Tektronix®
COMMITTED TO EXCELLENCE

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 ______________

070-3964-01
Product Group 61

First Printing SEP 1981
Revised FEB 1984

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.

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

Section 7C
CONTENTS
Z80 EMULATOR SPECIFICS

CONTENTS (cont)

ILLUSTRATIONS

TABLES

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 boad 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

Symbol	Description	Size in Bits	Value After RESET[a]	Alterable by S Command?
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 [b]	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 [b]	8	NC	yes
H	Register H	8	NC	yes
HC	Auxiliary carry flag [b]	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 [c]
IFF2	Interrupt flip-flop 2	1	0	yes [c]
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 [b]	1	NC	yes
O	Overflow flag [b]	1	NC	yes
P	Parity flag [b]	1	NC	yes
PC	Program counter	16	0000	no
R	Memory refresh register	8	00	yes
S	Sign flag [b]	1	NC	yes
SP	Stack pointer	16	NC	yes
Z	Zero flag [b]	1	NC	yes

[a] NC = not changed by RESET

[b] The flag register is illustrated in Fig. 7C-1.

[c] The S command performed on either IFF1 or IFF2 sets both.

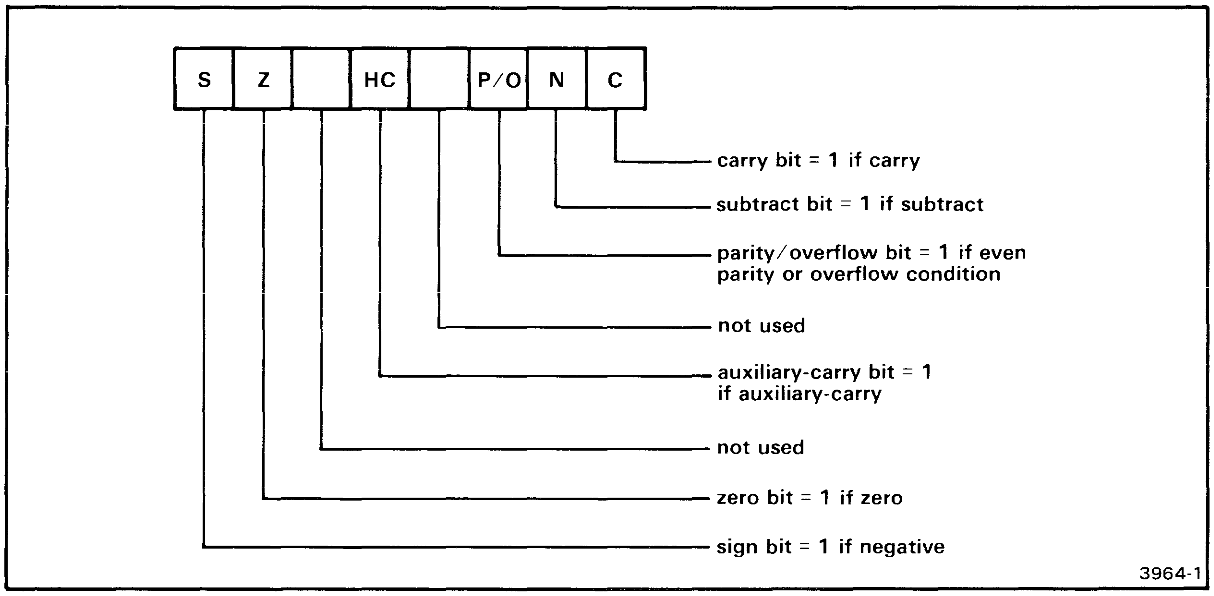

S
Z
HC
P/O
N
C
carry bit = 1 if carry
subtract bit = 1 if subtract
parity/overflow bit = 1 if even parity or overflow condition
not used
auxiliary-carry bit = 1 if auxiliary-carry
not used
zero bit = 1 if zero
sign bit = 1 if negative
3964-1

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
   ▲     ▲       ▲     ▲
   !     !       !     !
   !     !       !     !------ operand (s): address, register, or data
   !     !       !             being operated on
   !     !       !
   !     !       !------------- instruction mnemonic
   !     !
   !     !--------------------- machine language instruction
   !
   !--------------------------- address of the instruction
```

TRA—Sample Z80 TRAce Display

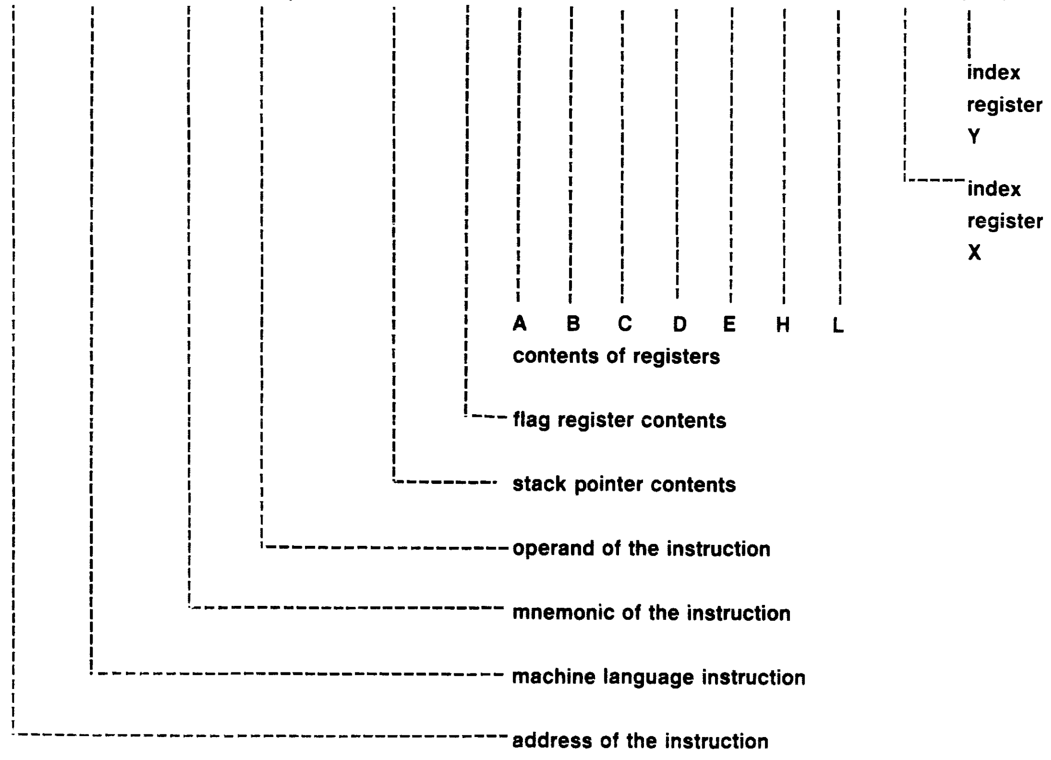
The TRA (TRAce) command selects the range and type of instructions to be displayed as your program executes. With the Z80 emulator, the TRA -N format is the same as the TRA -L format.

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

	Z80 Service Calls		Default Address of
SVC Number	mnemonic[a]	hexadecimal	SRB pointer
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

[a]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
        ORG     1000H           ; ENTRY POINT INTO PROGRAM
START   OUT     (0F7H),A        ; CALL SVC1
        NOP                     ; TO ASSIGN "CONI"
        LD      A,(SRB1ST)      ; CHECK THE STATUS TO SEE
        CP      00H             ; IF ALL WENT WELL
        JP      NZ,ABORT        ; NO? STOP EVERYTHING
        OUT     (0F6H),A        ; YES? CALL SVC2
        NOP                     ; TO ASSIGN "LPT"
        LD      A,(SRB2ST)      ; CHECK THE STATUS TO SEE
        CP      00H             ; IF ALL WENT WELL
        JP      NZ,ABORT        ; NO? STOP EVERYTHING
LOOP    OUT     (0F5H),A        ; CALL SVC3
        NOP                     ; TO READ A "CONI" LINE
        LD      A,(SRB3ST)      ; INTO "BUFFER"
        CP      00H             ; ALL OK?
        JP      NZ,ABORT        ; NO? STOP EVERYTHING
        OUT     (0F4H),A        ; CALL SVC4
        NOP                     ; TO WRITE TO "LPT"
        LD      A,(SRB4ST)      ; CHECK TO SEE IF
        CP      00H             ; ALL IS OK
        JP      Z,LOOP          ; YES? BACK TO READ ANOTHER LINE
;                                 NO? FALL THROUGH TO TERMINATION
ABORT   OUT     (0F3H),A        ; CALL SVC5
        NOP                     ; TO DO THE ABORT
        HALT                    ; SHOULD NEVER REACH HERE
        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.

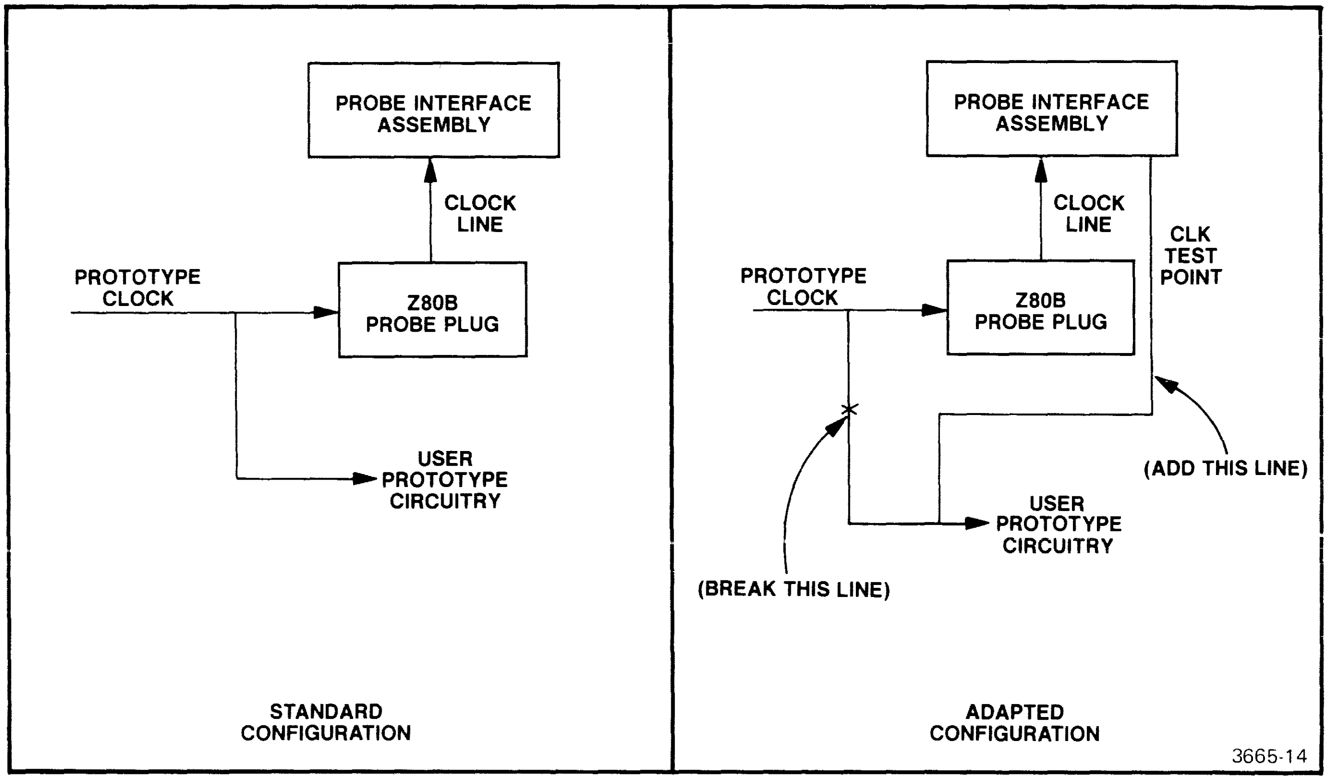


Fig. 7C-3. Connecting the prototype clock input directly to the Prototype Control Probe.

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 Ω 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 Min.	Processor Max.	Emulator Min.	Emulator Max.	Units
t(c)	Clock period	250	[a]	250	[a]	ns
t(w(ϕH))	Clock pulse width, clock high	110	[b]	110	[b]	ns
t(w(ϕL))	Clock pulse width, clock low	110	2000	110	2000	ns
t(r,f)	Clock rise and fall time		30	[c]	30	ns
t(D(AD))	Address output delay		110		130	ns
t(F(AD))	Delay to float				85[d]	ns
t(D(D))	Data output delay		150		170	ns
t(F(F))	Delay to float during write cycle		90		110[e]	ns
t(Sϕ(D))	Data setup time to rising edge of clock during M1 cycle	50		70[f]		ns
t(Sϕ(D))	Data setup time of falling edge of clock during M2 to M5	60		80[f]		ns
t(DLϕ(MR))	MREQ delay from falling edge of clock, MREQ low	20	85	35[g]	100[g]	ns
t(DHϕ(MR))	MREQ delay from rising edge of clock, MREQ high		85		100[g]	ns
t(DHϕ(MR))	MREQ delay from falling edge of clock, MREQ high		85		100[g]	ns
t(DLϕ(IR))	IORQ delay from rising edge of clock, IORQ low		75		90[g]	ns
t(DLϕ(IR))	IORQ delay from falling edge of clock, IORQ low		85		100[g,h]	ns
t(DHϕ(IR))	IORQ delay from rising edge of clock, IORQ high		85		100[g]	ns
t(DHϕ(IR))	IORQ delay from falling edge of clock, IORQ high		85		100[g]	ns
t(DLϕ(RD))	RD delay from rising edge of clock, RD low		85		95[g]	ns
t(DLϕ(RD))	RD delay from falling edge of clock , RD low		95		105[g]	ns
t(DHϕ(RD))	RD delay from rising edge of clock, RD high		85		95[g]	ns
t(DHϕ(RD))	RD delay from falling edge of clock, RD high		85		95[g]	ns
t(DLϕ(WR))	WR delay from rising edge of clock, WR low		65		75[g]	ns

Table 7C-4 (cont)
Representative Z80A Emulator/Z80A Microprocessor Timing Differences

Symbol	Parameter	Processor		Emulator		Units
		Min.	Max.	Min.	Max.	
t(DLϕ(WR))	WR delay from falling edge of clock, WR low		80		90[g]	ns
t(DHϕ(WR))	WR delay from falling edge of clock, WR high		80		90[g]	ns
t(DL(M1))	M1 delay from rising edge of clock, M1 low		100		110[g]	ns
t(DH(M1))	M1 delay from rising edge of clock, M1 high		100		110[g]	ns
t(DL(RF))	RFSH delay from rising edge of clock, RFSH low		130		140[g]	ns
t(DH(RF))	RFSH delay from rising edge of clock, RFSH high		120		130[g]	ns
t(S(WT))	WAIT setup time to falling edge of clock	70		90[f]		ns
t(D(HT))	HALT delay time from falling edge of clock		300		315[g]	ns
t(s(IT))	INT setup time to rising edge of clock	80		125[f]		ns
t(s(BQ))	BUSRQ setup time to rising edge of clock	50		85[f]		ns
t(DL(BA))	BUSAK delay from rising edge of clock, BUSAK low		100		110[g]	ns
t(DH(BA))	BUSAK delay from falling edge of clock, BUSAK high		100		110[g]	ns
t(s(RS))	RESET setup time to rising edge of clock	60		105[f]		ns
t(F(C))	Delay to/from float (MREQ, IORQ, RD, and WR)		80		115[g]	ns
t(mr)	M1 stable prior to IORQ Interrupt Ack.)	[i]		[h,i]		ns

[a]t(c) = t(w(ϕH)) + t(w(ϕL)) + t(r) + t(f)

[b]Although static by design, testing guarantees a t(w(ϕH)) of 200 μs maximum.

[c]Clock delay from the prototype to emulator CPU is 20 ns maximum.

[d]Delay measured from BUSAK asserted at CPU.

[e]Data will go to an indeterminate state, but will not tri-state unless BUSREQ is asserted.

[f]Timing reference is to CPU clock. To reference prototype clock, subtract propagation delay through driver/receiver buffers (10 ns maximum).

[g]Timing reference to CPU clock. To reference prototype clock, add propagation delay through driver/receiver buffers (20 ns maximum.)

[h]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(ϕH)) + 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 Min.	Processor Max.	Emulator Min.	Emulator Max.	Units
t(c)	Clock period	165	[a]	165	[a]	ns
t(w(ϕH))	Clock pulse width, clock high	70	[b]	70	[b]	ns
t(w(ϕL))	Clock pulse width, clock low	70	2000	70	2000	ns
t(r,f)	Clock rise and fall time		30	[c]	30	ns
t(D(AD))	Address output delay		80		100	ns
t(F(AD))	Delay to float				33[d]	ns
t(D(D))	Data output delay		120		140	ns
t(F(F))	Delay to float during write cycle		60		80[e]	ns
t(Sϕ(D))	Data setup time to rising edge of clock during M1 cycle	25		4.5[d]		ns
t(Sϕ(D))	Data setup time to falling edge of clock during M2 to M5	30		5.0[f]		ns
t(DLϕ(MR))	MREQ delay from falling edge of clock, MREQ low	20	60	35[g]	75[g,h]	ns
t(DHϕ(MR))	MREQ delay from rising edge of clock, MREQ high		60		75[g]	ns
t(DHϕ(MR))	MREQ delay from falling edge of clock, MREQ high		60		75[g]	ns
t(DLϕ(IR))	IORQ delay from rising edge of clock, IORQ low		60		75[g,h]	ns
t(DLϕ(IR))	IORQ delay from falling edge of clock, IORQ low		60		75[g]	ns
t(DHϕ(IR))	IORQ delay from rising edge of clock, IORQ high		60		75[g]	ns
t(DHϕ(IR))	IORQ delay from falling edge of clock, IORQ high		60		75[g]	ns
t(DLϕ(RD))	RD delay from rising edge of clock, RD low		60		70[g]	ns
t(DLϕ(RD))	RD delay from falling edge of clock, RD low		70		80[g]	ns

Table 7C-5 (cont)
Representative Z80B Emulator/Z80B Microprocessor Timing Differences

Symbol	Parameter	Processor Min.	Processor Max.	Emulator Min.	Emulator Max.	Units
t(DHϕ(RD))	RD delay from rising edge of clock, RD high		60		70[g]	ns
t(DHϕ(RD))	RD delay from falling edge of clock, RD high		70		80[g]	ns
t(DLϕ(WR))	WR delay from rising edge of clock, WR low		60		70[g]	ns
t(DLϕ(WR))	WR delay from falling edge of clock, WR low		55		65[g]	ns
t(DHϕ(WR))	WR delay from falling edge of clock, WR high		55		65[g]	ns
t(DL(M1))	M1 delay from rising edge of clock, M1 low		80		90[g]	ns
t(DH(M1))	M1 delay from rising edge of clock, M1 high		80		90[g]	ns
t(DL(RF))	RFSH delay from rising edge of clock, RFSH low		100		110[g]	ns
t(DH(RF))	RFSH delay from rising edge of clock, RFSH high		100		110[g]	ns
t(S(WT))	WAIT setup time to falling edge of clock	40		60[f]		ns
t(D(HT))	HALT delay time from falling edge of clock		200		215[g]	ns
t(s(IT))	INT setup time to rising edge of clock	45		90[f]		ns
t(s(BQ))	BUSRQ setup time to rising edge of clock	40		75[f]		ns
t(DL(BA))	BUSAK delay from rising edge of clock, BUSAK low		65		75[g]	ns
t(DH(BA))	BUSAK delay from falling edge of clock, BUSAK high		55		65[g]	ns
t(s(RS))	RESET setup time to rising edge of clock	45		90[f]		ns
t(F(C))	Delay to/from float (MREQ, IORQ, RD, and WR)		60		95[g]	ns
t(mr)	M1 stable prior to IORQ (Interrupt Ack.)	[i]		[i,j]		ns

[a]t(c) = t(w(ϕH)) + t(w(ϕL)) + t(r) + t(f).

[b]Although static by design, tesing guarantees a t(w(ϕH)) of 200 μs maximum.

[c]Clock delay from prototype to emulator CPU is 20 ns maximum.

[d]Delay measured from BUSAK asserted at CPU.

[e]Data will go to an indeterminate state, but will not tri-state unless BUSREQ is asserted.

[f]Timing reference is to CPU clock. To reference prototype clock, subtract propagation delay through Driver/Receiver Buffers (10 ns maximum).

[g]Timing reference to CPU clock. To reference prototype clock, add propagation delay through Driver/Receiver Buffers (20 ns maximum).

[h]Emulation Mode 1, on a memory change from system to prototype, MREQ or IORQ would be delayed up to 184 ns.

[i]t(mr) = 2 * t(c) + t(w(ϕH)) + t(f) −50.

[j]IORQ can be delayed a maximum of 153 ns during INTA cycles while the Debug TRA MODE is active.

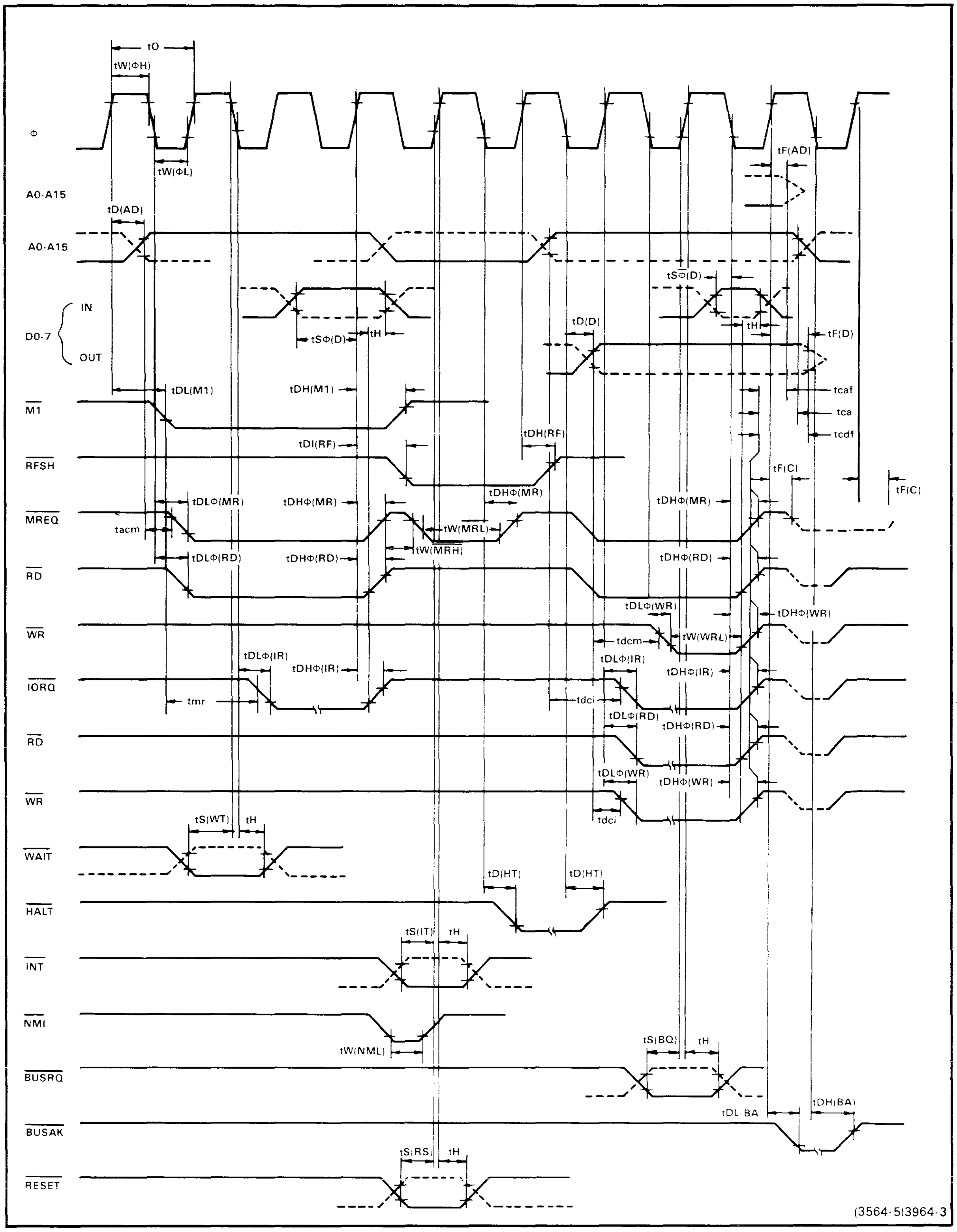


Fig. 7C-4. Z80 microprocessor bus timing.

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.

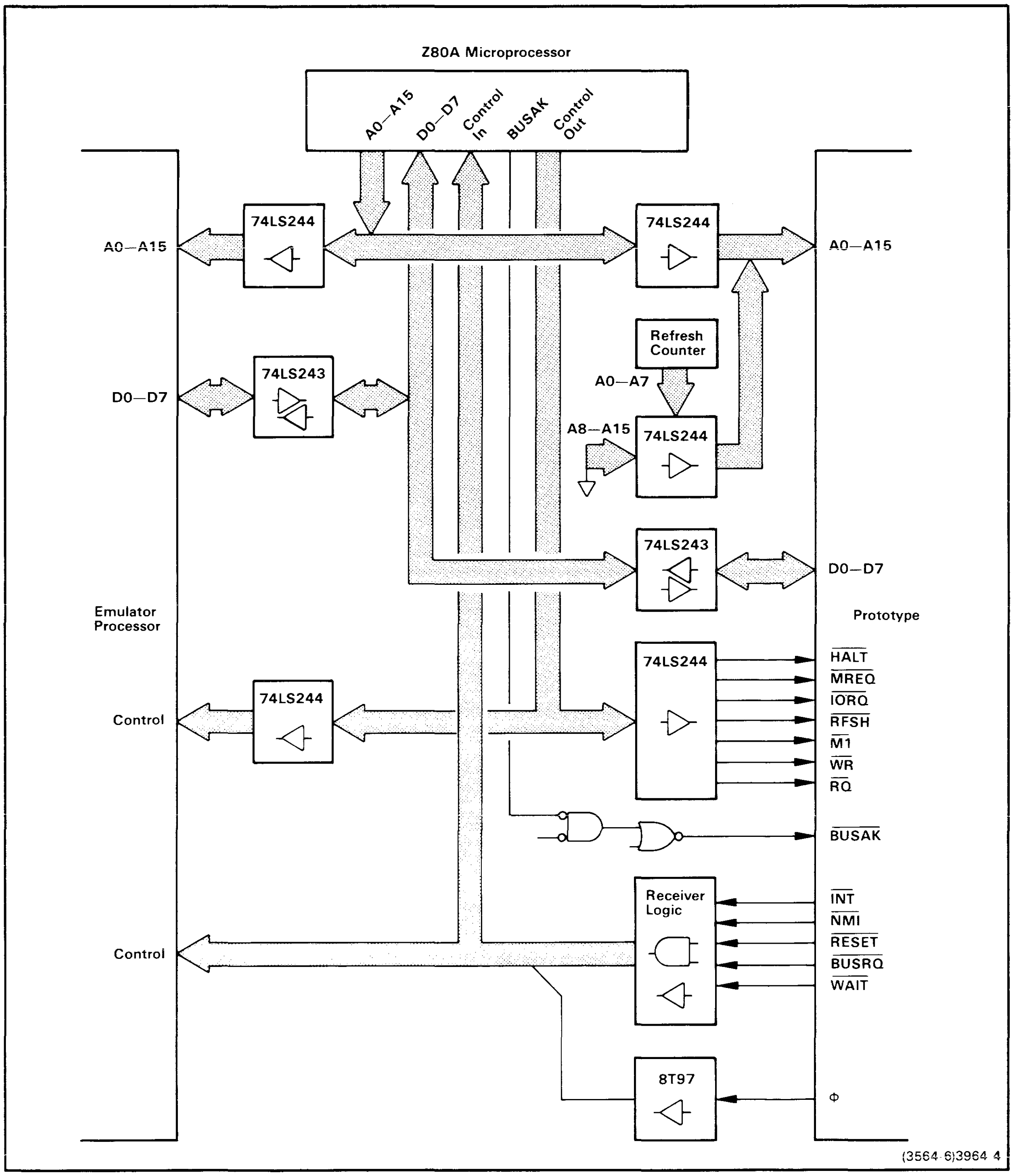


Fig. 7C-5. Block diagram of Z80A emulator/prototype interface.

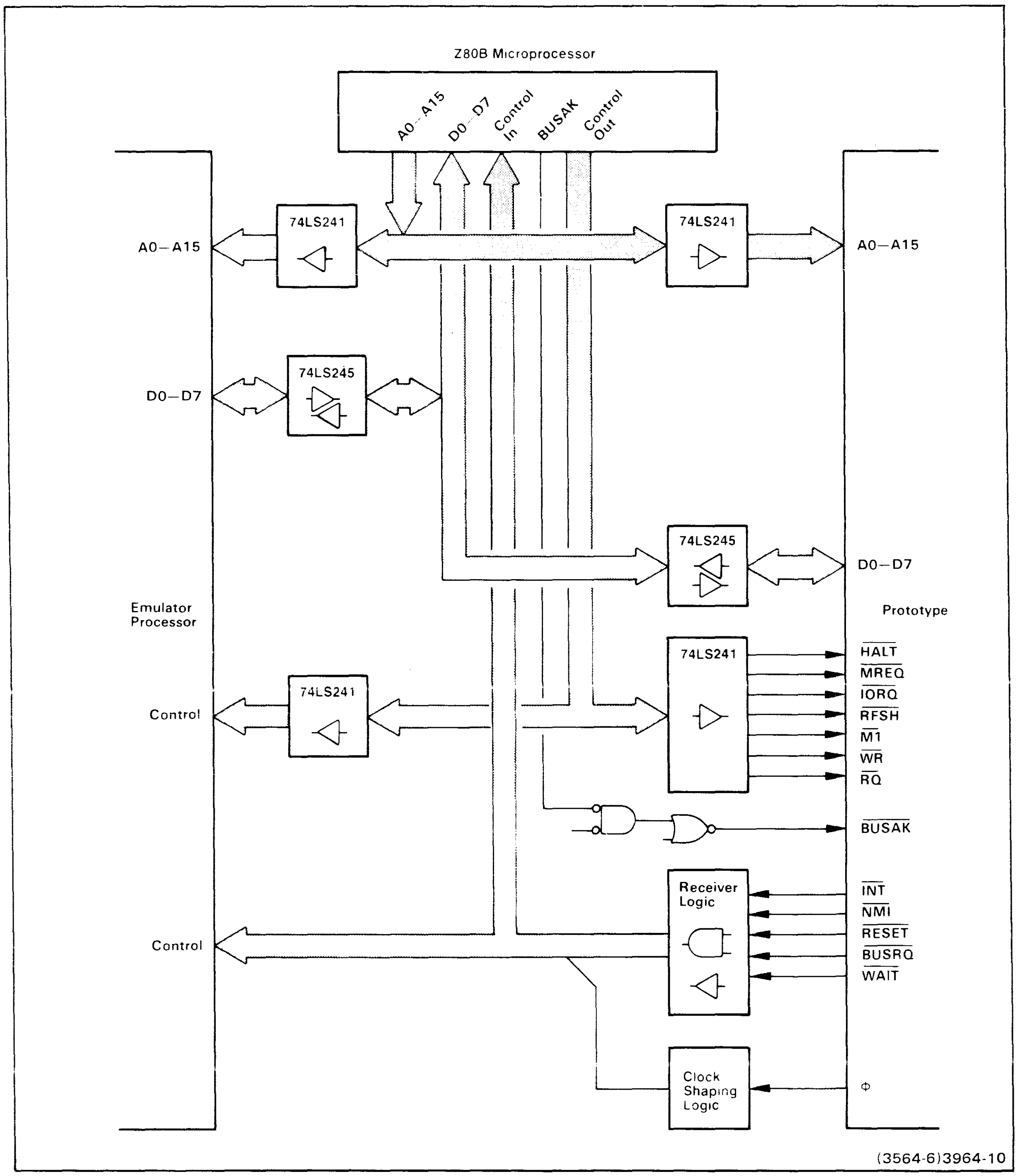


Fig. 7C-6. Block diagram of Z80B emulator/prototype interface.

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 CO -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 CO -A to continue execution of the command file:

```
> CO -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 CO -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 CO -A command again:

```
> CO -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.

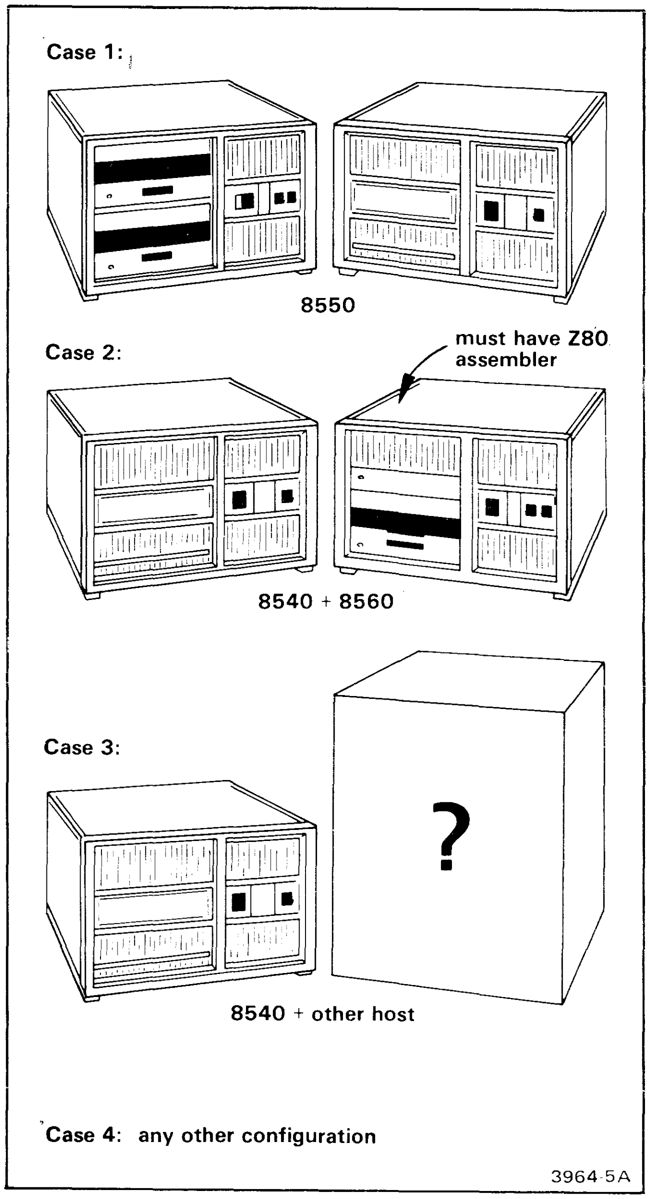


Fig. 7C-7. System configurations.

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
== ====== ======  ===================  ==================================
|    |      |              |                     |
|    |      |         source code            comments
|    |      |
|    |      +-- object code
|    |
|    +-------- address
|
+------------- source code line number
```

Fig. 7C-8. Demonstration program.

```
(A)

%2769231002100050605AF862305C20601D3F700
%0E62B24000421A
%3A3494DEM0010350514LOOP310615START310015TABLE350025TSIZE15
%09815 3100


(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  definition
  |   name    field      symbol definition fields
  |     |      |                     |
  |     |      |                     |
======----========-----------------------------------------
%3A3494DEM0010350514LOOP310615START310015TABLE350025TSIZE15


TERMINATION BLOCK

header
  |   transfer
  |   address
  |    |
  |    |
======----
%09815 3100
```

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 lable 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 shown 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 affect 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* (*a*ppend text) and type in the program. Use the BACK-SPACE key to erase any typing mistakes.

```
a <CR>
       column  column  column
         8       6       24
         |       |       |
         v       v       v
;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.

```
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, 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 *asml* and *obj:*

```
$ asm obj asml asm <CR>
      ^    ^    ^
      |    |    |
      |    |    L-----source file
      |    |
      |    L--------- assembler listing file
      |
      L------------ 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 asml <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 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 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
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 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:

```
$ lo <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 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 (LD HL, LD B, and XOR A) starting at location 100:

```
> P 100 210005 0605 AF <CR>
    === ====== ==== ==
     ^     ^     ^   ^
     |     |     |   |
     |     |     |   XOR A
     |     |     |
     |     |     LD B,TSIZE
     |     |
     |     LD HL,TABLE
     |
  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 + OF)
        ===== ==
> 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=OF.

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 begining 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:

```
> tra jmp loop 109 <CR>
             ==== ===
              ^   ^
              |   |
              |   '--- upper address      {   Within this range,
              |                           {   only jump instructions
              '------ lower address       {   are traced.
                      (106)
```

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 (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
+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 finishd 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:

```
  $ cs ..  <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