation Controller board has been installed.

E7—**System error on MAC board.** Unknown system error. Reboot and reselect. If the problem persists, contact your Tektronix service representative.

FE—**Process aborted.** This message is returned when the A command is used.

FF—**End of file.** Returned on a Read SVC if the file was at end of file before the read occurred. May be host-dependent.

## Section 11 TABLES

	Page
<b>Conversion Tables</b> .....	11-1
<b>Tektronix Hexadecimal Format (Tekhex)</b> .....	11-4
Standard Tekhex Format .....	11-4
Extended Tekhex Format .....	11-6
<b>Motorola Load Module Format</b> .....	11-8
<b>Intel Load Module Format</b> .....	11-10

### Tables

**Table  
No.**

11-1	ASCII-Binary-Hexadecimal-Decimal Conversion .....	11-1
11-2	Hexadecimal Multiplication .....	11-2
11-3	Decimal-Hexadecimal-Binary Equivalents .....	11-3
11-4	Standard Tekhex Data Block Format .....	11-4
11-5	Standard Tekhex Termination Block Format .....	11-5
11-6	Standard Tekhex Abort Block Format .....	11-5
11-7	Extended Tekhex Header Field .....	11-6
11-8	Character Values for Checksum Computation .....	11-6
11-9	Extended Tekhex Data Block Format .....	11-6
11-10	Extended Tekhex Termination Block Format .....	11-7
11-11	Extended Tekhex Symbol Block Format .....	11-7
11-12	Extended Tekhex Symbol Block: Section Definition Field .....	11-7
11-13	Extended Tekhex Symbol Block: Symbol Definition Field .....	11-7
11-14	Motorola Load Module Format .....	11-8
11-15	Intel Load Module Format .....	11-10

# Section 11

## TABLES

### CONVERSION TABLES

Table 11-1  
ASCII-Binary-Hexadecimal-Decimal Conversion

B I T S				0		1		1		1		1	
				CONTROL		SYMBOLS		UPPERCASE		LOWERCASE			
				0	10	20	30	40	50	60	70	80	90
0	0	0	0	NUL	DLE	SP	ø	@	P	\	p		
0	0	0	1	SOH	DC1	!	1	A	Q	a	q		
0	0	1	0	STX	DC2	"	2	B	R	b	r		
0	0	1	1	ETX	DC3	#	3	C	S	c	s		
0	1	0	0	EOT	DC4	\$	4	D	T	d	t		
0	1	0	1	ENQ	NAK	%	5	E	U	e	u		
0	1	1	0	ACK	SYN	&	6	F	V	f	v		
0	1	1	1	BEL BELL	ETB	'	7	G	W	g	w		
1	0	0	0	BS BACK SPACE	CAN	(	8	H	X	h	x		
1	0	0	1	HT	EM	)	9	I	Y	i	y		
1	0	1	0	LF	SUB	*	:	J	Z	j	z		
1	0	1	1	VT	ESC	+	;	K	[	k	{		
1	1	0	0	FF	FS	,	<	L	\	l	:		
1	1	0	1	CR RETURN	GS	-	=	M	]	m	}		
1	1	1	0	SO	RS	.	>	N	^	n			
1	1	1	1	SI	US	/	?	O	_	o	DEL		

Example:            ASCII            Binary            Hexadecimal            Decimal  
                          <                    011 1100            3C                    60

Table 11-2  
Hexadecimal Multiplication

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Example

HEX	9 x 8	=	48	HEX
HEX	40	=	64	DEC
HEX	8	=	8	DEC
HEX	48	=	72	DEC



**Table 11-3**  
**Decimal-Hexadecimal-Binary Equivalents**

Decimal	Hexadecimal	Binary 8-bit Code	Decimal	Hexadecimal	Binary 8-bit Code	Decimal	Hexadecimal	Binary 8-bit Code	Decimal	Hexadecimal	Binary 8-bit Code
0	00	0000 0000	64	40	0100 0000	128	80	1000 0000	192	C0	1100 0000
1	01	0000 0001	65	41	0100 0001	129	81	1000 0001	193	C1	1100 0001
2	02	0000 0010	66	42	0100 0010	130	82	1000 0010	194	C2	1100 0010
3	03	0000 0011	67	43	0100 0011	131	83	1000 0011	195	C3	1100 0011
4	04	0000 0100	68	44	0100 0100	132	84	1000 0100	196	C4	1100 0100
5	05	0000 0101	69	45	0100 0101	133	85	1000 0101	197	C5	1100 0101
6	06	0000 0110	70	46	0100 0110	134	86	1000 0110	198	C6	1100 0110
7	07	0000 0111	71	47	0100 0111	135	87	1000 0111	199	C7	1100 0111
8	08	0000 1000	72	48	0100 1000	136	88	1000 1000	200	C8	1100 1000
9	09	0000 1001	73	49	0100 1001	137	89	1000 1001	201	C9	1100 1001
10	0A	0000 1010	74	4A	0100 1010	138	8A	1000 1010	202	CA	1100 1010
11	0B	0000 1011	75	4B	0100 1011	139	8B	1000 1011	203	CB	1100 1011
12	0C	0000 1100	76	4C	0100 1100	140	8C	1000 1100	204	CC	1100 1100
13	0D	0000 1101	77	4D	0100 1101	141	8D	1000 1101	205	CD	1100 1101
14	0E	0000 1110	78	4E	0100 1110	142	8E	1000 1110	206	CE	1100 1110
15	0F	0000 1111	79	4F	0100 1111	143	8F	1000 1111	207	CF	1100 1111
16	10	0001 0000	80	50	0101 0000	144	90	1001 0000	208	D0	1101 0000
17	11	0001 0001	81	51	0101 0001	145	91	1001 0001	209	D1	1101 0001
18	12	0001 0010	82	52	0101 0010	146	92	1001 0010	210	D2	1101 0010
19	13	0001 0011	83	53	0101 0011	147	93	1001 0011	211	D3	1101 0011
20	14	0001 0100	84	54	0101 0100	148	94	1001 0100	212	D4	1101 0100
21	15	0001 0101	85	55	0101 0101	149	95	1001 0101	213	D5	1101 0101
22	16	0001 0110	86	56	0101 0110	150	96	1001 0110	214	D6	1101 0110
23	17	0001 0111	87	57	0101 0111	151	97	1001 0111	215	D7	1101 0111
24	18	0001 1000	88	58	0101 1000	152	98	1001 1000	216	D8	1101 1000
25	19	0001 1001	89	59	0101 1001	153	99	1001 1001	217	D9	1101 1001
26	1A	0001 1010	90	5A	0101 1010	154	9A	1001 1010	218	DA	1101 1010
27	1B	0001 1011	91	5B	0101 1011	155	9B	1001 1011	219	DB	1101 1011
28	1C	0001 1100	92	5C	0101 1100	156	9C	1001 1100	220	DC	1101 1100
29	1D	0001 1101	93	5D	0101 1101	157	9D	1001 1101	221	DD	1101 1101
30	1E	0001 1110	94	5E	0101 1110	158	9E	1001 1110	222	DE	1101 1110
31	1F	0001 1111	95	5F	0101 1111	159	9F	1001 1111	223	DF	1101 1111
32	20	0010 0000	96	60	0110 0000	160	A0	1010 0000	224	E0	1110 0000
33	21	0010 0001	97	61	0110 0001	161	A1	1010 0001	225	E1	1110 0001
34	22	0010 0010	98	62	0110 0010	162	A2	1010 0010	226	E2	1110 0010
35	23	0010 0011	99	63	0110 0011	163	A3	1010 0011	227	E3	1110 0011
36	24	0010 0100	100	64	0110 0100	164	A4	1010 0100	228	E4	1110 0100
37	25	0010 0101	101	65	0110 0101	165	A5	1010 0101	229	E5	1110 0101
38	26	0010 0110	102	66	0110 0110	166	A6	1010 0110	230	E6	1110 0110
39	27	0010 0111	103	67	0110 0111	167	A7	1010 0111	231	E7	1110 0111
40	28	0010 1000	104	68	0110 1000	168	A8	1010 1000	232	E8	1110 1000
41	29	0010 1001	105	69	0110 1001	169	A9	1010 1001	233	E9	1110 1001
42	2A	0010 1010	106	6A	0110 1010	170	AA	1010 1010	234	EA	1110 1010
43	2B	0010 1011	107	6B	0110 1011	171	AB	1010 1011	235	EB	1110 1011
44	2C	0010 1100	108	6C	0110 1100	172	AC	1010 1100	236	EC	1110 1100
45	2D	0010 1101	109	6D	0110 1101	173	AD	1010 1101	237	ED	1110 1101
46	2E	0010 1110	110	6E	0110 1110	174	AE	1010 1110	238	EE	1110 1110
47	2F	0010 1111	111	6F	0110 1111	175	AF	1010 1111	239	EF	1110 1111
48	30	0011 0000	112	70	0111 0000	176	B0	1011 0000	240	F0	1111 0000
49	31	0011 0001	113	71	0111 0001	177	B1	1011 0001	241	F1	1111 0001
50	32	0011 0010	114	72	0111 0010	178	B2	1011 0010	242	F2	1111 0010
51	33	0011 0011	115	73	0111 0011	179	B3	1011 0011	243	F3	1111 0011
52	34	0011 0100	116	74	0111 0100	180	B4	1011 0100	244	F4	1111 0100
53	35	0011 0101	117	75	0111 0101	181	B5	1011 0101	245	F5	1111 0101
54	36	0011 0110	118	76	0111 0110	182	B6	1011 0110	246	F6	1111 0110
55	37	0011 0111	119	77	0111 0111	183	B7	1011 0111	247	F7	1111 0111
56	38	0011 1000	120	78	0111 1000	184	B8	1011 1000	248	F8	1111 1000
57	39	0011 1001	121	79	0111 1001	185	B9	1011 1001	249	F9	1111 1001
58	3A	0011 1010	122	7A	0111 1010	186	BA	1011 1010	250	FA	1111 1010
59	3B	0011 1011	123	7B	0111 1011	187	BB	1011 1011	251	FB	1111 1011
60	3C	0011 1100	124	7C	0111 1100	188	BC	1011 1100	252	FC	1111 1100
61	3D	0011 1101	125	7D	0111 1101	189	BD	1011 1101	253	FD	1111 1101
62	3E	0011 1110	126	7E	0111 1110	190	BE	1011 1110	254	FE	1111 1110
63	3F	0011 1111	127	7F	0111 1111	191	BF	1011 1111	255	FF	1111 1111



**Table 11-5**  
Standard Tekhex Termination Block Format

Field	Number of ASCII Characters	Description
/	1	The slash specifies that the block is in Standard Tekhex format.
Transfer Address	4	Starting execution address (high-byte, low-byte format) of the code transmitted in the data blocks.
Byte	2	Always 00 in a termination block
Checksum	2	Sum, mod 256, of the six hex digits of the transfer address and byte count.

**Table 11-6**  
Standard Tekhex Abort Block Format

Field	Number of ASCII Characters	Description
/	1	The slash specifies that the block is in Standard Tekhex format.
/	1	Another slash to identify the abort block.
Message	1 to 69	A message to be displayed when the transfer is aborted.

Here is an example of a Standard Tekhex termination block:

```

+--Transfer address
|   +--Checksum
|   |
|   |
|   v   v
-----
/10000001
-   -
^   ^
|   |
|   |
|   +--Byte count
+--Header
    
```

Here is an example of a Standard Tekhex abort block:

```

Header
|
|   Message
|
|
|   |
v   v
-----
//5 CONSECUTIVE FAILURES. TRANSMISSION ABORTED
    
```

## Extended Tekhex Format

Each Extended Tekhex block begins with a six-character header field.

Extended Tekhex has no specially defined abort block. To abort a formatted transfer, use a Standard Tekhex abort block

**Table 11-7**  
Extended Tekhex Header Field

Item	Number of ASCII Characters	Description
%	1	A percent sign specifies that the block is in Extended Tekhex format.
Block Length	2	The number of characters in the block: a two-digit hex number. This count does not include the leading % or the end-of-line.
Block Type	1	6 = data block 3 = symbol block 8 = termination block
Checksum	2	A two-digit hex number representing the sum, mod 256, of the values of all the characters in the block, except the leading %, the checksum digits, and the end-of-line. Table 11-8 gives the values for all characters that may appear in Extended Tekhex message blocks.

**Table 11-8**  
Character Values for Checksum Computation

Characters	Values (Decimal)
0..9	0..9
A..Z	10..35
\$	36
%	37
(period)	38
_ (underscore)	39
a..z	40..65

**Table 11-9**  
Extended Tekhex Data Block Format

Field	Number of ASCII Characters	Description
Header	6	Standard header field. Block type = 6.
Load Address	2 to 17	The address where the object code is to be loaded: a variable-length number.
Object	2n	n bytes, each represented as two hex digits.

Here is an example of an Extended Tekhex data block:

```

+----- Block length: 15H = 21
|
| +----- Checksum: 1CH = 28 = 1+5+6+3+1+0+0+2+0+2+...
| |
| | +-- Object code: 6 bytes
| |
| | =====
| | %1561C3100020202020202
| | = = =====
| |
| | +-- Load address: 100H
| |
| | +----- Block type: 6
| |
| +----- Header character
    
```

**Table 11-10**  
Extended Tekhex Termination Block Format

Field	Number of ASCII Characters	Description
Header	6	Standard header field. Block type = 8.
Transfer Address	2 to 17	The address where program execution is to begin: a variable-length number.

**Table 11-11**  
Extended Tekhex Symbol Block Format<sup>a</sup>

Field	Number of ASCII characters	Description
Header	6	Standard header field. Block type = 3.
Section Name	2 to 17	The name of the section that contains the symbols defined in this block: a variable-length symbol.
Section Definition	5 to 35	This field must be present in exactly one symbol block for each section. This field may be preceded or followed by any number of symbol definition fields. Table 11-12 gives the format for this field.
Symbol Definition(s)	5 to 35 each	Zero or more symbol definition fields, as described in Table 11-13.

<sup>a</sup> For an example of Extended Tekhex symbol block format, refer to Section 4 of this manual, Intersystem Communication.

Here is an example of an Extended Tekhex termination block:

```

+----- Block length: 8
|
| +--- Checksum: 1AH = 26 = 0+8+8+2+8+0
| |
| == ==
| %0881A280
| = = ==
| | |
| | +--- Transfer address: 80H
| |
| +----- Block type: 8
|
+----- Header character
    
```

**Table 11-12**  
Extended Tekhex Symbol Block:  
Section Definition Field

Item	Number of ASCII Characters	Description
0	1	A zero signals a section definition field.
Base Address	2 to 17	The starting address of the section: a variable-length number.
Length	2 to 17	The length of the section: a variable-length number, computed as 1 + (high address - base address).

**Table 11-13**  
Extended Tekhex Symbol Block:  
Symbol Definition Field

Item	Number of ASCII Characters	Description
Type	1	A hex digit that indicates the global/local designation of the symbol, and the type of value the symbol represents: 1 = global address 2 = global scalar 3 = global code address 4 = global data address 5 = local address 6 = local scalar 7 = local code address 8 = local data address
Symbol	2 to 17	A variable-length symbol.
Value	2 to 17	The value associated with the symbol: a variable-length number.

## MOTOROLA LOAD MODULE FORMAT

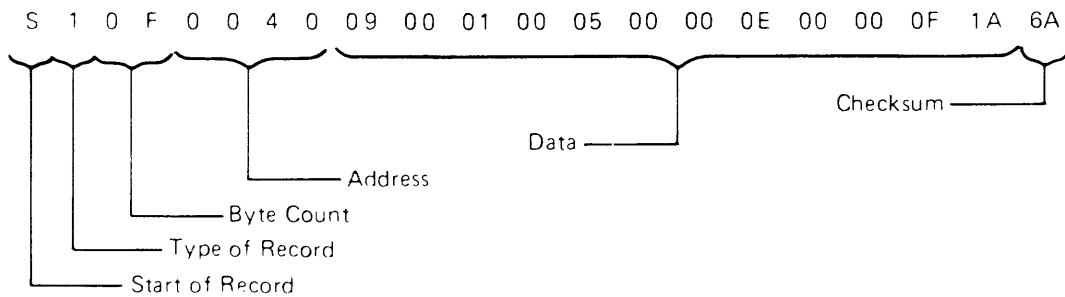
### General Format

Start of Record	Type of Record	Byte Count	Address	Data	Checksum
-----------------	----------------	------------	---------	------	----------

**Table 11-14**  
**Motorola Load Module Format**

Name	No. of ASCII Characters	Content Description
Start of Record	1	Always an 'S'.
Type of Record	1	0 = header record 1 = data record 9 = end of file record
Byte Count	2	A two-digit hexadecimal number representing the number of data bytes in the block plus three.
Address	4	Four hexadecimal digits representing the address of the memory location where this record is stored.
Data	2*N	N data bytes, each represented as two hexadecimal digits.
Checksum	2	Two-digit hexadecimal number representing the one's complement of the sum, modulo 100H, of the data bytes, address, and byte count.

**Example**



**Type of Record 1.** Specifies the data record.

**Byte Count 0F.** Indicates that there are 0F pairs of hexadecimal digits in the block following the byte count.

**Address 0040.** Specifies that the first data byte (09) is stored at address 0040. The next bytes (00,01,00,...) are stored at the next contiguous memory locations (0041, 0042, etc.).

**Data Bytes.** Pairs of hexadecimal digits representing data bytes stored on the record.

**Checksum 6A.** The one's complement of the number 95, the sum of the data bytes, address, and byte count:

$$0F+00+40+09+00+01+00+05+00+00+0E+00+00+0F+1A = 95H$$

## INTEL LOAD MODULE FORMAT

### General Format

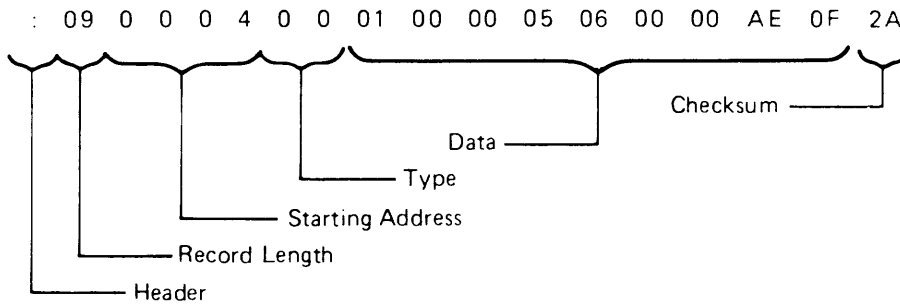
Header Character	Record Length	Starting Address	Type of Record	Data	Checksum
------------------	---------------	------------------	----------------	------	----------

**Table 11-15**  
**Intel Load Module Format**

Name	No. of ASCII Characters	Content Description
Header Character	1	Always a colon.
Record Length	2	Two-digit hexadecimal value representing the number of data bytes. A record length of 0 indicates the last record of a file.
Starting Address	4	Four hexadecimal digits representing the program memory address of the first data byte in the record.
Type of Record	2	00 = normal data record. 01 = last record of a file.
Data	2*N	N data bytes, each represented as two hexadecimal digits.
Checksum	2	Two-digit hexadecimal number representing the two's complement of the sum, modulo 100H, of the preceding data bytes, record type, address, and record length.



**Example**



**Record Length 09.** Indicates that there are nine data bytes in the record.

**Starting Address 0004.** Specifies that the first data byte (01) is stored at program memory address 0004. The next bytes (00,00,05...) are stored at the next contiguous memory locations (0005, 0006, etc.).

**Type of Record 00.** Indicates a normal data record.

**Data Bytes.** Pairs of hexadecimal digits representing data bytes stored on the record.

**Checksum 2A.** The two's complement of D6, the sum of the data bytes, record type, address, and record length.

$$09+00+04+00+01+00+00+05+06+00+00+AE+0F = D6$$

---

# Section 12

## GLOSSARY

**Acquisition Memory.** The buffer in the TTA that holds the 255 bus transactions most recently captured during program execution. This buffer stores up to 62 bits of information from each bus transaction.

**Address.** A number or symbol that specifies a byte in memory.

**Assembler.** A host system program that translates assembly language programs into machine language.

**Assembly Language.** A microprocessor-specific programming language that allows the symbolic representation of any microprocessor operation. Each operation is coded as one assembly language statement.

**Assign.** To associate a channel with a device or host file.

**Break.** A suspension of program execution by OS/40, accompanied by a display of the status of the emulator. A break may be set using an OS/40 BK command or one of several TTA commands. A break may also result from a special action, such as attempting to write to protected memory or typing CTRL-C.

**Breakpoint.** A program instruction at which a break is set. You can set breakpoints by using the TTA or the OS/40 BK (Breakpoint) command. The TRA -S command causes a break to occur each time a trace line is displayed.

**Buffer.** An area of memory where a block of data may be stored.

**Bus Operation.** A transfer of information between a microprocessor and a memory or I/O device. The four basic types of bus operations are memory read, memory write, I/O read, and I/O write. Some emulators recognize other types of bus operations.

**Channel.** The logical link between a device or host file and the operating system. Channels are numbered 0-9. Channels 0-7 are user-defined; channels 8 and 9 are assigned to standard input and standard output, respectively. The AS (Assign) command assigns a channel for use by a device or host file.

**Characteristic Module.** A circuit card that configures the PROM Programmer for a particular set of PROM devices.

**Checksum.** A value obtained by summing together data bytes as they are received. If the checksum calculated by the receiving device differs from the checksum transmitted with the data, the data may have been transferred incorrectly.

**COM Interface.** A communications interface between an 8540 and a host computer, established via the OS/40 COM command. Object code can be transferred between the host and the 8540, and host commands are executed from the 8540 system terminal as if the 8540 were not present.

**Close.** To disassociate a channel from a device or host file.

**Control Character.** A character whose ASCII code is in the range 00 to 1F hexadecimal. RUBOUT (ASCII code 7F) is also a control character. Some control characters are entered using special keys, such as TAB or RETURN. Others are entered by pressing the CTRL key and some other key at the same time.

**Counter.** See **General Purpose Counter**.

**Device.** An instrument attached to the 8540 that is used for I/O. See the Learning Guide section of this manual for a table of standard device names used by OS/40.

**Disassemble.** To translate machine language back into assembly language mnemonics. The DI command performs disassembly.

**Download.** To transfer data from a host to the 8540.

**Editor.** A host system program that allows you to create and modify text files conveniently.

**EEPROM.** Electronically erasable PROMs. EEPROMS on the 8540 contain user-defined strings saved with the PERMSTR command, and system patch information created by the ROMPATCH command.

**Emulating Microprocessor.** The microprocessor on which your program executes during emulation. The emulating microprocessor resides on the emulator board or in the prototype control probe, and is usually the same type of microprocessor as the one being emulated.

**Emulation.** Performing the functions of a microprocessor, at or near the microprocessor's execution speed, with controllable hardware that facilitates debugging and testing of the microprocessor software in the prototype hardware. Emulation features of the 8540 include breakpoints, tracing, and event timing.

**Emulation Mode.** An 8540 operating mode in which the 8540 can provide some of the hardware functions needed by the microprocessor-based program. The hardware functions that can be provided are memory, a clock, and I/O facilities. The three modes are:

- **Mode 0:** System mode. Your program uses program memory and the emulator clock and uses SVCs for I/O. Until the prototype is built and connected to the emulator hardware, the program may execute only in mode 0.
- **Mode 1:** Partial emulation mode. Your program uses the prototype's clock and may access both program and prototype memory; the memory map determines whether a particular address refers to program or prototype memory. Some emulators may also use SVCs.
- **Mode 2:** Full emulation mode. Your program uses the prototype's memory, clock, and I/O facilities. Some emulators may also use SVCs.

**Emulator (or Emulator Processor).** A circuit board in the 8540 that emulates the microprocessor that will drive your prototype hardware. You may use the emulation and debugging features of the emulator to test the software that will run on the prototype and to integrate the software and hardware components of the prototype.

**Event.** The simultaneous occurrence of one or more specified conditions that can be detected by the TTA. An event is defined in terms of values on the address bus, data bus, and certain other lines connected to the emulating microprocessor or prototype.

**Expression.** A formula that contains numbers, register names, or user-defined symbols related by operators. Evaluation of the expression yields an integer value. Many OS/40 command parameters may be entered as expressions. Refer to the Command Dictionary for more information on expressions.

**Extended Tekhex.** See **Tekhex**.

**File Pointer.** A logical position in a file, maintained by OS/40 for use in processing SVCs. All reads from the file and writes to the file are performed beginning at the current file pointer. The file pointer is updated to the end of the data item read or written. The file pointer may also be moved using Seek SVCs. An independent file pointer is maintained for each channel.

**Filespec.** A string of characters that specifies a file on the host computer. The term "filespec" in a Command Dictionary syntax block may also refer to a standard device name such as CONI or REMO.

**Full Emulation Mode.** See **Emulation Mode**.

**General Purpose Counter.** One of four registers in the TTA that increments or decrements while the emulator is running. You may use these counters to measure the execution time of a program segment, to measure the time between two specified events, or to count the occurrences of an event.

**High-Level Language.** A language such as MDL/ $\mu$  or Pascal that reflects algorithms more easily than assembly language. A program written in a high level language is essentially processor independent, relying on a processor-specific compiler or interpreter to convert the program into object code.

**Host.** A separate computer system that is used to prepare and maintain programs that are tested and debugged on the 8540.

**Instruction.** A **machine** instruction is a sequence of bytes that directs a microprocessor to perform an elementary operation such as load, store, add, or branch. An **assembly language** instruction is an alphanumeric representation of a machine instruction. The assembler translates an assembly language instruction into the corresponding machine instruction.

**Instruction Fetch.** A bus operation during which the first byte of the next instruction to be executed appears on the data bus.

**Interrupt.** A suspension of normal processing to handle a special request. In general, when a microprocessor is interrupted by a peripheral device, the microprocessor finishes the instruction it is executing, then jumps to the routine that services the indicated peripheral, and finally returns to the stream of instructions it was executing before the interrupt occurred.

**Intersystem Communication.** A process by which the 8540 exchanges information with another computer system, via cable or phone line.

**K.** 1024 bytes (400 hexadecimal).

**Label.** A symbol that represents an address, variable, or constant in an assembly language program.

**Linker.** A host system program that combines object modules into a single executable load module.

**Load Module.** A collection of executable object code suitable for loading into program memory. A load module may be in binary format, as produced by a linker or the SAV command, or it may be in a hexadecimal format. Refer to the Tables section of this manual for information on hexadecimal formats recognized by OS/40.

**LOCAL Mode.** Stand-alone mode for an 8540, in which each command entered is interpreted by the 8540 itself. See also **TERM Interface**.

**MAC.** See **Memory Allocation Controller**.

**Machine Language.** The binary language of a microprocessor. A high-level or assembly language program must be translated to machine instructions before the microprocessor can execute the program.

**Memory Allocation Controller.** A hardware option that permits the expanded addressing capabilities of emulators such as the Z8001/Z8002 and 68000 to operate within the confines of 8540 program memory. For such emulators, you must have the MAC option installed in order to use the DEAL (DEALocate), MEM (MEMory), and NOMEM (NOMEMory) commands.

**Memory Map.** An internal table maintained by OS/40 that indicates which portions of memory used by the emulator are in program memory and which are in prototype memory. The memory map also indicates which parts of program memory are protected from write operations during program execution. Memory may be mapped in blocks as small as 128 bytes. Use the MAP command to change or display the memory map.

**Mnemonic.** A symbol that represents a machine instruction. Usually the symbol is an abbreviation that suggests the machine operation to be performed. For example, the 8085A mnemonic MOV represents a machine instruction that moves a value into a register.

**Module.** A program unit that is complete for purposes of compiling, assembling, linking, or loading. It may be combined with other modules to produce a complete program. See also **Load Module; Object Code; Source Code**.

**Object Code.** Machine language produced by an assembler or compiler from source statements. An **object module** contains one or more sections of object code, plus special information used by the linker, library generator, or LO command. Each time the assembler executes, it produces one object module. An **object file** is a file that contains an object module.

**Operand.** A number on which an operation is performed. The expression X+3 represents an add operation on the operands X and 3. The assembly language statement LDA NUM1 performs a load operation on the byte addressed by the operand NUM1.

**Operator.** A character or sequence of characters that represents an operation to be performed on one or more operands. The only operators allowed in OS/40 command lines are + and -.

**OS/40.** The ROM-based Operating System of the 8540 Integration Unit.

**Partial Emulation Mode.** See **Emulation Mode**.

**Patch.** To alter a program by changing the executable object code rather than the source code.

**Program Clock.** A counter in the 8540 that increments every 100 milliseconds while the emulator is running. You may use the CLOCK command to initialize or display the value in the program clock. This value is also accessible through the "Read Program Clock" SVC.

**Program Counter.** A register that contains the address of the next instruction to be executed.

**Program Memory.** Memory in the 8540 that is used as a substitute for prototype memory in the early stages of prototype development (emulation modes 0 and 1).

**Program/Prototype Memory.** This term refers either to program memory or to prototype memory, depending on the current emulation mode and memory map. In mode 0, this term always refers to program memory. In mode 2, this term always refers to prototype memory. In mode 1, the memory map dictates which address ranges are in program memory and which are in prototype memory.

**PROM.** Programmable Read-Only Memory. Nonvolatile read-only memory that is blank when it is manufactured and stores whatever information is written to it by a PROM Programmer.

**PROM Programmer.** An optional device that writes data from memory to a PROM chip or reads data from a PROM chip into memory. You must have a PROM Programmer in order to use the following commands: CPR, RPR, WPR, PSTAT, and PTYPE.

**Prototype.** The microprocessor-based device that you are developing using the 8540.

**Prototype Control Probe.** A probe that takes the place of the prototype microprocessor and connects the prototype to the appropriate emulator hardware in the 8540.

**Prototype Memory.** Memory that resides in the prototype being developed.

**RAM.** Random Access Memory. Memory that may be read from or written to.

**Rewind.** To position a file pointer to the beginning of a file, so that information in the file can be re-processed. See also **Seek**.

**ROM.** Read-Only Memory: memory that is manufactured with predetermined contents and cannot be written to.

**Seek.** To position a file pointer to a given location in a file. Your program can use Seek SVCs to select any position in the file at which to read or write data. See also **Rewind**.

**Service Call (SVC).** A request for OS/40 to perform a specified I/O or maintenance function for an executing program. System programs use SVCs, as do most user programs that do not rely entirely on prototype I/O. The Service Calls section of this manual explains how to set up and initiate a service call, and describes each type of service call.

**Source Code.** Program statements written in high-level or assembly language. A **source module** is a set of statements that are compiled or assembled as a unit. A **source file** is a file that contains all or part of a source module.

**Stack.** A last-in, first-out (LIFO) data structure. Only the last item pushed onto the stack is immediately available. When that item is popped off the stack, the item stored before it becomes available. Many microprocessor applications use a stack in memory for temporary storage or for passing parameters between subroutines.

**Stack Pointer.** A register that contains the address of the top of the stack.

**Standard Input.** The file or device from which a command takes its input. Defaults to the system terminal unless you redirect input by using a <**filespec** parameter in the command line.

**Standard Output.** The file or device to which a command directs its output. Defaults to the system terminal unless you redirect output by using a >**filespec** parameter in the command line.

**Standard Tekhex.** See **Tekhex**.

**String.** A sequence of ASCII characters. OS/40 allows you to assign a name to a string and then refer to the string by name in a command line. If a string contains a delimiter such as a space, comma, or semicolon, the string should be enclosed in single or double quotes. If it contains a dollar sign, backslash, or double quote, the string should be enclosed in single quotes.

Permanent strings, created with the OS/40 PERMSTR command, are stored in the 8540's EEPROM storage area and thus are available whenever the system is powered up or restarted.

**SVC.** See **Service Call**.

**Symbol.** A string of up to 16 characters that begins with a letter and contains only letters, digits, periods, underscores, or dollar signs. In an assembly language program, predefined symbols include assembler directives and functions, mnemonics, and register names; user-defined symbols represent addresses, data items, variables, macros, sections, or modules.

**Symbol Table.** A table in system memory that contains program symbols and their values. This table is used in symbolic debug. Use the SYMLO or COM commands to place symbols from a load file in the symbol table. Use the ADDS command to create other symbols.

**Symbolic Debug.** The use of symbols in place of expressions or hexadecimal numbers during debugging. You can use symbols as parameters in any OS/40 command that accepts expressions as parameters. During output, each hexadecimal address is replaced with either a symbol or an offset relative to the start of a program section. Some emulators do not support symbolic debug; refer to your Emulator Specifics supplement for this information.

**System Memory.** Memory in the 8540 that is not accessible to the user. Most OS/40 commands execute in system memory.

**System Mode.** See **Emulation Mode**.

**System Terminal.** The CRT terminal or other RS-232-C-compatible I/O device through which you communicate with the 8540. Device names for input and output through the system terminal are CONI and CONO, respectively.

**Target Processor.** The microprocessor that the 8540 is to emulate.

**Tekhex.** Tektronix Hexadecimal Format: a format for representing the contents of a block of memory as an ASCII sequence of hexadecimal digits. Checksums in the Tekhex format permit verification of Tekhex data transmitted from one computer to another. The WH command can be used to write memory contents to a file or device in Tekhex format; similarly, RH can be used to read Tekhex data from a file or device into memory. There are two forms of Tekhex: Standard Tekhex and Extended Tekhex. Both forms of Tekhex are described in the intersystem Communication section of this manual.

**TERM Interface.** A mode of communication between an 8560 Multi-User Software Development Unit and an 8540. A TERM interface allows you to intermix OS/40 commands and TNIX commands, execute OS/40 commands from TNIX command files, and transmit data between the 8540 and 8560.

**TNIX.** The operating system of the TEKTRONIX 8560 Multi-User Software Development Unit.

**Trace.** To monitor the execution of a program by displaying the processor status each time a specified type of instruction is executed. The TRA command specifies the type and/or range of instructions to be displayed.

**Transfer Address.** The address of the first machine instruction to be executed in a program.

**Trigger Trace Analyzer (TTA).** An optional hardware device that enhances the 8540's debugging capabilities. The TTA allows you to capture and store up to 255 bus

transactions that precede or follow a selected event in the executing program. The TTA includes four general purpose counters and four triggers that can break program execution, start or stop counters, or signal other instruments.

You must have a TTA installed in your 8540 in order to use any of the following commands: ACQ, AD, BRE, BUS, CONS, COU, CTR, DATA, DISP, EVE, PRO, QUA, TCLR, TS.

**Upload.** To transfer data from the 8540 to a host.

# Section 13

## INDEX

### A

A command, 3-8  
 abort  
   command, 3-8  
   SVC, 6-11  
 abortable commands, 3-8  
 ACIA, 4-23, 4-25  
 ACK character, 4-15  
 ACQ command, 5-18  
   examples, 2-16-2-17  
 Acquisition Memory, 5-16-5-17, 12-1  
 AD command, 5-18  
 add:  
   contents of registers, 3-15  
   numbers, 3-15  
   symbol to symbol table, 3-9  
 address expressions, 3-5  
 address map:  
   allocate memory to, 3-10  
   deallocate memory from, 3-25  
 ADDS command, 3-9, 5-9  
   examples, 2-9  
 AL command, 3-10, 5-11  
   examples, 2-9  
 apostrophe. See Single quote  
 arithmetic expression, 3-5  
 arrows, input and output redirection, (< >), 3-3  
 AS command, 3-12  
 ASCII codes (table), 11-1  
 ASCII transfer, 4-8, 6-10  
   See also Intersystem communication  
 asm (8560 command), 1-25  
 assembler, 1-25, 12-1  
 assign  
   channel to device or file, 3-12  
   value to register or symbol, 3-51

### B

backslash (\), 3-4  
 BACKSPACE key, 3-7  
 baud rate for data transfers, 4-24  
 binary-decimal-hexadecimal equivalents, 11-3  
 binary transfer, 4-8, 6-10  
   See also Intersystem communication  
 BK command, 3-13, 5-4  
 boldface in syntax blocks, 3-2  
 braces ({}), 3-2  
 brackets ([ ]), 3-2  
 BRE command, 5-16  
 break, 12-1  
 breakpoint, 12-1  
   clearing, 2-13, 3-14  
   displaying, 2-13, 3-13  
   setting, 2-12, 3-13, 5-4  
   See also BK command  
 buffer (for an SVC), 6-4  
 BUS command, 5-15

### C

CALC command, 3-15, 5-9  
 capitalization, 1-15  
 carriage return, 1-10  
 channels. See I/O channels  
 characteristic module, Section 8, 12-1  
 characters, control, 3-7  
 characters, special:  
   ACK (0), 4-15  
   apostrophe. See Single quote  
   arrows (< >), 3-3  
   backslash (\), 3-4  
   braces ({}), 3-2  
   brackets ([ ]), 3-2  
   carriage return (<CR>), 1-10  
   dash (-), 3-3  
   dollar sign (\$), 1-14, 3-5  
   double quote ("), 3-5, 6-10  
   escape character (\), 3-4  
   NAK (7), 4-15  
   null, 3-21, 4-4  
   periods (...), 3-2  
   semicolon (;), 1-13, 3-2  
   single quote ('), 3-5, 6-10  
   slash (/), 4-1  
   See also Control characters  
 checksum, 12-1  
 CL command, 3-16  
 clear breakpoint, 3-14  
 CLOCK command, 3-17, 5-5  
 close  
   channel, 3-16  
   SVC, 6-11  
 CO command, 3-18  
 code address (symbol type), 4-13  
 COM command, 3-19, 4-22  
   examples, 2-3-2-5  
 COM interface, 1-12, 4-2  
 COM Interface Package, 1-5  
 command strings, 1-13  
   STARTUP, 1-9, 1-14  
 command line:  
   capitalization, 3-2  
   format, 3-2  
   notational conventions, 3-2  
 commands, Section 3  
   abortable, 3-8  
   affected by MEMSP, 3-39  
   dash modifiers, 3-3  
   delimiters, 3-3  
   memory areas, 3-7  
   multiple, 3-4  
   names, 3-3  
   parameters, 3-3  
   reference manuals for other commands, 3-2  
   suspendable, 3-8  
   syntax notation conventions, 3-2  
 CONFIG command, 2-1-2-2, 3-22  
 configuring the 8540, 1-1, 3-22, 3-23  
   with an 8560, 1-15  
 CONI, 1-5  
 CONO, 1-5  
 CONS command, 5-16

- continuing execution of suspended command, 3-18
- control characters, 1-10, 3-7, 12-1
  - null, 4-4
- conversion tables, 11-2
- correcting a typing mistake, 1-11
- COU command, 5-16-5-17
  - examples, 2-15, 2-17-2-18
- CPR command, Section 8
- <CR>, 1-10
- CTR command, 5-15
- CTRL (control) characters. See Control characters
  
- D**
- D command, 3-24
  - examples, 1-32, 1-34, 2-19
- dash (-), 3-3
- dash modifiers, 3-3
- data address (symbol type), 4-13
- DATA command, 5-15
- Data Transfer Types table (COM Interface), 4-3
- data transfers. See Intersystem Communication
- DCE, 4-23
- DEAL command, 3-25, 5-12-b-13
- deallocate memory from logical address map, 3-25
- decimal-hexadecimal-binary equivalents, 11-3
- deleting:
  - permanent strings, 3-43
  - temporary strings, 3-4
  - text being entered, 1-11
- delimiters, command, 3-3
- demonstration run, 1-17-1-40
- demonstrations of:
  - emulation, 1-17-1-40
  - SVCs, 6-6-6-9
- designators (memory space), 3-6
- devices. See I/O devices
- DI command, 3-26
  - examples, 2-20
- disassemble object code into mnemonics, 3-26
- DISP command, 2-16, 3-27, 5-18
- display:
  - breakpoint conditions, 3-13
  - contents of emulator registers, 3-27
  - DISP command, 3-27
  - executed instructions, 3-62
  - memory contents, 3-24, 3-29
  - memory map, 3-36
  - memory map assignments, 3-35
  - strings:
    - permanent, 3-43
    - temporary, 3-55
  - symbolic, 3-59
  - system status, 3-55
- dollar sign (\$), 1-14, 3-5
- don't-care expressions, 3-6
- DOS/50:
  - compared with OS/40, 9-1
  - HOST command, 4-4
- double quote ("), 3-5
- download, 3-21, 4-1
  - See also Intersystem communication
- DS command, 3-27
  - examples, 2-9, 5-5
- DTE, 4-23
- dump memory contents, 3-24
  
- E**
- EEPROM:
  - patching, 3-49
  - storing permanent strings in, 3-43
- electrical considerations, 1-1
  - for prototype control probe, Section 7
- EM command, 2-8, 3-28
- emulating microprocessor, 5-1
- emulation, Section 5
  - limits of, 9-1
  - mode 0, 5-1
  - modes 1 and 2, 5-6-5-7
- emulation modes, 3-28, 5-1, 12-1
- emulator, 5-1
  - display registers, 3-27
  - read from emulator port, 3-45
  - reinitialize, 3-47
  - starting, 5-4
  - stopping, 5-4
  - write to emulator port, 3-65
- emulator processor. See Emulator
- emulator-specific information, Section 7
- escape character:
  - CTRL-C, 3-7
  - (\), 3-4
- EVE command, 5-15
  - examples, 2-14-2-18
- evaluate arithmetic expression, 3-15
- event, 5-13-5-16, 12-2
  - consecutive, 5-16
- EX command, 3-29
- examine or alter memory contents, 3-29, 5-5
- execution:
  - abort command or program execution, 3-8
  - begin program execution, 3-32
  - load and execute program, 3-65
  - resume execution of suspended command, 3-18
  - SVC execution, 3-57
  - time execution of program segment, 3-17
- exit (from program), 6-12
- expressions:
  - address, 3-5
  - arithmetic expression evaluation, 3-15
  - don't-care, 3-6
  - operators in expressions, 3-6
- Extended Tekhex, 4-11
  - sample load module, 1-18
  
- F**
- F command, 3-30
  - examples, 2-20, 5-5
- file pointer, 6-11
- files:
  - assign channel to file, 3-12, 6-11
  - close channel to file, 3-16, 6-11
  - limitations of file handling SVCs, 6-4
  - save memory contents in file, 3-52
- fill program/prototype memory with data, 3-30
- formatted transfers, 4-9-4-21
  - See also Intersystem communication



**G**

G command, 2-8, 3-32  
 general purpose counter, 5-16-5-17, 12-2  
   examples, 2-15, 2-17-2-18  
 global symbol, 4-13  
 go (begin program execution), 3-32

**H**

halting program execution, 5-4, 5-7  
 handshaking (in formatted data transfers), 4-16  
 hardware requirements for host computer, 4-23  
 hardware specification, installation,  
   configuration, and verification, 1-1  
 hexadecimal:  
   formats. See Load module  
   read, 3-48  
   write, 3-64  
 hexadecimal-decimal-binary equivalents, 11-3  
 hexadecimal multiplication, 11-2  
 HOST (DOS/50 command), 4-4  
   examples, 2-6  
 host computer, 1-7  
   See also Intersystem communication

**I**

input/output. See I/O  
 installation of 8540 hardware and software, 1-1  
 instruction, 12-2  
 Intel hexadecimal load module format, 11-10  
 interface, define host, 3-22  
 interrupts, 12-2  
   in emulation mode 0, 5-6  
   in emulation modes 1 and 2, 5-7  
 intersystem communication:  
   COM command parameters, 4-22  
   communication with an 8002A, 2-7, 4-6  
   communication with an 8550, 2-5-2-7, 4-4  
   communication with an 8560, 1-11, 2-1-2-2, 4-1  
   communication with other host computers, 1-12,  
   2-3-2-5, 4-7  
   data transfers, 4-25  
   establishing communication, 4-25  
   exiting from COM, 4-25  
   formatted transfers, 4-9-4-21  
   hardware requirements, 4-23  
   host computer programs:  
     convert to Tekhex, 4-18-4-21  
     receive Tekhex, 4-17  
     transmit Tekhex, 4-16  
   protocols for data transfers:  
     formatted, 4-15  
     unformatted, 4-8  
   Types of Data Transfer table, 4-3  
   unformatted transfers, 4-7-4-9

**I/O:**

buffer (for an SVC), 6-4  
 channels, 6-4, 6-11  
   assigning, 2-21  
   closing, 2-21  
   displaying assignments, 2-20  
 devices:  
   identification and type codes, 6-13  
   jack numbers, 1-8  
   log terminal I/O to device, 3-34  
   redirection of standard I/O, 3-3-3-4

**J**

jack numbers, 1-8

**K**

keys, special:  
 BACKSPACE, 3-7  
 CTRL, 1-10, 3-7  
 escape (CTRL-C), 3-7  
 RETURN (carriage return), 1-10  
 RUBOUT, 3-7

**L**

LAS format for SRB, 6-3  
 line printer, 1-6  
   accessing the line printer in TERM mode, 1-37  
 LIST DBG (assembler directive), 1-20  
 LO command, 3-34  
 load:  
   module, 12-3  
     Tektronix Hexadecimal Format (Tekhex), 4-9-4-21  
     See also Intersystem communication  
   program, 3-34, 3-48, 3-65  
   symbols into symbol table, 3-61  
 LOCAL mode, 1-11, 3-22, 12-3  
 LOG command, 2-10, 3-34  
 logging in to 8560, 1-11  
 logging in to other hosts, 1-12  
 logical address map:  
   allocating memory to, 3-10, 5-7  
   deallocating memory from, 3-25  
 lowercase. See Capitalization  
 LPT, 1-6

**M**

M parameter (of COM and CONFIG commands), 4-22  
 MAC. See Memory Allocation Controller  
 manuals:  
   overview of related user manuals, 1-41, 3-2  
   overview of this manual, 1-40  
 MAP command, 2-14, 2-20, 3-35  
 MEM command, 3-38, 5-13  
 memory:  
   allocation, 2-9, 3-10, 5-4, 5-9-5-13  
   alter contents, 3-29, 3-42  
   available to emulator, 3-38, 3-41  
   command memory areas, 3-7  
   deallocation, 3-25  
   display contents, 3-24, 3-29  
   display memory map, 3-36  
   fill program/prototype memory with data, 3-30  
   layout for SVCs, 6-2, 6-3  
   memory map, 3-35-3-36, 5-7  
   memory spaces, 3-6, 3-39, 5-11  
   program memory, 1-5  
     changing contents, 5-5  
     displaying contents, 5-4-5-6  
     reallocating address blocks in a 32K 8540 system, 5-4  
   program/prototype memory, 12-3  
   copying data between program and prototype memory,  
   3-40  
   prototype memory, 12-3  
   read-only memory (ROM), 12-4  
     detecting a write to ROM, 5-4, 5-7  
     MAP command, 2-20, 3-35  
     TTA, 2-15  
   save contents, 3-52, 3-64  
   search for value or string, 3-53  
   system memory, 12-4  
   write protection of prototype memory, 3-35

Memory Allocation Controller, 5-9  
 commands, 5-11-5-13  
 memory spaces, 3-6, 3-39, 5-11  
 MEMSP command, 3-39  
 mode. See Emulation modes  
 MODE SELECT switch, 4-24  
 modifiers, command (dash), 3-3  
 Motorola hexadecimal load module format, 11-4-11-7  
 MOV command, 3-40  
 multiple commands, 3-4

## N

NAK character, 4-15  
 names  
 command, 3-1, 3-3  
 register, 3-6  
 NOMEM command, 3-41, 5-13  
 notational conventions in a command line, 3-2  
 null character, 3-21, 4-4

## O

object code, 12-3  
 operators in expressions, 3-6  
 optional products compatible with 8540, 1-5  
 OS/40, 12-3  
 compared with DOS/50, 9-1  
 output. See I/O  
 overview of related user manuals, 1-41  
 overview of this manual, 1-40  
 overwrite, 6-11

## P

P command, 3-42  
 examples, 2-19, 5-4  
 paper tape reader/punch, 1-6  
 parameters, 3-3  
 patch (alter) memory contents, 3-42  
 periods (...) in syntax blocks, 3-2  
 peripherals. See I/O devices  
 permanent strings, 3-43  
 PERMSTR command, 3-43  
 examples, 1-14  
 plugging in the 8540, 1-8  
 PPTP, 1-6  
 PPTR, 1-6  
 PRO command, 5-15  
 proceed (in SVCs), 6-10  
 program (user's):  
 aborting, 3-8  
 breaking, 2-12, 2-14-2-15, 2-18, 3-13, 5-4, 5-16  
 executing, 5-4  
 executing repeatedly, 5-4  
 halting execution, 3-8, 3-56, 5-4, 6-12  
 loading, 3-34  
 modifying, 2-10-2-12  
 monitoring, 5-4  
 timing a segment, 2-17, 2-18, 3-17, 5-16-5-17  
 program clock, 3-17, 5-5, 12-3  
 program memory. See Memory

program section. See Section  
 PROM Programmer, 1-6, Section 8  
 protocols. See Intersystem communication  
 prototype, 12-3  
 hardware considerations, 5-6, Section 7  
 I/O, 5-2-5-3  
 prototype control probe, 12-3  
 timing considerations, Section 7  
 prototype memory. See Memory  
 PSTAT command, Section 8  
 PTYPE command, Section 8

## Q

QUA command, 5-15  
 QUERY command, 3-44  
 quote, double (") and single ('), 3-5

## R

radix selector letters, 3-5  
 RD command, 3-45  
 reading:  
 from emulator port, 3-45  
 hexadecimal code into memory, 3-48  
 read-only memory (ROM). See Memory  
 redirecting I/O, 3-3-3-4  
 registers:  
 add contents of, 3-15  
 changing contents of, 3-51  
 displaying contents of, 5-5  
 names, 3-6  
 reinitializing emulator, 3-47  
 REMI, 1-8  
 REMO, 1-8  
 REMOTE port status register, 4-25  
 remove symbol from symbol table, 3-46  
 REMS command, 3-46, 5-9  
 RESET command, 3-47  
 resident commands, 3-7  
 resuming display, 3-6-3-7  
 RETURN key (carriage return), 1-10  
 rewind, 6-15  
 RH command, 3-48  
 ROM. See Memory  
 ROMPATCH command, 3-49  
 RPR command, Section 8  
 RUBOUT key, 3-7

## S

S command, 2-9, 3-51  
 SAS format for SRB, 6-2  
 SAV command, 2-2, 3-52  
 save memory contents:  
 in file, 3-52  
 in hexadecimal format, 3-64  
 SEA command, 3-53  
 search memory for value or string, 3-53

- section (of a program), 1-19
    - in symbolic debug, 1-33
    - in Tekhex symbol blocks, 4-13
  - seek (in SVCs), 6-11, 6-15-6-16
  - SEL command, 3-54
  - select
    - emulation mode, 3-28
    - target processor, 3-54
  - SELF TEST light:
    - on 8301 front panel, 1-9
    - on PROM Programmer front panel module, 8-2
  - semicolon (;), 1-13, 3-2
  - SEND (TEKDOS command), 4-6
  - service calls. See SVCs
  - service request block. See SRB
  - set:
    - breakpoint, 3-13-3-14
    - memory map assignments, 3-35-3-36
    - register or symbol to value, 3-51
  - single quote ('), 3-5
  - special characters. See Characters, special
  - special keys. See Keys, special
  - SRB, 6-4
    - examples, 6-9
    - status codes, 6-4
  - SRB vector, 6-3
    - examples, 6-9
  - standard input and output, 12-4
    - redirecting, 3-4
  - Standard Tekhex, 4-10
  - starting up the 8540, 1-8
  - STARTUP string, 1-9, 1-14
  - STAT command, 2-20, 3-55
  - status of SVC, 6-4
  - STR command, 3-55
  - strings, 1-13, 3-5
    - permanent, 3-43
    - STARTUP, 1-9, 1-14, 3-23
    - temporary, 3-55
  - SUSP command, 3-56
  - suspendable (abortable) commands, 3-8
  - suspending display, 3-6-3-7
  - SVC command, 3-57, 6-3, 6-4
    - examples, 2-13-2-14
  - SVCs, Section 6
    - 8540 compared with 8550, 9-2
    - accessing host files, 6-4
    - applications, 6-5-6-6
    - breaking on an SVC, 5-6
    - demonstration, 6-6-6-9
    - I/O instruction, 6-3, Section 7
    - LAS memory layout, 6-3
    - phasing SVCs out of a program, 5-6
    - SAS memory layout, 6-2
  - SYMB command, 3-58, 5-9
  - symbolic debug, 1-33-1-34, 5-7-5-9
    - Tekhex symbol block, 4-13
  - symbols, 3-6
    - adding, 3-9
    - displaying, 3-59
    - loading, 3-61
    - removing, 3-46
    - returning symbolic equivalent of value, 3-58
  - SYMD command, 3-59
  - SYMLO command, 2-2, 3-61, 5-9
  - syntax block, 3-2
  - syntax notation conventions, 3-2
  - system I/O, 2-20
  - system terminal, 1-5
- T**
- target processor, 12-4
    - selecting the target processor, 3-54, 5-3
  - TCLR command, 5-16
  - TEKDOS SEND command, 2-7-2-8, 4-6
  - Tekhex (Tektronix Hexadecimal Format), 3-21, 4-9-4-21, 11-4
    - conversion program, 4-18-4-21
    - receive program, 4-17
    - transmit program, 4-16
  - Tektronix Hexadecimal Format. See Tekhex
  - temporary strings, 3-4-3-5, 3-55
  - TERM mode, 1-7, 1-11, 3-22, 4-1
  - terminal I/O, logging to device or file, 3-34
  - test clips, 5-14
  - timing, 5-5
    - considerations for prototype control probe, Section 7
    - factors that affect timing, 5-5
  - timing a program segment:
    - using CLOCK, 3-17
    - using SVCs, 6-5
    - using the TTA, 2-17, 2-18, 5-16-5-17
  - TNIX, 1-7
  - TRA command, 3-62
    - examples, 1-35-1-37, 2-10-2-12
  - tracing:
    - using the TRA command, 2-10-2-12, 3-62-3-63, 5-4
    - using the TTA, 2-16-2-18, 5-18
  - transfer, data, 3-21
  - transfer address, 12-5
  - trigger, 5-13
  - Trigger Trace Analyzer. See TTA
  - TS command, 3-2
  - TTA:
    - Acquisition Memory, 5-16-5-17, 12-1
    - breakpoints, 5-16
    - clearing, 5-16
    - commands, 5-19
    - compared with RTPA, 9-2
    - event, 5-14-5-16
      - consecutive, 5-16
    - examples of use, 2-14-2-18
    - general purpose counters, 5-16-5-17
    - information monitored by, 5-14
    - trigger, 5-13-5-18
  - turning off the 8540, 1-10
  - turning on the 8540, 1-8
  - type-ahead, 1-13
- U**
- underlined characters:
    - in examples, 1-10
    - in syntax blocks, 3-2
  - unformatted transfers, 4-7
    - See also Intersystem communication
  - UNIX. See TNIX
  - upload, 12-5
    - See also Intersystem communication
  - uppercase. See Capitalization
  - user prototype. See Prototype

**V**

variable-length numbers and symbols, 4-12  
verification of 8540 hardware, 1-1  
virtual memory, 2-9

**W**

wait (in SVCs), 6-10  
WH command, 3-64  
WPR command, Section 8  
write  
    hexadecimal, 3-64  
    protection:  
        of program memory, 5-4  
        of prototype memory, 3-35, 5-7  
    to emulator I/O port, 3-65  
WRT command, 3-65

**X**

X command, 3-65

**Z**

zero-insertion-force (ZIF) socket, 8-1  
zeroing out memory, 1-26