OPERATIONS MANUAL

FOR

DDT-80 VERSION 1.3 OPERATING SYSTEM (MK78118)

AND

ASMB-80 VERSION 1.0

TEXT EDITOR, ASSEMBLER, LINKING LOADER (MK78119)

LIST OF FIGURES

LIST OF TABLES

SECTION 1

DDT-80 OPERATING SYSTEM

1-1. INTRODUCTION

1-2. This section describes the functions and operation of DDT-80 (Designer's Development Tool 80). The DDT-80 software is an integral part of the SDB-80 or MDX system and a knowledge of its operation is essential in order to use the system effectively.

1-3. OVERVIEW

1-4. The 'personality' of a microcomputer development system is in many ways determined by the software that communicates with the user. This software is generally known as the Operating System. It provides the necessary tools and techniques to operate the sytem, i.e., to efficiently and conveniently perform the tasks necessary to develop microcomputer software. The DDT-80 Operating System described here is designed to support the user from initial design through production testing. DDT-80 is a resident operating system that allows the user to display and update memory, registers, and ports, load and dump paper tape, set breakpoints, and execute programs.

1-5. The following paragraphs describe the functions and operation of DDT-80 as it applies to the SDB-80 and MDX microcomputer systems.

1-6. MEMORY AND PORT ALLOCATION

1-7. DDT-80 is a 2K byte program that resides at locations E000H - E7FFH in the memory map. In addition to the 2K of ROM, DDT-80 uses 256x8 of RAM for scratch RAM and temporary storage. This RAM resides at locations FF00H -FFFFH and is discussed in paragraph 1-10.

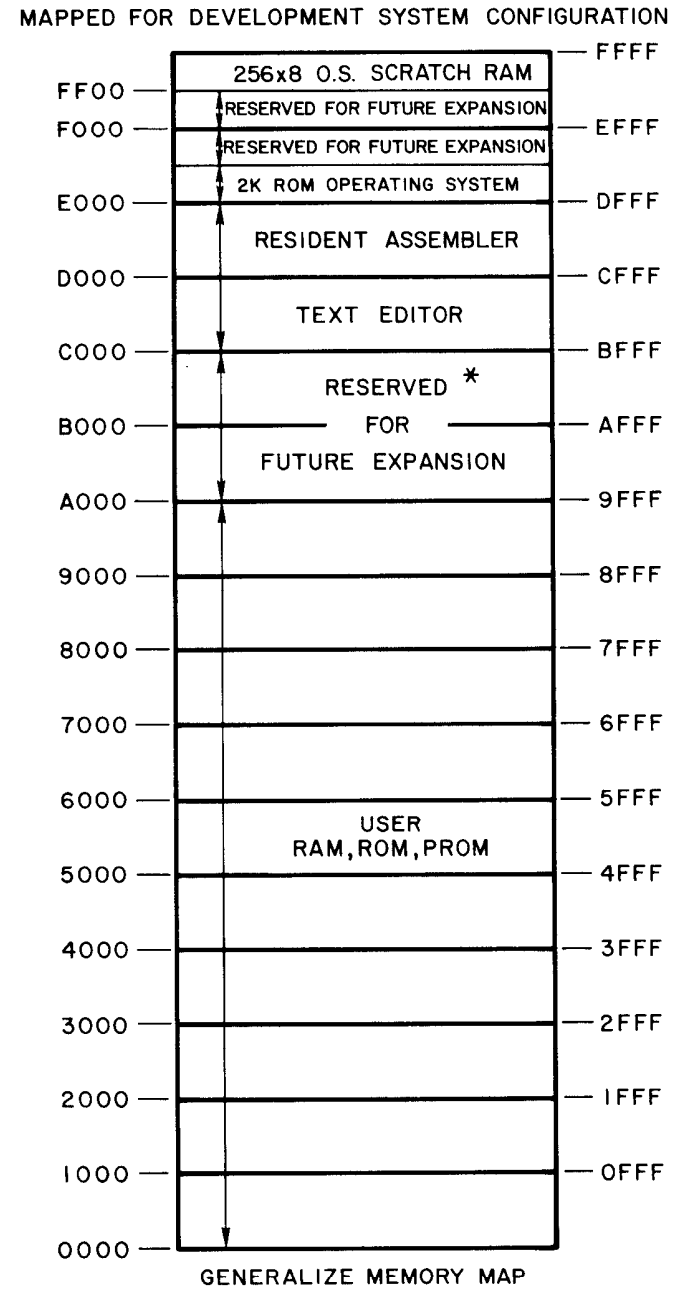
1-8. MEMORY MAP

1-9. Figure 1-1 depicts the memory address space for the ASMB-80/DDT-80 (development system configuration) showing the division between system and user memory. With the partitioning shown, it is intended that the user's program will generally reside at the low end of the memory map, with system software residing at the high end.

1-10. SCRATCHPAD RAM

1-11. The 256x8 Scratchpad RAM is used by the DDT-80 for temporary storage and a push down stack (for return address, etc.). This RAM also holds an image (or map) of all the user's internal CPU registers, a user's push down stack (separate from the DDT-80 stack), and space for user defined mnemonics (discussed in paragraph 1-43). Figure 1-2 is a detailed memory map of the 256x8 Scratchpad RAM. Note that the area between the user's mnemonics and the user's stack (the SP is initialized during

Figure 1-1 ASMB 80/DDT 80 MEMORY MAP



* DOES NOT APPLY TO THE MDX SYSTEM

power-up or reset) has no defined boundary. Thus the user must define this boundary with a tradeoff between stack size and the number of mnemonics defined.

1-12. An important concept in DDT-80 is preservation of the user's internal CPU registers. The state of the CPU is described by the contents of the registers. To preserve the state of the CPU for a user's program while debugging, DDT-80 keeps an image or map of all the user's registers. This image or map is referred to as the User Register Map throughout this documentation. DDT-80 installs or makes the CPU registers equal to the user register map when control is transferred from DDT-80 to a user program (as in the E command discussed in paragraph 1-62). DDT-80 saves the user register map when DDT-80 is commanded (breakpoint command discussed in paragraph 1-74) to interrupt a user program. DDT-80 allows modification to this register map with the display and/or update memory command (M command, discussed in paragraph 1-33). The user register map resides in the 256x8 Scratchpad location FFE6H thru FFFFH, as shown in Figure 1-3. Figure 1-4 shows the data paths between the user register map and the CPU registers. Also shown is the modification path between DDT-80 and the User Register Map.

Figure 1-2. SCRATCH PAD MEMORY MAP

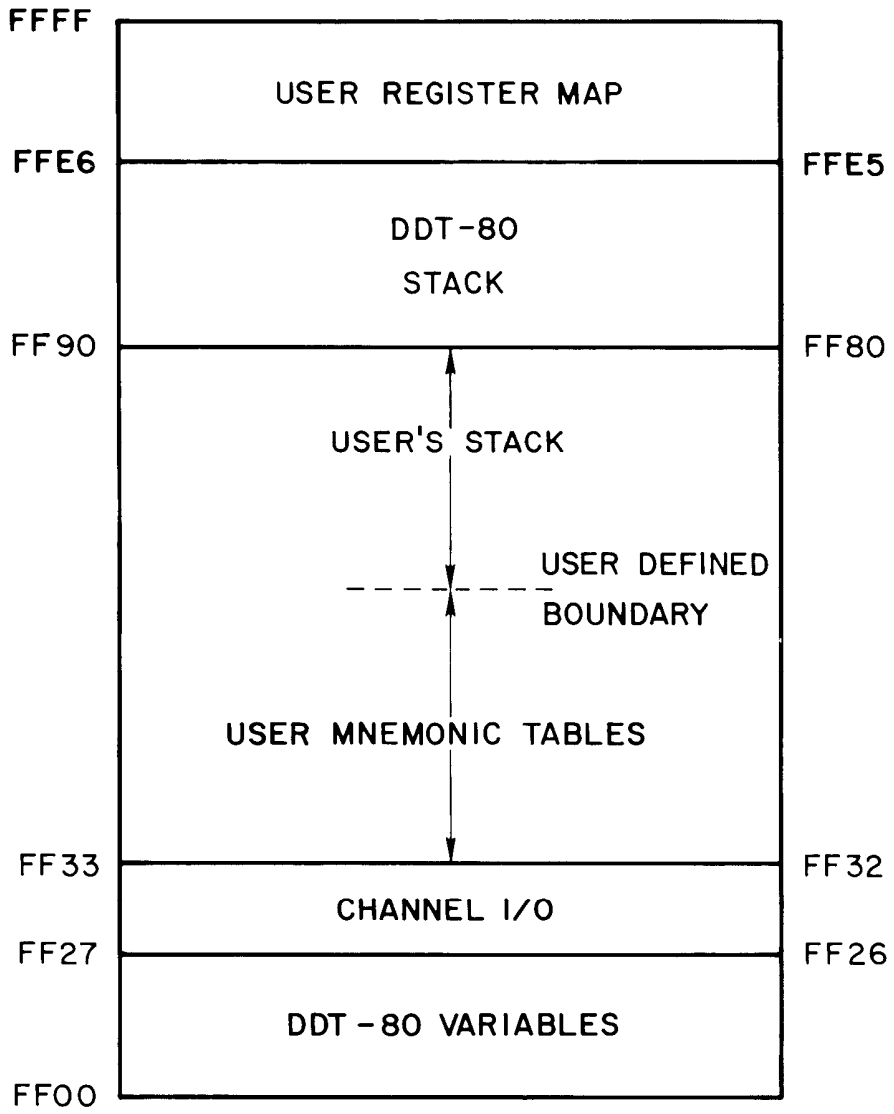
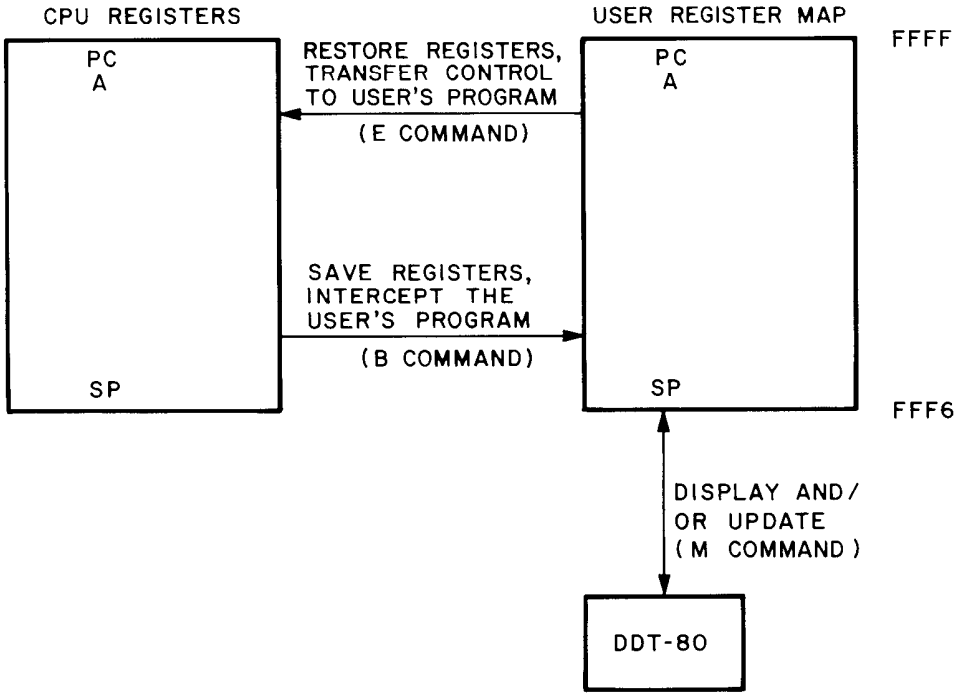


Figure 1-3 USER REGISTER MAP

MEMORY LOCATION	USER REGISTER		
FFFF	PC	PROGRAM	MSB
FFFE		COUNTER	LSB
FFFD		A	
FFFC		F	
FFFB		I	
FFFA		IF	
FFF9		B	
FFF8		C	
FFF7		D	
FFF6		E	
FFF5		H	
FFF4		L	
FFF3		A'	
FFF2		F'	
FFF1		B'	
FFF0		C'	
FFEF		D'	
FFEE		E'	
FFED		H'	
FFEC		L'	
FFEB		IX	MSB
FFEA			LSB
FFE9		IY	MSB
FFE8			LSB
FFE7	SP	STACK	MSB
FFE6		POINTER	LSB

Figure 1-4 DATA PATHS TO AND FROM THE USER REGISTER MAP



1-13. Another important concept in DDT-80 is channeled I/O. This concept gives the user the ability to select device drivers (software) to perform I/O to and from a peripheral through a channel. A channel is actually a fixed memory location where the address of that channel's selected I/O device drdiver is stored. When doing I/O through a specific channel, DDT-80 fetches the device driver address from the channel's fixed memory location. The address just fetched is then used to transfer control to the device driver for completion of the I/O function. A device driver is selected for a particular channel by storing the address of that driver in the channel's fixed location. These fixed locations reside in the 256x8 of Scratchpad RAM as shown in figure 1-5. This means the user selects the I/O driver for a peripheral merely by changing the contents of the fixed location for the channel through which the I/O is to be done (DDT-80 makes this easy, see paragraph 1-42).

1-14. Even more important, the user may write his own I/O driver for a particular I/O device he owns. The user then loads this I/O driver (software) into RAM and configures the device into his system by changing the contents of the I/O channel's fixed location. DDT-80 has 6 I/O channels:

Console Input
Console Output
Object Input
Object Output
Source Input
Source Output

Figure 1-5 I/O CHANNEL FIXED LOCATIONS

FF32	SOURCE OUT DRIVER	MSB
FF31		LSB
FF30	SOURCE IN DRIVER	MSB
FF2F		LSB
FF2E	OBJECT OUT DRIVER	MSB
FF2D		LSB
FF2C	OBJECT IN DRIVER	MSB
FF2B		LSB
FF2A	CONSOLE OUT DRIVER	MSB
FF29		LSB
FF28	CONSOLE IN DRIVER	MSB
FF27		LSB

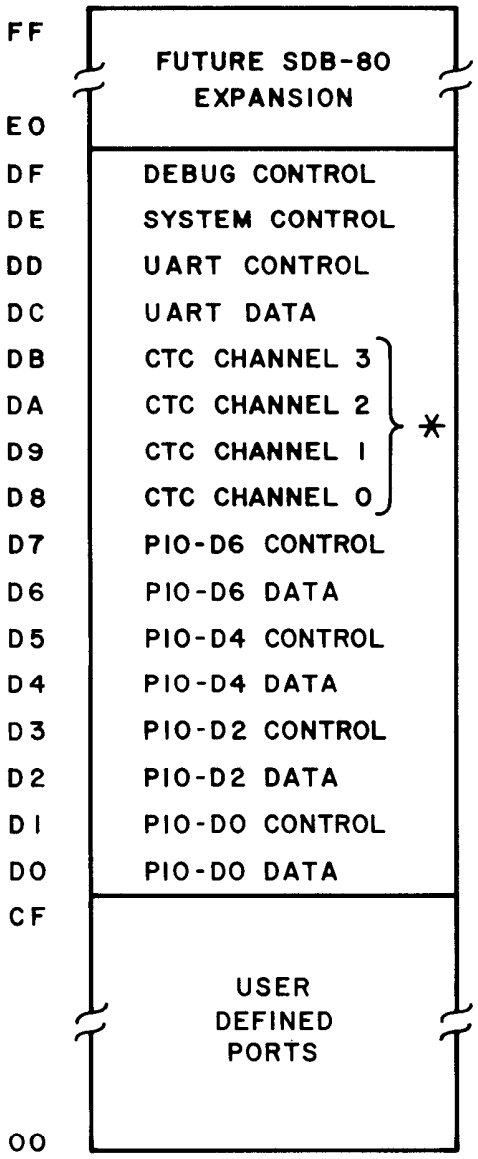
MSB-MOST SIGNIFICANT BYTE OF I/O DRIVER ADDRESS

LSB-LEAST SIGNIFICANT BYTE OF I/O DRIVER ADDRESS

1-15. PORT MAP

1-16. Figure 1-16 defines the port allocation on the SDB-80, and the MDX microcomputer system. Port DE is used for controlling data set ready (DSR), clear to send (CTS), and reader step (RS). Also, Port DE is used for sensing the state of data terminal ready (DTR), request to send (RTS), and serial bit string for measuring baud rate (used by the operating system). Ports DC and DD are the UART ports. MOSTEK is reserving ports E0H thru FFH for future expansion of its development system. It is recommended that the user limit his development system application to 00H thru CFH. Of course for an OEM application all 256 ports are available to the user. In the event any development system add on peripheral would exceed the assigned number of ports, we would start with CFH and work down.

Figure 1-6 PORT ALLOCATION



* THE CTC I/O PORTS FOR THE MDX-CPU1 CARD ARE 7C,7D,7E AND 7F THE MDX-SYSTEM DOES NOT USE CTC CHANNEL Ø FOR AN AUTO BAUD RATE GENERATOR

1-17. COMMAND FORMATS

1-18. DDT-80 recognizes commands which consist of three parts:

1. A single letter identifier.
2. An operand, or operands separated by commas or blanks.
3. A terminator to either abort the command or cause it to be executed.

1-19. Depressing the reset button for the SDB-80 causes DDT-80 to wait for the user to type in the character "S" or carriage return. This automatically sets the correct Baud rate. DDT-80 then types a carriage return, line feed, and a period "." to indicate that it is ready to receive a command. DDT-80 echos the command letter, prints a space, and then waits for the user to key-in the appropriate operand (s) in the format described below. A command is not executed until terminated by a carriage return (or one of the special terminators described below for display and update commands) and may be aborted at any time by a period. DDT-80 automatically supplies a line feed for the carriage return.

NOTE: The MDX-DEBUG card does not use the auto baud rate feature in DDT-80. To start DDT-80 executing from a reset, it is still necessary to type a character from the console terminal. See the MDX-DEBUG Operations Manual for selecting the baud rate.

1-20. COMMAND IDENTIFIERS

1-21. The following summarizes the 9 different single letter identifiers recognized as command identifiers.

1. M - Display, update, or tabulate the contents of memory.
2. P - Display and/or update the contents of an I/O port.
3. D - Dump the contents of memory in a format suitable to be read by the L command.

4. L - Load, into memory, data which is in the appropriate format.
5. E - Transfer control from DDT-80 to a user's program.
6. H - Perform 16 bit hexadecimal addition and/or subtraction.
7. C - Copy the contents of a block of memory to another location in memory.
8. B - Insert a breakpoint in the user's program (must be in RAM) which transfers control back to DDT-80. This allows the user to intercept his program at a specific point and examine memory and CPU register to determine if his program is working correctly.
9. R - Display the contents of the user registers.

1-22. COMMAND OPERANDS

1-23. A command operand represents 4 hexadecimal digits. The general definition of a command operand is as follows:

aaaa:=±bbbb±cccc±....±zzzz

1-24. DDT-80 allows arithmetic expressions (addition and/or subtraction), therefore the 4 hexadecimal digits, aaaa, can be calculated with a string of additions and/or subtractions. The sign of bbbb is assumed + if omitted. The values bbbb,...., and zzzz may be entered in one of the following forms:

1. 0-9,A-F hexadecimal digits (leading zeros need not be entered). If more than 4 digits are entered, then only the last 4 entered have meaning

2. :MN — The mnemonic MN is equivalent to 4 hex digits (see below for descripiton).

3. $ — Represent current address +1. This is valid for the M command and is used to calculate relative jump displacements.

1-25. An equal sign '=' may be entered at any time within the string to display the operand value so far as 4 hexadecimal digits.

1-26. Examples of typical operands are:

1. 4F7F — The operand value is equal to 4F7FH

2. :PC — The mnemonic PC is equivalent to address FFFEH and the operand value is equal to FFFEH.

3. 5038-5000 — The operand value will be 38H.

4. 5038-5000=0038 — The same as e) except '=' was entered to display the operand value.

5. 5038-$ — If current address = 5000H, then $=5001H and the operand value equals 37H.

6. 5038-$=0037 — The same as 5) except the equal sign was entered.

7. 305038 — More than 4 digits entered, therefore the last 4 have meaning . Operand value = 5038H.

8. 305038=5038 The same as 7) except the equal sign was entered.

1-27. Mnemonics are composed of 1 or 2 characters following a colon ":" and represent a 4 hex digit address. Table 1-1 lists the mnemonics recognized by DDT-80. Others may be added by the user as described by an example in paragraph 1-43.

1-28. Mnemonics are equivalent to a 4 hex digit address and the data at that address may represent either a single or double byte value (marked by * in the table). A single byte mnemonic causes the display of 2 hexadecimal digits to represent an 8-bit value. A double byte mnemonic causes the display of 4 hexadecimal digits (see examples in paragraph 1-33). If a command requires more than one operand, those operands have to be separated by either a blank or a comma.

Table 1-1. MNEMONICS RECOGNIZED BY DDT-80

Unrecognized mnemonics are resolved with a value of zero.

MNEMONIC	ADDRESS REPRESENTED BY THE MNEMONIC	DATA SAVED AT THAT ADDRESS
:PC*	FFFE	User's PC Register
:A	FFFD	User's A Register
:F	FFFC	User's F Register
:I	FFFB	User's I Register
:IF	FFFA	User's IFF Register
:B	FFF9	User's B Register
:C	FFF8	User's C Register
:D	FFF7	User's D Register
:E	FFF6	User's E Register
:H	FFF5	User's H Register
:L	FFF4	User's L Register
:A'	FFF3	User's A' Register
:F'	FFF2	User's F' Register
:B'	FFF1	User's B' Register
:C'	FFF0	User's C' Register
:D'	FFER	User's D' Register
:E'	FFEE	User's E' Register
:H'	FFED	User's H' Register
:L'	FFEC	User's L' Register
:IX*	FFEA	User's IX Register
:IY*	FFE8	User's IY Register
:SP*	FFE6	User's SP Register
:CI*	FF27	Address of the Console Input I/O Drive
:CO*	FF29	Address of the Console Output I/O Drive
:OI*	FF2B	Address of the Object Input I/O Drive
:OO*	FF2D	Address of the Object Output I/O Drive
:SI*	FF2F	Address of the Source Input I/O Drive
:SO*	FF31	Address of the Source Output I/O Drive
:TK*	E6B3	(Terminal Keyboard Driver Address)
:TT	E680	(Terminal Typehead Driver Address)
:ST	E67E	(Silent 700 Printer Driver Address)
"TR	E6A5	(Teletype Reader Drive Address)
:PR	E6C6	(Highspeed Papertape Reader Drive Address)
:PP	E6FA	(Highspeed papertape Punch Driver Address)
:AS	C000	(Calling Address for Assembler)
:ED	D48B	(Calling Address for the Editor)
:LP	E6F0	(Line Printer Driver)
:ER	D4D9	(Editor Reentry Address)
:TI	DF9B	(Silent 700 Tape Input Driver Address with ADC Option)
:TO	DF2F	(Silent 700 Tape Output Driver Address with ADC Option)

* = 2 byte mnemonics

1-29. COMMAND TERMINATORS

1-30. The command terminator immediately follows the operand(s) and signals DDT-80 that the command has been entered. Depending on the terminator, DDT-80 will do one of the following:

	Terminator	Action
1.	(CR)	Carriage return. DDT-80 will perform the command entered.
2.	^	Carat or up arrow. This terminator is valid only for the M and P commands. When updating a memory location (M) or a port (P), it signals DDT-80 to display the contents of the location or port just updated.
3.	.	Period. DDT-80 aborts the command, enter the command mode and be ready to accept another command.

1-31. DETAILED COMMAND DESCRIPTIONS

1-32. This section describes each DDT-80 command in detail. The command format is shown, followed by a description and examples. For the purposes of this section, the conventions used are:

1.	aaaa,....,zzzz	denote 4 hexadecimal digit operand value as described in paragraph 1-24.
2.	t	denotes the command terminator; carriage return, carat, or period.

3. - underline denotes the portion of the command entered by the user.

1-33. M COMMAND, DISPLAY AND UPDATE MEMORY. This command allows display and/or modification of specified memory locations or the CPU registers.

1-34. Format.

.M aaaat

1-35. Description. The user enters the command identifier M. DDT-80 collects the command and prints a space. The user then enters the operand aaaa followed by a terminator. DDT-80 responds by printing the memory address on the next line. This is followed by the contents of that particular address in hexadecimal. If the content is to be changed, the new value is entered. The new value entered is an operand as described in paragraph 1-22 except that the appropriate number of hexadecimal digits (2 or 4) is selected. For example, if the memory location 5001H was to be changed to FF:

```
.M 5001(CR)

5001 A3 FF(CR)      one memory location was changed

5002 A4 .           therefore the least significant 2 hex

.                   digits are used as the operand.
```

or if the PC register is to be changed to 7F50H:

```
.M :PC              The PC register is a 4 hex

:PC 433F 7F50(CR)   digit (16 bit) register, therefore
```

0000 20 . the least significant 4 hex digits

. are used as the operand.

1-36. When the user is examining and/or modifying a register or memory location, the accompanying terminator signals the action DDT-80 is to take. The possible operand (new value entered) and terminator combinations are:

Terminator	Meaning
1. (CR)	No operand entered, display next address
2. ^	No operand entered, display previous address
3. aa	Operand aa entered but "." aborts command with no change to value at address.
4. aa(CR)	Operand aa entered, change value at address to aa and step to next address.
5. aa^	Operand aa entered, change value at address to aa and display same address with the new value aa displayed.

1-37. A special feature of DDT-80 allows the user to conveniently compute relative addresses used in jump instructions. The value of the symbol "$" is defined as the value of the current location +1 and only has meaning during display and update commands. An example is given in paragraph 1-41.

1-38. Examples.

1-39. Accessing Memory Locations.

.M 16A(CR)	examine location 016AH
016A 3F(CR)	it contains 3FH, do not change, step to next location
016B 92^	next location contains 92H, do not change, go back to previous location.
016A 3F 34FF^	change contents of 016AH to FFH and display same location. Note that only the last 2 digits typed are stored in 016AH (the entry 34 was in error).
016A FF(CR)	new contents displayed step to next.
016B 92 .	abort
.	DDT-80 waiting for next command.

1-40. Examining User's Registers. The user may examine and change (if desired) his internal Z80 registers. They may be initialized, for example, prior to program execution, or after a breakpoint has been encountered in the program to be debugged. The contents of the user's registers may be accessed through the use of the mnemonics discussed in paragraph 1-28.

.M :A(CR)	Examine user's accumulator
:A 18 25(CR)	Change register A to 25H, examine next location

:PC 0400.	User's PC register, return to command mode
.M :PC(CR)	Examine user's PC register
:PC 5005 5000 ^	Change PC to 5000H, ' ' causes same address to be displayed with new contents
:PC 5000 .	Return to command mode
.	DDT-80 waiting for next command

When returning to execute the user's program, the new values will be inserted into the user's Z80 internal registers.

1-41. Computing Relative Jumps. This example shows the entering of a relative jump instruction at location 5000H to branch to location 5038H.

.M 4000+1000(CR)	Examine location 5000H (shows operand addition)
5000 20 18(CR)	Insert first byte of Jump (JR 5038-$) instruction
5001 F8 5038- $=0036^	Compute and display relative displacement for branch from 5001H to 5038H.
5001 36 .	Jump displacement of 36_H shown
.	DDT-80 waiting for command

It shoud be noted that the maximum legal displacement value for forward branches is 7FH and for backward branches is 80H. Hence the computed value XH for a branch should adhere to the following

Forward: 0000≤X≥007F

Backward: FF80≤X≥FFFE

For example,

```
5101 00 5000-$=FEFE
```

indicates that one cannot do the relative jump from location 5101H to 5000H since FEFEH is less than FF80H.

1-42. Examing Channel Assignments.

```
.M    :CI(CR)        Examine console input
:CI   :TK(CR)        Console input = terminal input
:CO   :TT(CR)        Console output = terminal output
                     (printer)
:OI   6363 :PR(CR)   Set object input to high speed paper
                     tape reader
:OO   0063(CR)       Object output not set to a device
                     driver
:SI   6363(CR)       Source input not set to a device driver

:SO   6300(CR)       Source output not set to a device
                     driver
FF33  80 .           End of mnemonic table, 80H terminates
                     table.
.                    DDT-80 waiting for next command
```

See paragraph 1-84 for a discussion of the I/O channels.

1-43. Adding Mnemonics. Add single byte mnemonic ":MN" to point to the first address of the DDT-80 register map.

```
.M FF33 (CR)        Examine user's mnemonic table

FF33 80 4D(CR)      Put "M" into table, 4DH=ASCII M

FF34 FF 4E(CR)      Put "N" into table, 4EH=ASCII N

FF35 FF E6(CR)      Lower address of mnemonic

FF36 FF FF(CR)      Upper address of mnemonic

FF37 FF 80(CR)      Close mnemonic table

FF38 FF .           Return to command mode

.M :MN(CR)          Now use new mnemonic to examine memory

.M FFE6(CR)         Mnemonic is equivalent to this

:MN 00 .

.                   DDT-80 waiting for command
```

In this example location FF33 did contain the 80H terminator, but it was written with a new mnemonic to extend the table. Hence, the 80H terminator had to be moved to the end, location FF37H in this case.

Caution: When adding mnemonics do not use a mnemonic unless the mnemonic table is correcly closed with an 80H terminator. Otherwise DDT-80 will not be able to locate the end of the table.

1-44. The next example shows the addition of a double-byte

mnemonic :HL to examine the user's HL register as a 16-bit value.

```
.M FF33(CR)          Examine mnemonic table

FF33 80 48(CR)       Insert ASCII H

FF34 FF 4C+80(CR)    Insert ASCII L+ 80H to make mnemonic
                     double byte

FF35 FF F6(CR)       Lower byte of address

FF36 FF FF(CR)       Upper byte of address

FF37 FF 80(CR)       Close table with terminator

FF38 FF .            Return to command mode

.M :HL(CR)           Now use new mnemonic

:HL 500E .           New mnemonic examines HL register

.                    DDT-80 waiting for next command
```

NOTE: User define mnemonics have precedence over system mnemonics (the user mnemonic table is searched first). So in the example above, :MN had precedence over :SP defining the same address.

1-45. M COMMAND, TABULATE MEMORY. This command allows the user to display, but not change, a block of memory. Up to 16 values are printed per line.

1-46. Format.

```
.M aaaa,bbbbt        tabulate memory locations aaaa through
                     bbbb
```

1-47. Description. The user enters the command identifier M followed by the starting (aaaa) and ending (bbbb) addresses of the memory block separated by a comma or a blank. Upon terminating with a carriage return DDT-80 prints a line feed, and then prints the contents of aaaaH to bbbbH inclusive with up to 16 values per line. DDT-80 then returns to the command mode. The tabulation may be stopped at any time by pressing the RESET button.

1-48. Example.

```
.M 4100,4127(CR)      display memory locations 4100H through
                      4127H inclusive

4100 1B 80 12 10 00 B7 A5 21 10 94 04 20 CA B7 44 18
4110 81 11 23 21 07 94 17 45 12 55 A5 18 21 80 C5 55
4120 90 0C A5 81 09 21 40 22
                      DDT-80 waiting for next command
```

1-49. P COMMAND, DISPLAY AND/OR MODIFY PORTS. This command allows the user to display and/or modify any of the possible 256 I/O ports. The reader should note that some ports are output only and cannot be read.

1-50. Format.

```
.P aat(CR)            Display port aa
```

1-51. Description. The user enters the command identifier P followed by the port address aa and a terminator. DDT-80 responds by printing the port address and the value at that port. If the value at that port is to be changed, the user enters the new value. The new value entered is a 2 hexadecimal digit operand. When the user is examining and/or modifying a port, the terminator signals the action DDT-80 is to take. The possible

operand (new value entered) and terminator combinations are:

	Terminator	Meaning
1.	(CR)	No operand entered, display next port
2.	^	No operand entered, display previous port
3.	aa.	Operand aa entered, but '.' aborts command with no change to the port.
4.	aa(CR)	Operand aa entered, change the port value to aa and step to display the value at the next port.
5.	aa^	Operand aa entered, change the port value to aa and display the same port with new value aa.

1-52. Example.

.P D1(CR)	Program PIO Port 1A (D0H) for BIT MODE
D1 FF CF^	CFH sets 1A Control (D1H) to BIT MODE. Port D1 is putput only.
D1 FF 0^	Program all bits of Port D0H as output bits.
D1 FF ^	Backup to D0H
D0 00 AA^	Output value AAH to Port D0
D0 AA .	
.	DDT-80 waiting for a command

The above is also an example of programming a PIO port.

1-53. D COMMAND, DUMP MEMORY. The dump command dumps the specified block of memory to the object output channel in an absolute format compatible with the object output produced by the Assembler. If the memory is dumped onto paper tape, it may later be read back into memory using the load command discussed in the next section.

1-54. Format.

.D aaaa,bbbt(CR) Dump memory from address aaaa, to address bbbb inclusive.

1-55. Description. The user enters the identifier D then the starting aaaa and ending addresses bbbb inclusively separated by a comma or a blank. Immediately after terminating (a period aborts the command) with a carriage return, the paper-tape punch should be turned on by the user. DDT-80 responds by outputting 8 inches of leader (blank tape), followed by the designated memory block in Z80 absolute hex loader compatible for not (discussed in paragraph 1-60), followed by 8 inches of trailer (blank tape). DDT-80 then returns to the command mode. The user may then turn off the tape punch. The dump process may be terminated at any time by pressing the RESET button.

1-56. Example.

.D 200,220(CR) Dump locations 200H through 220H

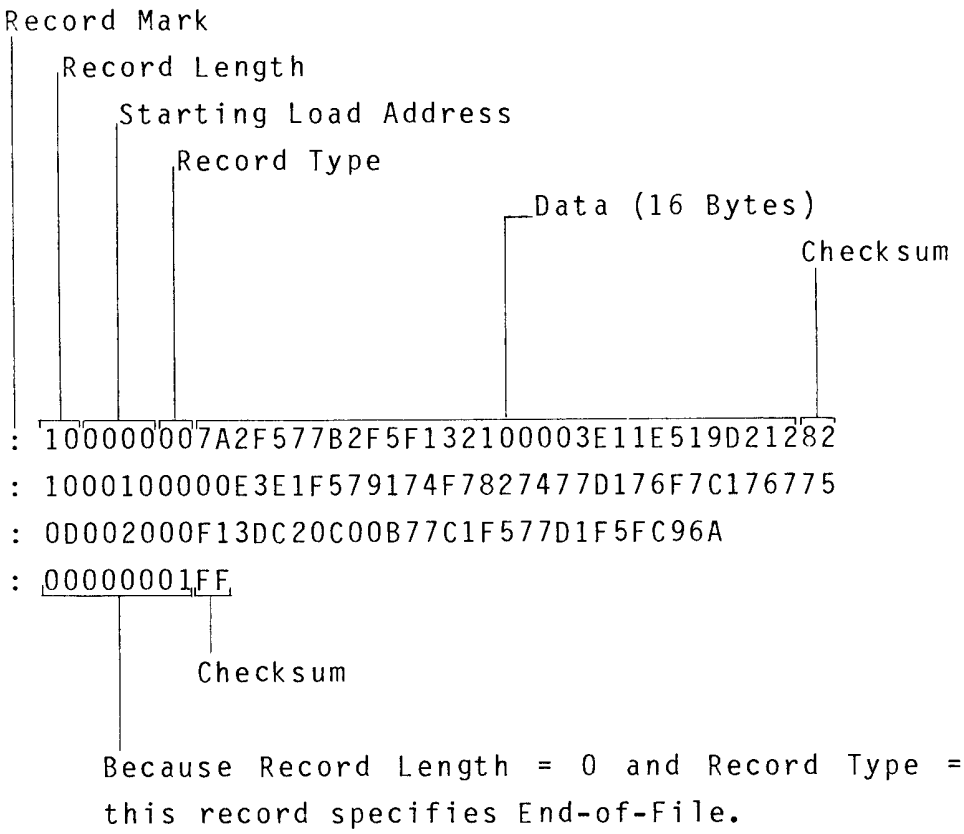
1-57. L COMMAND, LOAD MEMORY. The load command provides the capability to lod an absolute program and/or data into memory via the object input channel. The format of the data must be compatible with that produced as object output by the Assembler for non-relocatable, non-linkable assemblies. The L command will load tapes dumped by the D command described in the previous section.

1-58. Format.

.L t(CR) Load tape into memory

1-59. Description. The user inserts the tape into the reader positioned such that the leader is over the read mechanism. The user then enters the command letter "L" followed by a carriage return and places the tape reader switch to START. DDT-80 will then load data from the tape into RAM starting at the load address specified in each record on the load tape. Teletypes equipped with the reader step option should stop at the end of the load. DDT-80 then returns to the command mode. Those teletypes not so equipped will continue to read tape data as commands. If a checksum error is encountered, the start address of the line following the suspect line is printed on the console output channel.

1-60. The format for absolute hex load tapes is as follows:



1-61. CHECKSUM FIELD: Frames 9+2* (record length) to 9 + 2* (record length) +1

The checksum field contains the ASCII hexadecimal representation of the twos complement of the 8-bit sum of the 8-bit bytes that result from converting each pair of ASCII hexadecimal digits to one byte of binary, from the record length field to and including the last byte of the data field. Therefore, the sum of all the ASCII pairs in a record after converting to binary, from the record length field to and including the checksum field, is zero.

1-62. E COMMAND, EXECUTE A USER'S PROGRAM. The execute command is used to execute all programs including design aid programs

such as the Assembler and Text Editor.

1-63. Format.

.E aaaat(CR)	Transfer control to the program starting at address aaaa.
.E t(CR)	Transfer control to the address specified by PC in the register map.

1-64. Description. To cause execution of a program the user types the identifer E followed by the desired entry address of his program. Upon typing carriage return DDT-80 will load the user's internal registers from the save register map then transfer control to the program entry point. (It is therefore possible to enter a program with preset values in the registers if desired.) Since the register map is used for saving internal registers when a breakpoint (see paragraph 1-74) is encountered, the contents of the register map reflect the effect of the last instruction before the breakpoint was encountered. If no entry address is specified after the E command, DDT-80 will transfer control to the address specified by PC in the user's register map.

1-65. Example.

.E 1200(CR)	Execute the program starting at location 1200H.

To return control to DDT-80, the user's program must either encounter a breakpoint (see next section), or the RESET button should be pressed.

.	(User pressed RESET, back to DDT-80 command mode)

.M :PC(CR)	Examine user's program counter (PC)
:PC 62FF(CR)	Set user's PC to 1200H
:E(CR)	Execute program starting at location 120H

The execute command may also be used together with the breakpoint command to execute portions of programs while debugging.

1-66. H COMMAND, HEXADECIMAL ARTHMETIC. The arithmetic capability of DDT-80 allows hexadecimal addition and subtraction.

1-67. Format.

.H aaaa+bbbb+...+yyyy=zzzzt(CR)

1-68. Description. The user enters the command identifier and then enters the arithmetic expression. Only + and - are legal operations. If the sign of the first operand is omitted, it is assumed +. The equal sign causes the 4 digit (least significant 4 digits) result to be displayed. When the terminator is entered DDT-80 returns to accept another command.

1-69. Examples.

.H 5000-4FFF=0001(CR)	Subtract 4FFFH to 5000H
.H 5000+4FFF=9FFF(CR)	Add 4FFFH to 5000H
	DDT-80 waiting for a command
.	The equal sign caused the 4 digit result to be printed.

1-70. C COMMAND, COPY MEMORY BLOCKS. The copy command permits any block of memory to be moved to any area of memory. The move may be forward or backward and the new block may or may not overlap with theoriginal memory block.

1-71. Format.

.C aaaa,bbbb,cccct(CR)	Copy memory locations aaaa through bbbb inclusive to the memory block starting at address cccc.

1-72. Description. The user enters the command identifier C followed by the starting aaaa and ending address bbbb of the block to be moved, followed by the starting address cccc of the block receiving the data. The operands are separated by commas or blanks. Upon terminating with a carriage return, DDT-80 prints a line feed, performs the requested copy operation, and then prints a period "." to indicate that it is ready to accept another command. The text copied is not displayed.

1-73. Example.

.C 100,200,1200(CR)	Copy memory locations 100H through 200H inclusive to locations 1200H through 1300H
.C 100,0200,150(CR)	Copy memory locations, 100H through 200H inclusive to locations 150H through 250H. (overlapping copy)
.	DDT-80 waiting for command

Entire programs or subroutines may be moved aroung in this way

and still execute properly in their new locations. Care should be taken to copy complete instructions on both ends of the block when copying programs, and any relative branch instruction contained with a block to be moved should not branch outside the block.

1-74. B COMMAND, BREAKPOINT COMMAND. The breakpoint command causes the setting of a "trap" or breakpoint within the user's program. Upon encountering the breakpoint, the user's program will transfer control back to DDT-80 where the register, I/O ports, and memory contents may be inspected. Breakpoints may be set only in RAM, not ROM.

1-75. Formats.

.B aaaa,b(CR)	Set breakpoint at address aaaa. Where b indicates short (b=0) or long (b ±0) format for printing registers. Short implies print only the PC, A, and F. Long implies print all internal register. (see R command for the order the registers are output).
.B aaaa(CR)	Set breakpoint at address aaaa. b was omitted therefore use the short format for printing the registers.
.B (CR)	Clear any previous breakpoint.

1-76. Description. The user types the command identifier B followed by the address where it is desired to place a break point. Upon entering a carriage return DDT-80 proceeds to:

1. Remove any pre-existing breakpoint by restoring user's code.

2. Extract and save 3 bytes of the user's code starting at the breakpoint address.
3. Insert a 3 byte sequence into the user's program at the breakpoint address. (This sequence consists of a 3 byte JP instruction to return to the breakpoint entry of DDT-80.)
4. If b=0 a flag is set to indicate long format (print all internal CPU registers.)

1-77. DDT-80 then types a line feed and a period "." to return to the command mode. The user may now initiate execution of his program by usiny the execute command. When the address specified by the breakpoint command is encountered, control is transferred to DDT-80 where the following action are taken.

1. The three bytes of user code replaced by the trap instruction are restored.
2. All registers are recorded in RAM storage within DDT-80.
3. DDT-80 types: The breakpoint address (Program Counter), and value of the A and F registers for sort format output or all internal CPU registers for long format output.
4. DDT-80 prints a line feed and period and return to the command mode

1-78. A breakpoint can be cleared by executing its address or entering the command B(CR). If the user "misses" a breakpoint while executing, the 3 bytes of breakpoint code must replaced manually with the correct user's code after RESET. The set breakpoint command and execute command are closely related and are normally used together during the debug process for executing sections of a program and then evaluating the registers for correct data.

1-79. There are certain characteristics of the DDT-80 breakpoint facility which user should be aware of during debugging:

1. The trap sequence used by DDT-80 is as follows:
   JP DDT-80 Jump to DDT-80 Breakpoint Processor

2. Since DDT-80 replaces three bytes of the user program, a breakpoint should be set such that when the user program is executed, control can only be transferred to the first byte of the trap sequence. In addition, the breakpoint address must reference the first byte of an instruction. For example in the following sequence:

```
L1 JR NZ, L3-$
L2 LD A,0
L3 LD B, 0FH
```

A breakpoint should not be set at L2 because if the branch condition at L1 is met control would be transferred to the third byte of the trap sequence.

3. No error indication is given if one attempts to set a breakpoint in ROM.

4. After a breakpoint has been set, it can be changed simply by entering a new breakpoint. The act of entering a new breakpoint automatically clears the previous breakpoint.

5. When a breakpoint is encountered in a user program, DDT-80 saves the state of interrupts (through IFF) in the :IF register. The state of interrupts is restored or set according to the content of :IIF when control

is transferred to a user program.

1-80. R COMMAND, DISPLAY CPU REGISTERS

The display CPU registers command allows the user to dump the contents of all user registers to the console.

1-81. Formats.

.R t(CR) — Print the contents of the CPU registers

.R 1t(CR) — Print a heading to label the CPU registers on one line, on the next line prints the contents of the CPU registers.

1-82. Description. The user enters the command identifier R. If the user wants a heading to be printed that labels the register contents, the operand of 1 needs to be entered. If no heading is desired, then no operand is entered.

1-83. Examples.

```
.R (CR)
 A000 0100 0104 CFB3 C09A FFEE EDF6 9C3E C3DC FE9B D6ED F1BE FFB4

.R 1(CR)
  PC   AF  IIF   BC   DE   H   A'F' B'C' D'E' H'L'  IX    IY SP
A000 0100 0104 CFB3 C09A FFEE  EDF6 9C3E C3DC FE9B D6EC F1BE FFB4
```

PC contains A000H
A contains 01H
F contains 81H
I contains 01H
IF contains 04 (Bit 3 = 1 implies IFF = 1)
.
.

bit

F =

7							0
1	0	0	0	0	0	0	1
S	2	X	H	X	P/V	N	C

S = sign flag

```
                          Z = zero flag
IY contains F1BEH         X = indeterminate flag
I contains F1H            H = half carry (for BCD operations)
R contains BEH            P/V = parity or overflow flag
                          N = BCD add/subtract flag
SP contains FFB4H         C = carry flag
```

Register IF represents the interrupt flip-flop maintained by the Z-80. If IF=0, then interrupts were disabled when DDT-80 received control (encountered a breakpoint). If IF=4, then interrupts were enabled. Upon blank line reset IF is set to zero.

1-84. INPUT/OUTPUT

1-85. The SDB-80 has 3 I/O channels to which devices may be assigned. The console channel is used to receive and respond to commands and generally handle all editing, controlling, and monitoring information. The object channel is used by the Load (L) and Dump (D) commands to read and write machine-oriented data such as object tapes. The source channel is not used by DDT-80, but is used by The Resident Assembler and Text Editor to read and write user-oriented data such as source programs. The channel assignment for these devices is stored in the DDT-80 RAM area (see paragraph 1-13) and may therefore be updated using the M command with mnemonics.

1-86. DDT-80 contains a collection of I/O drivers which are used by programs internal to DDT-80 and may also be called from user programs using the procedures described below. The available drivers and corresponding mnemonics for their start addresses are:

```
:TK              terminal keyboard (uses UART)
```

:TR	teletype reader (uses UART: same as TK except a reader step pulse is sent first)
:TT	terminal typehead, or printer (uses UART: also operates teletype punch when switched on)
:PR	paper tape reader (uses PIO)
:PP	paper tape punch (uses PIO)
:LP	line printer (uses PIO)

1-87. Upon power-up or momentarily depressing RESET, DDT-80 initializes the logical/physical channel assignments as follows:

```
:CI :TK -     Console input = terminal keyboard
:CO :TT -     Console output = terminal typehead
.             DDT-80 waiting for command
```

1-88. Note that if the user changes the console channel assignments, then hits RESET, DDT-80 will change the assignments back to the above configuration. Also it is necessary for the user to initialize the object and source channels to the appropriate drivers at least once after power up. Since DDT-80 does not intialize the object and source channels upon reset, there is no need to reconfigure the object and source channel.

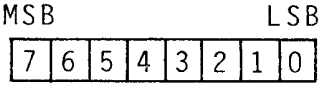
1-89. ADDING NEW I/O DRIVERS

1-90. The user may write I/O drivers (and define mnemonics for them) for additional devices and maintain compatibility with

DDT-80 drivers by observing the following:

1. Register E is the control register. Register D is the data register.

2. Register E - control register



   Bit 7 = 1 implies 'immediate return' I/O mode. Immediate return means; if data or device is ready, perform the I/O WHCN the function is done and data collected, clear bit 7. Otherwise leave bit 7 set.

   Bit 7 = 0 implies 'wait until done'. If bit 7 = 0, wait until the I/O function is complete before returning from the I/O drivers.

   Bit 3 = 1 implies initialize the device (usually this is done first and only once). Bit 3 is cleared upon exit from the drivers.

3. Register D contains data for outputting. Registers D and A contain input data.

4. The A register is destroyed on output

5. I/O drivers supplied:

   :TK - Terminal keyboard (uses the UART). Immediate return is done if flagged in register E.

   :TT - Terminal type head (uses the UART). Immediate return is done if flagged in register E.

:ST - Silent 700 printer (uses the UART). Immediate return is done if flagged in register E.

:TR - Tape Reader (uses the UART). Outputs the reader step signal. Immediate return is done if flagged in register E.

:PR - High Speed Paper Tape Reader (Uses PIO, Ports D0 - data, D1 - control). Immediate return is done if flagged in register E.

:PP - High Speed Paper Tape Punch (uses PIO, ports D2 - data, D3 - control). Immediate return is done if flagged in register E.

:LP - Line Printer (uses PIO. Ports D6 - data, D7 - control). Immediate return is done if flagged in register E.

1-91. SUBROUTINES CALLABLE IN DDT-80

1-92. The following is a list of callable subroutines in DDT-80. It should be noted that some of these routines could impact a user program in ways not discussed below. Studying the listing of DDT-80 would be appropriate for complete understanding of these subroutines.

1-93. RDCHR - READ AN ASCII CHARACTER

1-94. Calling Address. E522H

1-95. Parameters Upon Entry. The entry parameters are that the E register designates I/O channel as follows:

E = 0,1 Console channel

E = 2,3	Object channel
E = 4,5	Source Channel

If bit 7 (most significant bit) is set, then immediate return is requested if data is not ready. Some (but not all) I/O device drivers in DDT-80 allow immediate return.

1-96. Parameters Upon Exit. The exit parameters are as follows:

1. E register is unchanged except for the immediate return bit (bit 7) which will be cleared if data has been read.

2. Registers D and A contain the ASCII character. The parity bit (bit 7) is masked off.

1-97. Routines Called I/O Driver tied to the channel specified in register E.

1-98. WRCHR - WRITE AN ASCII CHARACTER

1-99. Calling Address. E527H

1-100. Parameters upon entry. The entry parameters are as follows:

1. E register designates I/O channel.

E = 0,1	Console Channel
E = 2,3	Object Channel
E = 4,5	Source Channel

If bit 7 (most significant bit) is set, then immdiate re-

turn is requested if the I/O device is not ready for data. Some (but not all) I/O drivers in DDT-80 allow immediate return.

2. D register must contain the data to be written.

1-101. Parameters upon Exit. The next parameters are as follows:

1. E register is unchanged except for the immediate return bit (bit 7). It will be cleared if data has been output.

2. D register contains the data output.

3. A register is destroyed.

1-102. Routines Called. I/O driver tied to the I/O channel specified in register E.

1-103. PACC - PRINT THE CONTENTS OF THE A REGISTER

1-104. Calling Address. E58BH

1-105. Parameters Upon Entry. The entry parameters are as follows:

1. E register designates I/O channel as for WRCHR. Immediate return is not valid when calling PACC.

2. A register contains the binary equivalent of the 2 hex digits to be printed in ASCII.

1-106. Parameters Upon Exit. The exit parameters are that A register contents are destroyed.

1-107. Routines Called. Routines called are as follows:

1. PRVAL
2. WRCHR

1-108. PRVAL - CONVERT THE LEAST SIGNIFICANT 4 BITS (1 hex digit) A REGISTER TO ASCII.

1-109. Calling Address. E5AFH

1-110. Parameters upon Entry. The entry parameters are that the A register least significant 4 bits must be equal to the hex digit being converted to its ASCII representation.

1-111. Parameters Upon Exit. The exit parameters are that the D and A register contain the ASCII representation for the hex digit converted.

1-112. ECHO- READ AND WRITE A CHARACTER THROUGH THE SAME I/O CHANNEL

1-113. Calling Address. E597H

1-114. Parameters Upon Entry. The entry parameters are that the E register designates the I/O channel as for RDCHR and WRCHR. Immediate return is not valid when calliing ECHO.

1-115. Parameters Upon Exit. The exit parameters are as follows:

1. D register contains the character read and printed

2. A register is destroyed

1-116. Routines Call. Routines called are as follows:

1. RDCHR

2. WRCHR

1-117. CRLF - OUTPUT A CARRIAGE RETURN AND LINE FEED

1-118. Calling Address. E59CH

1-119. Parameters Upon Entry. The entry parameters are that the E register designates the I/O channel as for WRCHR. Immediate return is not valid when calling CRLF.

1-120. Parameters Upon Exit. The exit parameters are as follows:

1. D register contains the ASCII representation for a line feed (0AH)

2. A register is destroyed

1-121. Routines Called. The routine called is WRCHR.

1-122. SPACE - OUTPUT A BLANK

1-123. Calling Address. E5A5H

1-124. Parameters Upon Entry. The entry parameters are that the E register designates the I/O channel as for WRCHR.

1-125. Parameters Upon Exit. The exit parameters are as fol-

lows:

1. D register contains an ASCII blank (20H).

2. A register is destroyed

1-126. Routines called. The routine called is WRCHR.

1-127. PTXT - PRINT A TEXT STRING

1-128. Calling Address. E3C7H

1-129. Parameters Upon Entry. The entry parameters are as follows:

1. E register designates the I/O channel as for WRCHR. Immediate return is not valid when calling PTXT.

2. HL register contains the beginning address where the text string is stored in memory.
3. The text string must terminate with an ASCII ETX character (03H). The ETX is not output.

1-130. Parameters Upon Exit. The exit parameters are as follows:

1. D register contains 03H (ETX character).

2. HL register contains the address where the ETX is stored in memory.

3. A register is destroyed.

1-131. Routines Called. The routine called is WRCHR.

1-132. ASBIN - CONVERT ASCII REPRESENTATION OF HEX DIGIT TO BINARY

1-133. Calling address. E583H.

1-134. Parameters Upon Exit. The exit parameters are as follows:

1. A register contains the corresponding binary value of the hex digit.

2. No error check takes place.

1-135. RENTRY - ENTRY ADDRESS TO DDT-80

1-136. Jump address. The jump address is E11DH. This address should be jumped to and not called. DDT-80 will print a carriage return and line feed and then a period. The user register map is not saved when jumping to RENTRY. DDT-80 is then ready to accept another command.

1-137. CALLABLE I/O DRIVERS

1-138. TK - read a character from terminal key board. (Uses the UART).

1. Calling address = E6B3H

2. Parameters upon entry:

   Bit 7 of register E is the immediate return bit. Bit 7 = 1 implies immediate return if character not ready.

3. Parameters Upon Exit:

   a. Bit 7 of register E is cleared if a character is read.

   b. Registers D and A contain the ASCII character. The parity bit is masked off.

1-139. TT - type a character to the terminal typehead or printer (uses the UART).

1. Calling Address = E680H

2. Parameters upon entry:

   a. Bit 7 of register E is the immediate return bit. Bit 7=1 implies immediate return if device is not ready for the character.

   b. Register D contains the ASCII character to output.

3. Parameters upon exit:

   a. Bit 7 of register E is cleared if data was output.

   b. Bit 4 of register E is used internally and is always cleared upon exit.

   c. Register D contains the character output.

   d. Register A is destroyed.

1-140. ST - the same at TT except a delay is inserted when a carriage return, line feed is output (uses the UART).

Calling address = E67EH

1-141. TR - the same as TK except that the reader step signal is output to get the next character on tape (uses the UART).

Calling address = E6A5H

1-142. PR - read a character from a high speed paper tape reader device (uses PIO. Ports D0-data, D1-control). Interrupts must be in mode 2 (IM=2).

1. Calling address = E6C6H

2. Parameters upon entry

   a. Bit 3 of register E indicates if the device is to be initialized.
      If bit 3=1 then initialize the device.

   b. Immediate return is done if flagged by bit 6 of register E.

3. Parameters upon exit:

   a. Bit 3 of register E is set to 0.

   b. Register D and A contains the ASCII data read.

   c. The parity bit is masked off.

   d. Interrupts are used and enabled upon exit.

1-143. PP- output a character to a high speed paper tape punch device (uses PIO, ports D2-data, D3-control). Interrupts must be in mode 2 (IM=2).

1. Calling address = E6FAH

2. Parameters upon entry:

   a. Bit 3 of register E indicates if the device is to be initialized. If bit 3=1 then initialize the device.

   b. Register D contains the ASCII character to output.

   c. Immediate return is done if flagged by bit 7 of register E.

3. Parameters upon exit:

   a. Bit 3 of register E is cleared.

   b. Register D contains the data output.

   c. Interrupts are used and enabled upon exit.

1-144. LP - output a character to a line printer (uses PIO, ports D6-data, D7-control). Interrupts must be in mode 2 (IM=2).

1. Calling address = E6F0H

2. Parameters upon entry:

   a. Bit 3 of register E indicates if the device is to be initialized. If bit 3=1 then initialize the device.

   b. Register D contains th ASCII character to output.

   c. Immediate return is done if flagged by bit 7 of register E.

3. Parameters upon exit:

   a. Bit 3 of register E is cleared.

   b. Register D contains the data output.

   c. Interrupts are used and enabled upon exit.

1-145. PROGRAMMING NOTES

1-146. The following is a list of items in DDT-80 that could affect a program the user is writing and debugging.

1. The user stack pointer is set by DDT-80 on power-up and reset (SP=FF90H).

2. DDT-80 uses 6 locations on the user stack for temporary storage when transferring control to a user program (E command). The user's stack is left unaffected and the stack pointer points to the correct value. The user needs to be aware that 6 locations past the stack pointer are used.

3. If the user writes a program that calls the I/O drivers PR, PP and LP, then the interrupt mode has to be equal to 2 (IM=2).

4. When a breakpoint (B command) has been entered and not encountered while running the program, the user must press reset to regain control. The breakpoint must be cleared manually by inserting the correct code for the 3 byte jump to DDT-80, or by setting location $FF0C_H$ to 1 and doing a B(CR) to clear breakpoints.

5. The user may write an extension to the operating system (DDT-80) if wanted. After a command and operands have been scanned and saved, DDT-80 does a jump to the executive routine through the address contained in locations $FF1F_H$ and $FF20_H$. The command letter is saved in location $FF1C_H$. The operands are saved in locations $FF14_H$ - FF15 (OPR1), $FF16_H$ -FF17 (OPR2), and $FF18_H$ - $FF19_H$ (OPR3). The extension must determine if the command is to be handled by the extension. If the command is to be handled by the extension, then the extension must take the appropriate action. When finished the extension must transfer control back to DDT-80 by a jump to location $E127_H$. If the extension does not handle the command, then the extension must transfer control back to DDT-80 by a jump to $E147_H$.

To install an extension the user must load into RAM the extension object code (L command). Then modify the contents of memory locations (M command) $FF1F_H$ (LSB) and $FF20_H$ (MSB) to the beginning address of the extension. If reset is ever pressed, then locations $FF1F_H$ - $FF20_H$ will be reset to jump to DDT-80 instead of the extension.

SECTION 2

ASMB-80 TEXT EDITOR

2-1. INTRODUCTION

2-2. The ASMB-80 Text Editor is a design aid written to assist the user in origination and modification of assembly language source programs. The Editor resides in ROM and permits random access editing on ASCII character strings which are typically read into memory from digital cassette or paper tape. The Editor allows line or character editing with the following set of edit commands:

1. An --- Advance record pointer n records.
2. Bn --- Backup record pointer n records.
3. CndS1dS2d - change string S1 to string S2 for n occurrences.
4. Dn --- delete next n records.
5. E --- exchange current record with records to be inserted.
6. I --- insert records.
7. Ln --- Go to line number n.
8. Mn --- Enter commands into one of two alternate command buffers (pseudo-macro).
9. N --- Print top, bottom, and current line number.
10. Pn --- Punch n records from buffer.
11. R --- Read source records into buffer.
12. Sn dS1d - Search for nth occurrence of string S1.
13. T --- Insert records at the top of the buffer.
14. Vn --- Output n records to console output channel.
15. Wn --- Output n records to source output channel.
16. Xn --- Execute alternate command buffer n. (Pseudo Macro)

2-3. TERMINOLOGY

2-4. The user should understand some of the following terms so the editor can be used properly.

1. Source - ASCII characters comprising the Z80 program instruction statements.

2. Record - a single source statement with carriage return as delimiter.

3. Buffer - computer memory area used to store the source.

4. Pointer - the position in the buffer (always the beginning of a record) where the next action of the editor will be initiated.

5. Current statement or record - the source statement in the buffer pointed to by the pointer.

6. Insert - installation of a record into the buffer immediately following the record pointed to by the pointer.

7. Delete - removal of the record pointed to by the pointer from the buffer.

2-5. TEXT EDITOR COMMANDS

In the following Text Editor command descriptions, the first alphabetic character designates the command, the n represents a decimal value from 0 to 9999. If n is omitted, it is assumed to be zero. The editor prints a greater than sign (>) when ready to accept a command string. When entering data from the console

channel (Key board usually), the user may delete previous characters by typing BS (backspace) or RUBOUT (DEL on some keyboards). When using RUBOUT, a backslash ( ) will printed for each character deleted. The user may also delete an entire line being entered by typing control-U. A CR/LF will be echoed. A control-shift-K (ESC) will return the user to DDT-80.

1. R - READ source statement records from the source channel. The read continues until the buffer is full, the end of the tape is read and time out occurs, or an ETX ASCII charaacter is encountered. The EXT is preserved in the buffer. Each initiation of the R command stores the new source records after source records already in the buffer. The R command leaves 83 or less bytes for editing if the buffer is filled by reading.

2. Bn - BACKUP the record pointer n number of records from the current statement. When n is zero, the pointer is positioned at the beginning of the first record in the buffer.

3. An - ADVANCE the record pointer n number of records from the current statement. When n is zero, the pointer is positioned at the beginning of the record in the buffer.

4. Pn - PUNCH n number of source records starting with the current statement or record. When n is zero the editor will punch the buffer from the current record to the end of the buffer. The user activates the punch by turning on the object output device (if needed) and typing control @. (ASCII null.

This is equivalent to control-shift-P on some terminals). At the end of the punch sequence, the user must turn off the object output device (if needed) and type control-@ (ASCII null). The editor will not initiate any other action unil this is done.

5. Sn -'SOURCE IMAGE CHARACTERS' (delimiter=') SEARCH buffer memory, starting at the current record, until the nth occurrence of the source image characters supplied between the delimiters is found. The n position the pointer at the beginning of this record. Any character that does not exist in the source image may be used as the delimiter. Both starting and terminating delimiters must be identical. A blank or comma must separate the command Sn and the first delimiter. When n is zero the first occurrence of the source image is sought. If the nth occurrence of the source image is not found by the end of the buffer, the pointer is positioned at the last record of the source and the editor awaits another command.

6. Cn -'STRING1'STRING2' (delimiter=') CHANGE the next n occurrences of STRING1 to STRING2 starting with the current record. STRING1 does not have to be in the current record. The Editor will search for STRING1 and start counting from there. The pointer will be positioned at the record where STRING1 occurred last. Any character not in STRING1 or STRING2 may be the delimiter and must be equal for all three delimiters. A blank or comma must separate Cn and the first delimiter. When n is zero the first occurrence of STRING1 is changed. If the nth occurrence of STRING1

is not found, the pointer is positioned at the last record of the source.

7. I - INSERT records after the current record. A carriage return (line feeds are supplied by the editor) must terminate each record to be inserted. A null line (no characters) must terminate an entire set of insertions. The prompt character '<' indicates the Editor is ready for a line to insert. The pointer is positioned at the record immediately following the set of inserted records. A warning message is printed if the buffer is full and the last record could be truncated. Also some of the Editor's work area could be corrupted if the buffer is overrun. Care should be taken when the 'BUFFER FULL' message is printed. The control-U character causes a null line to be returned to the I command and insertion is then terminated.

8. T - Insert records at the top of the buffer. Records are inserted before the first record in the buffer. Note the conditions under the I command for proper operation.

9. Dn - DELETE n number of records beginning with the current source statement. When n is zero, one record is deleted.

CAUTION: Do not perform a delete past the end of the Editor buffer because certain Editor variables could be altered incorrectly.

10. Wn - or Vn - Print n records starting with the current record. The pointer is unaffected. When n is zero, one record is printed. The W command directs output to the source I/O channel. The V command directs out-

put to the console I/O channel.

11. Ln - Position the pointer at the beginning of the source statement with line number n. N equal zero is not a valid line number. This command always advances in the source to find a requested line number. Thus, if the current record is beyond the requested record, use the backup (B) command before the L command.

12. N - The N command outputs to the console the line number of the first record in the buffer, the line number of the last record read by the R command, and the line number of the current record.

13. Mn - The M command tells the Editor to accept a command string and store that string in alternate command buffer 1 or 2 depending on the value of n (1 or 2). The alternate command buffers will accept character strings of 40 characters or less. No error is given if more than 40 characters are input and Editor variables could be corrupted if overrun occurs. The M command is the only way to enter commands into the alternate command buffers. If n equal zero, alternate command buffer 1 is selected.

14. Xn - The X command tells the editor to execute alternate command buffer 1 or 2 depending on the value of n (1 or 2). The alternate command buffers are executed as if included in the main command buffer. Once an alternate buffer has been executed, control is tranferred back to the main command buffer and it is completed. If n equal zero, alternate command buffer 1 is selected.

NOTE: The pseudo-macro command capability is implemented by the above 2 commands. The user puts his macro command string in alternate buffer 1 or 2 and then executes that macro string by the X command.

2-6. USING THE EDITOR

2-7. The user should first assign the channels for use with the Text Editor. The Editor uses the following channels:

1. :CI --- console input. All commands and data input via the 'I' and 'T' commands are accepted by the :CI channel.

2. :CO --- Console output. All interactive response to the user and output from the 'V' command is via the console output channel.

3. :OI --- Object input ...not used.

4. :OO --- Object output. Output of an edited module is on the object output channel via the 'P' command of the Editor.

5. :SI --- source input. A module to be edited must first be read into the Editor buffer via the 'R' command from the source input channel.

6. :SO --- source output. The contents of the Editor buffer with line numbers can be sent to the source output channel via the 'W' command.

2-8. The Text Editor may be entered by using the following

DDT-80 command:

.E :ED(CR)

2-9. The editor will then print a pointer (>) prompt. Any of the Editor commands may then be entered. Note that several commands may be entered on one line if they are separated by blanks. Each line of input is terminated by a carriage return.

Example:

```
. M   :00(CR)
 :00   aaaa   :PP(CR)
-assign edited output to the paper tape punch device
 driver.
:SI   aaaa   :PR(CR)
-read source to be edited from paper tape reader device
 driver.
:SO   aaaa   :LP(CR)
-assign output with line numbers to line printer driver.

aaaa .
 . E  :ED(CR)
-user executes the Editor
>
  .
  .
  .
.Editor commands
```

2-10. REENTERING THE EDITOR

2-11. In some some instances (e.g., RAM-based operation, see Section 5), the user will want to reenter the Editor to edit an existing source buffer. The normal procedure described above initializes the Editor's pointers so that the buffer appears to

be empty. To reenter the Editor without reinitializing the pointers, use the command:

```
:E    :ER(CR)
.     -user executes Editor Reentry address.
>
```

2-12. NOTES

2-13. Concerning Line Numbers . Lines (separated by carriage returns) are counted as they are read in. Line numbers reside in memory with their respective lines and correspond to line numbers assigned by the ASMB-80 Assembler. These numbers are fixed for data read and cannot be altered. When an insertion is done, each inserted line receives the same line number (0000). When a line is deleted, its corresponding line number is also deleted.

2-14. Concerning Buffer Full Conditions. The Editor reads or insert until the buffer is filled with 83 or less characters of the buffer end before the 'BUFFER FULL' message is printed . The 83 character padding helps preserve the last record in its entirety. The crucial point occurs when the user ignores the 'BUFFER FULL' message and continues reading or inserting . The buffer data will overrun into Editor Work space and variables, thus corrupting the buffer data (no EOB) as well as Editor variables. The edit session could then be a total loss. When the buffer is full, the user should remove as many of the records at the beginning of the buffer as possible. This is accomplished by punching and then deleting those records. An exchange or change could be executed if some editing is necessary.

2-15. Start of Text (STX) and End of Text (ETX) Characters. The STX and ETX characters are used by the ASMB-80 Assembler to connect a source module that had to be broken into pieces for efficient handling. The Editor views the STX character as if it

were a character in the first source statement of the module. It is unprintable but will be included in the buffer. It is recommeded that STX immediately precede a comment statement or the first statement in all but the first module in a multiple module set. (See Section 3-81). The reason for not having an STX followed by a carriage return at the beginning of the module is the Assembler will assemble a blank line and the line numbers might become confusing. The ETX character will cause the Editor to stop reading. It also will be included in the source buffer with a carriage return immediately following it. It is also unprintable and should not be on a line by itself. Therefore, it should be the last character of the last source statement in the module. Note: (CNTL - B = STX, CNTL - C = ETX)

2-16. Concerning Rubout and Backspace Characters. The Editor does not protect the user from backing up past the beginning of the command or insert buffers. If too many rubouts or backspaces are entered while keying in data, the results are unpredictable. If the possibility exists of having to go back to the beginning of the line with a large number of rubouts, the Control-U character is best to use.

SECTION 3

ASMB-80 ASSEMBLER

3-1. INTRODUCTION

3-2. MOSTEK ASMB-80 Z80 ASSEMBLER

This section describes the function and operation of the MOSTEK ASMB-80 Z80 Assembler. The ASMB-80 Assembler is provided in ROM. In conjunction with the resident Text Editor and the Relocating Linking Loader it provides the means for editing, assembling, and loading Z80 programs. The Assembler reads Z80 source mnemonics and pseudo-ops and outputs an assembly listing and object code. The object code is in industry standard hexadecimal format modified for relocatable, linkable assemblies.

3-3. CAPABILITIES

3-4. The ASMB-80 Assembler recognizes all standard Z80 source mnemonics. It supports global symbols, relocatable programs, and a printed symbol table. In conjunction with the MOSTEK Text Editor and Relocating Linking Loader, the user has state-of-the-art software for building, assembling and loading programs. The Assembler can assemble any length program, limited only by a symbol table size which is user selectable.

3-5. HARDWARE CONFIGURATION

3-6. The Assembler will work on a system with the following configuration:

1. Console I/O.
2. 4K RAM, minimum.
3. Resident DDT-80 Operating System.

3-7. SOFTWARE CONFIGURATION

3-8. The Assembler uses subroutines which are in the resident DDT-80, Text Editor, and Relocating Linking Loader. The Assembler is resident in ROM starting at location 0C000H.

3-9. DEFINITIONS

1. SOURCE MODULE - the user's source program. Each source module is assembled into one object module by the Assembler. The end of a source module is defined by an 'END' pseudo-op or an EOT CHARACTER (04H) on input. The source module is read on channel :SI.
2. OBJECT MODULE - the object output of the Assembler for one source module. The object module contains linking information, address and relocating information, machine code, and checksum information for use by the MOSTEK Relocating Linking Loader. The object module is in ASCII. A complete definition of the MOSTEK object format is in Appendix B. The object module is output on SDB-80 channel :00.
3. LOAD MODULE - the binary machine code of one complete program. The load module generally is defined in RAM. It is created by the MOSTEK Relocating Linking Loader from one or more object modules.
4. LOCAL SYMBOL - a symbol in a source module which appears in the label field of a source statement.
5. INTERNAL SYMBOL - a symbol in a source (and object) module which is to be made known to all other modules which are loaded with it by the Relocating Linking Loader. An internal symbol is also called global, defined, public, or common. Internal symbols are de-

fined by the GLOBAL pseudo-op. An internal symbol must appear in the label field of the same source module. Internal symbols are assumed to be addresses, not constants, and they will be relocated when loaded by the Loader.

6. EXTERNAL SYMBOL - a symbol which is used in a source (and object) module but which is not a local symbol (does not appear in the label field of a statement). External symbols are defined by the GLOBAL pseudo-op. External symbols may not appear in a expression which uses operators. An external symbol is a reference to a symbol that exists and is defined as internal in another program module.
7. GLOBAL DEFINITION - both internal and external symbols are defined as "GLOBAL" in a source module. The Assembler determines which are internal and which are external.
8. POSITION INDEPENDENT - a program which can be placed anywhere in memory. It does not require relocating information in the object module.
9. ABSOLUTE - a program which has no relocation information in the object module. An absolute program which is not position independent can be loaded only in one place in memory in order to work properly.
10. RELOCATABLE - a program which has extra information in the object module which allows the Relocating Linking Loader to place the program anywhere in memory.
11. LINKABLE - a program which has extra information in the object module which defines internal and external symbols. The Relocating Linking Loader uses the information to connect, resolve or link, external references to internal symbols.

3-10. ASSEMBLY LANGUAGE SYNTAX

3-11. An assembly language program (source module) consists of labels, opcodes, pseudo-ops, operands, and comments in a sequence which defines the user's program. The assembly language conventions are described below.

3-12. DELIMITERS

3-13. Labels, opcodes, operands, and pseudo-ops must be separated from each other by one or more commas, spaces, or tab characters (ASCII 09H). The label may be separated from the opcode by a colon, only, if desired.

3-14. LABELS

3-15. A label is composed of one or more characters. If more than 6 characters are used for the label, only the first 7 are recognized by the Assembler. The characters in the label cannot include ' ( ) * + , - < > = . / :; or space. In addition, the first character cannot be a number (0-9). Table 3-1 summarizes the allowed characters in a label or symbol. A label can start in any column if immediately followed by a colon (:). It does not require a colon if started in column one. For example:

```
allowed
--------
LAB
L923
$25
A25E:
not allowed
---------
9LAB                    ;STARTS WITH A NUMBER
```

```
L)AB                    ;ILLEGAL CHARACTER IN LABEL
L:ABC                   ;ILLEGAL CHARACTER IN LABEL
```

3-16. OPCODES

3-17. There are 74 generic opcodes (such as 'LD'), 25 operand key words (such as 'S'), and 693 legitimate combinations of opcodes and operands in the Z80 instruction set. The full set of these opcodes is documented in the "Z80 CPU TECHNICAL MANUAL" and listed in Appendix A of this manual. The ASMB-80 Assembler allows one other opcode which is not explicitly shown in the Z80 CPU Technical Manual:

```
IN   F,(C) ;SET THE CONDITION BITS ACCORDING
           ;TO THE CONTENTS OF THE PORT DEFINED BY THE C-
            REGISTER
```

Table 3-1. ASCII CHARACTER SET (7-BIT CODE)

LSD \ MSD	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 0000	NUL	DLE	SPACE	0	@	P	`	p
1 0001	SOH	DC1	!	1	A	Q	a	q
2 0010	STX	DC2	"	2	B	R	b	r
3 0011	ETX	DC3	#	3	C	S	c	s
4 0100	EOT	DC4	$	4	D	T	d	t
5 0101	ENQ	NAK	%	5	E	U	e	u
6 0110	ACK	SYN	&	6	F	V	f	v
7 0111	BEL	ETB	'	7	G	W	g	w
8 1000	BS	CAN	(	8	H	X	h	x
9 1001	HT	EM	)	9	I	Y	i	y
A 1010	LF	SUB	*	:	J	Z	j	z
B 1011	VT	ESC	+	;	K	[	k	{
C 1100	FF	FS	,	<	L	\	l	|
D 1101	CR	GS	-	=	M	]	m	}
E 1110	SO	RS	.	>	N	^	n	~
F 1111	SI	US	/	?	O	_	o	DEL

NOT ALLOWED

ADDITIONAL CHARACTERS NOT ALLOWED AS FIRST CHARACTER

3-18. PSEUDO-OPS

The following pseudo-ops are recognized by the Assembler:

1. ORG nn - origin- sets the program counter to the value nn.
2. label EQU nn - equate- sets the value of a label to nn in the program; can occur only once for any label.
3. label DEFL nn - define label- sets the value of a label to nn in the program; may be repeated in the program with different values for the same label.
4. DEFM 'aa' - define message- defines the contents of successive bytes of memory to be the ASCII equivalent code of characters within quotes. Maximum length of the message is 63 characters. Only the first 4 bytes of the object code are shown on the asssembly listing.
5. DEFB n - define byte- defines the contents of a byte located the current program counter address to be n.
6. DEFW nn - define word-defines the contents of a two-byte word to be nn. The least significant byte is located at the program counter address, while the most significant byte is located at the program counter address plus one.
7. DEFS nn - define storage- reserves nn bytes of memory starting at the current

program counter. This cannot be used to reserve storage at the start on end of a module.

8. END nn - end statement- defines the last line of the program. The 'END' statement is required. nn is optional and represents the transfer address (starting execution address) of the program. The transfer address can be used with the Relocating Linking Loader to automatically start execution of a loaded program. The transfer address defaults to the first address of the program.

9. GLOBAL symbol - define global symbol- any symbol which is to be made known among several separately assembled modules must appear in this type of statement. The assembler determines if the symbol is internal (defined as a label in the program), or external (used in the program but not defined as a label).

10. NAME symbol - module name- This pseudo-op defines the name of the program (source and object). The name is placed in the heading of the assembly listing and is placed in the first record of the object module to identify it. This pseudo-op is designed primarily to facilitate future compiler design. The name of a module defaults to 6 blanks.

11. PSECT op - program section- This pseudo-op may appear only once in a source module. It defines the program module attributes for the following operands:

REL - relocatable program (default).

ABS - absolute program. No relocating information is generated in the object module by the Assembler. The module will be loaded where it is origined.

3-19. ASSEMBLER DIRECTIVES

3-20. Assembler Directives are pseudo-ops which modify the assembly listing format. They are not listed in the assembly listing, but they are assigned statement numbers.

1. EJECT - eject, ejects a page of the listing.
2. TITLE s - title, ejects a page and prints the string 's' at the top of each page as a user heading. 's' can be up to 32 characters long.
3. LIST - listing on, turns the assembly listing on.
4. NLIST - listing off, turns the assembly listing off.

3-21. OPERANDS

3-22. There may be zero, one, or two operands present in a statement depending upon the opcode used. An operand which appears in a statement may take one of the following forms.

3-23. A generic operand, such as the letter 'A', which stands for the accumulator. Table 3-2 summarizes these operands and their meanings.

Table 3-2 GENERIC OPERANDS

A ---- A register (accumulator)
B ---- B register
C ---- C register
D ---- D register
E ---- E register
F ---- F register (flags)
H ---- H register
L ---- L register

AF ---- AF register pair
AF' --- AF' register pair
BC ---- BC register pair
DE ---- DE register pair
HL ---- HL register pair

SP ---- Stack Pointer register
$ ---- Program Counter

I ---- I register (interrupt vector MS byte)
R ---- Refresh register

IX ---- IX index register
IY ---- IY index register

NZ ---- Not zero
Z ---- Zero
NC ---- Not carry
C ---- Carry
PO ---- Parity odd/not overflow
PE ---- Parity even/overflow
P ---- Sign positive
M ---- Sign negative

3-24. A constant. The constant must be in the range 0 through OFFFFH. It can be in the following forms:

1. Decimal - this is the default mode of the Assembler. Any number may be denoted as decimal by following it with the letter 'D'.
   E.g., 35,249D.

2. Hexadecimal - must begin with a number (0-9) and end with letter 'H'.
   E.g., 0AF1H.

3. Octal - must end with the letter 'Q' or 'O'. E.g., 377Q, 277O

4. Binary - must end with the letter 'B'. E.g., 0110111B

5. ASCII - letters enclosed in quote marks will be converted to their ASCII equivalent value. E.g., 'A' = 41H

3-25. A label which appears elsewhere in the program. Note that labels cannot be defined by labels which have not yet appeared in the user program (this is an inherent limitation of a two pass assembler).

```
L EQU H
H EQU I
I EQU 7     IS NOT ALLOWED.
I EQU 7
H EQU I
L EQU H     IS ALLOWED.
```

3-26. The symbol '$' is used to represent the value of the program counter of the current instruction.

3-27. ASMB-80 EXPRESSIONS

3-28. The MOSTEK ASMB-80 Assembler accepts a limited set of expression types in the operand field of a statement. All expressions are evaluated from left to right. Table 3-3 shows the allowed operators. Parentheses may be used to ensure correct expression evaluation.

Table 3-3 ALLOWED OPERATORS IN ASMB-80 ASSEMBLER

Unary plus	(+)
Unary minus	(-)
Addition	(+)
Subtraction	(-)
Shift right 8	(.)

The dot operator (.) may be placed at the end of an expression. Its affect is to shift a 16-bit value right by 8 bits so the most significant byte can be accessed. Zeros are shifted into the higher order bits.

Examples:

-5	=	0FFFBH
+5	=	0005H
-5-(4+1)	=	0FFF6H
0AABBH	=	0AABBH
0AABBH.	=	00AAH

3-29. Note that enclosing expression wholly in parenthese indicates a memory address. The contents of the memory address equivalent to the expression value will be used as the operand value. Integer two's complement arithmetic is used throughout.

3-30. The negative (2's complement) of an expression or quantity may be formed by preceding it with a minus sign.

3-31. In doing relative addressing, the current value of the program counter must be subtracted from the label if a branch is to be made to that label address. E.g.:

```
JR NC,LOOP-$
```

...will jump relative to 'LOOP'

3-32. The allowed range of an expression depends on the context of its use.

3-33. An error message will be generated if this range is exceeded during its evaluation. In general, the limits on the range of relative jump ('JR') are -126 bytes and +129 bytes. Up to 20 total operators, constants, plus labels may appear in one expression.

3-34. For relocatable programs, the Assembler will output relocation information in the object module for those addresses which are to be relocated by the loader. Expressions are determined to be relocatable addresses or non-relocatable constants according to the following rules:

<constant> <operation> <constant>=<constant>
<constant> <operation <relocatable>=<relocatable>
<relocatable> <operation> <constant>=<relocatable>

```
<relocatable> <operation> <relocatable>=<constant>
Example:
  I EQU 1              ;CONSTANT DEFINITION
  DEFW I               ;CONSTANT WHICH WILL NOT BE RELOCATED


  LAB EQU $            ;RELOCATABLE DEFINITION

  .
  .
  .

  JP  LAB              ;RELOCATABLE OPERAND
  JR  LAB-$            ;CONSTANT OPERAND
  JR  +5+(I)           ;CONSTANT OPERAND
```

External symbols are not allowed in expressions. For relocatable programs, external symbols are always considered to be relocatable address constants.

3-35. COMMENTS

3-36. A comment is defined as any characters following a semicolon in a line. A semicolon which appears in quotes in an operand is treated as an expression rather than a comment starter. Comments are ignored by the assembler, but they are printed in the assembly listing. Comments can begin in any column. Note also that the ASMB-80 Assembler ignores any statements which have an asterisk (*) in column one.

3-37. OBJECT OUTPUT

3-38. The object output from the Assembler is output to channel :00. The object output of the Assembler can be loaded by an

Intel hexadecimal loader for nonlinkable, nonrelocatable programs. Extra information is inserted into the object output for linkable and relocatable programs for using the MOSTEK Relocating Linking Loader. For a complete discussion of the object format, see Appendix B.

3-39. ASSEMBLY LISTING OUTPUT

3-40. The assembly listing is output to channel :SO. The user must insert tabs in the source to obtain columns in the assembly listing. The value of each equated symbol will be printed with a pointer (>) next to it. Any address which is relocatable will be identified with a quote (') character. Macro expansions (macro option only) will be printed with a plus (+) character next to the statement number. The statement number and page number are printed in decimal. If the listing option is not selected, errors will be output on the console channel (:CO).

3-41. ABSOLUTE MODULE RULES

3-42. The pseudo-op 'PSECT ABS' defines a module to be absolute. The program will be loaded in the exact addresses at which it is assembled. This is useful for constants, or a common block of global symbols, or a software driver whose position must always be known.

3-43. This method can also be used to define a list global constants.

For example:

```
        PSECT  ABS              ;ABSOLUTE ASSEMBLY
        GLOBAL AA
AA      EQU    0
        GLOBAL AB
```

```
AB EQU    0E3H
   GLOBAL AC
AC EQU    25H
   GLOBAL AD
AD EQU    0AF3H
   END
```

All symbols in the above module will assume constant values which may be used by any other module.

3-44. RELOCATABLE MODULE RULES

3-45. The following rules apply to relocatable programs.

1. Programs default to relocatable if the 'PSECT ABS' psuedo-op is not used or if 'PSECT REL' is specified.

2. Only those values which are 16-bit address values will be relocated. 16-bit constants will not be relocated (internal symbols are exceptions).

For example:

```
AA EQU    0A13H          ;ABSOLUTE VALUE
   LD     A,(AA)         ;AA NOT RELOCATED
AR EQU    $              ;RELOCATED VALUE
   LD     A,(AR)         ;AR WILL BE RELOCATED
   END
```

Any 8-bit quantity, whether derived from a 16-bit address value or not, will NOT be relocated. For example:

```
B8 EQU    20H            ;ABSOLUTE VALUE
   LD     A,B8           ;B8 NOT RELOCATED
```

```
AR   EQU        $             ;RELOCATABLE VALUE
     LD         A,AR          ;AR NOT RELOCATED
     DEFB       AR            ;AR NOT RELOCATED
     LD         A,(IX+AR)     ;AR NOT RELOCATED
     END
```

3-46. Labels equated to labels which are constants will be treated as constants. Labels equated to labels which are relocatable values will be relocated. Internal symbols are exceptions. For example:

```
B8   EQU        20H           ;ABSOLUTE VALUE
C8   EQU        B8            ;ABSOLUTE VALUE
     LD         A,(C8)        ;C8 WILL NOT BE RELOCATED

AR   EQU        $             ;RELOCATABLE VALUE
BR   EQU        AR            ;RELOCATABLE VALUE
     LD         A,(BR)        ;BR WILL BE RELOCATED
```

3-47. Internal symbols will always be marked relocatable. This point is important because an internal symbol will be relocated even though it looks like a constant. This point is discussed further, below.

3-48. External symbols will always be marked relocatable.

3-49. GLOBAL SYMBOL HANDLING

3-50. A global symbol is a symbol which is known by more than one program. A global symbol has its value defined in one program. It can be used by that program any other program. A global symbol is defined as such by the GLOBAL Pseudo-op. For example:

```
GLOBAL  SYM1
  ;'SYM1' is a symbol which is defined as "global".
```

3-51. An internl symbol is one which is defined as global and also appears in the label field of a statement in the same program. The symbol value is thus defined for all programs which use that symbol.

3-52. An external symbol is one which is defined as global but does NOT appear in the label field of a statement in the same program. For example:

```
       GLOBAL   SYM1

       CALL     SYM1
       .
       .
       .
       END
          -'SYM1' is an external symbol
       GLOBAL   SYM1
  SYM1 EQU
       LD   A,(SYM1)
       .
       .
       .
       END
```

-'SYM1' is an internal symbol. Its value is the address of the LD instruction.

3-53. If these two programs were loaded by the MOSTEK Relocating Linking Loader, all global symbol references would be "resolved". This means that each address in which an external symbol was used would be modified to the value of the corresponding internal symbol. The loaded programs whould be equivalent (using one example) to one program written as follows.

```
      CALL SYM1
      .
      .
      .
SYM1  EQU
      LD   A,(SYM1)
      .
      .
      .
      END
```

3-54. Global symbols are used to allow large programs to be broken up into smaller modules. The smaller modules are used to ease programming, facilitate changes or allow programming by different members of the same team.

3-55. The ASMB-80 Assembler has several rules which apply to global symbols. The examples in the following paragraphs should be studied carefully.

3-56. GLOBAL SYMBOL BASIC RULES

Both passes of the Assembler must be done if global symbols are used. This restriction exists because symbols are defined as global during pass 1, and an external reference link list is built up during pass 1.

1. Global symbols follow the same syntax rules as labels. They may not start with a number(0-9) or a restricted character. They may not contain restricted. For example:

allowed

```
GLOBAL  SYM1
GLOBAL  A&&
GLOBAL   B
```

not allowed

```
GLOBAL 1AB              ;STARTS WITH A NUMBER
GLOBAL A=B              ;CONTAINS A RESTRICTED CHARACTER
```

2. An external symbol may not appear in an expression. For example:

```
GLOBAL  SYM1            ;EXTERNAL SYMBOL
CALL    SYM1            ;OK
LD      HL,(SYM1)       ;OK
LD      HL,SYM1+25H     ;NOT ALLOWED
JP      SYM1+2          ;NOT ALLOWED
```

3. An external symbol is always considered to be a 16-bit address; therefore, an external symbol may not appear in an instruction requiring an 8-bit operand. It may not be used for a displacement or an 8-bit constant.

For example:

```
GLOBAL  SYM1            ;EXTERNAL SYMBOL
CALL    SYM1            ;OK
LD      A,(SYM1)        ;OK
LD      A,SYM1          ;NOT ALLOWED
LD      (IX+SYM1),A     ;NOT ALLOWED
BIT     SYM1,A          ;NOT ALLOWED
```

4. In relocatable assembly, a global symbol is always considered to be a relocatable 16-bit address. This applies to both internal and external symbols. It does not apply to absolute assemblies (PSECT ABS).

5. By definition an external symbol cannot also be an internal symbol.

6. For a set of modules to be loaded, no duplication of internal symbol names is allowed. That is, an internal symbol can be defined only once in a set of modules to be loaded together.

3-57. GLOBAL SYMBOL ADVANCED RULES

1. An external symbol cannot appear in the operand field of a 'EQU' or 'DEFL' pseudo-op. Thus, an external symbol must be explicitly defined as global. For example:

```
      GLOBAL SYM1       ;EXTERNAL SYMBOL
SYM2  EQU    SYM1       ;NOT ALLOWED
SYM3  DEFL   SYM1       ;NOT ALLOWED
```

2. All references to an external symbol are marked relocatable, except the first reference in a program. The object code for these references is actually a backward link list, terminating in the constant OFFFFH. (See definition of object format in Appendix-B). This rule does not apply to absolute assemblies.

3. An internal symbol is always marked relocatable, except for absolute assemblies. This point is important, because an internal symbol will be relocated even though it looks like a constant. For example:

```
PSECT  REL              ;RELOCATABLE MODULE
GLOBAL YY                ;INTERNAL SYMBOL
YY     EQU    0AF3H     ;YY WILL ALWAYS BE MARKED RE-
                         LOCATABLE.
LD     A,(YY)            ;YY WILL BE RELOCATED WHEN
                         LOADED.
;THE ABOVE INSTRUCTION LOADS THE CONTENTS OF THE AD-
 DRESS YY.
```

```
;RELOCATED, INTO THE A-REGISTER.


        OR
        PSECT   ABS         ;ABSOLUTE ASSEMBLY
        GLOBAL  YY          ;INTERNAL SYMBOL
YY      EQU     0AF3H       ;YY IS AN ABSOLUTE VALUE
        LD      A,(YY)      ;THIS LOADS THE CONTENTS OF AD-
                             DRESS
                            ;0AF3H INTO THE A-REGISTER
         END
```

4. All other rules that apply to local symbols also apply to internal symbols.

3-58. USE OF THE 'NAME' PSEUDO-OP

3-59. The NAME pseudo-op can be used to identify both a source module and an object module. The name of the module being assembled can be assigned by the NAME pseudo-op. The name is placed in the heading of the assembler listing. The name is also placed in the first record of the object output. The first record is the module definition record (record type 05), and it is described in Appendix B.

3-60. USING THE ASSEMBLER

3-61. The ASMB-80 Assembler is resident in ROM starting at location C000H. The user first prepares his source module using the Editor on paper tape or magnetic tape. To use the Assembler, the user must first assure that the channels are properly assigned. Table 3-4 summarizes how the channels are used by the Assembler. The typical system uses a paper tape reader for source input, a paper tape punch for object output, and a line printer for source output (the assembly listing). The following

DDT-80 command sequence assigns the corresponding device drivers to the channels.

```
.M :00(CR)
:00 aaaa :PP(CR)
    -assign paper tape punch driver to object output chan-
     nel.
:SI aaaa :PR(CR)
    -assign paper tape reader driver to source input chan-
     nel.
:SO aaaa :LP(CR)
    -assign line printer driver to source output channel.
```

Table 3-4 CHANNEL ASSIGNMENTS FOR THE ASSEMBLER

```
:CI --- console input
:CO --- console output
:OI --- not used
:OO --- object module output
:SI --- source module input
:SO --- assembly listing output
```

The user then enters the following command from the DDT-80 Operating system:

```
.E  :AS(CR)
```

where E is the DDT-80 command for execute, and AS is the mnemonic which stands for the Assembler starting address. The Assembler then outputs the following message to the console output device:

```
OPTIONS?
```

Options are described in paragraph 3-63. If no options are to be entered, the user enters "carriage return".

3-62. If memory mapping has not been selected (see Section 5 of

this manual), then the Assembler outputs the following message:

SYMBOL TABLE LIMITS?

If the default symbol table limits are to be used, the user enters "carriage return". The default symbol table limits are 300H and 0E00H, allowing 312 symbols in one program. The Assembler then reads the source module for pass 1. During pass 1, the symbol table and external references are defined. The name of the module is defined, and the external symbol link list is built. At the end of reading, the following message is output to the console output device:

READY PASS2?

The user then reloads the source module for pass 2 of the Assembler. This may involve reloading paper tape into a paper tape reader or rewinding a magnetic tape. Then the user depresses any key on the console to start pass 2. During pass 2, the assembly listing and object output are output. At the end of pass 2, the following message is output on the console output device:

ERRORS = nn

where nn is the total number of errors (in decimal) which were found by the Assembler.

Control is then returned to the DDT-80 monitor.

3-63. OPTIONS

3-64. The Assembler allows the user to select the following options from the console. When the Assembler outputs the message:

OPTIONS?

the user may enter any of the following codes. Each letter entered will be automatically separated by a blank. A carriage return terminates the options.

K -- No listing. This suppresses the assembly listing output.

L -- Listing (default). The assembly listing is output to the source output channel (:SO).

N -- No object output. This suppresses object from the Assembler.

O -- Object output (default). The object output is sent to the object output channal (:00).

P -- Pass 2 only. This selects and runs only pass 2 of the Assembler. The symbol table is left intact from a previous run of pass 1 of the Assembler.

Q -- Quit. This returns control to the DDT-80 Operating System.

R -- Reset the symbol table. This option clears the symbol table of all previous symbol references. This operation is automatically done for pass 1. It is used primarily for single pass operations (described in paragraph 3-74).

S -- Symbol table. The symbol table is normally not output by the Assembler. This option prints a symbol table at the end of the assembly listing.

. -- Abort the command, default the options, and allow the user to try again.

For example:

OPTIONS? P N S(CR)

- the user has selected pass 2 only, no object output, and a symbol table.

3-65. ERROR MESSAGES

3-66. Any error which is found denotes the assembly listing. A single letter abbreviation is printed in the left margin next to the statement which is in error. Table 3-5 defines the Assembler error abbreviations.

For example:

```
O  H2:  LC  A,B
        -designates an opcode error.
```

Table 3-5. ASMB-80 ERROR ASSEMBLER ABBREVIATIONS

B -- invalid operator error. An operator exists in an expression which is not allowed by the Assembler.

D -- invalid digit error. A number exists in an operand which has a digit or character in it which is not allowed.

E -- external symbol usage error. An external symbol is being used in an expression or as the operand of an EQU or DEFL pseudo-op. This is not allowed.

*F - Symbol table full. The symbol table is full as the result of too many symbols being defined. Note that the symbol table limits are as follows:

default: 312 symbols.
auto memory mapping, 4K system: 28 symbols
auto memory mapping, 16K system: 369 symbols

Each symbol uses 9 bytes of memory . All unique symbol names used in expressions, defined as labels, and defined by the GLOBAL pseudo-op are stored in the symbol table. This is an 'abort' error.

I -- invalid operand error. An invalid operand or combination of operands exists for the given opcode.

L -- label error. An invalid character exists in a label or symbol. Table 3-1 shows which characters are allowed in a symbol. This error can also occur for certain expressions when the Assembler scans for a symbol.

M -- Multiple definition error. A symbol was defined in the label field of the source program more than once. This error can be circumvented by using the 'DEFL' pseudo-op.

N -- label required error. An EQU or DEFL pseudo-op is being used without a label in the statement.

O -- opcode error. An invalid opcode exists in the opcode field of the source statement.

P -- PSECT error. The PSECT pseudo-op exists more than once in the same program. This is not allowed.

Q -- unbalanced quote error. An expression has unbalanced quotes in it.

R -- range error. An operand exists which is out of the range allowed for the given opcode. Example: the range of a jump relative (JR) opcode is -126 through +129.

S -- syntax error. An error in an expression exists. This error usually refers to unbalanced parentheses or extra characters in the expression.

T -- truncation of input error. The input statement exceeded 127 characters in length which is the maximum allowed by the Assembler.

U -- undefined symbol error. A symbol used in an expression is undefined. Note that a symbol cannot be defined in terms of a symbol which has not yet appeared in the program. Example:

```
I  EQU  H
H  EQU  J
J  EQU  7      ...IS NOT ALLOWED

I  EQU  7
H  EQU  I
J  EQU  H      ... ALLOWED
```

This is an inherent limitation of a two-pass assembler. External symbols must be defined by the GLOBAL pseudo-op. In single pass operation, forward references and global symbols will be flagged with this error.

V -- overflow error. An expression caused an overflow error in the Z80 CPU when it was evaluated. This can occur for any expression involving arithmetic operators.

*X - memory mapping error. The Symbol Table limits must be defined as:
300H≤ lower limit,
lower limit<upper limit.

* -- These errors are 'abort' errors. They will abort the assembly process and print the message on the console device.

3-67. ABORT ERRORS

3-68. Several errors abort the Assembler when they are encountered. These are noted in Table 3-5. Abort error messages are outut to the console output device. Control is immediately returned to the DDT-80 Operating System. Abort errors may occur during pass 1 or pass 2. For example:

ABORT ERROR = F

- means the symbol table is full. The user must modify the size of the symbol table and retry the assembly.

3-69. ADVANCED OPERATIONS

3-70. CHANGING THE SYMBOL TABLE SIZE

3-71. The symbol table of the Assembler may be placed anywhere in the RAM by the user. The symbol table defaults to the following RAM locations 0300H through 0E00H. These range allows 312 symbols to be defined in one program. The user may enter up to two operands to change the size of the symbol table when the Assembler asks for the symbol table limits:

SYMBOL TABLE LIMITS? op1, op2. op1 is the address (hexadecimal) which defines the start of the symbol table. It must be greater than or equal to 300H. op2 defines the end of the symbol table. It must be greater than op1. Each symbol uses 9 bytes of RAM. The length defined by (op2-op1) does not have to be evenly divisible by 9. For example: if the user enters: SYMBOL TABLE LIMITS? 500,700, then the symbol table will use ram locations 500H through 700H, and up to (700H-500H) /9 = 56 symbols may be defined in a source module to be assembled.

3-72. USING ONE I/O DEVICE WITH THE ASSEMBLER

3-73. The ASMB-80 Assembler may be used with only one I/O device, such as a teletype. The source input channel is assigned to the teletype tape reader driver (reader step control is required). The object output channel is assigned to the teletype printer as follows:

```
        teletype
.M  :00(CR)
:00   aaaa :TT(CR)
.     -assign object output to teletype.
:SI   aaaa :TR(CR)
      -assign source input to teletype reader.
:SO aaaa :TT(CR)
      -assign source output (listing) to teletype.
aaaa aaaa .
```

Then the Assembler is executed with the option 'K' for no listing:

```
.E  :AS(CR)
OPTIONS? __
```

Pass 1 and pass 2 are done. The object output will be output on the teletype during pass 2. The Assembler is then executed one more time with option 'N' for no object and 'P' for pass 2 only:

```
.E :AS(CR)
OPTIONS? PN(CR)
```

The source paper tape is read once more during this pass, then the listing is printed on the teletype.

3-74. PASS 2 ONLY OPERATION (SINGLE PASS OPERATION)

3-75. The ASMB-80 Assembler can be used as a single pass

assembler under the following restrictions:

1. No GLOBAL symbols are defined.
2. No forward symbol references occur.
3. The NAME pseudo-op is not in the source.

3-76. The Assembler will correctly assemble Z80 programs under the above restrictions during pass 2. This is useful for assembling data tables and certain types of programs. The Assembler symbol table should be initiatlized to assure proper operation in this mode. This may be done by using the 'R' option to reset the symbol table prior to assembling using pass 2 only as follows:

```
.M  :00(CR)
:00  aaaa :PP(CR)
      -user assigns object output to paper tape punch
:SI aaaa :PR(CR)
     -user assigns source input to paper tape reader
:SO aaaa :LP(CR)
     -user assigns source output to line printer
aaaa aaaa .
.E  :AS(CR)
OPTIONS?  P R(CR)
     -user selects pass 2 only operation and resets the
      symbol table prior to assembly.
     .
     .
     .
```

The symbol table initialization described above only has to be done after power up and after symbols are left in the table from a previously assembly.

3-77. USING THE ASSEMBLER AS A LEARNING TOOL

3-78. The ASMB-80 can be used as a Z80 learning tool. Z80 opcodes and operands can be assembled from the console device to help the novice become acquainted with the instruction set. In this mode, pass 2 alone is run with all I/O directed to the console device. The user can enter source statements from the console keyboard. The assembled listing is directed back to the console output device, so the results of assembling each statement can be seen.

3-79. To perform this operation, a special driver must be assigned to the console input channel. The driver is shown in Figure 3-1. The user assigns the channel as follows:

```
.M :00(CR)
:00 aaaa :TT(CR)
    - assign object output to console output device (e.g.,
      teletype)
:SI aaaa (ECHO)(CR)
    - assign source input channel to address of special
      driver (Figure 3-1)
:SO aaaa :TT
    - assign source output to console output device (e.g.,
      teletype).
aaaa aaaa.
```

The assembler is executed with option 'P' (pass 2 only). If you do not wish to see the object output use option 'N' (no object output). The symbol table should be reset using the 'R' option as follows:

```
.E  :AS(CR)
OPTIONS? N P R(CR)
```

3-80. Note that the same restrictions apply to this mode of

operation as to pass 2 only operation:

1. No GLOBAL symbols.
2. No forward symbol references.
3. No NAME pseudo-op.

Figure 3-1 - ECHO DRIVER

```
      FIGURE 3-1  ECHO DRIVER              MOSTEK FLP-80 ASSEMBLER V2.0 PAGE 0001
ADDR  OBJECT    ST # SOURCE STATEMENT          DATASET = DK0:FIG3D1.

                0002 ; GATHER THE CHARACTER
'0000  CDB3E6   0003          CALL    TK      ;KEYBOARD INPUT DRIVER
                0004 ; ECHO THE CHARACTER
'0003  CD7EE6   0005          CALL    ST      ;PRINTER OUTPUT
                0006 ; SUBSTITUTE TT FOR ST ABOVE IF TELETYPE IS
                0007 ; BEING USED INSTEAD OF SILENT 700
'0006  D5       0008          PUSH    DE      ;SAVE CHARACTER
'0007  7A       0009          LD      A,D     ;CHARACTER INTO A-REG
'0008  FE0D     0010          CP      0DH     ;CARRIAGE RETURN?
'000A  CC9CE5   0011          CALL    Z,CRLF  ;YES. OUTPUT CR AND LF
'000D  D1       0012          POP     DE      ;RESTORE CHARACTER
'000E  7A       0013          LD      A,D
'000F  C9       0014          RET             ;RETURN TO CALLER
                0015 ;
 >E59C          0016 CRLF     EQU     0E59CH  ;CR AND LF
 >E6B3          0017 TK       EQU     0E6B3H
 >E680          0018 TT       EQU     0E680H
 >E67E          0019 ST       EQU     0E67EH
                0020          END

ERRORS=0000
```

3-81. ASSEMBLING SEVERAL SOURCE MODULES TOGETHER

3-82. Several source modules may be assembled together to form one object module. The 'END' pseudo-op must appear only in the last module. Each source tape to be read except the first should start with a comment record in which the first characer is an STX

control character (02H), (e.g. <STX> ;comment). Each module except the last one should end with an ETX control character (03H)in a comment e.g., (e.g. ;<ETX>). When the Assembler reads the ETX character, it will continue reading the tape until it runs out of the reader. However, all information except an STX character will be ignored. The next source tape in sequence may be loaded. The Assembler will continue reading but ignoring characters until an STX character is encountered. After STX or 'END' pseudo-op. (AN EOT character, 04H, can be used in place of the 'END' pseudo-op to terminate the last source module).

3-80. Note that, for normal operation, all source tapes must be read in sequence for both passes of the assembler.

3-81. MACRO OPTION

3-82. The ASMB-80 Assembler can be expanded to handle macros. The option 'M' activiates 4 linkages to a RAM-loadable macro handler program. The macro handler program must first be loaded into RAM before the Assembler is executed. The macro option can be used only with 16K or more of RAM.

3-83. Complete documentation on linking a macro handler to the Assembler is included in the ASMB-80 Source Listing, MOSTEK Part Number 78536. MOSTEK does not supply a macro handler package for this version of the ASMB-80 Assembler.

ERRATA

1. The displacement in indexed addressing ((IX+D), (IX-d), (IY+d), (IY-d))is not checked by the Assembler for valid range for the following instructions: RLC, RL, RRC, RR, SLA, SRA, SRL.

2. The Assembler will not properly assemble a series of 'DEFM' pseudo-ops if the No object ('N') option is selected. The result is that control of the system is lost and 'reset' must be performed.

SECTION 4

ASMB-80 RELOCATING LINKING LOADER

4-1. INTRODUCTION

4-2. This section describes the operation of the ASMB Relocating Linking Loader. This Loader will load and link both relocatable and non-relocatable programs produced by the ASMB-80 assembler. Non-linkable and non-relocatable programs can also be loaded by the Relocating Linking Loader or by the absolute loader in DDT-80 (See Para. 1-57).

4-3. The Relocating Linking Loader enables separately assembled object modules to be linked together and to be relocated to any place in user's RAM memory. This enables the program designer to utilize a modular approach in software development. A large program, for example, can be created as a collection of relative short individual modules. These modules can be separately assembled, debugged, and then combined into a complete program when loaded. In many cases this approach can significantly reduce the amount of assembly and debug time required during program development.

4-4. The Relocating Linking Loader automatically links global symbols which provide communication or linkage between program modules. A global symbol is a symbol which is defined within a program module but can also be referenced by other program modules. As object programs are loaded, a table containing global symbol references and definitions is built up. At the end of each module, the Loader resolves all references to global symbols that have been referenced. The symbol table can be printed to list all global symbols and their load addresses. The number of object modules which can be loaded by the loader is limited only by the amount of RAM available for the modules and

the symbol table.

4-5. The beginning and ending addresses of each program module are printed on the console device as it is loaded. The transfer or execute address as defined by the Assembler "END" pseudo-op is also printed for the first module loaded. The Loader execute command(E) can be used to automatically start execution at the transfer address.

4-6. The Relocating Linking Loader allows loading of both relocatable and non relocatable modules. Non-relocatable modules will never be relocated and are always loaded at therir starting address (ssss) as defined by the ORG pseudo-op during assembly. Relocatable modules are located start at an offset address plus module starting address (ssss). The offset address is specified as an operand for the load command and if not specified defaults to the end of the previously loaded module. The PSECT pseudo-op of the Assembler can be used to define a module as either relocatable or absolute (non-relocatable).

4-7. COMMANDS FOR RELOCATING LINKING LOADER

4-8. LOADER ENTRY. To enter the Relocating Linking Loader from DDT-80 the user types the load command L and one or more operands. If no operands are specified, the absolute loader in the operating system is executed to load a program not requiring relocation or linkage to other modules (see Para. 1-57). The operand addresses (aaaa and bbbb) are in hexadecimal and can be up to 4 digits in length (see Para. 1-22. for syntax).

.L aaaa bbbb(CR) aaaa = Offset Address
bbbb = Origin of global symbol table. If this address is not specified it defaults to an address specified by the Loader (see Para. 4-12).

NOTES

1. BEGINNING ADDRESS for an Absolute Module = ssss
2. BEGINNING ADDRESS for a Relocatable Module = aaaa+ssss
3. ssss = Module starting address as defined by ORG pseudo-op.
4. Space for the global symbol table is allocated downward in memory from the table orgin (see Para. 4-11). The table is initialized after execution of the L command from DDT-80.

4-9. LOADER COMMANDS. The loader commands are as follows:

1.	*L aaaa(CR)	Load Next Program Module BEGINNING ADDRESS for an Absolute Module = ssss. BEGINNING ADDRESS for a Relocatable Module = aaaa+ssss. If aaaa is not specified, it defaults to the end of the last module loaded.
2.	*T(CR)	Print Global Symbol Table (see Para.4-13).
3.	*E(CR)	Start execution at the EXECUTE or transfer address defined by the END pseudo-op of the first module loaded.
4.	*.	Return to DDT-80 Operating System.
5.	*Any other Command	Returns loader prompting character(*).

NOTES:

1. Loader prompting character = *
2. DDT-80 prompting character = .

4-10. LOADER SYMBOL TABLE

4-11. In the linking process the Relocating Linking Loader builds a symbol table of global references between program modules and resolves these references as each individual module is loaded. The number of modules that can be loaded by the Loader is limited only when the size of all the linked modules combined plus the global symbol table exceeds th user's RAM memory space. Space for this table is allocated dynamically downward in memory from its origin.

Table Length = (N+1)X11 N=Number of unique global symbols

4-12. The symbol table origin is specified by either the user as the second operand of the load command or by the Loader if the second operand is not entered. When the Loader specifies the origin it first determines if the system is in the auto-mapping mode (see Para. 5-7). If the system is in the auto-mapping mode, the table origin is placed at location ENDC. ENDC is a location in memory which is a distance D up from the bottom of memory. D is equal to 25% of the total RAM memory length (E.G., ENDC=OFFFH for a 16K system). If the system is not in the auto-mapping mode, the table origin is then placed at the top of RAM memory minus 512 bytes. The space of 512 bytes is reserved for user I/O drivers.

4-13. At any time during a load sequence the user can list the global symbol table with the T command printing each symbol and its address on the console device. Global symbol addresses which are unknown are marked undefined (UNDEF=****). A global symbol

is defined when a module is loaded which contains the symbol in the label field and also a reference of the symbol by the GLOBAL pseudo-op.

4-14. The end address of the global symbol table varies dynamically depending upon the total number of global symbols defined during the load process. If during a load sequence, the user wishes to know the position or length of the symbol table in memory, he can return to the DDT-80 operating system and interrogate (M command) four locations in scratchpad RAM. Locations FF04H and FF05H contain the symbol table beginning or origin address and locations FF0AH and FF0BH contain the symbol table address. Since space for the table is allocated downward in memory, the beginning address will always be greater than the ending address.

4-15. REENTERING THE LOADER TO PRESERVE THE SYMBOL TABLE

4-16. To reenter the Relocating Linking Loader without modifying the existing symbol table the user can execute address DBFBH using the E command. It should be noted that when the Loader is entered initially from the operating system using the L command the table end address is reset to indicate zero symbol entries. This causes any table from a previous load sequence to be overwritten. The reentry point which does not alter the existing symbol table can often be useful. After executing a program, for example, the user may wish to print out the existing global symbol table or load a program module which was accidently omitted. It should be noted that the execute or transfer address of the first module loaded is not saved upon exit from the loader. On reentry this causes the loader (E) command to be ignored.

4-17. LOADER ERROR MESSAGES. The loader error messages are de-

fined as follows:

Number	Comment	Return To
1	Checksum	Loader
2	Double definition of a global symbol	Loader
3	Attempt to overwrite loader symbol table	Operating System
4	Attempt to load outside of available memory	Operating System
5	Symbol table full. This is caused by the end of the symbol table reaching the bottom of RAM memory (0000H).	Operating System

4-18. Errors which return to the loader are classified as non-fatal with the loader symbol table remaining intact allowing continuation of the load sequence. However, errors which return to the operating system are fatal and the loading process must be restarted from the beginning with the first module.

4-19. As modules are read global symbol definitions and references are placed in the symbol table. A global symbol is defined by a module if it occurs in the label field of the module. A global symbol is referenced by a module if it occurs in the operand field. A global symbol followed by an ERROR 2 message is printed by the Loader if during the process of loading a module, a global symbol definition is encountered, which already existed in the table from a previously loaded module, (see Example 3 Para 4-26). In this case the second definition of the symbol is ignored because the previous definition could have been used to resolve global references from other modules. ERROR 2 can also occur. When a program is accidently loaded twice.

4-20 If a checksum error (ERROR 1) occurs in a data record, the address of the next location above where the last byte of the record was stored is printed on the console device. This gives the user the capability to correct the data in memory using DDT-80 if he does not wish to reload the module. If a user decides to reload a program after a checksum error in a data record he should always reload the module at the same offset that was used by the initial load. This is required when the module contains global symbol definitions because the second definition of the global symbols will be ignored during the reload (see Para. 4-19). In this case ERROR 2 messages for double definition of global symbols should be ignored.

4-21. If no addresses are printed out and a checksum error occurs, then the error was detected in a non-data type record (External, Internal, relocating, EOF of Mod Def). In this situation in some cases the module can be reloaded at the same offset address and the proper linkage will be obtained. However, if the checksum error was caused by error in global symbol definition (Internal Record) reloading would not generate the proper linkage. To safeguard against this case it is recommended that if a checksum error occurs that is not in a data record, then the load sequence should be started over at the beginning from DDT-80 (see Para. 4-8) initializing the global symbol table.

4-22. LOAD SEQUENCE EXAMPLE

4-23. Commands entered by the operator are followed by a terminator indicating a carriage return.

4-24. EXAMPLE 1 Three relocatable modules each having an origin of 0000H are loaded and linked together starting at the load address 0100H.

.L 0100(CR)

```
        BEG ADDR 0100
        EXECUTE  0140
        END ADDR 03FF
        UNDEF SYM  04
         *T(CR)

SYMBOL TABLE (UNDER=****)

   ASBIN          ****  BINDEC  01A0  CRLF  024C  DIV     ****
   ENTRY          0140  ECHO    ****  PTXT  ****  RDCHR   0123
   SQRT           02A6  WRCHR   0180
   *L(CR)
   BEG ADDR       0400
   END ADDR       059F
   UNDEF SYM        01

   *L(CR)

   BEG ADDR       05A0
   END ADDR       0800
   UNDEF SYM        00

   *T(CR)

   SYMBOL TABLE (UNDER=****)

   ASBIN          0430  BINDEC  01A0  CRLF  024C  DIV     05A0
   ENTRY          0140  ECHO    0400  PTXT  04B0  RDCHR   0123
   SQRT           02A6  WRCHR   0180
   *.
   .
```

4-25. EXAMPLE 2 A non-relocatable module (ORG=0500H) is loaded followed by a relocatable module (ORG = 0100H).

```
 .L 0(CR)
BEG ADDR 0500
EXECUTE  0500
END ADDR 063E
UNDEF SYM  02
*L(CR)
BEG ADDR 073F
EXECUTE  073F
END ADDR 0810
*E(CR)
```

4-26. Example 3 Two relocatable modules (ORG=0000H) are loaded but a checksum error (ERROR 1) is encountered in a data record while reading the second module. After correcting a problem with the tape reader the second module is reloaded at the same Offset Address. During the reload of module 2 ERROR 2 (Double Definition of a global symbol) should be ignored.

```
.L 0600(CR)
BEG ADDR 0600
EXECUTE  0600
END ADDR 072F
UNDER SYM 02
*L(CR)
0750 **** ERROR 1
BEG ADDR 0730
EXECUTE 0730
END ADDR 07A0
```

```
UNDEF SYM  00
*L   0730(CR)
SUB1 **** ERROR 2
ASBIN ****ERROR 2
BEG ADDR 0730
EXECUTE 0730
END ADDR 07A0
UNDEF SYM  00
*E(CR)
```

Start Execution of Loaded programs at EXECUTE Address 0600H.

SECTION 5

ASMB-80 RAM-BASED OPERATION

5-1. INTRODUCTION

5-2. The ASMB-80 Editor, Assembler and Loader may operate directly on data stored in RAM. This type of operation eliminates the more time consuming intermediate steps of inputting and ouputting data from an external media (e.g., paper or cassette tape). The key to RAM-based operation is to keep program modules relatively small so both the source and object modules can be kept in RAM at the same time (see figure 5-1). During RAM based operation it is also recommended that the porgram source be backed up on an external media after major editing changes. This will prevent loss of the source in case of accidental over-writing during execution of the program load module.

5-3. CONCEPTS

5-4. The Editor works on a source buffer in RAM. This buffer is updated as editing progresses. At the end of an editing session it is usually dumped to an external media. The Assembler then reads the external media (e.g. paper tape) twice and generates an object module. If RAM based operation is desired, the Assembler can also read the source buffer directly. This is accomplished by entering the Auto Mapping Mode (see Para. 5-6) prior to creating the source buffer and by assigning a RAM driver (INA) to the source input channel (see Example 1 Para. 5-14).

5-5. The ASMB-80 Assembler outputs a source listing using the source output channel (:SO) and an object a module using the object output channel (:OO). Typically the object output is

directed to an external media. If RAM based operation is used, however, the object output can be directed to a buffer in RAM by assigning a RAM driver (OUTB) to the channel (:OO). The loader can then read the object module directly from RAM by assigning a RAM driver (INB) to the object input channel (:OI). Even though the loader can place the load module any place in RAM it is recommended that it be positioned to avoid overwriting the source or object modules. This allows the user the flexibility during debug to go back and reload the object module or Edit and Assemble the source without reloading.

5-6. AUTO MAPPING MODE

5-7. The RAM based operation contains resident in firmware a memory mapping or allocation program. This program, called MEMMAP automatically partitions memory into 3 functional sections based on the amount of RAM available (e.g., 4k or 16K).

AUTO MAPPING MEMORY PARTITIONS

Area A - Reserved for Source Buffer in Auto Mapping Mode

Area B - Reserved for Object Buffer

Area C - Reserved for Assembler scratchpad and symbol table and also used by loader symbol table and program load module. The first section designated Area A (see figure 5-1) is used as the source buffer for the Editor and Assembler. With a 16K RAM system its length would be 9.5K bytes of characters. The second section, Area B is used to store the object module output of the Assembler. Area C is utilized by the Assembler with locations 0 to 300H acting as a scratchpad area. The assembler symbol table occupies the space between 300H and the top of Area C. With a 16K system a maximum of 369 symbols would be al-

lowed during assembly and 28 symbols (9 bytes/symbol) with a 4K system. Area C is also available for loading executable program modules without destroying either the source on the object modules. However the top portion of Area C is used by the global symbol table of the Relocating Linking Loader. This table is a push down stack whose size depends on the number of global symbols used in the program (11 bytes/global symbol). The top 512 bytes of RAM are reserved for user I/O drivers. Execution of MEMMAP which resides at location DE79H causes the following:

1. MEMMAP partitions memory by assigning values to the pointers ENDA, ENDB and ENDC depending on the size of the user's RAM space. These pointers define the boundaries between memory sections (see figure 5-1).

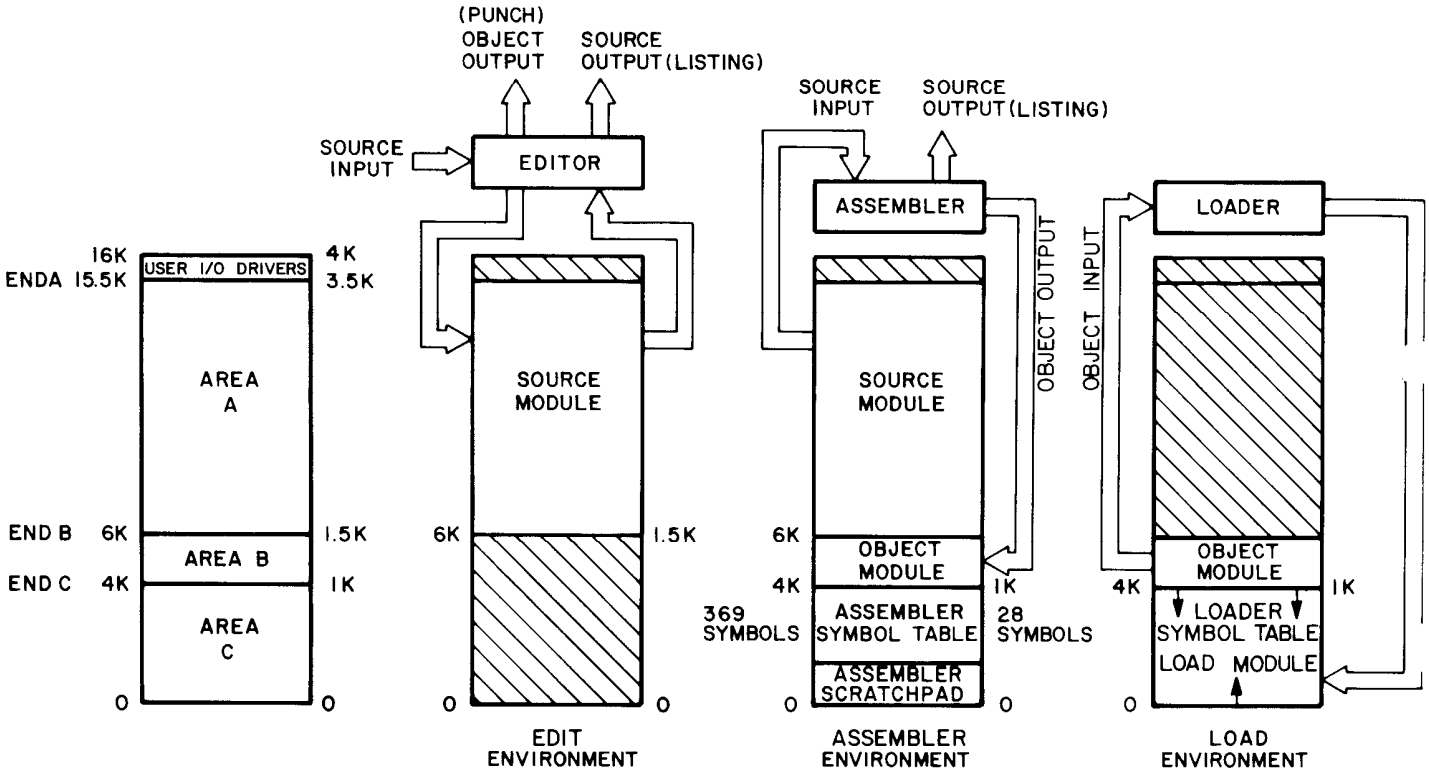
   ENDA = Top of RAM - 512 bytes

   ENDB = 37.5% point in RAM

   ENDC = 25% point in RAM

2. MEMMAP sets a two byte flag in scratch pad RAM (FF06H and FF07H) indicating that the system is in the Auto Mapping Mode. This flag is used by the Relocatable Linking Loader to position its symbol table (see Para. 4-12) and also by the Editor to position the source buffer in RAM. In the auto mapping mode the Editor builds the source starting at the bottom of Area A. If the system is not in the auto mapping mode, the buffer starts at location 0.

Figure 5-1. AUTO MAPPING OPTION



NOTES: 1) ENDA = TOP OF RAM - 512 BYTES
ENDB = 37.5 % POINT IN RAM
ENDC = 25 % POINT IN RAM

2) EXECUTION OF MEMMAP PARTIONS RAM ACCORDING TO THE FUNCTIONS ENDA, ENDB AND ENDC REGARDLESS OF THE SIZE OF RAM (4K, 16K, 24K AND ETC.)

NOTES: 1) ENDA = Top of RAM -512
ENDB = 37.5% point in RAM
ENDC = 25% point in RAM

2) Execution of MEMMAP partitions RAM according to the functions ENDS, ENDB and ENDC regardless of the size of RAM (4K, 16K, 24K and etc.).

3. MEMAP echos the message 'AUTO MAPPING MODE' to verify the system is in that mode.

5-8. If desired the user can modify the RAM memory allocations for the individual functional areas (A,B, and C). This is done by modifying the values of ENDA, ENDB and ENDC with the DDT-80 M command after executing MEMAP.

POINTER	ADDRESS OF 2 BYTES POINTER
ENDA	FF00H
ENDB	FF02H
ENDC	FF04H

5-9. EXIT FROM AUTO MAPPING MODE

5-10. To exit Auto Mapping Mode and return to the normal operational mode the user should reset the auto mapping flag. This is done by setting locations FF06H and FF07H to zero.

5-11. RAM DRIVERS

5-12. The RAM drivers (INA, INB and OUTB) can be assigned to I/O channels during RAM based operations. Pointers used by these drivers and MEMMAP reside in the DDT-80 256x8 scratch pad RAM. This RAM resides at locations FF00H to FFFFH and should not be utilized as a program load area by the user.

NAME	ADDRESS	DESCRIPTION
INA ----	DEE1H	--- Input driver from memory Area A. INA read a character from RAM area A and increments a pointer to the next location from which to read data. Source records in

area A contain line numbers delimited by a carriage return. INA automatically increments its pointer 3 times after reading a carriage return in order ignore line numbers when reading program data. If the initialize bit (bit 3) is set in the E register, INA will read the third location from the bottom of Area A ignoring the first line number. Data read by INA is returned in registers D and A.

INB DF05H --- Input driver from memory Area B. INB reads a character from RAM Area B and increments a pointer to the next location from which to read data. If the initialize bit (bit 3) is set in the E register, then INB will read the first or bottom location in Area B. The data read is returned in registers D and A.

OUTB ---- DF1AH --- Output driver to memory Area B. OUTB outputs the character in the D register to memory Area B. A pointer is incremented to point to the next location for outputting data. If the initialize bit (bit 3) is set in the E register, then the first character is outputted to the first or bottom

location in Area B.

5-13. EXAMPLES OF RAM BASED OPERATIONS

5-14. EXAMPLE 1. The operator using the Auto-Mapping Mode edits, assembles and loads a program from RAM. The address aaaa represents any random address previously assigned to the I/O channels.

Step 1. Enter Auto-Mapping Mode by executing MEMMAP.

```
.E DE79(CR)
AUTO MAPPING MODE
.
```

Step 2. Editor I/O Setup

.M :00(CR)	Assign I/O channels as follows:
:00 aaaa :PP(CR)	paper tape punch to object output
:SI aaaa :PR(CR)	paper tape reader to source input.
:SO aaaa :TT(CR)	teletype typehead to source output.

Step 3. Edit the Source Module

.E :ED(CR)	Enter editor program.
>R(CR)	Read the source module from paper tape into the source buffer.
Other Editor Command	The user edits the source module.
>P (CR)	After the editing session is complete the user saves the updated source on paper tape.

At this point the source buffer exists in RAM starting at location 0600H for a 4K system or 1800H for a 16K system (see figure 5-1).

Step 4. Assembler I/O Setup

.M :00(CR)

:00 :PP DF1A(CR) Assign the RAM output driver OUTB which has an address of DF1A (see Para. 5-10) to the object output channel.

:SI :PR DEE1(CR) Assign the RAM input driver INA which has an address of DEE1 (see Para. 5-10) to the source input channel.

Step 5. Assemble Source Module

.E :AS(CR) User executes the assembler. After the assembler completes execution, the source buffer remains intact in memory Area A and the object is stored in Area B. It should be noted that memory Area C is overwritten by the assembler symbol table and scratchpad (locations 0-300H).

Step 6. Loader I/O Setup

.M :0I(CR)

.0I aaaa DF05(CR) Assign the RAM input driver INB which has an address of DF05 (see para. 5-12.) to the object input channel.

Step 7. Load Object Module

.L 0100(CR) The object module is read from memory Area B and the data is loaded in Area C to prevent

overwriting of the source or object modules.

Step 8. Execute Program

If errors occur during program execution, the program can be edited and reassembled from the RAM source buffer by simply returning to Step 2. This however assumes that program execution does not modify the source buffer in Area A.

5-15. Example 2. The user increases the source buffer length to accommodate a source module that is greater than memory Area A (see memory map figure 5-1). This is accomplished by outputting the object module on paper tape and extending the source buffer to the bottom of Area B which is normally reserved for the object module during RAM based operation. In a 16K system the source buffer would be extended by 2K bytes.

Step 1. Enter Auto-Mapping Mode by executing MEMMAP.

```
.E DE79(CR)
AUTO MAPPING MODE
```

Step 2. Expand Source buffer in Area A (see figure 5-1) by modifying the location of ENDB to equal ENDC.

```
.M FF02(CR)          Change ENDB location from
FF02 FF FF(CR)       17FFH to 0FFFH. This
FF03 17 0F(CR)       example assumes 16K of user
                     RAM.
```

Step 3. Edit the Source Module

```
.E :ED(CR)           Enter SDB-80 editor program
>R(CR)               Read the source module from an
                     external media.
```

Other Editor Commands	The user edits the source module
>PO(CR)	Save updated source on paper tape

Step 4. Assembler I/O Setup

.M :OO(CR)	
.OO aaaa :PP(CR)	Assign the paper tape punch to the object output channel.
.SI :PR DEE1(CR)	Assign the RAM input driver INA which has an address of DEE1 (see Para. 5-12) to the source input channel.

Step 5. Assemble Source Module

.E :AS(CR) User executes assembler.

The assembler reads the expanded source buffer and outputs the object module on paper tape.

It should be noted that the source buffer could be expanded even further by reducing the size of memory Area C. This in turn, however, would reduce the number of symbols allowed in the Assembler symbol table (9 bytes/symbol).

5-16 MEMTOP

The subroutine MEMTOP (DEBBH) which is called by MEMMAP is a useful system routine that is available to the user. MEMTOP starts at location 0 and determines the top of contiguous RAM by writing and reading a test pattern from each location. After each memory location is tested it is restored so RAM is not modified on return from MEMTOP.

On Exit:	ENDMEM(FF24H)=Top of RAM
	HL=Top of RAM - 512
Registers Modified:	A,B,DE,HL

SECTION 6

ASMB-80 SILENT 700 I/O DRIVERS

6-1. INTRODUCTION

6-2. The I/O interface discussed in this section allows the user to connect the Texas Instruments Silent 700 Model 733 ASR to the system as a terminal. The Silent 700 must be equipped with the Automatic Device Control (ADC) option. The interface is a pure software interface (except for the plug into the serial I/O socket) with routines which drive the devices on the terminal. These device drivers enhance the use of digital cassettes and protect the user from data loss. Only 300 baud operation is supported. The following drivers are resident in ASMB-80 firmware.

1. :TK - Silent 700 keyboard driver
2. :ST - Silent 700 printer driver
3. :TI - Silent 700 playback cassette driver (tape input)
4. :TO - Silent 700 record cassette driver (tape output)
5. ZSK - Silent 700 keyboard driver. Same as :TK except that record and playback units are controlled.
6. ZSP - Silent 700 printer driver. Same as :ST except that record and playback units are controlled.

6-3. USING THE INTERFACE

6-4. On power-up or reset the console in and console out channels are configured for operation with a Silent 700 as a terminal. The drivers :TK and :ST are automatically assigned for Silent 700 keyboard input and terminal printing. In addition the drivers :TI and :TO can be used for cassette tape input and output operations.

6-5. The following sequence illustrates I/O channel assignments used when performing Silent 700 cassette tape operations. In this configuration both the source and the object channels can communicate with the tape units.

```
.M  :CI(CR)
:CI :TK(CR)          Address of keyboard input driver
:CO :ST(CR)          Address of printer driver
:OI aaaa :TI(CR)     Address of tape input driver
:OO aaaa :TO(CR)     Address of tape output driver
:SI aaaa :TI(CR)     Address of tape input driver
:SO aaaa :ST(CR)     Address of printer driver
```

6-6. The drivers ZSK and ZSP perform the same function as :TK and :ST except that they turn off the playback and record units before execution. Since :TI and :TO turn the record and playback units off at the end of logical record, the drivers :TK and :ST are recommended for use with the ASMB-80 Assembler, Editor and Loader programs.

However, in an application program ZSK and ZSP should be used if there could be a keyboard input or printer request while the tape unit was still running in the middle of a record.

6-7. INITIALIZATION OF SILENT 700 DRIVERS

6-8. The Silent 700 drivers (:TI, :TO, ZSK and ZSP) use a flag byte in scratchpad RAM (location 0FF26H) to determine the status (on/off) of the playback and record units. When a S700 driver is called with initialize bit set (bit 3 of the E register) both the playback and record units will be turned off and the flag byte will be reset. After the call the initialize bit is also reset and the normal driver function is performed (see para. 6-9). If a user wishes to write an application program utilizing the S700 drivers, it is recommended that the initialize bit be set for the

first I/O call (e.g. RDCHR or WRCHR). This will guarantee that the flag byte accurately reflects the current status of the terminal.

6-9. DESCRIPTION OF SILENT 700 DRIVERS

For each driver the E register must contain the channel number.

NAME		ADDRESS		DESCRIPTION
:TK	----	E6B3	----	Silent 700 Keyboard Driver. :TK reads a character from the keyboard device and returns the data in the D and A registers.
:ST	----	E67E	----	Silent 700 Printer. :ST outputs a character to the S700 printer. The D register contains the ASCII character to be printed. :ST also delays 200 msec after outputting a carriage return.
:TI	----	DF9B	----	Silent 700 play back (Tape Input). Each time :TI is called it checks the status (on-/off) of the Playback Unit and turns it on if necessary. It then reads a character from tape and returns with it in the D and A register. If bit 7 (most significant bit) of the E register is set, an immediate

		return will be executed if data is not ready (see Para. 1-93).
:TO	DF2F	Silent 700 Reccrd Tape Output. Each time :TO is called it checks the status (on/off) of the Record Unit and turns it on if necessary. :TO then outputs the ASCII character in the D register. The character is recorded on the tape and also echoed on the printer since the devices are in parallel. If the character outputted was a carriage return, :TO delays 200 msec. If the character was a LF indicating the end of a Record the Record Unit is automatically turned off.
ZSK	DFCD	Silent 700 Keyboard Driver. ZSK reads a character from the keyboard device and returns the data in the D and A registers. If either the playback or record units are on, they are turned off by ZSK before reading a character. ZSK does not have an assigned mnemonic requiring the driver address to be assigned to a channel.

NAME	ADDRESS	DESCRIPTION
ZSP	DF89	Silent 700 Printer. ZSP outputs a character to the S700 printer. The D register contains the ASCII character to be printed. ZSP also delays 200 msec after outputting a carriage return. If either the playback or record units are on, they are turned off by ZSP before outputting a character. ZSP does not have an assigned mnemonic requiring the driver address to be assigned to a channel.

6-10. SILENT 700, 1200 BAUD OPTION

The Silent 700 drivers are not specified to work at 1200 baud. Only the ST: and TI: drivers need modification. The following drivers can be used for 1200 baud operation. Note that when assembling from cassette, the source module must be terminated with two END statements. After pass 1 is done, the Assembler will automatically go into pass 2, read the END statements, and finish. The the cassette must be rewound and the Assembler executed again with the "pass 2 only" (P) option.

APPENDIX A

Z80 OPCODE LISTING

APPENDIX B

MOSTEK OBJECT OUTPUT DEFINITION

B-1. Each record of an object module begins with a delimiter (colon or dollar sign) and ends with carriage return and line feed. A colon (:) is used for data records and end of file record. A dollar sign ($) is used for records containing relocation information and linking information. An Intel loader will ignore such information and allow loading of non-relocatable, non-linkable programs. All information is in ASCII.

B-2. Each record is identified by a "type". The type appears in the 8th and 9th bytes of the record and can take the following values:

00 - data record
01 - end-of-file
02 - internal symbol
03 - external symbol
04 - relocation information
05 - module definition

B-3. DATA RECORD FORMAT (TYPE 00)

Byte 1	Colon (:) delimiter.
2-3	Number of binary bytes of data in this record. The maximum is 32 binary bytes (64 ASCII bytes).
4-5	Most significant byte of the start address of data.
6-7	Least significant byte of start address of data.
8-9	ASCII zeros. This is the "record type" for data.
10-	Data bytes.
Last two bytes	- Checksum of all bytes except the de-

limiter, carriage return, and line feed. The checksum is the negative of the binary sum of all bytes in the record.

CRLF Carriage return - line feed

B-4. END-OF-FILE (TYPE 01)

Byte 1	Colon (:) delimiter.
2-3	ASCII zeros.
4-5	Most significant byte of the transfer address of the program. This transfer address appears as an argument in the 'END' pseudo-op of a program. It represents the starting execution address of the program.
6-7	Least significant byte of the transfer address.
8-9	Record type 01.
10-11	Checksum.
CRLF	Carriage return line feed

B-5. INTERNAL SYMBOL RECORD (TYPE 02)

Byte 1	Dollar sign ($) delimiter.
2-7	Up to 6 ASCII character of the internal symbol name. The name is left justified, blank filled.
8-9	Record type 02
10-13	Address of the internal symbol, most significant byte first.
14-15	Binary checksum. Note that the ASCII letters of the symbol are converrted to binary before the checksum is calculated. Binary conversion is done without regard to errors.
CRLF	Carriage return, line feed.

B-6. EXTERNAL SYMBOL RECORD (TYPE 03)

Byte 1	Dollar Sign ($) Delimiter.
2-7	Up to 6 ASCII characters of the external symbol name. The name is left justified, blank filled.
8-9	Record type 03.
10-13	Last address which uses the external symbol. This is the start of a link list in the object data records which is described below. The most significant byte is first.
14-15	Binary checksum.
CRLF	Carriage return, line feed.

The ASMB-80 Assembler outputs the external symbol name and the last address in the program where the symbol is used. The data records which follow contain a link list pointing to all occurrences of that symbol in the object code. This is illustrated in Figure B-1.

1. The external symbol record shows the symbol ('LAB') and the last location in the program which uses the symbol (212AH).
2. The object code at 212AH has a pointer which shows where the previous reference to the external symbol occurred (200FH).
3. This backward reference list continues until a terminator ends the list. This terminator is OFFFFH.

B-7. RELOCATING INFORMATION RECORD (TYPE 04). The addresses in the program which must be relocated are explicitly defined in these records. Up to 16 addresses (64 ASCII characters) may be defined in each record.

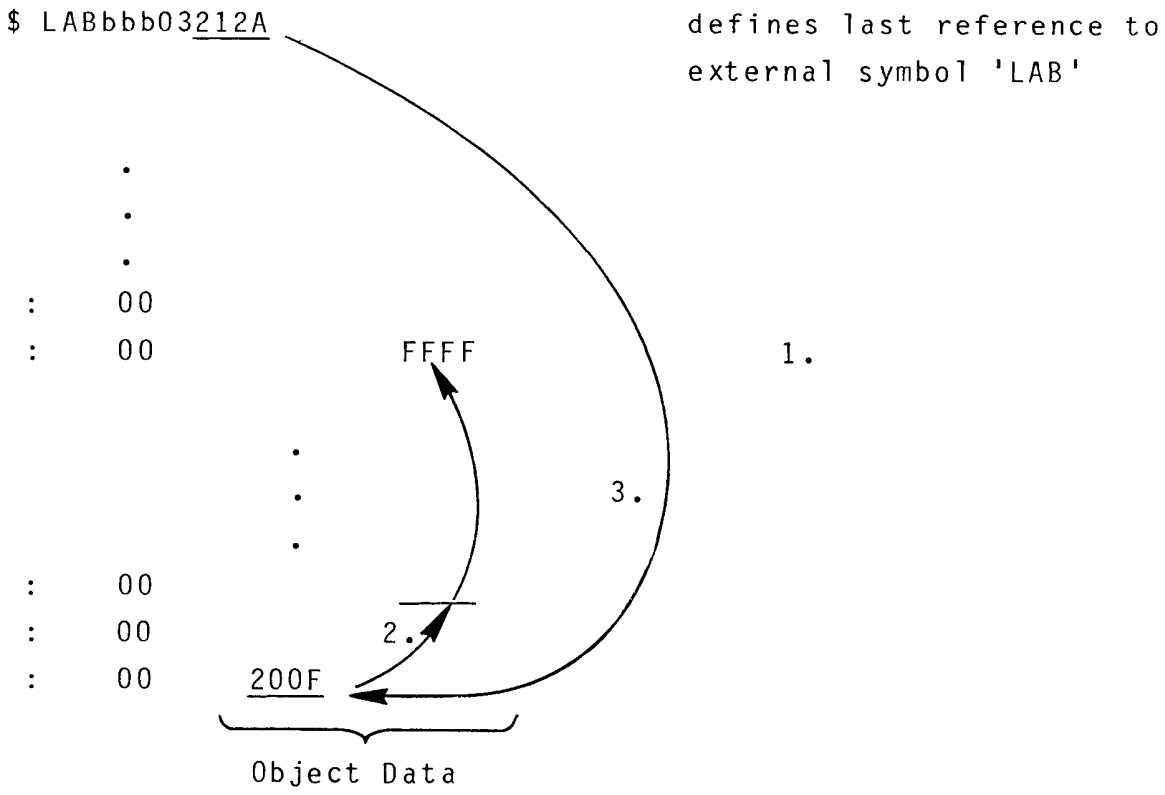
Byte 1	Dollar sign ($) delimiter.
2-3	Number of sets of 2 ASCII characters,

	where 2 sets define an address.
4-7	ASCII zeros.
8-9	Record type 04.
10-	Addresses which must be relocated, most significant byte first.
Last two bytes-	Binary checksum.
CRLF	Carriage return, line feed.

B-8. MODULE DEFINITION RECORD (TYPE 05). This record has the name of the module (defined by the 'NAME' pseudo-op) and a loading information flag byte. The flag byte is determined by the 'PSECT' pseudo-op.

Byte 2	Dollar sign ($) delimiter.
2-7	Name of the module, left justified, blank filled.
8-9	Record type 05.
10-11	Flag byte When converted to binary, the flag byte is defined as follows: BIT 0 - 0 for absolute assemblies 1 for relocatable assemblies BIT 1 - 0 for Z80 assemblies
12-13	Binary checksum.
CRLF	Carriage return, line feed.

Figure B-1. External Symbol Link List



```
 ADDR  OBJECT    ST #

                 0002 ; PSEUDO OPS
                 0003 ;
                 0004  NAME OPCODES
                 0005  ORG  0
                 0006  PSECT REL
                 0007 ;
'0000  AA        0008  DEFB 0AAH
'>0001           0009 L2 DEFL $
 >55AA           0010 L2 DEFL 55AAH
'0001  41424344  0011  DEFM 'ABCD'
                 0012 NN DEFS 2
'0007  BBAA      0013  DEFW 0AABBH
 >AABB           0014 L1 EQU 0AABBH
 >0005           0015 IND EQU 5
 >0020           0016 N EQU 20H
 >0030           0017 DIS EQU 30H
                 0018  GLOBAL NN
                 0019  IF 0
                 0020 ; SHOULD NOT BE ASSEMBLED
                 0021  LD A,B
                 0022  ENDIF
                 0023  IF 1
                 0024 ; SHOULD BE ASSEMBLED
'000A  78        0025  LD A,B
                 0026  ENDIF
                 0027 ; TURN LISTING OFF
                 0032 ; LISTING SHOULD BE ON
                 0033 ;
                 0034 ;
                 0035 ;
                 0036 ; Z80 OPCODES
                 0037 ;
'000C  8E        0038  ADC A,(HL)
'000D  DD8E05    0039  ADC A,(IX+IND)
'0010  FD8E05    0040  ADC A,(IY+IND)
'0013  8F        0041  ADC A,A
'0014  88        0042  ADC A,B
'0015  89        0043  ADC A,C
'0016  8A        0044  ADC A,D
'0017  8B        0045  ADC A,E
'0018  8C        0046  ADC A,H
'0019  8D        0047  ADC A,L
'001A  CE20      0048  ADC A,N
'001C  ED4A      0049  ADC HL,BC
'001E  ED5A      0050  ADC HL,DE
'0020  ED6A      0051  ADC HL,HL
'0022  ED7A      0052  ADC HL,SP
                 0053 ;
'0024  86        0054  ADD A,(HL)
'0025  DD8605    0055  ADD A,(IX+IND)
'0028  FD8605    0056  ADD A,(IY+IND)
'002B  87        0057  ADD A,A
'002C  80        0058  ADD A,B
'002D  81        0059  ADD A,C
'002E  82        0060  ADD A,D
'002F  83        0061  ADD A,E
'0030  84        0062  ADD A,H
'0031  85        0063  ADD A,L
```

```
OPCODE Z80 OPCODE LISTING
   ADDR  OBJECT    ST #

  '0032  C620      0064  ADD A,N
  '0034  09        0065  ADD HL,BC
  '0035  19        0066  ADD HL,DE
  '0036  29        0067  ADD HL,HL
  '0037  39        0068  ADD HL,SP
  '0038  DD09      0069  ADD IX,BC
  '003A  DD19      0070  ADD IX,DE
  '003C  DD29      0071  ADD IX,IX
  '003E  DD39      0072  ADD IX,SP
  '0040  FD09      0073  ADD IY,BC
  '0042  FD19      0074  ADD IY,DE
  '0044  FD29      0075  ADD IY,IY
  '0046  FD39      0076  ADD IY,SP
                   0077 ;
  '0048  A6        0078  AND (HL)
  '0049  DDA605    0079  AND (IX+IND)
  '004C  FDA605    0080  AND (IY+IND)
  '004F  A7        0081  AND A
  '0050  A0        0082  AND B
  '0051  A1        0083  AND C
  '0052  A2        0084  AND D
  '0053  A3        0085  AND E
  '0054  A4        0086  AND H
  '0055  A5        0087  AND L
  '0056  E620      0088  AND N
                   0089 ;
  '0058  CB46      0090  BIT 0,(HL)
  '005A  DDCB0546  0091  BIT 0,(IX+IND)
  '005E  FDCB0546  0092  BIT 0,(IY+IND)
  '0062  CB47      0093  BIT 0,A
  '0064  CB40      0094  BIT 0,B
  '0066  CB41      0095  BIT 0,C
  '0068  CB42      0096  BIT 0,D
  '006A  CB43      0097  BIT 0,E
  '006C  CB44      0098  BIT 0,H
  '006E  CB45      0099  BIT 0,L
                   0100 ;
  '0070  CB4E      0101  BIT 1,(HL)
  '0072  DDCB054E  0102  BIT 1,(IX+IND)
  '0076  FDCB054E  0103  BIT 1,(IY+IND)
  '007A  CB4F      0104  BIT 1,A
  '007C  CB48      0105  BIT 1,B
  '007E  CB49      0106  BIT 1,C
  '0080  CB4A      0107  BIT 1,D
  '0082  CB4B      0108  BIT 1,E
  '0084  CB4C      0109  BIT 1,H
  '0086  CB4D      0110  BIT 1,L
                   0111 ;
  '0088  CB56      0112  BIT 2,(HL)
  '008A  DDCB0556  0113  BIT 2,(IX+IND)
  '008E  FDCB0556  0114  BIT 2,(IY+IND)
  '0092  CB57      0115  BIT 2,A
  '0094  CB50      0116  BIT 2,B
  '0096  CB51      0117  BIT 2,C
  '0098  CB52      0118  BIT 2,D
  '009A  CB53      0119  BIT 2,E
  '009C  CB54      0120  BIT 2,H
  '009E  CB55      0121  BIT 2,L
```

```
  ADDR  OBJECT    ST #

                  0122 ;
'00A0  CB5E       0123  BIT 3,(HL)
'00A2  DDCB0055E  0124  BIT 3,(IX+IND)
'00A6  FDCB0055E  0125  BIT 3,(IY+IND)
'00AA  CB5F       0126  BIT 3,A
'00AC  CB58       0127  BIT 3,B
'00AE  CB59       0128  BIT 3,C
'00B0  CB5A       0129  BIT 3,D
'00B2  CB5B       0130  BIT 3,E
'00B4  CB5C       0131  BIT 3,H
'00B6  CB5D       0132  BIT 3,L
                  0133 ;
'00B8  CB66       0134  BIT 4,(HL)
'00BA  DDCB0566   0135  BIT 4,(IX+IND)
'00BE  FDCB0566   0136  BIT 4,(IY+IND)
'00C2  CB67       0137  BIT 4,A
'00C4  CB60       0138  BIT 4,B
'00C6  CB61       0139  BIT 4,C
'00C8  CB62       0140  BIT 4,D
'00CA  CB63       0141  BIT 4,E
'00CC  CB64       0142  BIT 4,H
'00CE  CB65       0143  BIT 4,L
                  0144 ;
'00D0  CB6E       0145  BIT 5,(HL)
'00D2  DDCB056E   0146  BIT 5,(IX+IND)
'00D6  FDCB056E   0147  BIT 5,(IY+IND)
'00DA  CB6F       0148  BIT 5,A
'00DC  CB68       0149  BIT 5,B
'00DE  CB69       0150  BIT 5,C
'00E0  CB6A       0151  BIT 5,D
'00E2  CB6B       0152  BIT 5,E
'00E4  CB6C       0153  BIT 5,H
'00E6  CB6D       0154  BIT 5,L
                  0155 ;
'00E8  CB76       0156  BIT 6,(HL)
'00EA  DDCB0576   0157  BIT 6,(IX+IND)
'00EE  FDCB0576   0158  BIT 6,(IY+IND)
'00F2  CB77       0159  BIT 6,A
'00F4  CB70       0160  BIT 6,B
'00F6  CB71       0161  BIT 6,C
'00F8  CB72       0162  BIT 6,D
'00FA  CB73       0163  BIT 6,E
'00FC  CB74       0164  BIT 6,H
'00FE  CB75       0165  BIT 6,L
                  0166 ;
'0100  CB7E       0167  BIT 7,(HL)
'0102  DDCB057E   0168  BIT 7,(IX+IND)
'0106  FDCB057E   0169  BIT 7,(IY+IND)
'010A  CB7F       0170  BIT 7,A
'010C  CB78       0171  BIT 7,B
'010E  CB79       0172  BIT 7,C
'0110  CB7A       0173  BIT 7,D
'0112  CB7B       0174  BIT 7,E
'0114  CB7C       0175  BIT 7,H
'0116  CB7D       0176  BIT 7,L
                  0177 ;
'0118  DC0500'    0178  CALL C,NN
'011B  FC0500'    0179  CALL M,NN
```

```
OPCODE Z80 OPCODE LISTING
   ADDR  OBJECT    ST #

  '011E  D40500'   0180  CALL NC,NN
  '0121  CD0500'   0181  CALL NN
  '0124  C40500'   0182  CALL NZ,NN
  '0127  F40500'   0183  CALL P,NN
  '012A  EC0500'   0184  CALL PE,NN
  '012D  E40500'   0185  CALL PO,NN
  '0130  CC0500'   0186  CALL Z,NN
                   0187 ;
  '0133  3F        0188  CCF
                   0189 ;
  '0134  BE        0190  CP (HL)
  '0135  DDBE05    0191  CP (IX+IND)
  '0138  FDBE05    0192  CP (IY+IND)
  '013B  BF        0193  CP A
  '013C  B8        0194  CP B
  '013D  B9        0195  CP C
  '013E  BA        0196  CP D
  '013F  BB        0197  CP E
  '0140  BC        0198  CP H
  '0141  BD        0199  CP L
  '0142  FE20      0200  CP N
                   0201 ;
  '0144  EDA9      0202  CPD
  '0146  EDB9      0203  CPDR
  '0148  EDA1      0204  CPI
  '014A  EDB1      0205  CPIR
                   0206 ;
  '014C  2F        0207  CPL
                   0208 ;
  '014D  27        0209  DAA
                   0210 ;
  '014E  35        0211  DEC (HL)
  '014F  DD3505    0212  DEC (IX+IND)
  '0152  FD3505    0213  DEC (IY+IND)
  '0155  3D        0214  DEC A
  '0156  05        0215  DEC B
  '0157  0B        0216  DEC BC
  '0158  0D        0217  DEC C
  '0159  15        0218  DEC D
  '015A  1B        0219  DEC DE
  '015B  1D        0220  DEC E
  '015C  25        0221  DEC H
  '015D  2B        0222  DEC HL
  '015E  DD2B      0223  DEC IX
  '0160  FD2B      0224  DEC IY
  '0162  2D        0225  DEC L
  '0163  3B        0226  DEC SP
                   0227 ;
  '0164  F3        0228  DI
                   0229 ;
  '0165  102E      0230  DJNZ DIS
                   0231 ;
  '0167  FB        0232  EI
                   0233 ;
  '0168  E3        0234  EX (SP),HL
  '0169  DDE3      0235  EX (SP),IX
  '016B  FDE3      0236  EX (SP),IY
  '016D  08        0237  EX AF,AF'
```

```
  ADDR  OBJECT    ST #

'016E  EB        0238  EX DE,HL
'016F  D9        0239  EXX
                 0240 ;
'0170  76        0241  HALT
                 0242 ;
'0171  ED46      0243  IM 0
'0173  ED56      0244  IM 1
'0175  ED5E      0245  IM 2
                 0246 ;
'0177  ED78      0247  IN A,(C)
'0179  DB20      0248  IN A,(N)
'017B  ED40      0249  IN B,(C)
'017D  ED48      0250  IN C,(C)
'017F  ED50      0251  IN D,(C)
'0181  ED58      0252  IN E,(C)
'0183  ED70      0253  IN F,(C)
'0185  ED60      0254  IN H,(C)
'0187  ED68      0255  IN L,(C)
                 0256 ;
'0189  34        0257  INC (HL)
'018A  FD3405    0258  INC (IY+IND)
'018D  DD3405    0259  INC (IX+IND)
'0190  3C        0260  INC A
'0191  04        0261  INC B
'0192  03        0262  INC BC
'0193  0C        0263  INC C
'0194  14        0264  INC D
'0195  13        0265  INC DE
'0196  1C        0266  INC E
'0197  24        0267  INC H
'0198  23        0268  INC HL
'0199  DD23      0269  INC IX
'019B  FD23      0270  INC IY
'019D  2C        0271  INC L
'019E  33        0272  INC SP
                 0273 ;
'019F  EDAA      0274  IND
'01A1  EDBA      0275  INDR
'01A3  EDA2      0276  INI
'01A5  EDB2      0277  INIR
                 0278 ;
'01A7  E9        0279  JP (HL)
'01A8  DDE9      0280  JP (IX)
'01AA  FDE9      0281  JP (IY)
'01AC  DA0500'   0282  JP C,NN
'01AF  FA0500'   0283  JP M,NN
'01B2  D20500'   0284  JP NC,NN
'01B5  C30500'   0285  JP NN
'01B8  C20500'   0286  JP NZ,NN
'01BB  F20500'   0287  JP P,NN
'01BE  EA0500'   0288  JP PE,NN
'01C1  E20500'   0289  JP PO,NN
'01C4  CA0500'   0290  JP Z,NN
                 0291 ;
'01C7  382E      0292  JR C,DIS
'01C9  182E      0293  JR DIS
'01CB  302E      0294  JR NC,DIS
'01CD  202E      0295  JR NZ,DIS
```

```
OPCODE Z80 OPCODE LISTING
   ADDR  OBJECT    ST #

  '01CF  282E      0296  JR Z,DIS
                   0297 ;
  '01D1  02        0298  LD (BC),A
  '01D2  12        0299  LD (DE),A
  '01D3  77        0300  LD (HL),A
  '01D4  70        0301  LD (HL),B
  '01D5  71        0302  LD (HL),C
  '01D6  72        0303  LD (HL),D
  '01D7  73        0304  LD (HL),E
  '01D8  74        0305  LD (HL),H
  '01D9  75        0306  LD (HL),L
  '01DA  3620      0307  LD (HL),N
                   0308 ;
  '01DC  DD7705    0309  LD (IX+IND),A
  '01DF  DD7005    0310  LD (IX+IND),B
  '01E2  DD7105    0311  LD (IX+IND),C
  '01E5  DD7205    0312  LD (IX+IND),D
  '01E8  DD7305    0313  LD (IX+IND),E
  '01EB  DD7405    0314  LD (IX+IND),H
  '01EE  DD7505    0315  LD (IX+IND),L
  '01F1  DD360520  0316  LD (IX+IND),N
                   0317 ;
  '01F5  FD7705    0318  LD (IY+IND),A
  '01F8  FD7005    0319  LD (IY+IND),B
  '01FB  FD7105    0320  LD (IY+IND),C
  '01FE  FD7205    0321  LD (IY+IND),D
  '0201  FD7305    0322  LD (IY+IND),E
  '0204  FD7405    0323  LD (IY+IND),H
  '0207  FD7505    0324  LD (IY+IND),L
  '020A  FD360520  0325  LD (IY+IND),N
                   0326 ;
  '020E  320500'   0327  LD (NN),A
  '0211  ED430500' 0328  LD (NN),BC
  '0215  ED530500' 0329  LD (NN),DE
  '0219  220500'   0330  LD (NN),HL
  '021C  DD220500' 0331  LD (NN),IX
  '0220  FD220500' 0332  LD (NN),IY
  '0224  ED730500' 0333  LD (NN),SP
                   0334 ;
  '0228  0A        0335  LD A,(BC)
  '0229  1A        0336  LD A,(DE)
  '022A  7E        0337  LD A,(HL)
  '022B  DD7E05    0338  LD A,(IX+IND)
  '022E  FD7E05    0339  LD A,(IY+IND)
  '0231  3A0500'   0340  LD A,(NN)
  '0234  7F        0341  LD A,A
  '0235  78        0342  LD A,B
  '0236  79        0343  LD A,C
  '0237  7A        0344  LD A,D
  '0238  7B        0345  LD A,E
  '0239  7C        0346  LD A,H
  '023A  ED57      0347  LD A,I
  '023C  7D        0348  LD A,L
  '023D  3E20      0349  LD A,N
  '023F  ED5F      0350  LD A,R
                   0351 ;
  '0241  46        0352  LD B,(HL)
  '0242  DD4605    0353  LD B,(IX+IND)
```

```
  ADDR   OBJECT     ST #

'0245   FD4605     0354   LD B,(IY+IND)
'0248   47         0355   LD B,A
'0249   40         0356   LD B,B
'024A   41         0357   LD B,C
'024B   42         0358   LD B,D
'024C   43         0359   LD B,E
'024D   44         0360   LD B,H
'024E   45         0361   LD B,L
'024F   0620       0362   LD B,N
                   0363 ;
'0251   ED4B0500'  0364   LD BC,(NN)
'0255   010500'    0365   LD BC,NN
                   0366 ;
'0258   4E         0367   LD C,(HL)
'0259   DD4E05     0368   LD C,(IX+IND)
'025C   FD4E05     0369   LD C,(IY+IND)
'025F   4F         0370   LD C,A
'0260   48         0371   LD C,B
'0261   49         0372   LD C,C
'0262   4A         0373   LD C,D
'0263   4B         0374   LD C,E
'0264   4C         0375   LD C,H
'0265   4D         0376   LD C,L
'0266   0E20       0377   LD C,N
                   0378 ;
'0268   56         0379   LD D,(HL)
'0269   DD5605     0380   LD D,(IX+IND)
'026C   FD5605     0381   LD D,(IY+IND)
'026F   57         0382   LD D,A
'0270   50         0383   LD D,B
'0271   51         0384   LD D,C
'0272   52         0385   LD D,D
'0273   53         0386   LD D,E
'0274   54         0387   LD D,H
'0275   55         0388   LD D,L
'0276   1620       0389   LD D,N
                   0390 ;
'0278   ED5B0500'  0391   LD DE,(NN)
'027C   110500'    0392   LD DE,NN
                   0393 ;
'027F   5E         0394   LD E,(HL)
'0280   DD5E05     0395   LD E,(IX+IND)
'0283   FD5E05     0396   LD E,(IY+IND)
'0286   5F         0397   LD E,A
'0287   58         0398   LD E,B
'0288   59         0399   LD E,C
'0289   5A         0400   LD E,D
'028A   5B         0401   LD E,E
'028B   5C         0402   LD E,H
'028C   5D         0403   LD E,L
'028D   1E20       0404   LD E,N
                   0405 ;
'028F   66         0406   LD H,(HL)
'0290   DD6605     0407   LD H,(IX+IND)
'0293   FD6605     0408   LD H,(IY+IND)
'0296   67         0409   LD H,A
'0297   60         0410   LD H,B
'0298   61         0411   LD H,C
```

```
OPCODE Z80 OPCODE LISTING
   ADDR  OBJECT    ST #

  '0299  62        0412  LD H,D
  '029A  63        0413  LD H,E
  '029B  64        0414  LD H,H
  '029C  65        0415  LD H,L
  '029D  2620      0416  LD H,N
                   0417 ;
  '029F  2A0500'   0418  LD HL,(NN)
  '02A2  210500'   0419  LD HL,NN
                   0420 ;
  '02A5  ED47      0421  LD I,A
                   0422 ;
  '02A7  DD2A0500' 0423  LD IX,(NN)
  '02AB  DD210500' 0424  LD IX,NN
                   0425 ;
  '02AF  FD2A0500' 0426  LD IY,(NN)
  '02B3  FD210500' 0427  LD IY,NN
                   0428 ;
  '02B7  6E        0429  LD L,(HL)
  '02B8  DD6E05    0430  LD L,(IX+IND)
  '02BB  FD6E05    0431  LD L,(IY+IND)
  '02BE  6F        0432  LD L,A
  '02BF  68        0433  LD L,B
  '02C0  69        0434  LD L,C
  '02C1  6A        0435  LD L,D
  '02C2  6B        0436  LD L,E
  '02C3  6C        0437  LD L,H
  '02C4  6D        0438  LD L,L
  '02C5  2E20      0439  LD L,N
                   0440 ;
  '02C7  ED4F      0441  LD R,A
                   0442 ;
  '02C9  ED730500' 0443  LD SP,(NN)
  '02CD  F9        0444  LD SP,HL
  '02CE  DDF9      0445  LD SP,IX
  '02D0  FDF9      0446  LD SP,IY
  '02D2  310500'   0447  LD SP,NN
                   0448 ;
  '02D5  EDA8      0449  LDD
  '02D7  EDB8      0450  LDDR
  '02D9  EDA0      0451  LDI
  '02DB  EDB0      0452  LDIR
                   0453 ;
  '02DD  ED44      0454  NEG
                   0455 ;
  '02DF  00        0456  NOP
                   0457 ;
  '02E0  B6        0458  OR (HL)
  '02E1  DDB605    0459  OR (IX+IND)
  '02E4  FDB605    0460  OR (IY+IND)
  '02E7  B7        0461  OR A
  '02E8  B0        0462  OR B
  '02E9  B1        0463  OR C
  '02EA  B2        0464  OR D
  '02EB  B3        0465  OR E
  '02EC  B4        0466  OR H
  '02ED  B5        0467  OR L
  '02EE  F620      0468  OR N
                   0469 ;
```

```
   ADDR  OBJECT    ST #

  '02F0  EDBB      0470  OTDR
  '02F2  EDB3      0471  OTIR
                   0472 ;
  '02F4  ED79      0473  OUT (C),A
  '02F6  ED41      0474  OUT (C),B
  '02F8  ED49      0475  OUT (C),C
  '02FA  ED51      0476  OUT (C),D
  '02FC  ED59      0477  OUT (C),E
  '02FE  ED61      0478  OUT (C),H
  '0300  ED69      0479  OUT (C),L
  '0302  D320      0480  OUT (N),A
                   0481 ;
  '0304  EDAB      0482  OUTD
  '0306  EDA3      0483  OUTI
                   0484 ;
  '0308  F1        0485  POP AF
  '0309  C1        0486  POP BC
  '030A  D1        0487  POP DE
  '030B  E1        0488  POP HL
  '030C  DDE1      0489  POP IX
  '030E  FDE1      0490  POP IY
  '0310  F5        0491  PUSH AF
  '0311  C5        0492  PUSH BC
  '0312  D5        0493  PUSH DE
  '0313  E5        0494  PUSH HL
  '0314  DDE5      0495  PUSH IX
  '0316  FDE5      0496  PUSH IY
                   0497 ;
  '0318  CB86      0498  RES 0,(HL)
  '031A  DDCB0586  0499  RES 0,(IX+IND)
  '031E  FDCB0586  0500  RES 0,(IY+IND)
  '0322  CB87      0501  RES 0,A
  '0324  CB80      0502  RES 0,B
  '0326  CB81      0503  RES 0,C
  '0328  CB82      0504  RES 0,D
  '032A  CB83      0505  RES 0,E
  '032C  CB84      0506  RES 0,H
  '032E  CB85      0507  RES 0,L
                   0508 ;
  '0330  CB8E      0509  RES 1,(HL)
  '0332  DDCB058E  0510  RES 1,(IX+IND)
  '0336  FDCB058E  0511  RES 1,(IY+IND)
  '033A  CB8F      0512  RES 1,A
  '033C  CB88      0513  RES 1,B
  '033E  CB89      0514  RES 1,C
  '0340  CB8A      0515  RES 1,D
  '0342  CB8B      0516  RES 1,E
  '0344  CB8C      0517  RES 1,H
  '0346  CB8D      0518  RES 1,L
                   0519 ;
  '0348  CB96      0520  RES 2,(HL)
  '034A  DDCB0596  0521  RES 2,(IX+IND)
  '034E  FDCB0596  0522  RES 2,(IY+IND)
  '0352  CB97      0523  RES 2,A
  '0354  CB90      0524  RES 2,B
  '0356  CB91      0525  RES 2,C
  '0358  CB92      0526  RES 2,D
  '035A  CB93      0527  RES 2,E
```

OPCODE Z80 OPCODE LISTING

```
   ADDR  OBJECT    ST #

  '035C  CB94      0528  RES 2,H
  '035E  CB95      0529  RES 2,L
                   0530 ;
  '0360  CB9E      0531  RES 3,(HL)
  '0362  DDCB059E  0532  RES 3,(IX+IND)
  '0366  FDCB059E  0533  RES 3,(IY+IND)
  '036A  CB9F      0534  RES 3,A
  '036C  CB98      0535  RES 3,B
  '036E  CB99      0536  RES 3,C
  '0370  CB9A      0537  RES 3,D
  '0372  CB9B      0538  RES 3,E
  '0374  CB9C      0539  RES 3,H
  '0376  CB9D      0540  RES 3,L
                   0541 ;
  '0378  CBA6      0542  RES 4,(HL)
  '037A  DDCB05A6  0543  RES 4,(IX+IND)
  '037E  FDCB05A6  0544  RES 4,(IY+IND)
  '0382  CBA7      0545  RES 4,A
  '0384  CBA0      0546  RES 4,B
  '0386  CBA1      0547  RES 4,C
  '0388  CBA2      0548  RES 4,D
  '038A  CBA3      0549  RES 4,E
  '038C  CBA4      0550  RES 4,H
  '038E  CBA5      0551  RES 4,L
                   0552 ;
  '0390  CBAE      0553  RES 5,(HL)
  '0392  DDCB05AE  0554  RES 5,(IX+IND)
  '0396  FDCB05AE  0555  RES 5,(IY+IND)
  '039A  CBAF      0556  RES 5,A
  '039C  CBA8      0557  RES 5,B
  '039E  CBA9      0558  RES 5,C
  '03A0  CBAA      0559  RES 5,D
  '03A2  CBAB      0560  RES 5,E
  '03A4  CBAC      0561  RES 5,H
  '03A6  CBAD      0562  RES 5,L
                   0563 ;
  '03A8  CBB6      0564  RES 6,(HL)
  '03AA  DDCB05B6  0565  RES 6,(IX+IND)
  '03AE  FDCB05B6  0566  RES 6,(IY+IND)
  '03B2  CBB7      0567  RES 6,A
  '03B4  CBB0      0568  RES 6,B
  '03B6  CBB1      0569  RES 6,C
  '03B8  CBB2      0570  RES 6,D
  '03BA  CBB3      0571  RES 6,E
  '03BC  CBB4      0572  RES 6,H
  '03BE  CBB5      0573  RES 6,L
                   0574 ;
  '03C0  CBBE      0575  RES 7,(HL)
  '03C2  DDCB05BE  0576  RES 7,(IX+IND)
  '03C6  FDCB05BE  0577  RES 7,(IY+IND)
  '03CA  CBBF      0578  RES 7,A
  '03CC  CBB8      0579  RES 7,B
  '03CE  CBB9      0580  RES 7,C
  '03D0  CBBA      0581  RES 7,D
  '03D2  CBBB      0582  RES 7,E
  '03D4  CBBC      0583  RES 7,H
  '03D6  CBBD      0584  RES 7,L
                   0585 ;
```

```
   ADDR  OBJECT    ST #

  '03D8  C9        0586  RET
  '03D9  D8        0587  RET C
  '03DA  F8        0588  RET M
  '03DB  D0        0589  RET NC
  '03DC  C0        0590  RET NZ
  '03DD  F0        0591  RET P
  '03DE  E8        0592  RET PE
  '03DF  E0        0593  RET PO
  '03E0  C8        0594  RET Z
                   0595 ;
  '03E1  ED4D      0596  RETI
  '03E3  ED45      0597  RETN
                   0598 ;
  '03E5  CB16      0599  RL (HL)
  '03E7  DDCB0516  0600  RL (IX+IND)
  '03EB  FDCB0516  0601  RL (IY+IND)
  '03EF  CB17      0602  RL A
  '03F1  CB10      0603  RL B
  '03F3  CB11      0604  RL C
  '03F5  CB12      0605  RL D
  '03F7  CB13      0606  RL E
  '03F9  CB14      0607  RL H
  '03FB  CB15      0608  RL L
                   0609 ;
  '03FD  17        0610  RLA
                   0611 ;
  '03FE  CB06      0612  RLC (HL)
  '0400  DDCB0506  0613  RLC (IX+IND)
  '0404  FDCB0506  0614  RLC (IY+IND)
  '0408  CB07      0615  RLC A
  '040A  CB00      0616  RLC B
  '040C  CB01      0617  RLC C
  '040E  CB02      0618  RLC D
  '0410  CB03      0619  RLC E
  '0412  CB04      0620  RLC H
  '0414  CB05      0621  RLC L
                   0622 ;
  '0416  07        0623  RLCA
                   0624 ;
  '0417  ED6F      0625  RLD
                   0626 ;
  '0419  CB1E      0627  RR (HL)
  '041B  DDCB051E  0628  RR (IX+IND)
  '041F  FDCB051E  0629  RR (IY+IND)
  '0423  CB1F      0630  RR A
  '0425  CB18      0631  RR B
  '0427  CB19      0632  RR C
  '0429  CB1A      0633  RR D
  '042B  CB1B      0634  RR E
  '042D  CB1C      0635  RR H
  '042F  CB1D      0636  RR L
                   0637 ;
  '0431  1F        0638  RRA
                   0639 ;
  '0432  CB0E      0640  RRC (HL)
  '0434  DDCB050E  0641  RRC (IX+IND)
  '0438  FDCB050E  0642  RRC (IY+IND)
  '043C  CB0F      0643  RRC A
```

```
OPCODE Z80 OPCODE LISTING
   ADDR  OBJECT    ST #

   '043E  CB08      0644  RRC B
   '0440  CB09      0645  RRC C
   '0442  CB0A      0646  RRC D
   '0444  CB0B      0647  RRC E
   '0446  CB0C      0648  RRC H
   '0448  CB0D      0649  RRC L
                    0650 ;
   '044A  0F        0651  RRCA
                    0652 ;
   '044B  ED67      0653  RRD
                    0654 ;
   '044D  C7        0655  RST 0
   '044E  CF        0656  RST 08H
   '044F  D7        0657  RST 10H
   '0450  DF        0658  RST 18H
   '0451  E7        0659  RST 20H
   '0452  EF        0660  RST 28H
   '0453  F7        0661  RST 30H
   '0454  FF        0662  RST 38H
                    0663 ;
   '0455  9E        0664  SBC A,(HL)
   '0456  DD9E05    0665  SBC A,(IX+IND)
   '0459  FD9E05    0666  SBC A,(IY+IND)
   '045C  9F        0667  SBC A,A
   '045D  98        0668  SBC A,B
   '045E  99        0669  SBC A,C
   '045F  9A        0670  SBC A,D
   '0460  9B        0671  SBC A,E
   '0461  9C        0672  SBC A,H
   '0462  9D        0673  SBC A,L
   '0463  DE20      0674  SBC A,N
                    0675 ;
   '0465  ED42      0676  SBC HL,BC
   '0467  ED52      0677  SBC HL,DE
   '0469  ED62      0678  SBC HL,HL
   '046B  ED72      0679  SBC HL,SP
                    0680 ;
   '046D  37        0681  SCF
                    0682 ;
   '046E  CBC6      0683  SET 0,(HL)
   '0470  DDCB05C6  0684  SET 0,(IX+IND)
   '0474  FDCB05C6  0685  SET 0,(IY+IND)
   '0478  CBC7      0686  SET 0,A
   '047A  CBC0      0687  SET 0,B
   '047C  CBC1      0688  SET 0,C
   '047E  CBC2      0689  SET 0,D
   '0480  CBC3      0690  SET 0,E
   '0482  CBC4      0691  SET 0,H
   '0484  CBC5      0692  SET 0,L
                    0693 ;
   '0486  CBCE      0694  SET 1,(HL)
   '0488  DDCB05CE  0695  SET 1,(IX+IND)
   '048C  FDCB05CE  0696  SET 1,(IY+IND)
   '0490  CBCF      0697  SET 1,A
   '0492  CBC8      0698  SET 1,B
   '0494  CBC9      0699  SET 1,C
   '0496  CBCA      0700  SET 1,D
   '0498  CBCB      0701  SET 1,E
```

```
   ADDR  OBJECT    ST #

  '049A  CBCC      0702  SET 1,H
  '049C  CBCD      0703  SET 1,L
                   0704 ;
  '049E  CBD6      0705  SET 2,(HL)
  '04A0  DDCB05D6  0706  SET 2,(IX+IND)
  '04A4  FDCB05D6  0707  SET 2,(IY+IND)
  '04A8  CBD7      0708  SET 2,A
  '04AA  CBD0      0709  SET 2,B
  '04AC  CBD1      0710  SET 2,C
  '04AE  CBD2      0711  SET 2,D
  '04B0  CBD3      0712  SET 2,E
  '04B2  CBD4      0713  SET 2,H
  '04B4  CBD5      0714  SET 2,L
                   0715 ;
  '04B6  CBDE      0716  SET 3,(HL)
  '04B8  DDCB05DE  0717  SET 3,(IX+IND)
  '04BC  FDCB05DE  0718  SET 3,(IY+IND)
  '04C0  CBDF      0719  SET 3,A
  '04C2  CBD8      0720  SET 3,B
  '04C4  CBD9      0721  SET 3,C
  '04C6  CBDA      0722  SET 3,D
  '04C8  CBDB      0723  SET 3,E
  '04CA  CBDC      0724  SET 3,H
  '04CC  CBDD      0725  SET 3,L
                   0726 ;
  '04CE  CBE6      0727  SET 4,(HL)
  '04D0  DDCB05E6  0728  SET 4,(IX+IND)
  '04D4  FDCB05E6  0729  SET 4,(IY+IND)
  '04D8  CBE7      0730  SET 4,A
  '04DA  CBE0      0731  SET 4,B
  '04DC  CBE1      0732  SET 4,C
  '04DE  CBE2      0733  SET 4,D
  '04E0  CBE3      0734  SET 4,E
  '04E2  CBE4      0735  SET 4,H
  '04E4  CBE5      0736  SET 4,L
                   0737 ;
  '04E6  CBEE      0738  SET 5,(HL)
  '04E8  DDCB05EE  0739  SET 5,(IX+IND)
  '04EC  FDCB05EE  0740  SET 5,(IY+IND)
  '04F0  CBEF      0741  SET 5,A
  '04F2  CBE8      0742  SET 5,B
  '04F4  CBE9      0743  SET 5,C
  '04F6  CBEA      0744  SET 5,D
  '04F8  CBEB      0745  SET 5,E
  '04FA  CBEC      0746  SET 5,H
  '04FC  CBED      0747  SET 5,L
                   0748 ;
  '04FE  CBF6      0749  SET 6,(HL)
  '0500  DDCB05F6  0750  SET 6,(IX+IND)
  '0504  FDCB05F6  0751  SET 6,(IY+IND)
  '0508  CBF7      0752  SET 6,A
  '050A  CBF0      0753  SET 6,B
  '050C  CBF1      0754  SET 6,C
  '050E  CBF2      0755  SET 6,D
  '0510  CBF3      0756  SET 6,E
  '0512  CBF4      0757  SET 6,H
  '0514  CBF5      0758  SET 6,L
                   0759 ;
```

```
OPCODE Z80 OPCODE LISTING
   ADDR  OBJECT    ST #

  '0516  CBFE      0760  SET 7,(HL)
  '0518  DDCB05FE  0761  SET 7,(IX+IND)
  '051C  FDCB05FE  0762  SET 7,(IY+IND)
  '0520  CBFF      0763  SET 7,A
  '0522  CBF8      0764  SET 7,B
  '0524  CBF9      0765  SET 7,C
  '0526  CBFA      0766  SET 7,D
  '0528  CBFB      0767  SET 7,E
  '052A  CBFC      0768  SET 7,H
  '052C  CBFD      0769  SET 7,L
                   0770 ;
  '052E  CB26      0771  SLA (HL)
  '0530  DDCB0526  0772  SLA (IX+IND)
  '0534  FDCB0526  0773  SLA (IY+IND)
  '0538  CB27      0774  SLA A
  '053A  CB20      0775  SLA B
  '053C  CB21      0776  SLA C
  '053E  CB22      0777  SLA D
  '0540  CB23      0778  SLA E
  '0542  CB24      0779  SLA H
  '0544  CB25      0780  SLA L
                   0781 ;
  '0546  CB2E      0782  SRA (HL)
  '0548  DDCB052E  0783  SRA (IX+IND)
  '054C  FDCB052E  0784  SRA (IY+IND)
  '0550  CB2F      0785  SRA A
  '0552  CB28      0786  SRA B
  '0554  CB29      0787  SRA C
  '0556  CB2A      0788  SRA D
  '0558  CB2B      0789  SRA E
  '055A  CB2C      0790  SRA H
  '055C  CB2D      0791  SRA L
                   0792 ;
  '055E  CB3E      0793  SRL (HL)
  '0560  DDCB053E  0794  SRL (IX+IND)
  '0564  FDCB053E  0795  SRL (IY+IND)
  '0568  CB3F      0796  SRL A
  '056A  CB38      0797  SRL B
  '056C  CB39      0798  SRL C
  '056E  CB3A      0799  SRL D
  '0570  CB3B      0800  SRL E
  '0572  CB3C      0801  SRL H
  '0574  CB3D      0802  SRL L
                   0803 ;
  '0576  96        0804  SUB (HL)
  '0577  DD9605    0805  SUB (IX+IND)
  '057A  FD9605    0806  SUB (IY+IND)
  '057D  97        0807  SUB A
  '057E  90        0808  SUB B
  '057F  91        0809  SUB C
  '0580  92        0810  SUB D
  '0581  93        0811  SUB E
  '0582  94        0812  SUB H
  '0583  95        0813  SUB L
  '0584  D620      0814  SUB N
                   0815 ;
  '0586  AE        0816  XOR (HL)
  '0587  DDAE05    0817  XOR (IX+IND)
```

```
   ADDR  OBJECT    ST #

  '058A  FDAE05    0818  XOR (IY+IND)
  '058D  AF        0819  XOR A
  '058E  A8        0820  XOR B
  '058F  A9        0821  XOR C
  '0590  AA        0822  XOR D
  '0591  AB        0823  XOR E
  '0592  AC        0824  XOR H
  '0593  AD        0825  XOR L
  '0594  EE20      0826  XOR N
                   0827 ;
                   0828  END

ERRORS=0000
```

MOSTEK®
Z80·F8
3870
Covering the full spectrum of microcomputer applications.

1215 W. Crosby Rd. • Carrollton, Texas 75006 • 214/242-0444
In Europe, Contact: MOSTEK Brussels
150 Chaussee de la Hulpe, B1170, Belgium;
Telephone: (32) 02/660-2568/4713

PRINTED IN USA April 1979
Publication No. MK78522