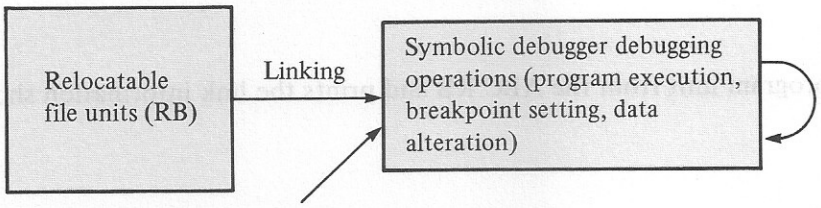
— CONTENTS —

# INTRODUCTION

The SHARP MZ-80 series symbolic debugger links and loads one or more program units from relocatable files to form an object program in memory in an immediately executable form and runs the object program for debugging. It provides the programmer with facilities for taking a memory dump of the object program in the link area, for setting a breakpoint in the program, for displaying and altering the contents of the CPU internal registers and for starting execution of the program at a given address with the CPU internal registers set to specified values (indicative start).

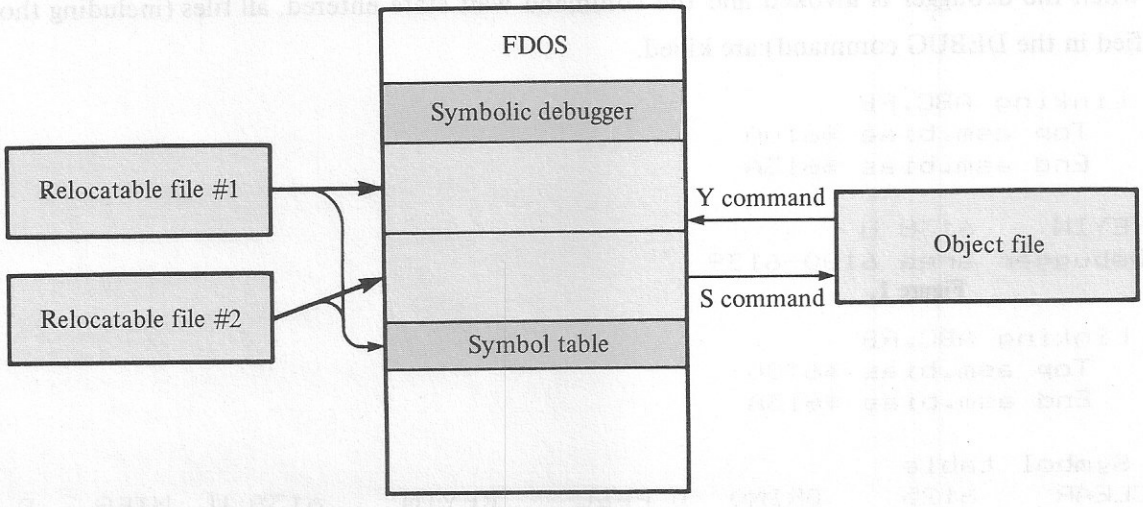



**Debugging with the symbolic debugger**

The debugger is said to be "symbolic" since it permits the programmer to reference addresses (e.g., breakpoints) during debugging not only in absolute hexadecimal representation but with global symbols declared as entry symbols in the source program with the ENT assembler directive. This releases the programmer from the burden of remembering relative addresses in relocatable programs and offset values specified when they are loaded.

In normal program development process, the programmer debugs each object program unit with the symbolic debugger and, if he finds errors, he reedits its source program and reassembles it. After debugging all object program units, the programmer links and loads them with the linker to form the final object program.

Symbolic debugger commands are summarized in the table on page 3. Commands marked with a dagger permit symbolic operations. The debugger creates the symbol table in the same way as the linker.




**Symbolic debugger file processing**

## —Starting the Symbolic Debugger—

The symbolic debugger is started by entering one of the commands below in the FDOS command mode.

1. **DEBUG** [CR]

The debugger is invoked and the debugger command wait state entered.

2. **DEBUG** [filename 1, ..... filename N] [CR]

The debugger links and loads program units from relocatable files filename 1 through filename N and waits for entry of a debugger command.

3. **DEBUG/P** ABC [CR]

The debugger loads the program unit from file ABC.RB and prints the link information shown in Figure 1 on the printer.

4. **DEBUG/P/T** ABC [CR]

The debugger loads the program unit from file ABC.RB and prints the link and symbol table information on the printer.

5. **DEBUG** ABC, XYZ, TBL$20 [CR]

The debugger links and loads program units from relocatable files ABC.RB and XYZ.RB and waits for entry of a debugger command. It also reserves 2000 (hex) bytes (approximately 8K bytes) of space for the symbol table. Approximately 6K bytes of space are reserved when the table size is not specified.

6. **DEBUG** ABC, $1000, XYZ, DEF/O [CR]

The debugger links and loads program units from relocatable files ABC.RB and XYZ.RB to generate an object program in object program file DEF.OBJ, then waits for entry of a debugger command. It reserves 4K bytes of free space (offset of 1000 (hex)) between program units ABC and XYZ.

Note: When the debugger is invoked and the command wait state entered, all files (including those specified in the DEBUG command) are killed.

```
 Linking ABC.RB
   Top asm.bias $6100
   End asm.bias $613A

KEYIN     6138 U
Debugger area 6100-6139
```

**Figure 1.**

```
 Linking ABC.RB
   Top asm.bias $6100
   End asm.bias $613A

  Symbol table
CLEAR     6125     DRINO   D FFEC     KEYIN     6138 U  MTFG    D FFE6
START     6100
Debugger area 6100-6139
```

**Figure 2.**

# SYMBOLIC DEBUGGER COMMAND TABLE

| Command type | Command name | Function |
|---|---|---|
| Symbol table command | T | Displays the contents of the symbol table; i.e., the label symbol name, its absolute address and the definition status for each table entry. (Table Dump) |
| Debugging commands | B† | Displays, sets or alters a breakpoint. (Breakpoint) |
| | & | Clears all breakpoints set. (Clear Breakpoints) |
| | M† | Displays the contents of the specified block in the link area in hexadecimal representation or alters them. (Memory Dump) |
| | D† | Displays the contents of the specified block in the link area in hexadecimal representation with one instruction on a line. (Memory List Dump) |
| | W† | Writes hexadecimal data, starting at the specified address in the link area. (Write) |
| | G† | Executes the program at the specified address. (GOTO) |
| | I | Executes the program at the address designated by PC with the register buffer data set to the CPU internal reigsters. (Indicative Start) |
| | A | Displays the contents of registers A, F, B, C, D, E, H and L in hexadecimal representation or alters them. (Accumulator) |
| | C | Displays the contents of complementary registers A′, F′, B′, C′, D′, E′, H′ and L′ in hexadecimal representation or alters them. (Complementary) |
| | P | Displays the contents of registers PC, SP, IX, IY and I in hexadecimal representation or alters them. (Program Counter) |
| | R | Displays the contents of all registers in hexadecimal representation. (Register) |
| | X | Transfers the specified memory block to the specified address. (Transfer) |
| File I/O commands | S | Saves the object program in the link area in an output file with the specified name. (Save) |
| | Y | Reads the object program from the object file with the specified file name into memory. (Yank) |
| Special commands | \ | Executes the specified FDOS built-in command. |
| | # | Switches the printer list mode for listing printout. |
| | ! | Transfers control to FDOS. |

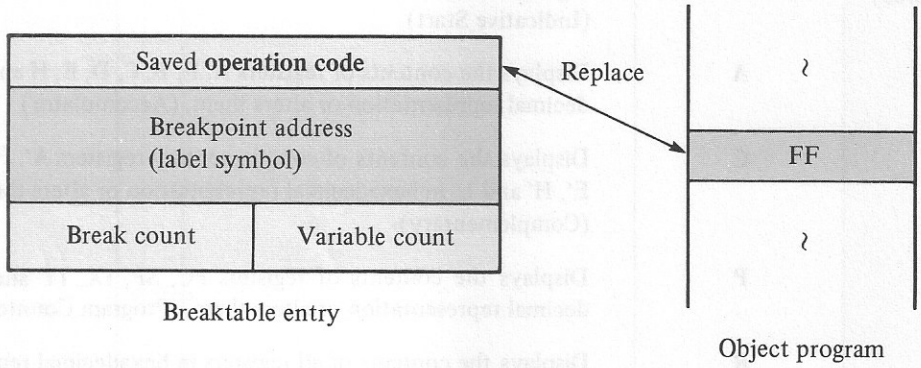
Note: Commands marked by a dagger permit symbolic operations.

# BREAKPOINTS

A breakpoint is a checkpoint set up in the program at which program execution is stopped and the contents of the CPU registers are saved into the register buffer. At this point, the programmer can examine and alter the memory and register contents. He can also restart the program at this point. Thus, breakpoints facilitate program checking and debugging.

The symbolic debugger allows a maximum of nine breakpoints. When setting a breakpoint, the programmer must specify not only its address but also its count. The count specifies the number of allowable passes through the breakpoint in a looping program before a break actually occurs. The maximum allowable value of the break count is E in hexadecimal (14 in decimal).

When a breakpoint is set in a program, the debugger saves the operation code at that location (address) in the break table and replaces it with code FFH. The debugger creates one breaktable entry for each breakpoint as shown below.




Hexadecimal code FF is the operation code for RST 7, which initiates a break operation. When the RST 7 instruction, which is a 1-byte CALL instruction, is executed, the contents of the program counter are pushed into the stack and the program counter is loaded with new data 0038H; that is, program control jumps to address 0038H in the monitor, from which point control is immediately passed to the debugger. The debugger searches the breaktable for the pertinent breakpoint. If the breakpoint is not found, the debugger displays error message "RST7?." Thus, the RST 7 instruction is used in the system and cannot be used by user programs.

When the debugger finds the required breakpoint in the table, it checks the corresponding count and decrements the variable count (this count is initially set to the break count) by one. If the variable count reaches zero, the debugger performs break processing; otherwise, it continues program execution.

## —T (Table dump) Command—

The T command displays the contents of the symbol table, that is, the label symbol name, its absolute address and its definition status.

| | |
|---|---|
| ＊ **DT** | Displays the contents of the symbol table. |

— Enter a **T** command in response to the prompt "＊D".

— The debugger displays the label symbol name, its absolute address (in hexadecimal) and the definition status for each symbol table entry. The programmer can detect symbol definition errors by checking the definition status of the displayed label symbols.

— Messages pertaining to the symbol table definition status are identical to those issued by the linker. The definition status messages are listed below, followed by examples.

— Two symbol table entries are displayed on a line.

| Message | Definition status |
|---|---|
| U | Undefined symbol (address or data) |
| M | Multi-defined symbol (address or data) |
| X | Cross-defined symbol (address or data) |
| H | Half-defined symbol (data) |
| D | EQU-defined symbol (data) |

No message is attached to symbols for which an address has been defined.

U, M, X and H indicate error conditions.

## Link message examples

**First program unit loaded (UNIT–#1)**

```
TMDLYH:   LD      HL, START
COUNT:    ENT
          DEC     HL
          LD      A, H
          CP      COUNT0
          JR      NZ, COUNT
          LD      A, L
          CP      COUNT1
          JR      NZ, COUNT
          CP      COUNT2
          JR      NZ, COUNT
          RET
PEND:     ENT
          DEFM    'TMDLYH'
          DEFB    0DH
COUNT1:   EQU     00H
COUNT0:   EQU     50H
          END
```

"START" X

START is not defined as an address in the first program, but is defined as data in the second or subsequent program with the START : EQU statement.

Note: The EQU statement should be placed at the beginning of the program unit.

"COUNT2" H

COUNT2 is not defined as data in the first program, but is defined as data in the third program with the COUNT2 : EQU statement.

"COUNT1" D

COUNT1 is defined as data (D indicates no error condition).

**Second program unit loaded (UNIT–#2)**

```
TMDLYL:   LD      HL, START
LOOP1:    DEC     H
          LD      A, H
          CP      COUNT
          JR      NZ, LOOP1
          RET
PEND:     ENT
          DEFM    'TMDLYL'
          DEFB    0DH
START:    EQU     1000H
COUNT:    EQU     00H


          END
```

"COUNT" X

COUNT is defined as an address in the first program while it is simultaneously defined as data in the second program.

"PEND" M

PEND is defined as an address in the first program while it is simultaneously defined as an address in the second program (duplicated definition).

**Third program unit loaded (UNIT–#3)**

```
INPUT:    CALL    001BH
          CALL    TMDLYL
          CALL    001BH
          LD      HL, START
          CP      0DH
          JR      Z, END
          LD      (HL), A
          INC     HL
          JR      INPUT
END:      JP      0000H
COUNT2:   EQU     12

          END
```

"TMDLYL" U

TMDLYL is neither defined as an address nor declared with the ENT directive in any other external program unit.

## —B (Breakpoint) Command—

The B command sets or changes a breakpoint. A breakpoint occurs after instructions immediately preceding the breakpoint are executed the number of times specified in the break counter. When a breakpoint is taken, program execution is interrupted and control is passed to the debugger. The debugger saves the contents of the CPU registers into the register buffer and waits for a debugger command. The programmer can specify the breakpoint with either an absolute hexadecimal address or a label symbol (the label symbol can be given a displacement of from −65535 to 65535 in decimal).

| | |
|---|---|
| ＊ **DB**<br>**addr count** | Sets a breakpoint. |
| 1 **7530␣2** | The breakpoint is address 7530 and the break count is 2. |
| 2 **SORT3␣1** | The breakpoint is the address represented by label symbol "SORT3" and the break count is 1. |
| 3 **SORT3+5L␣1** | The breakpoint is the address of the instruction 5 lines away from the address represented by label symbol "SORT3" and the break count is 1. |
| 4 **MAIN0−9␣2** | The breakpoint is the address of the instruction 9 bytes before the address represented by label symbol "MAIN0" and the break count is 2. |
| 5 ■ | (The breakpoint and the break count must be separated by at least one blank (denoted by ␣ ).) |

— Enter the **B** command in response to the prompt "＊D".

— The debugger carries out a new line operation and displays "addr count". It then performs a new line operation and displays the breakpoint number followed by a space and the cursor to prompt the programmer to enter a breakpoint address and a break count.

The programmer may specify a breakpoint address with a 4-digit hexadecimal number or a global symbol (see the example above). In either case, enter an address followed by a space and a break count. The break count specifies the number of allowable passes through the breakpoint before a break actually occurs. **The programmer can specify a hexadecimal value from 1 to E.**

When a break count is entered, the debugger performs a new line operation and displays the next breakpoint number to prompt for the next breakpoint address.

— When a label symbol is entered as a breakpoint address, the debugger displays message "???" and waits for a new command if the pertinent symbol is not defined or if the symbol is a data defining symbol.

— No breakpoint can be specified for **the DJNZ instruction** When a breakpoint is specified for the DJNZ instruction, the debugger displays message "DJNZ?" and waits for entry of a new command.

— No breakpoint can be specified for **the CALL instruction** either. Breakpoints cannot be specified for any instructions which push the program counter contents into the stack. The debugger will display the message "CALL?" if such an attempt is made.

To check a CALL instruction, set a breakpoint at the beginning of the called routine.

— **To clear a previously set breakpoint**, enter that breakpoint address with a break count of 0 (use the & command to clear all breakpoints).

The debugger displays message "???" and waits for a command when an attempt is made to clear an undefined breakpoint.

— **The programmer can specify a maximum of nine breakpoints.** When the programmer specifies nine breakpoints, the debugger displays "X" on the next line instead of the next breakpoint number. This requests the programmer to clear a breakpoint or change a break count, not to set a new breakpoint. If the programmer attemps to set a new breakpoint, the debugger will not accept it and prompts for a new command with message "Over".

— When a B command is entered after breakpoints are set, the debugger displays them; in this case, the hexadecimal address is displayed first, followed by the break count format.

— The programmer can use the DEL key while setting breakpoints. When the CR key is pressed, the debugger is returned to the command wait state.

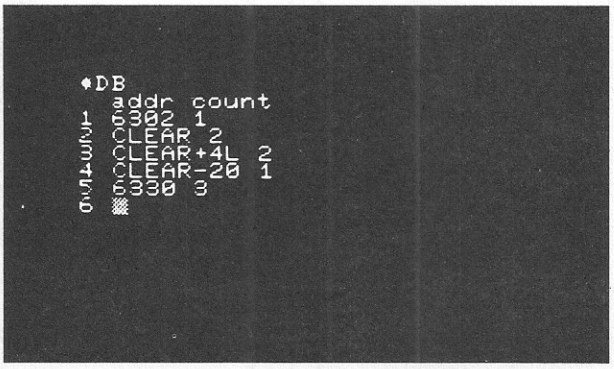
## —& (Clear B.P) Command—

| | |
|---|---|
| ＊ D& | Clears all the breakpoints which have been set. |

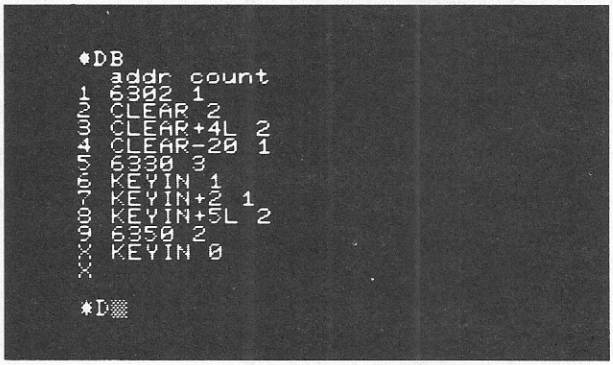
— Enter the & command in response to the prompt "＊D".

— The debugger clears all breakpoints set and waits for entry of a new command.

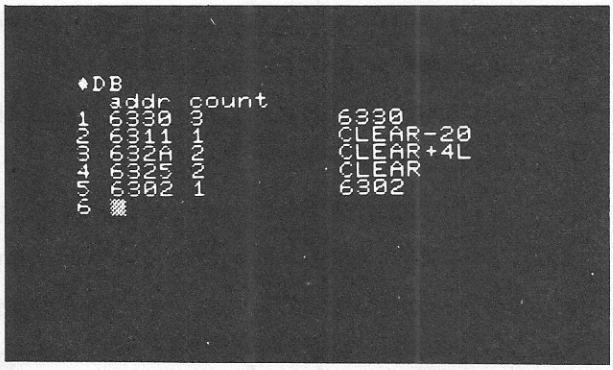
— The photo at right shows an example of setting breakpoints. The breakpoints are set with a 4-digit hexadecimal number (absolute address), a global label symbol, a label symbol plus a line specification and a label symbol plus a byte displacement.



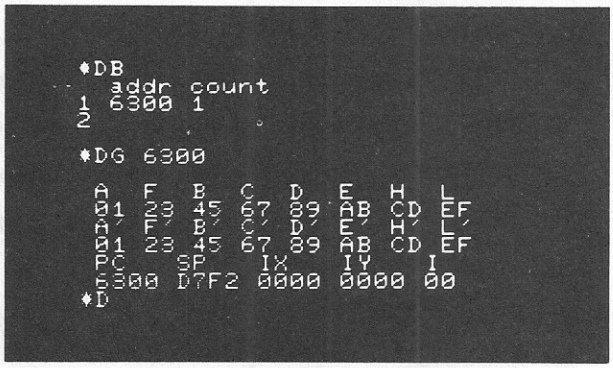

— The photo at right shows that breakpoint "KEYIN" has been cleared on the line identified by "X".




— The photo at right shows an example of displaying previously set breakpoints with a B command. Breakpoints are displayed with hexadecimal absolute addresses shown first, followed by the break counts and the label symbols.




— The photo at right shows that a break occurred immediately when the program was executed from address 6300 with a G command with a breakpoint at 6300 and a count of 1. As soon as a breakpoint was taken, an R command was executed to display the status of the CPU registers.



## —M (Memory dump) Command—

The M command displays the contents of the specified memory block in hexadecimal representation. The memory block may be specified with either absolute hexadecimal addresses or label symbols. The M command permits the programmer to alter data with the cursor.

| | |
|---|---|
| ＊ **DM 7800␣7850** CR | Displays the contents of the memory block from 7800 to 7850. |
| ＊ **DM MAIN7␣MAIN9** CR | Displays the contents of the memory block from the address identified by "MAIN7" to the address identified by "MAIN9". |
| ＊ **DM STEP0−9␣STEP3+15L** CR | Displays the contents of the memory block from the address 9 bytes before the address identified by label symbol "STEP0" to the address of the instruction 15 lines away from label symbol "STEP3". |

— Enter the M command in response to the prompt "＊D".

— The debugger displays the cursor with a space between the cursor and the letter M and waits for the programmer to enter the starting and ending addresses of the memory block to be dumped. The programmer may specify the starting and ending addresses of the memory block with either 4-digit hexadecimal numbers or global symbols.

— **The starting address must be smaller than or equal to the ending address.** Otherwise, the debugger will display the message " ? ".

— When a memory block in the link area is specified, the debugger displays a dump of memory contents on the screen with 8 bytes on a line.

— If the printer is placed in the enable mode, the debugger prints the memory dump on the printer with 16 bytes on a line.

— The cursor appears on the screen when the memory block dump is completed. The programmer can then alter byte data in the memory dump by moving the cursor to the desired byte position on the screen, entering the new byte data in hexadecimal and pressing CR. The byte data under the cursor is overwritten with the new data. The debugger displays the message "Error" if the data entered does not match the byte format.

— When CR is pressed with the cursor on a memory dump line, the data on that line is reentered into memory. The debugger is returned to the command mode, however, when CR is pressed with the cursor at the beginning of a line containing no data.

— Press the SPACE key to suspend display of the memory dump. To resume display, press the SPACE key again.

— Press the BREAK key to force the debugger into the command mode.

## —D (Memory list dump) Command—

The D command displays the contents of the specified memory block in hexadecimal representation with one instruction on a line. The memory block may be specified with either absolute hexadecimal addresses or label symbols. The programmer cannot alter memory contents through cursor manipulation.

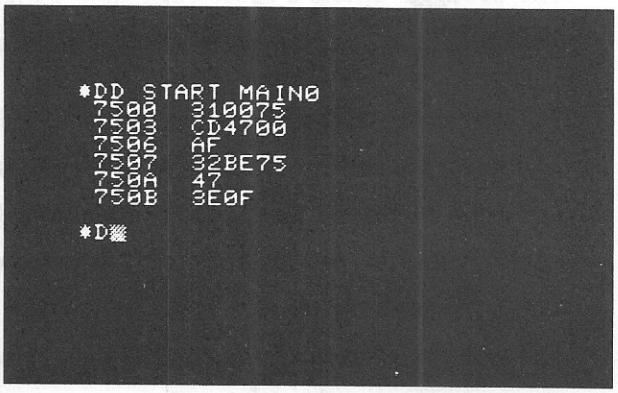
| | |
|---|---|
| ＊ DD 7800␣7850 CR | Displays the contents of the memory block from addresses 7800 to 7850 with one instruction on a line. |
| ＊ DD START␣MAIN0 CR | Displays the contents of the memory block from the addresses identified by "START" to the address identified by "MAIN0" with one instruction on a line. |
| ＊ DD 7500␣START+12L CR | Displays the contents of the memory block from address 7500 to the address of the instruction 12 lines away from the label symbol "START" with one instruction on a line. |

— Enter the D command in response to the prompt "＊D".

— The debugger displays the cursor with a space between it and the letter D, then waits for the programmer to enter the starting and ending addresses of the memory block to be dumped. The programmer may specify the starting and ending addresses of the memory block either with 4-digit hexadecimal numbers or global symbols. As with the M command, the starting address must be smaller than or equal to the ending address.

— Press the CR key after specifying the required memory block; the degugger then displays an address and machine language code on each line.

Consider the source program shown below, which contains the label symbols "START" and "MAIN0". Assume that the corresponding object code is loaded in memory starting at address 7500. When a D command is entered, the debugger displays a dump listing on the screen as shown in the photo at right.




```
START : ENT
        LD    SP, START
        CALL  MSTP
        XOR   A
        LD    (?TABP), A
        LD    B, A
MAIN0 : ENT
        LD    A, 0FH
```

— It must be noted that **the memory block starting address specified in the D command must contain an operation code.** If the starting address contains a data byte, subsequent lines dumped will display meaningless instructions which read that data byte as an operation code. The same note applies to the data areas (defined by DEFB and DEFW, etc.) in the memory block.

— Display of the memory dump listing can be suspended and resumed with the SPACE key.

— The D command does not allow memory alteration; after the memory dump is completed, the debugger is returned to the command wait state.

— Press the BREAK key to terminate this command in the middle of a dump.

## —W (Data write) Command—

The W command writes hexadecimal data, starting at the specified memory address. The memory address may be either an absolute hexadecimal address or a label symbol.

| | |
|---|---|
| ＊DW 8000 CR | Writes machine language data, starting at address 8000. |
| ＊DW DATA1 CR | Writes machine language data, starting at the address identified by label symbol "DATA1". |

— Enter the W command in response to the prompt "＊D".

— The debugger displays the cursor with a space between it and the letter W, then waits for the programmer to enter the starting address of the memory area to be written.

The programmer may specify the memory block starting address with a 4-digit hexadecimal number or a global symbol.

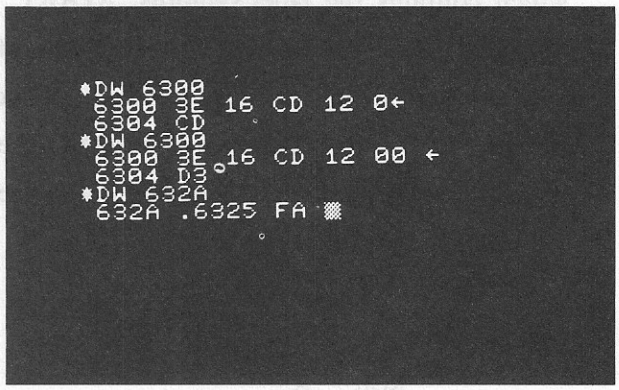
— The memory area to be written must be inside the link area.

```
＊DW 1111
  1111
  ???
```
} Address 1111 is not in the link area.

— When the programmer presses the CR key after specifying an address, the debugger displays that address on the next line to prompt the programmer to enter 2-digit hexadecimal data.

The debugger enters a space each time 2-digit data is entered and performs a new line operation and displays a new address each time eight bytes of data are entered.

— To correct the data just entered, press the ← key to return the cursor to the byte of data just entered and correct it. The photo on the right shows an example. As the photo shows, when the ← key is pressed, the cursor is placed on the next line and the address of the byte of data to which the cursor is moved is displayed.




— To specify a displacement for a JR, DJNZ or other Z80 relative jump instruction, enter a period; the debugger waits for the programmer to enter an absolute address (no label is allowed) with a 4-digit hexadecimal number as the destination of the jump. When the programmer enters a 4-digit hexadecimal address, the debugger computes the displacement and stores the 1-byte result in the current byte position. The seventh and eighth lines in the photo above show an example of specifying a displacement.

— After the necessary data has been written, press CR; the debugger then returns to the command wait state.

## —G (Goto) Command—

The G command transfers program control to the specified address. This command is also used to restart the program following a break.

| | |
|---|---|
| ＊ DG 7700 CR | Executes the program at address 7700. |
| ＊ DG START CR | Executes the program at the address identified by label symbol "START". |
| ＊ DG CR | Restarts the program at the breakpoint. The restart address and CPU register data are stored in the register buffer. |

— Enter a G command in response to the prompt "＊D".

— The debugger then waits for entry of an execution address. The programmer can specify the execution address with either a 4-digit hexadecimal number or a global label symbol defined with the ENT assembler directive.

When using a label symbol, the programmer can specify the execution address on a line or byte basis.

| | |
|---|---|
| ＊ DG MAIN0 | Executes the program at address "MAIN0". |
| ＊ DG MAIN0+3L | Executes the program at the address 3 lines after "MAIN0". |
| ＊ DG MAIN0−12 | Executes the program at the address 12 bytes before the address identified by "MAIN0". |

— To restart the program at a breakpoint, enter a G command and press CR. If this operation is initiated when no breakpoint is taken, the debugger returns to the command wait state without executing the program.

The contents of the CPU registers to be restored when the program is restarted are displayed with the R command. The value in the program counter (PC) is used as the restart address. Since the PC value can be changed with the P command, it is possible to restart the program at an address other than the breakpoint.

— Press the BREAK key to terminate entry of a G command.

## —I (Indicative start) Command—

The I command executes the program with the CPU registers loaded with the register buffer contents. The execution address is designated by the program counter. The contents of the CPU registers can be specified by the programmer through use of the A, C and P commands.

```
* DI
  A   F   B   C   D   E   H   L
  01  23  45  67  89  AB  CD  ED
  A'  F'  B'  C'  D'  E'  H'  L'
  01  23  45  67  89  AB  CD  EF
  PC      SP      IX      IY      I
  78AB    1FEA    5F70    4F50    00
  Start OK? ■
```

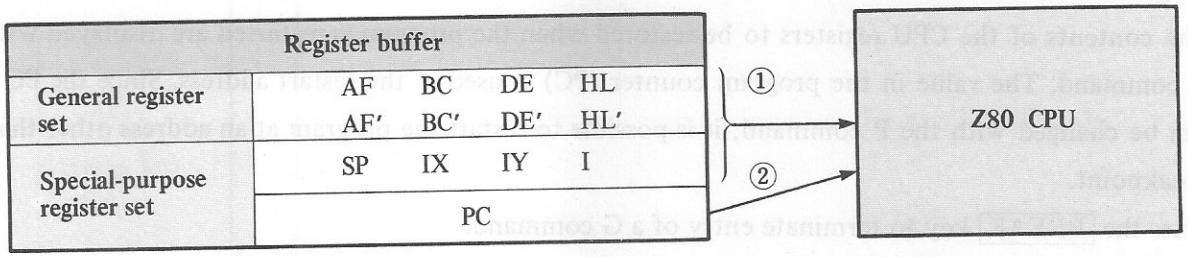
Executes the program at the address designated by the program counter with the data shown on the screen loaded in the CPU registers.

— Enter the I command in response to the prompt " * D".

— The debugger displays the 2- and/or 4-digit hexadecimal values to be loaded into the CPU registers. These values are stored in the register buffer. They can be displayed with the R command.

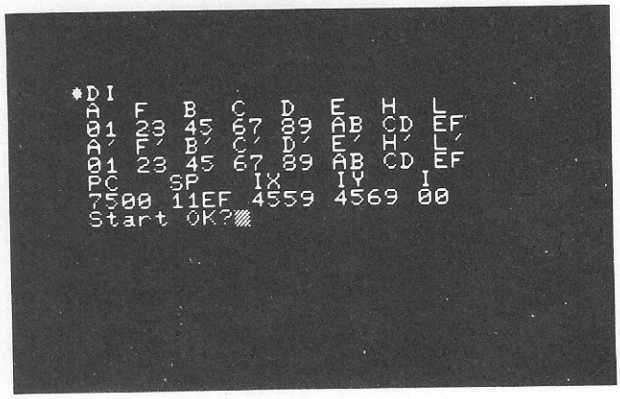
— The debugger then displays message "Start OK?". To start the program in this environment, press CR. **The debugger then executes the program, starting at the address designated in the program counter.** To change register values or terminate the I command, press the BREAK key; the debugger then returns to the command wait state.

— The figure below shows how the CPU registers are set with the I command.




The CPU general registers and special-purpose registers SP, IX, IY and I are loaded first; the program counter is then loaded with the execution address and the program is executed.

— The photo at right shows how the debugger responds to the I command and executes the program (at address 7500 in this example.)



## —A (Accumulator) Command—

The contents of the Z80 CPU registers are saved in the register buffer when a breakpoint is taken; the contents of the primary general registers saved can be displayed with the A command. The buffer contents can also be altered using cursor manipulation.

| | |
|---|---|
| ＊ DA<br>A F B C D E H L<br>01 23 45 67 89 AB CD EF | Displays the contents of primary register pairs AF, BC, DE and HL. |

— Enter the A command in response to the prompt "＊D".

— The debugger displays the contents of accumulator A, flag register F, and general register pairs BC, DE and HL with 2-digit hexadeciaml numbers. These values represent **the contents of the primary CPU registers set up when a break occurs at a breakpoint.** They are stored in the register buffer for use in subsequent restart operations (see the G command description) at the breakpoint.

— If necessary, the programmer can alter the register contents. To change a register value, place the cursor on the desired register value, overwrite it with a new value, and press [CR] key.

**The register values displayed or altered with the A command are those values which will be restored to the CPU internal registers on a restart at a breakpoint or on an indicative start with the I command.**

— Press [CR] key ; the debugger then returns to the command wait state.

## —C (Complementary) Command—

The C command displays the contents of the complementary general-purpose registers set up on the last break. The programmer can alter their contents through cursor manipulation.

| | |
|---|---|
| ＊ DC<br>A′ F′ B′ C′ D′ E′ H′ L′<br>01 23 45 67 89 AB CD EF | Displays the contents of complementary register pairs AF′, BC′, DE′ and HL′. |

— Enter the C command in response to the prompt "＊D".

— The debugger displays the contents of accumulator A′, flag register F′ and general-purpose register pairs BC′, DE′ and HL′ with 2-digit hexadecimal numbers. The contents of the registers and the meanings of the register contents and data altered through cursor manipulation are the same as with the A command. They are used for restart at a breakpoint or with the I command.

— Press the [CR] key; the debugger then returns to the command wait state.

## —P (Program counter) Command—

The P command displays the contents of the special-purpose registers set up on the last break. The programmer can alter their contents through cursor manipulation.

```
＊DP
 PC    SP    IX    IY    I
 78AB  1FEA  5F70  5F50  00
```

Displays the contents of special-purpose registers PC, SP, IX, IY and I.

— Enter the P command in response to the prompt "＊D".

— The debugger displays the contetns of special-purpose registers PC, SP, IX, IY and I with 2- and/or 4-digit hexadecimal numbers. The meanings of the register contents and the data altered through cursor manipulation are the same as with the A and C commands.

The register values displayed or altered through cursor manipulation are restored into the pertinent registers upon restartat a breakpoint or upon indicative start with the I command. The program does not have to restart at the breakpoint; **the programmer can specify another restart address by altering the PC value.**

— Press CR key; the debugger then returns to the command wait state.

## —R (Register) Command—

The R command displays the contents of all CPU internal registers set up on the last break or altered with the A, C or P commands. The programmer cannot alter their contents.

```
＊DR
 A   F   B   C   D   E   H   L
 01  23  45  67  89  AB  CD  EF
 A'  F'  B'  C'  D'  E'  H'  L'
 01  23  45  67  89  AB  CD  EF
 PC      SP      IX      IY      I
 78AB    1FEA    5F70    5F50    00
```

Displays the contents of all CPU registers.

— Enter the R command in response to the prompt "＊D".

— The debugger displays the contents of all CPU registers with 2- and/or 4-digit hexadecimal numbers. The cursor does not appear in the screen, so the programmer cannot alter their values.

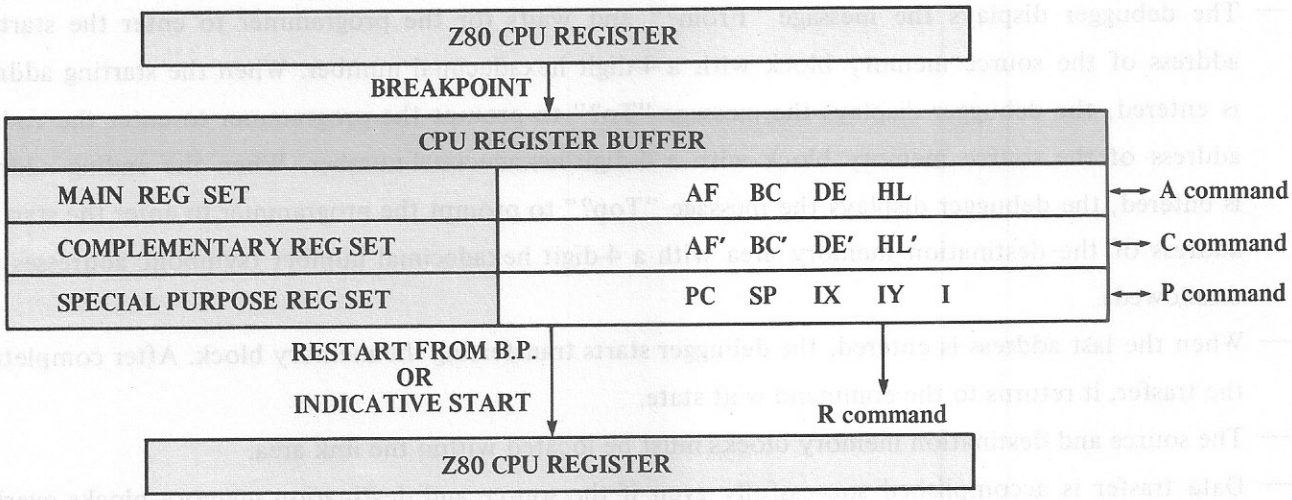
The same data is automatically displayed when a break occurs or when an indicative start is initiated with the I command.

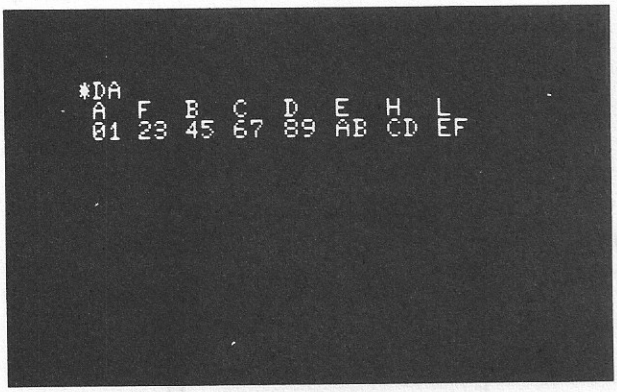
— The debugger enters the command wait state after displaying all the register contetns.

## Using register commands A, C, P and R

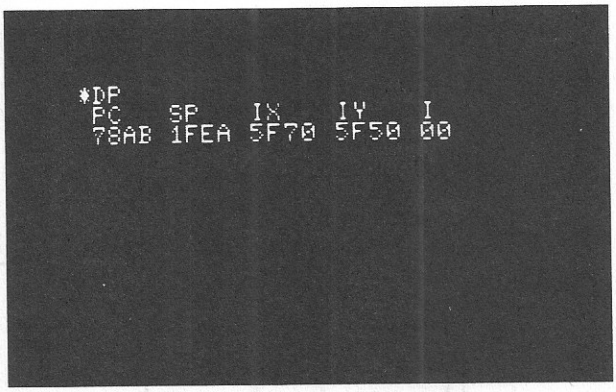
Values displayed with register commands (A, C, P and R) are the actual contents of the register buffer in the debugger. The register buffer in the debugger contains values loaded when breaks occur or when changes are made through cursor manipulation with the A, C or P command. The values are restored to the CPU registers when a restart is made from a breakpoint or when an indicative start is made.

The figure below shows the relationship between the CPU registers and the register commands; the photos show examples of use of the register commands.

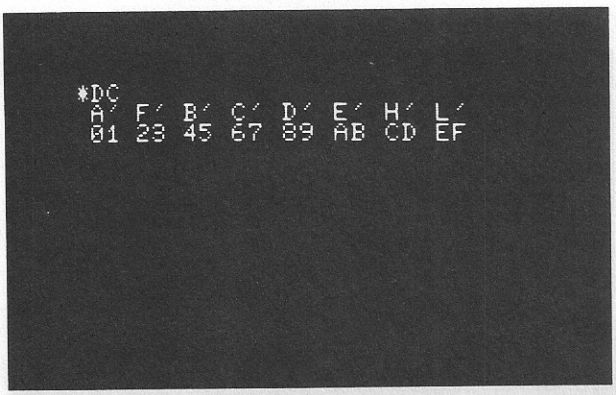


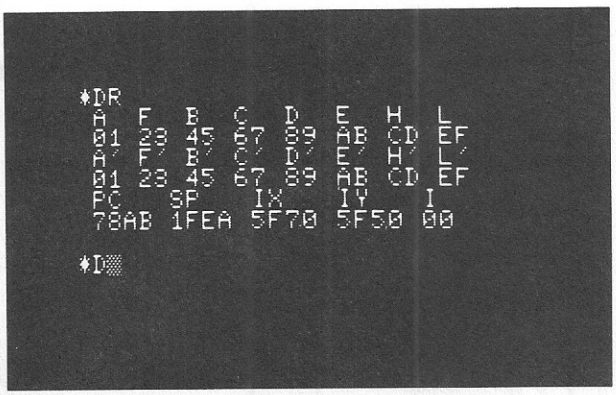

A command




P command




C command




R command

## —X (Data transfer) Command—

The X command trasfers the contents of the specified memory block to the specified memory area.

| | |
|---|---|
| ＊DX<br>**From? 7500 To? 811F Top? 9000** | Transfers the contents of the memory block from addresses 7500 to 811F to the memory area starting at address 9000. |

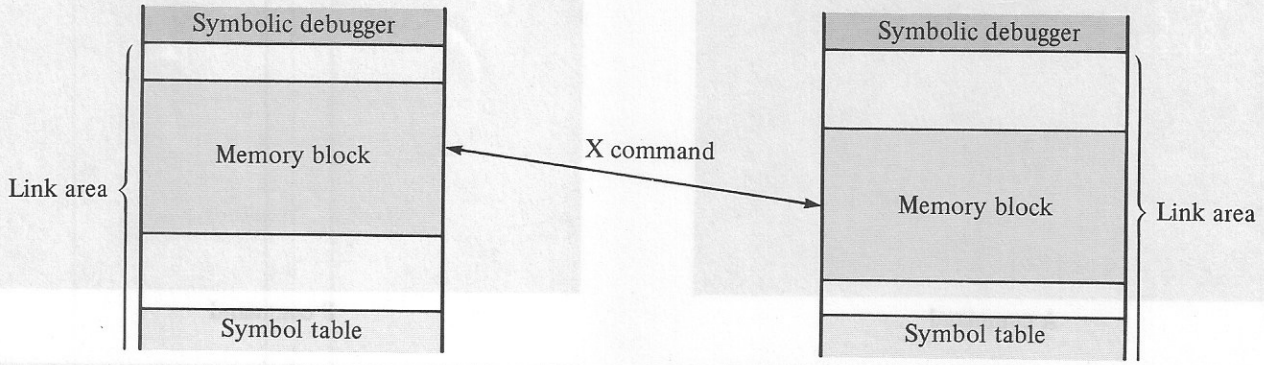
— Enter the X command in response to the prompt "＊D".

— The debugger displays the message "From?" and waits for the programmer to enter the starting address of the source memory block with a 4-digit hexadecimal number. When the starting address is entered, the debugger displays the message "To?" to prompt the programmer to enter the ending address of the source memory block with a 4-digit hexadecimal number. When the ending address is entered, the debugger displays the message "Top?" to prompt the programmer to enter the starting address of the destination memory area with a 4-digit hexadecimal number (symbolic addresses are disallowed).

— When the last address is entered, the debugger starts transferring the memory block. After completing the trasfer, it returns to the command wait state.

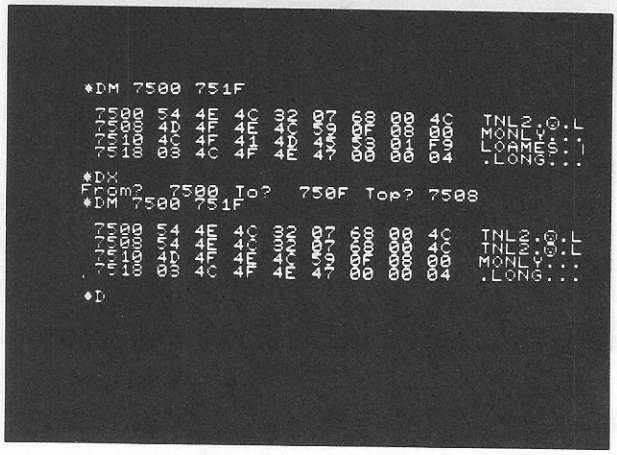
— The source and destination memory blocks must be located within the link area.

— Data trasfer is accomplished successfully even if the source and destination memory blocks overlap as shown below. The memory block shown in the figure at left may be trasferred to the memory block shown in the figure at right and vice versa.




— The photo at right shows how the debugger transfers the memory block starting at address 7500 and ending at address 750F to the memory area starting at address 7508.

Compare the memory contents displayed with the two M commands.



## —S (Save) Command—

The S command saves a specified block of the object program in the symbolic debugger link area into a named output file in immediately executable form. The contents of this file can be restored to the link area with the Y command.

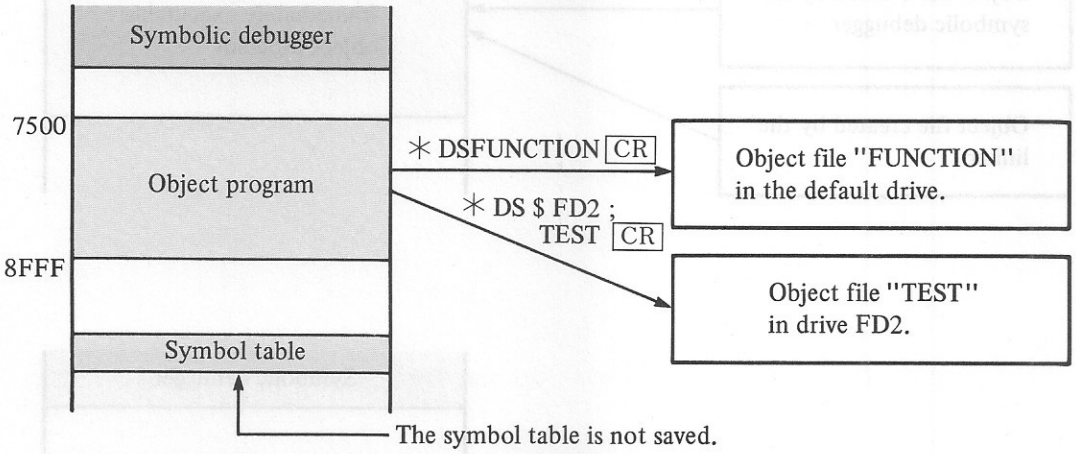
| | |
|---|---|
| ＊ **DSfilename** CR<br>**TBE␣7500␣8FFF␣7500** CR | Saves the immediately executable object program from addresses 7500 to 8FFF in the link area to an object file with a file name of filename. OBJ. |

— Enter the S command followed by a file name in response to the prompt "＊D".

— Press CR after entering a file name. The debugger displays a TBE (Top-Bottom-Execute) message after verifying that the specified file does not exist on the specified diskette.

— Enter the starting and ending addresses of the block to be saved and the execution address with 4-digit hexadecimal numbers or symbolic label names. When the execution address is omitted, the debugger assumes the block starting address as the execution address.

— The figure below shows how the object program block from addresses 7500 to 8FFF is saved to an output file with the file name "FUNCTION".



## —Y (Yank) Command—

The Y command reads the object file identified by filename into the link area.

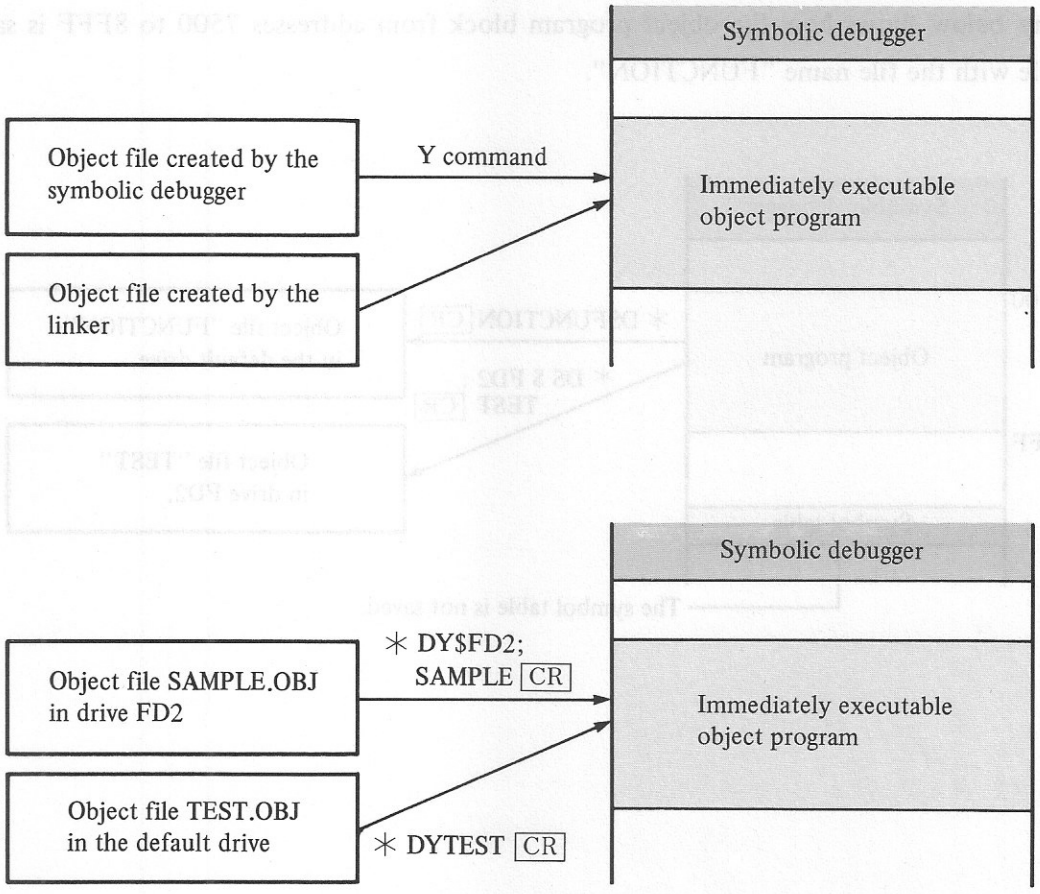
| | |
|---|---|
| ＊ **DY filename** [CR] | Reads the object file named filename into the link area under loading |
| **Loading address $7500–8FFF** | conditions established when the file was saved. |
| **Execute address $7500** | |

— Enter the **Y** command followed by a file name in response to the prompt "＊D".

— Press [CR] after entering the file name. The debugger then searches for the file named filename. OBJ and reads it.

— The program in the filename.OBJ file is loaded into the link area block between the starting and ending addresses specified when the file was saved with the S command.

Note: Files opened before the Y command is issued are all killed.



## —\(FDOS) Command—

The \ command executes a built-in FDOS command. "＊D" is displayed to prompt for the next command.

| | |
|---|---|
| ＊ D \ FREE $FDn [CR] | Outputs the number of used and unused sectors on the floppy disk in the disk drive indicated by $FDn. |

— Enter the \ command followed by the desired built-in FDOS command in response to the prompt "＊D".

— Press the [CR] key; the debugger then executes the specified FDOS command and displays "＊D" to prompt for the next command.

— The XFER and EXEC commands cannot be executed. The RUN command cannot be executed when the program to be executed by the RUN command is too long.

## —# Command—

| | |
|---|---|
| ＊D# | Switches the list mode for printout on the printer. |

— Enter the # command in response to the prompt "＊D".

— The debugger then switches the list mode. When the debugger is invoked, the printer list mode is set to the disable mode. The mode alternates between enable and disable each time a # command is entered. In the enable mode, all output is directed to both the screen and the printer (except with the M command).

## —! Command—

| | |
|---|---|
| ＊ D! | Returns control to FDOS. |

— Enter the ! command in response to the prompt "＊D".

— Control is then transferred to FDOS.

# ERROR MESSAGES

| Error message | Description | Related commands |
|---|---|---|
| ??? | ○ The command operand fields does not match the 4-digit hexadecimal format.<br>○ A symbolic label is missing.<br>○ A data defining symbol is used as a label. | M, D, W, B, G |
| **Error** | ○ An invalid number of digits was entered when altering register or memory contents, or a key other than 0 through 9 or A through F was pressed. | A, C, P, M |
| **DJNZ?** | A breakpoint was set for a DJNZ instruction. | B |
| **CALL?** | A breakpoint was set for a CALL instruction. | B |
| **RST 7?** | A breakpoint was set for a RST 7 instruction. | B |
| **Over** | An attempt was made to set more than 9 breakpoints. | B |
| ? | ○ An attempt was made to access outside the link area.<br>○ The starting address is greater than the ending address.<br>○ An attempt was made to clear an undefined breakpoint.<br>○ The breakpoint counter was set to F (the maximum permissible value is E in hexadecimal). | M, D, W, B, G, X<br>M, D<br>B<br>B |

Note: Refer to the System Messages in the System Command manual for other system error messages.