PART 11

THE DOS/BATCH DEBUGGING PROGRAM

ODT

PART 11

CHAPTER 1

INTRODUCTION TO THE DEBUGGING PROGRAM ODT

ODT (on-line-debugging technique for the PDP-11) is a system program that aids in debugging programs. From the keyboard the user interacts with ODT and a loaded object program. The major features provided by ODT include:

1. Printing the contents of any location for examination or alteration.
2. Running all or any portion of a user program using the breakpoint feature.
3. Searching the program area for specific bit patterns.
4. Searching the program area for words which reference a specific word.
5. Calculating offsets for relative addresses.
6. Filling a block of words or bytes with a designated value.

During a debugging session at the terminal the user should have the assembly listing of the program to be debugged. Minor corrections to the program may be made on-line during the debugging session. The program may then be run under control of ODT to verify changes made. Major corrections, however, such as a missing subroutine, should be noted on the assembly listing and incorporated in a subsequent updated program assembly.

1.1 RELOCATION

When the assembler produces a binary relocatable object module, the base address of the module is taken to be location ØØØØØØ, and the addresses of all program locations as shown in the assembly listing are indicated relative to this base address. After the module is linked by the Linker, many values within the program, and all the addresses of locations in the program, will be incremented by a constant whose value is the actual absolute base of the module after it has been relocated. This constant is called the relocation bias for the module. Since a linked program may contain several relocated modules, each with its own relocation bias, and since, in the process of debugging, these biases will have to be subtracted from absolute addresses continually in order to relate relocated code to assembly listings, ODT provides an automatic relocation facility.

The basis of the relocation facility lies in 8 relocation registers, numbered Ø through 7, which may be set to the values of the relocation biases you are interested in at a given time during debugging. Relocation biases should be obtained by consulting the link map produced by the Linker. Once set, a relocation register is used by ODT to relate relocatable code to relocated code. For more information on the exact nature of the relocation process, consult Part 9, the DOS/BATCH Linker.

1.2 RELOCATABLE EXPRESSIONS

The symbol h below stands for an integer in the range Ø to 7 inclusive.

The symbol k stands for an octal number up to six digits long, with a maximum value of 177777. If more than six digits are typed, ODT takes the last six digits, truncated to the low-order 16 bits. k may be preceded by a minus sign, in which case its value will be the two's complement of the number typed. For example:

k (number typed)	value
1	ØØØØØ1
-1	177777
4ØØ	ØØØ4ØØ
-17773Ø	ØØØ5Ø
1234567	Ø34567

The symbol r is called a relocatable expression and is evaluated by ODT as a 16-bit (6 octal digit) number. It may be typed in any one of three forms.

Form A	k	The value r is simply the value of k.
Form B	n,k	The value of r is the value of k plus the contents of relocation register n. If the n part of this expression is greater than 7, ODT takes only the last octal digit of n.
Form C	C or C,k or n,C or C,C	Whenever the letter C is typed, ODT replaces C with the contents of a special register called the constant register. This value has the same role as the k or n that it replaces. The constant register is designated by the symbol $C and may be set to any value.

In the following examples, assume that relocation register 3 contains ØØ34ØØ and that the constant register contains ØØØØØ3.

r	value of r
5	ØØØØØ5
-17	177761
3,Ø	ØØ34ØØ
3,15Ø	ØØ355Ø
3,-1	ØØ3377
C	ØØØØØ3
3,C	ØØ34Ø3
C,Ø	ØØ34ØØ
C,1Ø	ØØ341Ø
C,C	ØØ34Ø3

1.3 COMMANDS

ODT's commands are composed using the following characters and symbols. They are often used in combination with the address upon which the operation is to occur, and are offered here for familiarization prior to their thorough coverage, which follows.

r/	Open the word at location r.
/	Reopen the last opened location.
r \(SHIFT/L)	Open the byte at location r.
\	Reopen the last opened byte.
nR	After a word has been opened, retype the contents of the word relative to relocation register n; i.e., subtract contents of relocation register n from the contents of the opened word and print the result. If n is omitted, ODT selects the relocation register whose contents are closest but less than or equal to the contents of the opened location.
n!	After a word or byte has been opened, print the address of the opened location relative to relocation register n. If n is omitted, ODT selects the relocation register whose contents are closest, but less than or equal to the address of the opened location.
↓ (LINE FEED Key)	Open next sequential location.
↑ or ^	Open previous location.
RETURN	Close open location and wait for the next command.
← or _	Take contents of opened location, index by contents of PC, and open that location.
@	Take contents of opened location as absolute address and open that location.

>	Take contents of opened location n as relative branch instruction and open referenced location.
<	Return to sequence prior to last @, >, or ↑ command and open succeeding location.
X	Perform a Radix-5Ø unpack of the binary contents of the current opened word; then permit the storage of a new Radix 5Ø binary number in the same location.
r;O	Calculate offset from currently open location to r.
$n/	Open general register n (Ø-7).
$y/	Open special register y, where y may be one of the following letters.

- S status register (saved by ODT after a breakpoint)
- M mask register
- B first word of the breakpoint table
- P priority register
- C constant register
- R first relocation register (register Ø)
- F format register

;F	Fill memory words with the contents of the constant register.
;I	Fill memory bytes with the contents of the low-order 8 bits of the constant register.
;	Separate commands from command arguments (used with alphabetic commands below); separate a relocation register specifier from an addend.
;B	Remove all breakpoints.
r;B	Set breakpoint at location r.
r;nB	Set breakpoint n at location r.
;nB	Remove breakpoint n.
r;E	Search for instructions that reference effective address r.

x;W	Search for words with bit patterns that match x.
;nS	Enable single-instruction mode (n can have any value and is not significant); disable breakpoints.
;S	Disable single-instruction mode; reenable breakpoints.
r;G	Go to location r and start program run.
;P	Proceed with program execution from breakpoint; stop when next breakpoint is encountered or at end of program. In single-instruction mode only, proceed to execute next instruction only.
k;P	Proceed with program execution from breakpoint; stop after encountering the breakpoint k times. In single-instruction mode only, proceed to execute next k instructions.
;R	Set all relocation registers to -1 (highest address value).
;nR	set relocation register n to -1.
r;nR	set relocation register n to the value of r; if n is omitted, it is assumed to be Ø.
r;C	Print the value of r and store it in the constant register.
r;nA	Print n bytes in their ASCII format, starting at location r; then allow n bytes to be typed in, starting at location r.
CTRL/C	Prepare Monitor to accept a command from the keyboard.

PART 11

CHAPTER 2

COMMANDS AND FUNCTIONS

When ODT is started as explained in Chapter 11-4, it will indicate its readiness to accept commands by printing an asterisk on the left margin of the teleprinter output. In response to the asterisk the user can issue most commands; for example, you can examine and, if desired, change a word, run the object program in its entirety or in segments, or even search core for certain words or references to certain words. The discussion below explains these features.

All commands to ODT are stated using the characters and symbols shqwn in Sections 11-1.2 and 11-1.3.

2.1 PRINTOUT FORMATS

Normally, when ODT prints addresses (as with the commands ↓, ↑, ←, @, <, and >), it attempts to print them in relative form (Form B in Section 11-2.1). ODT looks for the relocation register whose value is closest but less than or equal to the address to be printed, and then represents the address relative to the contents of the relocation register. However, if no relocation register fits the requirement, the address is printed in absolute form. Since the relocation registers are initialized to -1 (the highest number), the addresses are initially printed in absolute form. If any relocation register subsequently has it contents changed, it may then, depending on the command, qualify for relative form.

For example, suppose relocation registers 1 and 2 contained 1ØØØ and 1ØØ4 respectively, and all other relocation registers contained numbers much higher. Then the following sequence might occur.

```
*774/ØØØØØØ↓
ØØØ776/ØØØØØØ↓
1,ØØØØØØ /ØØØØØØ↓
1,ØØØØØ2 /ØØØØØØ↓
2,ØØØØØØ /ØØØØØØ
```

The format is controlled by the format register, $F. Normally this register contains Ø, in which case ODT prints addresses relatively whenever possible. $F may be opened and changed to a nonzero value, however, in which case all addresses will be printed in absolute (see Section 11-2.2.1Ø).

2.2 OPENING, CHANGING AND CLOSING LOCATIONS

An open location is one whose contents ODT has printed for examination, and whose contents are available for change. A closed location is one whose contents are no longer available for change.

The contents of an open location may be changed by typing the new contents followed by a single character command which requires no argument (i.e., ↓, ↑, RETURN, ←, @, >, <). Any command typed to open a location when another location is already open, will first cause the currently open location to be closed.

2.2.1 Slash /

One way to open a location is to type the address followed by a slash.

```
*1000/012746
```

Location 1000 is open for examination and is available for change. (Note that in all examples ODT's printout is underlined; user typed input is not.)

Should the user not wish to change the contents of an open location, he should type the RETURN key and the location will be closed; ODT will print another asterisk and wait for another command. However, should the user wish to change the word, he should simply type the new contents before giving a command to close the location.

```
*1000/012746 012345
*
```

In the example above, location 1000 now contains 012345 and is closed since the RETURN key was typed after entering the new contents, as indicated by ODT's second asterisk. Used alone, the slash will reopen the last location opened.

```
*1000/012345 2340
*/002340
```

In the example above, the open location was closed by typing the RETURN key. ODT changed the contents of location 1000 to 002340 and then closed the location before printing the *. A single slash then directed ODT to reopen the last location opened. This allowed verifying that the word 002340 was correctly stored in location 1000.

Note again that opening a location while another is currently open will automatically close the currently open location before opening the new location.

Also note that if you specify the opening of an odd-numbered address with a slash, ODT will open the location as a byte, and subsequently will behave as if a backslash had been typed.

2.2.2 Backslash \

In addition to operating on words, ODT operates on bytes. One way to open a byte is to type the address of the byte followed by a backslash (\ is printed by typing SHIFT/L). This not only causes the byte value at the specified address to be printed out, it also causes the value to be interpreted as ASCII code, and the corresponding character to be echoed (if possible) on the terminal.

```
*1ØØ1\1Ø1=A
```

A backslash typed alone will reopen the last open byte. If a word was previously open, the backslash will reopen its even byte.

```
*1ØØ2/ØØØØØ4\ØØ4=
```

2.2.3 LINE FEED Key ↓

If the LINE FEED key is typed when a location is open, ODT closes the open location and opens the next sequential location.

```
*1ØØØ/ØØ234Ø↓          (↓ denotes typing the LINE FEED key)
ØØ1ØØ2/Ø1274Ø
```

In this example, the LINE FEED key instructed ODT to print the address of the next location along with its contents, and to wait for further instructions. After the above operation, location 1ØØØ is closed and 1ØØ2 is open. The open location may be modified by typing the new contents.

If the opened location was a byte, the LINE FEED opens the next byte.

2.2.4 Up-Arrow ↑

The up-arrow (or circumflex) symbol results from typing the SHIFT and N key combination. If the up-arrow is typed when a location is open, ODT closes the open location and opens the previous location (as shown by continuing from the example above).

```
ØØ1ØØ2/Ø1274Ø↑
ØØ1ØØØ/ØØ234Ø
```

Now location 1ØØ2 is closed and 1ØØØ is open. The open location may be modified by typing the new constants.

If the opened location was a byte, then ↑ opens a byte as well.

The LINE FEED and up-arrow (or circumflex) keys will operate on bytes if a byte is open when the command is given (see Sections 11-2.2.3 and 11-2.2.4). For example:

```
*1ØØ1\1Ø1=A↓
ØØ1ØØ2\ØØ4=↑
ØØ1ØØ1\1Ø1=A
*
```

2.2.5 Back-Arrow ←

The back-arrow (or underline) symbol results from typing the SHIFT and O key combination. If the back-arrow is typed to an open word, ODT interprets the contents of the currently open word as an address indexed by the program counter (PC) and opens the location so addressed.

```
*1ØØ6/ØØØØØ6←
ØØ1Ø16/1ØØ4Ø5
```

Notice in this example that the open location, 1ØØ6, was indexed by the PC as if it were the operand of an instruction with address mode 67 as explained in the Processor Handbook.

A modification to the opened location can be made before a ↓, ↑, or ←, is typed. Also, the new contents of the location will be used for address calculations using the ← command. Example:

```
*100/000222      4↓     (modify to 4 and open next location)
000102/000111     6↑    (modify to 6 and open previous location)
000100/000004     100←  (change to 100 and open location indexed
000202/(contents)       by PC)
```

2.2.6 Open the Addressed Location @

The symbol @ will optionally modify, close an open word, and use its contents as the address of the location to open next.

```
*1006/001024     @          (open location 1024 next)
001024/000500
*1006/001024     2100     @(modify the 2100 and open
002100/177774               location 2100)
```

2.2.7 Relative Branch Offset >

The right angle bracket > will optionally modify, close an open word, and use its low-order byte as a relative branch offset to the next word opened.

```
*1032/000407  301>          (modify to 301 and interpret
000636/000010                as a relative branch)
```

2.2.8 Return to Previous Sequence <

The left-angle bracket < will optionally modify, close an open location, and open the next location of the previous sequence interrupted by a ←, @, or > command. Note that ←, @, or > will cause a sequence change to the word opened. If a sequence change has not occurred, < will simply open the next location as a LINE FEED does. The command will operate on both words and bytes.

```
*1032/000407     301 >       (> causes a sequence change)
000636/000010 <              (< causes a return to original sequence)
001034/001040 @              (@ causes a sequence change)
001040/000405\005=   <       (< now operates on byte)
001035  \002= <              (< acts like ↓)
001036  \004=
```

2.2.9 Accessing General Registers Ø-7

The program's general registers Ø-7 can be opened using the following command format:

```
*$n/
```

where n is the integer representing the desired register (in the range Ø through 7). When opened, these registers can be examined or changed by typing in new data as with any addressable location. For example:

```
*$Ø/ØØØØ33              (RØ was examined and closed)
*
```

and

```
*$4/ØØØ474   464        (R4 was opened, changed, and
*                        closed)
```

The example above can be verified by typing a slash in response to ODT's asterisk.

```
*/ØØØ464
```

The ↓, ↑, ←, or @ commands may be used when a register is open.

2.2.1Ø Accessing Internal Registers

The program's status register contains the condition codes of the most recent operational results and the interrupt priority level of the object program. It is opened using the following command:

```
*$S/ØØØ311
```

where $S represents the address of the status register. In response to $S in the example above, ODT printed the 16-bit word of which only the low-order 8 bits are meaningful: bits Ø-3 indicate whether a carry, overflow, zero, or negative (in that order) has resulted, and bits 5-7 indicate the interrupt priority level (in the range Ø-7) of the object program. (See Part 3, DOS/BATCH Monitor, for the status register format.

The $ is used to open certain other internal locations:

$B location of the first word of the breakpoint table (see Section 11-2.3).

$M mask location for specifying which bits are to be examined during a bit pattern search (see Section 11-2.6).

$P location defining the operating priority of ODT (see Section 11-2.12).

$S location containing the condition codes (bits Ø-3) and interrupt priority level (bits 5-7).

$C location of the constant register (see Section 11-2.7).

$R location of relocation register Ø, the base of the relocation register table (see Section 11-2.1Ø).

$F location of format register (see Section 11-2.1).

2.2.11 Radix-5Ø Mode X

The Radix-5Ø mode of packing certain ASCII characters three to a word is employed by many DEC-supplied PDP-11 system programs, and may be employed by any programmer via the assembler's .RAD5Ø directive.

ODT provides a method for examining and changing memory words packed in this way with the X command.

When a word is opened, the user may type X, in which case ODT will convert the contents of the opened word to its 3-character Radix-5Ø equivalent, and will type these characters on the terminal. The user may then type one of the following.

	Type	Effect
a.	RETURN key	closes the currently open location.
b.	LINE FEED key	closes the location and opens the next one in sequence.
c.	↑ key	closes the location and opens the previous one in sequence.
d.	Any three characters whose octal code is Ø4Ø (space) or greater	converts the three specified characters into packed Radix-5Ø format.

Here are the legal Radix-5Ø characters.

. $ Space Ø through 9 A through Z

If any other characters are typed, the resulting binary number is unspecified. However, exactly three characters must be typed before ODT resumes its normal mode of operation.

After the third character is typed, the resulting binary number is available to be stored into the opened location by closing the location in any one of the usual ways (carriage-return, line feed, etc.). Example:

```
*1ØØØ/Ø42431      X=KBI    CBA
*1ØØØ/Ø11421      X=CBA
```

WARNING

> After ODT has converted the three characters to binary, the binary number can be interpreted in one of many different ways, depending on the command that follows. For example,
>
> *1234/Ø63337 X=PRO XIT/
>
> Since the Radix-5Ø equivalent of XIT is 113574, the final slash typed in the example will cause ODT to open location 113574 if it is a legal address. (See Section 11-2.15 for a discussion of command legality and detection of errors.)

2.3 BREAKPOINTS

The breakpoint feature facilitates monitoring the progress of program execution. A breakpoint may be set at any instruction that is not referenced by the program for data. When a breakpoint is set, ODT replaces the contents of the breakpoint location with a trap instruction. When the program is executed and the breakpoint is encountered, program execution is suspended, the original contents of the breakpoint location are restored, and ODT regains control.

With ODT the user can specify up to eight breakpoints set, numbered Ø through 7. The r;B command will set the next available breakpoint. Specific breakpoints may be set or changed by the r;nB command where n is the number of the breakpoint. For example:

```
*1Ø2Ø;B          (sets breakpoint Ø)
*1Ø3Ø;B          (sets breakpoint 1)
*1Ø4Ø;B          (sets breakpoint 2)
*1Ø32;1B         (resets breakpoint 1)
*
```

The ;B command removes all breakpoints. To remove only one of the breakpoints the ;nB command is used, where n is the number of the breakpoint. For example:

```
*;2B            (removes the second breakpoint)
*
```

The $B/ command opens the location containing the address of breakpoint Ø. The next seven locations contain the addresses of the other breakpoints in order, and thus can be opened using the LINE FEED key. (The next location is for single-instruction mode, explained in Section 11-2.5). Example:

```
*$B/ØØ1Ø2Ø  ↓
nnnnnn/ØØ1Ø32↓
nnnnnn/ (address internal to ODT)
```

In this example, breakpoint 2 is not set. The contents will be an address internal to ODT. After the table of breakpoints is the table of proceed command repeat counts, first for each breakpoint, and then for the single instruction mode (see Section 11-2.5).

```
.
.
.                ↓
nnnnnn/ØØ1Ø36    ↓        (breakpoint 7)
nnnnnn/nnnnnn    ↓        (single-instruction address)
nnnnnn/ØØØØØØ    15  ↓    (count for breakpoint Ø)
nnnnnn/ØØØØØØ             (count for breakpoint 1)
```

2.4 RUNNING THE PROGRAM r;G AND r;P

Program execution is under control of ODT. There are two commands for running the program: r;G and r;P. The r;G command is used to start execution (go) and r;P to continue (proceed) execution after having halted at a breakpoint. For example,

```
*1ØØØ;G
```

starts execution at location 1ØØØ. The program will run until encountering a breakpoint or until program completion, unless it gets caught in an infinite loop, where you must either restart or re-enter as explained in Section 11-4.2.

When a breakpoint is encountered, execution stops and ODT prints Bn; (where n is the breakpoint number), followed by the address of the breakpoint. You may then examine desired locations for expected data. For example:

```
*1Ø1Ø;3B        (breakpoint 3 is set at location
                1Ø1Ø)
*1ØØØ;G         (execution started at location 1ØØØ)
*B3;ØØ1Ø1Ø      (execution stopped at location 1Ø1Ø)
*
```

When a breakpoint is set in a loop, it may be desirable to allow the program to execute a certain number of times through the loop before recognizing the breakpoint. This may be done by typing the k;P command and specifying the number of times the breakpoint is to be encountered before program execution is suspended (on the kth encounter).

The count k is associated only with the numbered breakpoint that most recently occurred. A different proceed count may be associated with each numbered breakpoint, and will apply to that breakpoint only. Example:

```
B3;ØØ1Ø1Ø       (execution halted at breakpoint)
*125Ø;B         (set breakpoint at location 125Ø)
*4;P            (continue execution, loop through
B3;ØØ125Ø       breakpoint 3 times and halt on 4th
*               occurrence of the breakpoint)
```

The breakpoint repeat counts can be inspected by typing $B/ and following that with the typing of nine LINE FEEDs. The repeat count for breakpoint Ø will then be printed. The repeat counts for breakpoints 1 through 7, and the repeat count for the single instruction trap follow in sequence (see Section 11-2.5). Opening any one of these provides an alternative way of specifying the count. The location, being open, can have its contents modified in the usual manner by the typing of new contents and then the RETURN key.

Breakpoints are inserted when performing an r;G or k;P command. Upon execution of the r;G or k;P command, the general registers Ø-6 are set to the values in the locations specified as $Ø-$6 and the processor status register is set to the value in the location specified as $S.

2.5 SINGLE-INSTRUCTION MODE

With this mode the user can specify the number of instructions he wishes executed before suspension of the program run. The proceed command, instead of specifying a repeat count for a breakpoint encounter, specifies the number of succeeding instructions to be executed. Note that breakpoints are disabled when single-instruction mode is operative. Commands for single-instruction mode follow.

;nS	Enables single-instruction mode (n can have any value and serves only to distinguish this form from the form ;S); breakpoints are disabled.
k;P	Proceeds with program run for next k instructions before re-entering ODT (if k is missing, it is assumed to be 1). (Trap instructions and associated handlers can affect the proceed repeat count. See Section 11-3.2.)
;S	Disables single-instruction mode.

When the repeat count for single-instruction mode is exhausted and the program suspends execution, ODT prints

```
B8;k
*
—
```

where k is the address of the next instruction to be executed. The $B breakpoint table contains this address following that of breakpoint 7. However, unlike the table entries for breakpoints Ø-7, direct modification has no effect.

Following the repeat count for breakpoint 7 in the table is the repeat count for single-instruction mode. This table entry, though, may be directly modified, and thus is an alternative way of setting the single-instruction mode repeat count. In such a case, ;P implies the argument set in the $B repeat count table rather than 1.

2.6 SEARCHES

With ODT the user can search all or any specified portion of core memory for any specific bit pattern or for references to a specific location.

2.6.1 Word Search x;W

Before initiating a word search, the mask and search limits must be specified as shown in the example below. The location represented by $M is used to specify the mask of the search. $M/ opens the mask register. The next two sequential locations (opened by LINE FEEDs) contain the lower and upper limits of the search. Bits set to 1 in the mask will be examined during the search; other bits will be ignored. Then the search object and the initiating command are given using the x;W command where x is the search object. When a match is found the address of the unmasked matching word is printed. For example:

```
*$M/000000  177400 ↓          (test high order eight bits)
nnnnnn/000000  1000 ↓         (set low address limit)
nnnnnn/000000  1040           (set high address limit)
*400;W                        (initiate word search)
001010/000770
001034/000404
*
```

In the search process, the word currently being examined and the search object are exclusively ORed (XOR), and the result is ANDed with the mask. If this result is zero, a match has been found, and is reported on the teleprinter. Note that if the mask is zero, all locations within the limits will be printed.

Typing CTRL/U during a search printout will terminate the search.

2.6.2 Effective Address Search r;E

ODT enables the user to search for words that address a specified location. Open the mask register only to gain access to the low- and high-limit registers. After specifying the search limits (Section 11-2.6.1) the command r:E is typed initiating the search.

Words which are either an absolute address (argument r itself), a relative address offset, or a relative branch to the effective address will be printed after their addresses. For example:

```
*$M/177400↓                   (open mask register only to gain
nnnnnn/001000  1010↓          access to search limits)
nnnnnn/001040  1060
*1034;E                       (initiating search)
001016/001006                 (relative branch)
001054/002767                 (relative branch)
*1020;E                       (initiating a new search)
001022/177774                 (relative address offset)
001030/001020                 (absolute address)
*
```

Particular attention should be given to the reported references to the effective address, because a word may have the specified bit pattern of an effective address without actually being so used. ODT will report these as well.

Typing CTRL/U during a search printout will terminate the search.

2.7 CONSTANT REGISTER r;C

It is often desirable to convert a relocatable address into a relocated address or to convert a number into its two's complement, and then to store the converted value into one or more places in your program. The constant register provides a means of accomplishing this and other useful functions.

When r;C is typed, the relocatable expression r is evaluated to its six digit octal value and is both printed on the terminal and stored in the constant register. The contents of the constant register may be invoked in subsequent relocatable expressions by typing the letter C.

Examples:

*-4432;C=173346	(The two's complement of 4432 is placed in the constant register.)
*1ØØØ/ØØ1ØØØ C	(The contents of the constant register are stored in location 1ØØØ.)
*1ØØØ;1R	(Relocation register 1 is set to 1ØØØ.)
*1,4272;C=ØØ5272	(Relative location 4272 is reprinted as an absolute location and stored in the constant register.)

2.8 CORE BLOCK INITIALIZATION ;F AND ;I

The constant register can be used in conjunction with the commands ;F and ;I to set a block of memory to a given value. While the most common value required is zero, other possibilities are plus one, minus one, ASCII space, etc.

When the command ;F is typed, ODT stores the contents of the constant register in successive memory words starting at the memory word address specified in the lower search limit, and ending with the address specified in the upper search limit.

When the command ;I is typed, the low-order 8 bits in the constant register are stored in successive bytes of memory starting at the byte address specified in the lower search limit and ending with the byte address specified in the upper search limit.

Example: Assume relocation register 1 contains 1ØØØ, 2 contains 2ØØØ, 3 contains 3ØØØ. The following sequence sets word locations 1ØØØ-1776 to zero, and byte locations 2ØØØ-2777 to ASCII spaces.

```
*$M/ØØØØØØ ↓                    (open mask register to gain access
                                 to search limits)
nnnnnn/ØØØØØØ  1,Ø↓             (sets lower limit to 1ØØØ)
nnnnnn/ØØØØØØ  2,-2             (sets upper limit to 1776)
*Ø;C=ØØØØØØ                     (constant register set to zero)
*;F                             (locations 1ØØØ-1776 set to zero)
*$M/ØØØØØØ↓
nnnnnn/ØØ1ØØØ  2,Ø↓             (sets lower limit to 2ØØØ)
nnnnnn/ØØ1776  3,-1             (sets upper limit to 2777)
*4Ø;C=ØØØØ4Ø                    (constant register set to 4Ø
                                 (SPACE))
*;I                             (byte locations 2ØØØ-2777 are set
*                                to value in low order 8 bits of
                                 constant register)
```

2.9 CALCULATING OFFSETS r;O

Relative addressing and branching involve the use of an offset--the number of words or bytes forward or backward from the current location to the effective address. During the debugging session it may be necessary to change a relative address or branch reference by replacing one instruction offset with another. ODT calculates the offsets in response to its r;O command.

The command r;O causes ODT to print the 16-bit and 8-bit offsets from the currently open location to address r. For example:

```
*346/ØØØØ34  414;O    ØØØØ44  Ø22  22
*/ØØØØ22
```

In the example, location 346 is opened and the offsets from that location to location 414 are calculated and printed. The contents of location 346 are then changed to 22 (the 8-bit offset) and verified on the next line.

The 8-bit offset is printed only if it is in the range $-128_{1Ø}$ to $127_{1Ø}$ and the 16-bit offset is even, as was the case above. For example, the offset of a relative branch is calculated and modified as follows.

```
*1Ø34/1Ø3421  1Ø34;O  177776  377\Ø21  377
*1Ø3777
```

Note that the modified low-order byte 377 must be combined with the unmodified high-order byte.

2.10 RELOCATION REGISTER COMMANDS r;nR, ;nR, ;R

The use of the relocation registers has been defined in Section 11-1.1. At the beginning of a debugging session it will be desirable to preset the registers to the relocation biases of those relocatable modules that will be receiving the most attention.

This can be done by typing the relocating bias, followed by a semicolon and the specification of relocation registers.

```
r;nR
```

r may be any relocatable expression and n is an integer from Ø to 7. If n is omitted it is assumed to be Ø.

As an example:

```
*1ØØØ;5R                    (puts 1ØØØ into relocation register 5)
*5,1ØØ;5R                   (effectively adds 1ØØ to the contents
*                           of relocation register 5)
```

In certain uses, programs may be relocated to an address below that at which they were assembled. This could occur with PIC code which is moved without the use of the linker.

In this case the appropriate relocation bias would be the 2's complement of the actual downward displacement. One method for easily evaluating the bias and putting it in the relocation register is illustrated in the following example.

Suppose the program was assembled at location 5ØØØ and was moved to location 1ØØØ. Then the sequence

```
*1ØØØ;1R
*1,-5ØØØ;1R
*
```

puts the 2's complement of 4ØØØ in relocation register 1, as desired.

Relocation registers are initialized to -1, so that unwanted relocation registers will never enter into the selection process when ODT searches for the most appropriate register.

To set a relocation register to -1, type ;nR. To set all relocation registers to -1, type ;R.

ODT maintains a table of relocation registers, beginning at the address specified by $R. Opening $R ($R/) opens relocation register Ø. Successively typing the LINE FEED key opens the other relocation registers in sequence. When a relocation register is opened in this way, it may be modified just as any other memory location.

2.11 THE RELOCATION CALCULATORS nR and n!

When a location has been opened, it is often desirable to relocate the relocated address and the contents of the location back to their relocatable values. To calculate the relocatable address of the opened location relative to a particular relocation bias, type n!, where n specifies the relocation register. This calculator works with opened bytes and words. If n is omitted, the relocation register whose contents are closest but less than or equal to the opened location is selected automatically by ODT. In the following example, assume that these conditions are fulfilled by relocation register 2, which contains 2ØØØ. To find the most likely module that a given opened byte is in, use n! as shown here.

```
*25ØØ\Ø11=   !=2,ØØØ5ØØ
```

Typing nR after opening a word causes ODT to print the octal number which equals the value of the contents of the opened location minus the contents of relocation register n. If n is omitted, ODT selects the relocation register whose contents are closest but less than or equal to the contents of the opened location. For example, assume the relocation bias stored in relocation register 1 is ØØ1234; use nR as shown here.

```
*1,5ØØ/Ø2455Ø     1R=1,Ø23314
```

The value 23314 is the content of 1,5ØØ, relative to the base 1234. An example of the use of both:

If relocation register 1 contains 1ØØØ, and relocation register 2 contains 2ØØØ, then to calculate the relocatable addresses of relocation 3ØØØ and its content, relative to 1ØØØ and 2ØØØ, the following can be performed.

```
*3ØØØ/ØØ567Ø 1!=1,ØØ2ØØØ 2!=2,ØØ1ØØØ 1R=1,ØØ467Ø 2R=2,ØØ367Ø
```

2.12 ODT PRIORITY LEVEL $P

$P represents a location in ODT that contains the priority level at which ODT operates. If $P contains the value 377, ODT will operate at the priority level of the processor at the time ODT is entered. Otherwise $P may contain a value between Ø and 7 corresponding to the fixed priority at which ODT will operate.

To set ODT to the desired priority level, open $P. ODT will print the present contents, which may then be changed.

```
*$P/ØØØØØ6    377
*
```

If $P is not specified, its value will be seven.

Breakpoints may be set in routines at different priority levels. For example, a program running at low priority level may use a device service routine which operates at a higher priority level. If a breakpoint occurs from a low priority routine, if ODT operates at a low priority, and if an interrupt does occur from a high priority routine, then the breakpoints in the high priority routine will not be executed since they have been removed.

2.13 ASCII INPUT AND OUTPUT r;nA

ASCII text may be inspected and changed by the command

r;nA

where r is a relocatable expression, and n is a character count. If n is omitted it is assumed to be 1. ODT prints n characters starting at location r, followed by a <CR> <LF>. You may then type one of the following.

a. <CR> ODT outputs <CR> <LF> <*> and waits for another command.

b. <LF> ODT opens the byte following the last byte output.

c. <up to n characters of text>

ODT inserts the text into core, starting at location r.

If less than n characters are typed, you must terminate the command by typing CTRL/U, causing <CR> <LF> <*> to be output as in case a. above. However, if exactly n characters are typed, ODT responds with <CR> <LF> <address of next available byte> <CR> <LF> <*>.

Note that n may actually be expressed as a relocatable expression, and could be quite large, accidentally. There is no safeguard against this in ODT.

2.14 RETURN TO MONITOR CTRL/C

If ODT is awaiting a command, a CTRL/C from the keyboard will simulate a TTY interrupt and call the Monitor. The Monitor responds with a ↑C on the terminal and returns to ODT at a wait loop. See Chapter 3-3 of this handbook (DOS/BATCH Monitor) for a more detailed description of CTRL/C.

2.15 ERROR DETECTION

ODT informs the user of two types of error: illegal or unrecognizable command and bad breakpoint entry. ODT does not check for the legality of an address when commanded to open a location for examination or modification. Thus the command:

```
177774/
```

will reference nonexistent memory, thereby causing a trap through the vector at location 4. If this vector has not been properly initialized, unpredictable results will occur.

Similarly, a command such as

```
$2Ø/
```

which references an address eight times the value represented by $2, may cause an illegal (nonexistent) memory reference.

Typing something other than a legal command will cause ODT to ignore the command, print

```
*?
```

and wait for another command. Therefore, to cause ODT to ignore a command just typed, type any illegal character (such as 9 or RUBOUT) and the command will be treated as an error, i.e., ignored.

ODT suspends program execution whenever it encounters a breakpoint, i.e., a trap to its breakpoint routine. If the breakpoint routine is entered and no known breakpoint caused the entry, ODT prints

```
BEØØ1542
*
```

and waits for another command. In the example above, BEØØ1542 denotes a bad entry from location ØØ1542. A bad entry may be caused by an illegal trace trap instruction, setting the T-bit in the status register, or by a jump to the middle of ODT.

PART 11

CHAPTER 3

PROGRAMMING CONVENTIONS

Information in this chapter is not necessary for the efficient use of ODT. However, it provides a better understanding of how ODT performs some of its functions.

3.1 FUNCTIONAL ORGANIZATION

The internal organization of ODT is almost totally modularized into independent subroutines. The internal structure consists of three major functions: command decoding, command execution, and various utility routines.

The command decoder interprets the individual commands, checks for command errors, saves input parameters for use in command execution, and sends control to the appropriate command execution routine.

The command execution routines take parameters to be saved by the command decoder and use the utility routines to execute the specified command. Command execution routines exit either to the object program or back to the command decoder.

The utility routines are common routines such as SAVE-RESTORE and I/O. They are used by both the command decoder and the command executers. (See Figure 11-1.)

3.2 BREAKPOINTS

The function of a breakpoint is to give control to ODT whenever the user program tries to execute the instruction at the selected address. Upon encountering a breakpoint, the user can utilize all of the ODT commands to examine and modify his program.

When a breakpoint is executed, ODT removes all the breakpoint instructions from the user's code so that the locations may be examined and/or altered. ODT then types a message to the user of the form Bn;k where k is the breakpoint address (and n is the breakpoint number). The breakpoints are automatically restored when execution is resumed.

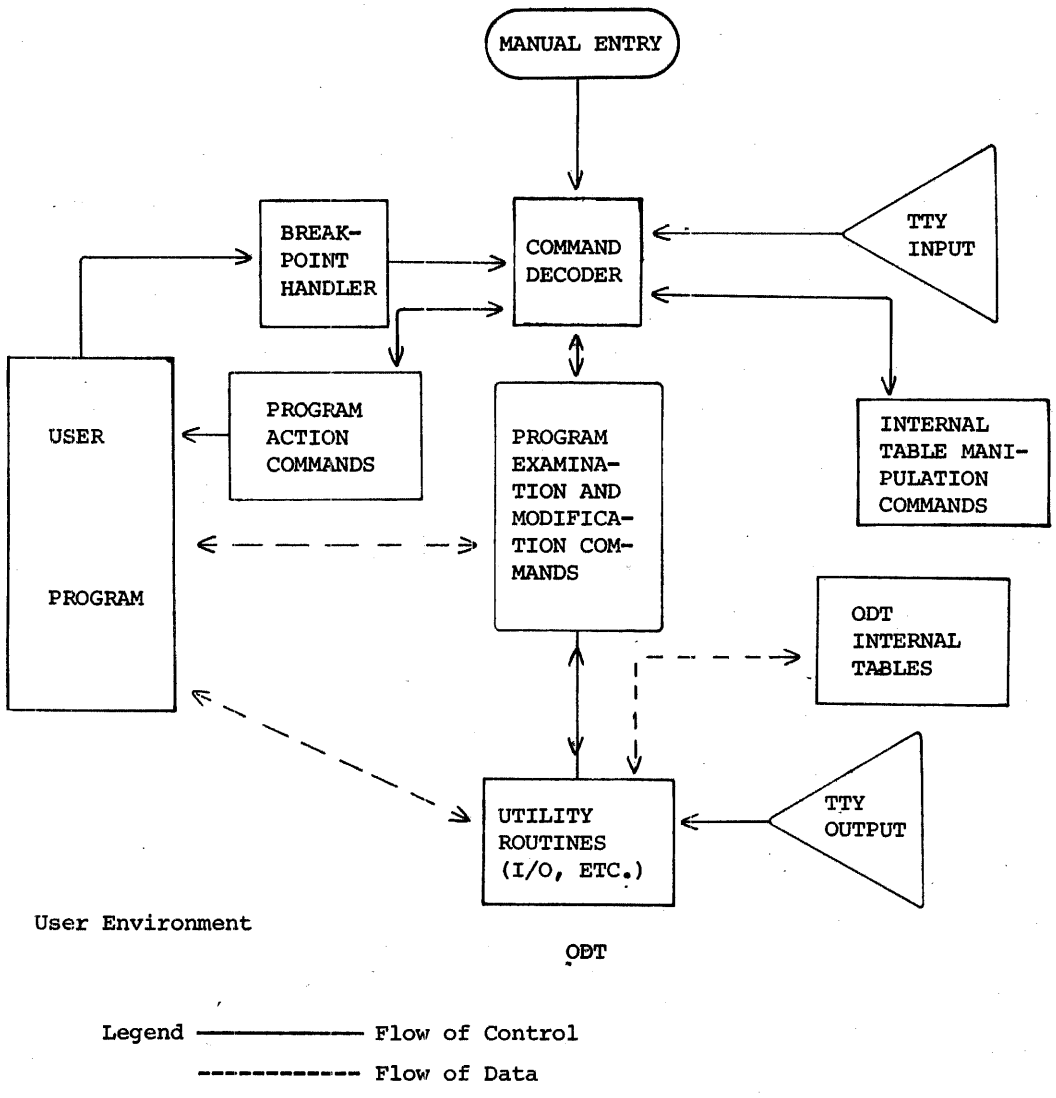

Figure 11-1

ODT Communication and Data Flow

A major restriction in the use of breakpoints is that the word where a breakpoint has been set must not be referenced by the program in any way since ODT has altered the word. Also, no breakpoint should be set at the location of any instruction that clears the T-bit. For example:

```
MOV  #240,177776          ;SET PRIORITY TO LEVEL 5
```

Note that instructions that cause or return from traps (e.g., EMT, RTI) are likely to clear the T-bit, since a new word from the trap vector or the stack will be loaded into the status register.

A breakpoint occurs when a trace trap instruction (placed in the user program by ODT) is executed. When a breakpoint occurs, the following steps are taken.

1. Set processor priority to seven (automatically set by trap instruction).
2. Save registers and set up stack.
3. If internal T-bit trap flag is set, go to step 13.
4. Remove breakpoints.
5. Reset processor priority to ODT's priority or user's priority.
6. Make sure a breakpoint or single-instruction mode caused the interrupt.
7. If the breakpoint did not cause the interrupt, go to step 15.
8. Decrement repeat count.
9. Go to step 18 if nonzero; otherwise reset count to one.
10. Save terminal status.
11. Type message to user about the breakpoint or single-instruction mode interrupt.
12. Go to command decoder.
13. Clear T-bit in stack and internal T-bit flag.
14. Jump to the GO processor.
15. Save terminal status.
16. Type BE (bad entry) followed by the address.

17. Clear the T-bit, if set, in the user status and proceed to the command decoder.

18. Go to the proceed processor, bypassing the TTY restore routine.

Note that steps 1-5 inclusive take approximately 1ØØ microseconds during which time interrupts are not permitted to occur (ODT is running at level 7).

When a proceed (;P) command is given, the following occurs.

1. The proceed is checked for legality.

2. The processor priority is set to seven.

3. The T-bit flags (internal and user status) are set.

4. The user registers, status, and program counter are restored.

5. Control is returned to the user.

6. When the T-bit trap occurs, steps 1, 2, 3, 13, and 14 of the breakpoint sequence are executed, breakpoints are restored, and program execution resumes normally.

When a breakpoint is placed on an IOT, EMT, TRAP, or any instruction causing a trap, the following occurs.

1. When the breakpoint occurs as described above, ODT is entered.

2. When ;P is typed, the T-bit is set and the IOT, EMT, TRAP, or other trapping instruction is executed.

3. This causes the current PC and status (with the T-bit included) to be pushed on the stack.

4. The new PC and status (no T-bit set) are obtained from the respective trap vector.

5. The whole trap service routine is executed without any breakpoints.

6. When an RTI is executed, the saved PC and PS (including the T-bit) are restored. The instruction following the trap-causing instruction is executed. If this instruction is not another trap-causing instruction, the T-bit trap occurs, causing the breakpoints to be reinserted in the user program, or the single-instruction mode repeat count to be decremented. If the following instruction, is a trap-causing instruction, this sequence is repeated starting at step 3. All exits from the trap handler must be via the RTI instruction. Otherwise, the T-bit will be lost. ODT will not gain control again since the breakpoints have not yet been reinserted.

Note that the ;P command is illegal if a breakpoint has not occurred (ODT will respond with ?); ;P is legal, however, after any trace trap entry.

The internal breakpoint status words have the following format.

1. The first eight words contain the breakpoint addresses for breakpoints Ø-7. (The ninth word contains the address of the next instruction to be executed in single-instruction mode.)

2. The next eight words contain the respective repeat counts. (The following word contains the repeat count for single-instruction mode.)

These words may be changed at will by the user, either by using the breakpoint commands or by direct manipulation with $B.

When program runaway occurs (that is, when the program is no longer under ODT control, perhaps executing an unexpected part of the program where a breakpoint has not been placed) ODT may be given control by pressing the HALT key to stop the machine and restart ODT (see Section 11-4.2). ODT will print *, indicating that it is ready to accept a command.

If the program being debugged uses the terminal for input or output, the program may interact with ODT to cause an error since ODT uses the terminal as well. This interactive error will not occur when the program being debugged is run without ODT. The user should note the following when using the terminal as an I/O device under ODT.

1. If the terminal interrupt is enabled upon entry to the ODT break routine, and no output interrupt is pending when ODT is entered, ODT will generate an unexpected interrupt when returning control to the program.

2. If the interrupt of the terminal reader (the keyboard) is enabled upon entry to the ODT break routine, and the program is expecting to receive an interrupt to input a character, both the expected interrupt and the character will be lost.

3. If the terminal reader (keyboard) has just read a character into the reader data buffer when the ODT break routine is entered, the expected character in the reader data buffer will be lost.

3.3 SEARCH

The word search allows the user to search for bit patterns in specified sections of memory. Using the $M/ command, the user specifies a mask, a lower search limit ($M+2), and an upper search limit ($M+4). The search object is specified in the search command itself.

The word search compares selected bits (where ones appear in the mask) in the word and search object. If all of the selected bits are equal, the unmasked word is printed.

The search algorithm:

1. Fetch a word at the current address.
2. XOR (exclusive OR) the word and search object.
3. AND the result of step 2 with the mask.
4. If the result of step 3 is zero, type the address of the unmasked word and its contents. Otherwise, proceed to step 5.
5. Add two to the current address. If the current address is greater than the upper limit, type * and return to the command decoder, otherwise go to step 1.

Note that if the mask is zero, ODT will print every word between the limits, since a match occurs every time (i.e., the result of step 3 is always zero).

In the effective address search, ODT interprets every word in the search range as an instruction that is interrogated for a possible direct relationship to the search object. The mask register is opened only to gain access to the search limit registers.

The algorithm for the effective address search (where (X) denotes contents of X, and K denotes the search object):

1. Fetch a word at the current address X.
2. If (X)=K [direct reference], print contents and go to step 5.
3. If (X)+X+2=K [indexed by PC], print contents and go to step 5.

4. If (X) is a relative branch to K, print contents.

5. Add two to the current address. If the current address is greater than the upper limit, perform a carriage return/line feed and return to the command decoder; otherwise, go to step 1.

3.4 TERMINAL INTERRUPT

Upon entering the TTY SAVE routine, the following occurs.

1. Save the status register (TKS).

2. Clear interrupt enable and maintenance bits in the TKS.

3. Save the TTY status register (TPS).

4. Clear interrupt enable and maintenance bits in the TPS.

To restore the TTY:

1. Wait for completion of any I/O from ODT.

2. Restore the TKS.

3. Restore the TPS.

WARNINGS

If the TTY printer interrupt is enabled upon entry to the ODT break routine, the following may occur.

1. If no output interrupt is pending when ODT is entered, an additional interrupt will always occur when ODT returns control to the user.

2. If an output interrupt is pending upon entry, the expected interrupt will occur when the user regains control.

If the TTY reader (keyboard) is busy or done, the expected character in the reader data buffer will be lost.

If the TTY reader (keyboard) interrupt is enabled upon entry to the ODT break routine, and a character is pending, the interrupt (as well as the character) will be lost.

PART 11

CHAPTER 4

OPERATING PROCEDURES

This section describes loading procedures for ODT, restarting and reentering procedures, and error recovery.

4.1 LOADING PROCEDURES

ODT is supplied as a relocatable object module. It should be linked with other object modules which form the program to be debugged. The resultant load module is loaded into core memory using the System Loader, as explained in Part 3, DOS/BATCH Monitor.

4.2 STARTING AND RESTARTING

After loading the load module (including ODT) into core via the Monitor GET command, ODT may be started by means of the Monitor command OD. ODT indicates its readiness to accept input by printing the following.

```
ODT11R Vnnnn          (nnnn is the ODT version number)
*
```

When ODT is started at its start address, the SP register is set to an ODT internal stack, registers RØ-R6 are saved in $Ø-$6, the terminal status and the processor status information are saved, and the trace trap vector is initialized. If ODT is started at its start address after breakpoints have been set in a program, ODT will forget about the breakpoints and will leave the program modified; i.e., the breakpoint instructions will be left in the program.

ODT's start address is determined from the LINK's load map; i.e., the low limit address plus 172_8 equals ODT's start address.

There are two ways of restarting ODT.

1. Restart at start address + 2 (use the DOS/BATCH command OD R).
2. Re-enter at start address + 4 (use the DOS/BATCH command OD K).

To restart, use the Monitor command OD R. A restart will save the general registers, clear the relocation registers, remove the breakpoint instructions from the user program, and then forget all breakpoints; i.e., simulate the ;B command.

To re-enter, use the Monitor command OD K. A re-enter will save the processor status and general registers, remove the breakpoint instructions from the user program, and ODT will type the BE (bad entry) error message. ODT will remember which breakpoints were set and will reset them on the next ;G command (;P is illegal after a bad entry).

4.3 USING ODT WITH STAND-ALONE SYSTEMS

Since DOS/BATCH supports console devices with various fill character counts and external page addresses, it is necessary for ODT to have this information. In the normal case ODT obtains this information from the DOS/BATCH SVT table. If ODT is to be used in a nonDOS/BATCH environment, the user program must supply the necessary information. This is done by defining a global symbol $$CON that has in that location the fill count, and in the following location the terminal status register address. The fill count is set by turning on successive high-order bits for each null required on <CR><LF>. For example, if the word contained 17ØØØØ, 4 nulls would be inserted; if the word contained 177777, the maximum number of nulls (16) would be inserted.

Example:

```
$$CON:    .WORD    174ØØØ          ;5 null fill
          .WORD    177564          ;external page addr.
```

NOTE

The word specified by the global $$CON is used by ODT to determine whether or not to access the DOS/BATCH SVT table. It must not be used in user programs for any purpose other than that described above or ODT will not operate correctly.