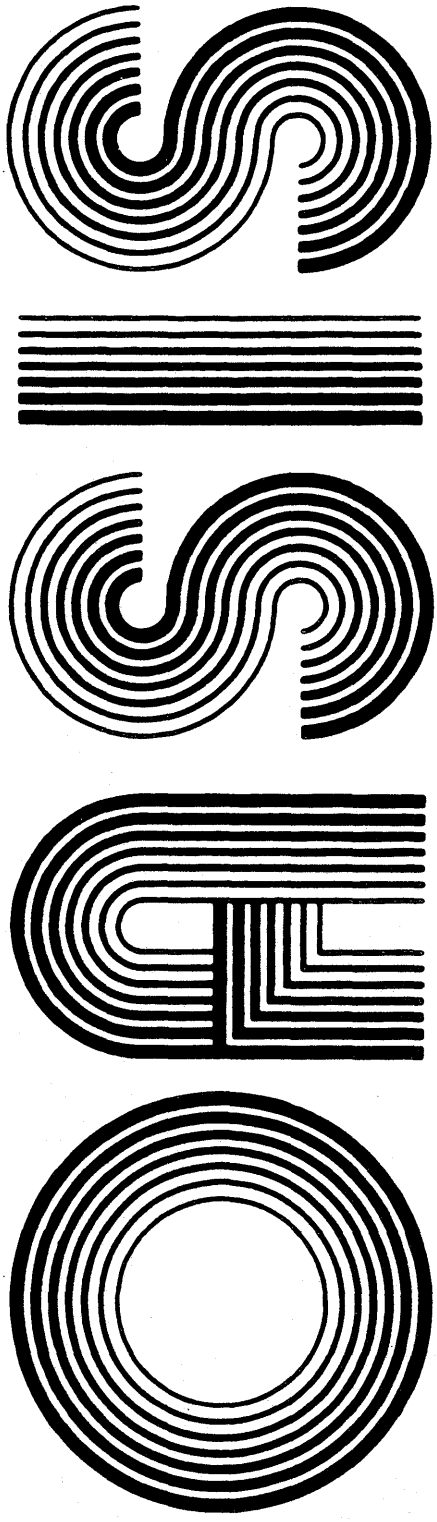
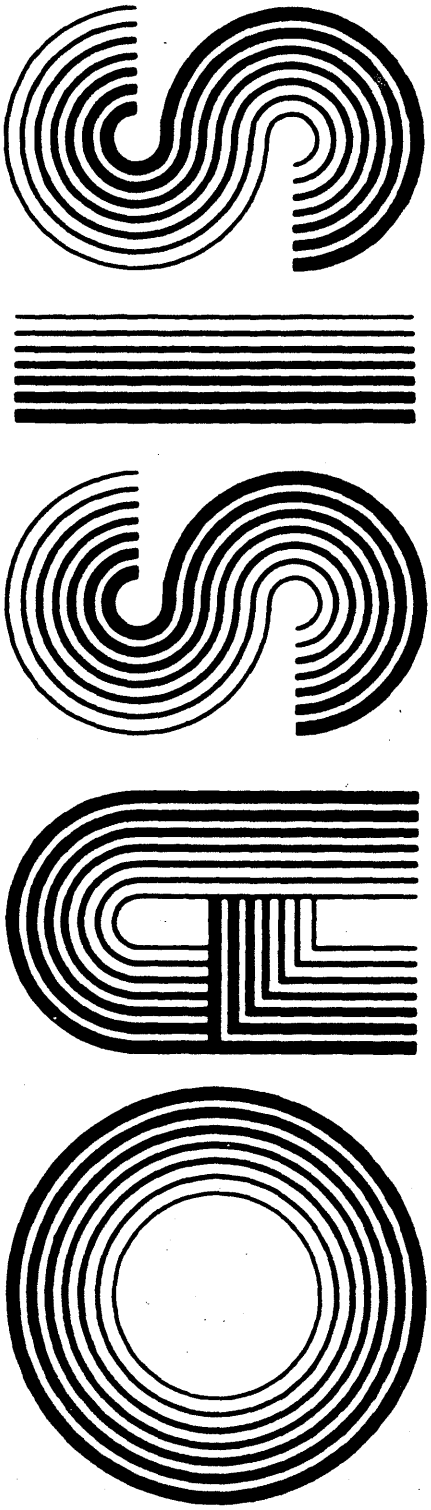


DYNAMIC DEBUGGING REFERENCE MANUAL

OPERATING SYSTEM SOFTWARE

MAKES MICROS RUN LIKE MINIS

PHASE ONE
SYSTEMS, INC.



DYNAMIC DEBUGGING REFERENCE MANUAL

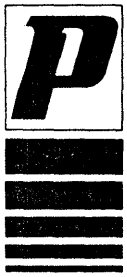
Second Edition

Revised

Documentation by: C. P. Williams
Software by: Timothy S. Williams

OPERATING SYSTEM SOFTWARE

MAKES MICROS RUN LIKE MINIS



PHASE ONE SYSTEMS, INC.

7700 EDGEWATER DRIVE SUITE 830
OAKLAND, CALIFORNIA 94621 USA

P R E F A C E

This manual describes the OASIS assembly language Debugger. It provides sufficiently detailed information necessary to the use of this Debugger in conjunction with the OASIS Operating System.

This manual, named DEBUG , like all OASIS documentation manuals, has the manual name and revision number (if applicable) in the lower, inside corner of each page of the body of the manual. In most chapters of the manual the last primary subject being discussed on a page will be identified in the lower outside corner of the page.

Related Documentation

The following publications provides additional information that may be required in the use of the OASIS Debugger:

OASIS System Reference Manual

OASIS MACRO Assembler Reference Manual

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The DEBUG command allows you to perform on-line, interactive debugging of a program. The format of the DEBUG command is:

DEBUG

When this command is executed the DEBUG program is loaded into high memory and executed. When the DEBUG program is first executed it will display on the console:

```
OASIS Z80 Debugger version n.n

To exit DEBUG, use ESC-Q; to re-enter, use ESC-D.
Type "HELP" to list command syntax.
```

1.1 DEBUG Prompting Character

When the DEBUG command is in control the equal sign (=) character will be displayed at the left side of the console. This is the prompting character for the DEBUG program and indicates that the DEBUG program is waiting for operator input.

Once the Debugger has been loaded the user may return control to the CSI by using the GO command. When the user wishes to re-enter the Debugger he must type the System Debug-key. This will transfer control to the Debugger and the program counter (PC) will be displayed.

1.2 System Control Keys

When the Debugger is in control the System Debug-key is inoperative as this would cause a break into the Debugger itself. Instead, the Program Cancel-key may be used to quit the current operation of the Debugger and return to the Debug command mode. For example, a Trace of several thousand instructions can be aborted by typing the Program Cancel-key.

The System Cancel-key should be used to quit the program being debugged, return control to the CSI but leave the Debugger loaded in memory. The Debugger may be re-entered by using the System Debug-key.

It is not advised that you use the GO 0 command to return control to the CSI because the program that was being debugged may have set a QET to perform required clean-up duties. The best procedure for returning control to the CSI is to issue a GO command followed by the entry of the System Cancel-key.

1.3 Protected Memory

The Debugger will not allow you to examine or change memory areas outside of the user area. (The user area is the contiguous memory area from the end of the NUCLEUS to the beginning of device drivers, the Debugger, etc.) When an attempt is made to access these areas with the commands: ASM, DUMP, FILL, LIST, MOVE, READ, SAVE, SEARCH, SET, TRACE, VERIFY, WRITE, the Debugger will not perform the operation but instead display "Protected Area".

1.4 DEBUG Commands

The commands available to the user when the DEBUG command is in control of the system are:

```
ASM     addr
BASE    [num[,org]]
CALC    expr (ops=+,-,*,/,\,&,!,^,<,>,~,-,@)
DUMP    [addr]
FILL    start,end,value
GO      [addr/*[,brk...]
INPUT   port[,start,end]
LIST    [addr]
LOAD    name[addr]
MOVE    start,end,to
OUTPUT  port,[value/start,end]
PAGE
PRINT
QUIT
READ    sect,addr[,count]
REG     [name value]
SAVE    name start,end
SEARCH  start,end,value
SELECT  drive
SET     addr[,value...]
TRACE   [addr[,][count]]
TRCALL
VERIFY  start,end,to
WRITE   sect,addr[,count]
.Z80inst      (immediate execute)
```

The above list of commands comprises the HELP message and will be displayed in response to the user typing HELP.

1.5 Numeric Expressions

The OASIS DEBUG utility allows the user to use numeric expressions wherever a numeric value may be used. Numeric expressions may contain numeric literals (assumed hexadecimal base), string literals (one or two characters enclosed within quotes), and arithmetic and logical operators (listed under the CALC command). Numeric expressions in the debugger have no hierarchy: strict left to right evaluation is performed.

The indirect address operator (@) may be used in one of two ways: prefix and postfix. When the operator is used with prefix notation (i.e., @6000) it means that the following numeric literal is the address of the value to be used. When the operator is used with postfix notation (i.e., 123+6000@) it means that the previous numeric expression is the address of the value to be used (123+6000@ is identical to @6123).

CHAPTER 2

DEBUG COMMANDS

2.1 Assemble Command

The DEBUG ASM command allows the user to assemble Z80 code directly into memory without using the system assembler. The format of the ASM command is:

ASM <addr-exp>

Where:

addr-exp Indicates the address that the assembled code is to be saved at.

When the ASM command is executed the Debugger will prompt the user with the address that the next instruction will be saved at. The user then may type the Z80 assembly code that he wishes assembled. The opcode and operand must be separated by at least one space. Labels are not allowed. Only one instruction per line is allowed. When the user is finished entering code a carriage return with no instruction preceding it will transfer control out of the assemble command.

2.2 Base Command

The DEBUG BASE command provides a means of debugging relocatable programs. The format of the BASE command is:

BASE [<n>[,<addr-exp>]]

Where:

n Indicate the "base register" number to be used. Base register numbers are in the range of 0 - 8. When the Debugger is first entered base register 0 is set with a base address of 0000H. Omitting the base register number will cause the display of all base registers defined with an asterisk (*) by the register number currently in use.

addr-exp Indicates the address that the base register indicated by the preceding number is to be set to.

For information describing the use of base registers see the appendix "Using Base Registers" at the back of this manual.

2.3 Calculate Command

The DEBUG CALC command allows the user to perform hexadecimal arithmetic and logical functions on hexadecimal numbers. The format of the CALC command is:

[CALC] <nnnn>[<op><nnnn>...]

Where:

nnnn Indicates a number (or ASCII characters) that the operation is to be performed on.

op Indicates the arithmetic or logic operation to be performed. Do not separate with spaces!

The available operators in the CALC command include:

Op	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Modulo
<	Shift left (second value indicates number of bits to shift)
>	Shift right (second value indicates number of bits to shift)
!	OR
&	AND
^	XOR Exclusive OR
-	Unary two's complement
~	Unary one's complement
@	Unary indirect address contents

For example:

```
=C 23*15
002DF, 735T, '..'
=C 23T*15T
00159, 345T, '.Y'
=C 23*15+3/2
00171, 369T, '.q'
=C 'A'
00041, 65T, 'A'
=C ' A'
02041, 8257T, ' A'
=C '      A'
02041, 8257T, ' A'
=C -12
0FFEE, 65518T, '..'
=C ~12
0FFED, 65517T, '..'
=C 7\3
00001, 1T, '.'
=C @4000
0FEC9, 65225T, '..'
=
```

2.4 Dump Command

The DEBUG DUMP command allows the user to display the contents of memory on the console terminal. The format of the display is identical to the CSI DUMPDISK command except that the header information is not displayed in the Debugger. The format of the DUMP command is:

DUMP [<addr-exp>]

Normally the user will specify the address that he wishes the display to start at. If the address is omitted then the dump will display memory starting with the last address DUMPed, LISTed, or ASseMbled. The DUMP command will display one screen of information, or 16 lines, whichever is less. A down arrow indicates that the dump is to continued; an up arrow indicates that the previous block of memory is to be displayed.

The format of the display of a memory dump is divided into three sections. The first section is the address of memory being displayed on that line. This address is the address of the first byte of data displayed on each line. The second section is the contents of memory for that address. The data is broken by spaces every four bytes for readablity. The third section is the conversion to ASCII of the line of data. Any byte that cannot be converted to an ASCII character will be displayed as a period (.).

For example:

```
=D 4000
4000: C9FE7F20 13CDD19B 1820FD36 0049FD23 '... ..... .6.I.#'
4010: C9FD3600 52FD23C9 FEBF2005 CDD89B18 '..6.R.#... .....'
4020: 09E6F01F 1F1F1FCD 419C8947 E60FCA4C '........A.vG...L'
4030: 9B4FFD36 002CFD23 78FE1A20 06CDB59B '.O.6.,.#x.. ....'
4040: C34C9BFE 7C2006CD D19BC34C 9BFE7E20 '.L..| .....L..~ '
4050: 06CDD89B C34C9B79 FE0F2018 CD0F9EC3 '.....L.y.. .....'
4060: 4C9BFD36 0028FD23 CDC59CFD 360029FD 'L..6.(.#....6.).'
4070: 23C34C9B CD419CC3 4C9B3D5F 87835F16 '#.L..A..L.=_.._.'
4080: 0021509C 193ABDAF E9C37D9C C3829CC3 '.!P..:....}.....'
4090: 269DC32D 9DC3459D C38E9DC3 9E9DC3AA '&..-..E.........'
40A0: 9DC3BA9D C3CA9DC3 E69DC3FC 9DC3C59C '................'
40B0: C3029EC3 079EE638 1F1F1FE6 07FE0620 '.......8....... '
40C0: 54FD3600 28FD23CD C59C3ABE AFB72827 'T.6.(.#...:...('
40D0: 2AB9AF3A BFAFB728 032B1804 2322B9AF '*..:...(.+..#"..'
40E0: 5E16003E 2BCB7B28 067BED44 5F3E2DFD '^..>+.{(.{.D^>-.'
40F0: 7700FD23 CD4F9DFD 360029FD 23C93ABE 'w..#.O..6.).#.:.'
=
```

2.5 Fill Command

The DEBUG FILL command allows you to fill a section of memory with a constant. The format of the FILL command is:

FILL <start-exp>,<end-exp>,<value-exp>

Where:

start-exp Indicates the first address to be filled.

end-exp Indicates the last address to be filled.

value-exp Is the value that the memory space is to be filled with.

2.6 Go Command

The DEBUG GO command allows the user a means of specifying that execution is to be resumed or started. The format of the GO command is:

GO [<addr-exp>|* [<brk-exp> ...]]

Where:

addr-exp Indicates the address that control is to be transfered to. This may be an asterisk or blank, indicating that execution is to continue at the point that it was interupted by the Debugger. An address of 0 will return control to the CSI (the Debugger will remain in memory).

brk-exp Indicates a breakpoint address. Up to eight breakpoints may be set at any one time. When the Debugger detects that a breakpoint address is about to be executed a break will occur--control will return to the Debugger and the breakpoint address will be displayed on the console. Breakpoints must be reset every time a GO command is executed because the Debugger clears all unused breakpoints when it regains control.

The GO command can be used to set a breakpoint that will be encountered when a COMMAND program is loaded and executed from the CSI.

To specify this type of breakpoint is to occur the Debugger is invoked and the operator types: GO,* In this command the * is a special indicator that tells the Debugger to break at the entry point address of the next COMMAND executed from the CSI.

2.7 Input Command

The INPUT comand provides an easy method of getting data from an input port. The format of the INPUT command is:

INPUT <port-exp>[,<start-exp>,<end-exp>]

Where:

port-exp Indicates the port number (physical device number) that the input is to come from.

start-exp Indicates the starting address of memory to be used for storage of the data input. When this field is not specified the data is displayed on the console and only one byte of data is accepted.

end-exp Indicates the ending address of memory to be used for storage of the data input. This field is used to determine the number of bytes of data to be input.

When the INPUT command is executed the port specified by <port> is read and the byte(s) is either displayed on the console or saved in memory.

2.8 List Command

The DEBUG LIST command allows the user to "dis-assemble" machine code into Z80 mnemonics. The format of the LIST command is:

LIST [<addr-exp>[,<# lines>]]

Where:

addr-exp Indicates the address that the dis-assembly is to begin at. If this field is not specified then the list will begin at the last address DUMPed, LISTed, or ASseMbled.

lines Indicates the number of dis-assembled lines of code to be displayed. When this field is not specified sixteen lines will be displayed.

The LIST command assumes that the starting address is the address of a Z80 opcode. Since almost all values could be interpreted as an opcode no error will be detected if the wrong address is specified. However the listing may be meaningless.

When the list command is executed one screen of dis-assembly will be displayed on the console device. The display is divided into four columns of information. The first column is the address of the opcode for that line. The second column contains the machine code representation of the instruction. The third column is the Z80 mnemonic of the opcode. The forth column is the operand of the instruction. Labels are displayed as absolute addresses.

All values displayed are hexadecimal.

Refer to the appendix "Using Base Registers" in the back of this manual for addition information regarding the LIST command.

For example:

```
=L 4000
4000 C9       RET
4001 FE7F     CP   7F
4003 2013     JR   NZ,4018
4005 CDD198   CALL 9BD1
4008 1820     JR   402A
400A FD360049 LD   (IY+0),49
400E FD23     INC  IY
4010 C9       RET
4011 FD360052 LD   (IY+0),52
4015 FD23     INC  IY
4017 C9       RET
4018 FEBF     CP   0BF
401A 2005     JR   NZ,4021
401C CDD89B   CALL 9BD8
401F 1809     JR   402A
4021 C9       RET
=
```

2.9 Load Command

The LOAD command provides an easy means of loading a program to be tested. The format of the command is:

LOAD fd [addr-exp]

Where:

fd Is the file description of the program to be loaded.

addr-exp Indicates the load address to be used. When this field is not specified the program will be loaded in its normal location.

2.10 Move Command

The MOVE command allows the user to move blocks of data in memory. The format of the MOVE command is:

MOVE <start-exp> <end-exp> <to-exp>

Where:

start-exp Indicates the first address that is to be moved.

end-exp Indicates the last address that is to be moved.

to-exp Indicates the destination address that the data is to be moved to.

When the MOVE command is executed the block of data starting with the start address is move to the area starting with to address, one byte at a time.

For example:

```
=D 5000
5000: 01020304 05060708 090A0B0C 0D0E0F10 ................
5010: 090A0B0C 0D0E0F10 00000000 00000000 ................
5020: 00000000 00000000 00000000 00000000 ................
.
.
.
=M 5000 500F 5008
=D 5000
5000: 01020304 05060708 01020304 05060708 ................
5010: 01020304 05060708 00000000 00000000 ................
.
.
.
```

2.11 Output Command

The OUTPUT command provides an easy method of outputting data to a port. The format of the command is:

OUTPUT <port-exp>,<value-exp>|<<start-exp>,<end-exp>-exp>

Where:

port-exp Indicates the port number (physical device number) to be accessed.

value-exp Indicates the 8-bit value to be output to the port.

start-exp Indicates the 16-bit memory address of the first byte to be output to the port.

end-exp Indicates the 16-bit memory address of the last byte to be output to the port.

When the OUTPUT command is executed no device drivers are accessed.

When only one number is specified following the port address the OUTPUT command will output that number (<value>) to the specified port. When two numbers are specified following the port address then these numbers are interpreted as memory addresses for the <start> and <end>. In this later situation the data in the locations specified is output to the port.

2.12 Page List Command

The DEBUG PAGE command, similar to the LIST command, disassembles and displays a portion of the program in memory. The PAGE command will display one full console screen of disassembled code. (The LIST command always displays sixteen lines of code.) The format of the PAGE command is:

PAGE

Refer to the DEBUG LIST command for information regarding the display format.

2.13 Print Command

The DEBUG PRINT command allows the user to specify that output from the debugging process is to be output on the printer device. The format of the PRINT command is:

PRINT

When this command is executed the first time all subsequent output generated by the debugger will be output to the device PRINTER1. The next time this command is executed the output from the debugger will be displayed on the console. The output generated by the program being debugged is not affected by this command.

2.14 Quit Command

The DEBUG QUIT command allows the user to unload the system Debugger and return control to the CSI. The format of the QUIT command is:

QUIT

When this command is executed the memory used by the Debugger is released and control returns to the Command String Interpreter. Any program in the memory area

will be lost (the process of reloading the Debugger causes most, or all, of the user memory to be changed).

It is possible that no memory will be returned to the user area by this command. This would happen if a new device was attached while the Debugger was loaded. This can be avoided (if known in advance) by first detaching any device drivers loaded since the DEBUG command was first loaded, then entering the debugger and executing the QUIT sub-command.

2.15 Read Disk Command

The READ command allows you to read a sector or sectors of data from the disk into memory. The format of the READ command is:

READ <sect-exp> <addr-exp> [<count-exp>]

Where:

sect-exp Indicates the relative sector number of the disk to be read. The disk drive number is specified by the SELECT command.

addr-exp Indicates the address in memory that the data is to be read into.

count-exp Indicates the number of contiguous sectors to be read. If this field is not specified then a value of one is assumed.

2.16 Register Command

The REG command allows you to display or set the Z80 registers. The format of the REG command is:

REG [<name> <value-exp>]

Where:

name Indicates the name of the register to set. If this field is not specified then all of the registers will be displayed.

value-exp Indicates the value that the register is to be set to.

For example:

```
=REG
E ZP  AF=0044 BC=0901 DE=1F7D HL=3108 PC=34BC XOR  A
 I=10 AF'C898 BC'FFFF DE'FFFF HL'FFFF SP=9083 IX=0000 IY=0000
(BC): 01020304 05060708 090A0B0C 0D0E0F10 '................'
(DE): 2100113E C3180403 C30811BE 20151803 '!..>........... .'
(HL): 7700FD23 CD4F9DFD 360029FD 23C93ABE 'w..#.O..6.).#.:.'
(SP): EC31

=REG AF 5A5A
=REG
E Z   AF=5A5A BC=0901 DE=1F7D HL=3108 PC=34BC XOR  A
 I=10 AF'C898 BC'FFFF DE'FFFF HL'FFFF SP=9083 IX=0000 IY=0000
(BC): 01020304 05060708 090A0B0C 0D0E0F10 '................'
(DE): 2100113E C3180403 C30811BE 20151803 '!..>........... .'
(HL): 7700FD23 CD4F9DFD 360029FD 23C93ABE 'w..#.O..6.).#.:.'
(SP): EC31
```

As can be seen from the examples the REG command displays all of the registers, including the alternate set, the stack pointer, index registers, program counter, the status of the flags, interrupt enable status, and the mnemonic of the next instruction to be executed.

The four lines following the register display are partial dumps of memory corresponding to the values contained in the BC, DE, HL, and SP register pairs.

Refer to the chapter "Z80 CPU Overview" in the MACRO Assembler Language Reference Manual for information regarding these registers and flags. Also refer to ZILOG's Z80 Technical Manual for more detailed information.

2.17 Save Program Command

The SAVE command provides an easy method of saving a program written with the debugger. The format of the command is:

SAVE fd <start-exp>,<end-exp>

Where:

fd Is the file description of the program to be saved.

start-exp Indicates the starting address of the program in memory.

end-exp Indicates the ending address of the program in memory.

2.18 Search Command

The SEARCH command provides a method to search memory for a specific value. The format of the command is:

SEARCH <start-exp>,<end-exp>,<value-exp>...

Where:

start-exp Indicates the starting address of the block of memory to be searched.

end-exp Indicates the ending address of the block of memory to be searched.

value-exp Is the value(s) to be searched for.

When the SEARCH command is executed the block of memory specified by the <start> and <end> fields will be searched for the <value> list. When a match occurs the address of the location that matched will be displayed on the console. When there is more than one match found in the block of memory searched the question 'Again?' will be asked. Any response other than an N will be interpreted as a yes response and the next matching location address will be displayed on the console. When no match occurs the DEBUG prompt character will be displayed.

2.19 Disk Select Command

The SELECT command provides the user with the ability to specify the disk drive that future reads or writes by the Debugger are to be performed on. The format of the SELECT command is:

SELECT <drv>

Where:

drv Indicates the logical disk drive code (A, B, etc.) that future disk reads or writes are to access.

2.20 Set Memory Command

The DEBUG SET command allows the user to change the contents of memory. The format of the SET command is:

SET <addr-exp> [,<value-exp>]...

Where:

addr-exp Indicates the starting address to set.

value-exp Indicates a list of values, separated by commas (or spaces), that the addresses are to be set to. When values are specified the SET command executes in immediate mode, returning to the command level when done. When a value is not specified the SET command will be in "set" mode. In this latter mode the Debugger will display the address to be set, followed by the current contents in hexadecimal and ASCII. At this point the Debugger awaits input from the user. The user may type the value that the address is to be set to or he may use the arrow commands to change the address to be set:

The up key (default CNTRL/Z) will decrement the address by one (back up)

The down key (default CNTRL/J) will increment the address by one (advance)

The right key (default CNTRL/F) will increment the address by one (advance)

The carriage return will exit from the "set" mode.

Any of the above commands may be preceded with a value indicating that the current address is to be set to that value and then the address is to be changed according to the command.

2.21 Trace Command

The TRACE command allows the user to "trace" the flow of execution of a program. The format of the TRACE command is:

TRACE [<addr-exp>[,][<count-exp>]]

Where:

addr-exp Indicates the address that execution and tracing is to begin at. When this field is not specified execution will begin at the current program counter. A comma may be used as a positional filler to allow the user to specify a count.

count-exp Indicates the number of consecutive instructions to be traced. When this field is not specified a value of one is assumed (single step mode).

When the TRACE command is in effect an instruction will be executed and then an abbreviated REG will be displayed. If the count has not been reached this sequence will be repeated.

The Debugger normally does not allow the tracing of the execution of a system call (SC) instruction as most of these are time critical.

For example:

```
>@15E0'
=T ,5
E ZP  AF=0044 BC=3F3E DE=4D97 HL=0017 PC=15D2' LD   A,(15FA')
E ZP  AF=0044 BC=3F3E DE=4D97 HL=0017 PC=15D5' OR   A
E ZP  AF=0044 BC=3F3E DE=4D97 HL=0017 PC=15D6' RET
E ZP  AF=0044 BC=3F3E DE=4D97 HL=0017 PC=15E3' JR   Z,15E0'
E ZP  AF=0044 BC=3F3E DE=4D97 HL=0017 PC=15E0' CALL 15D2'
```

2.22 Trace Call Command

The DEBUG TRCALL command provides an easy method of tracing through a call to a subroutine without tracing the subroutine itself. The format of the TRCALL command is:

TRCALL

When the TRCALL command is executed and the PC is addressing a CALL instruction the call will be traced but the subroutine called will not. If the PC is addressing an instruction that is not a CALL the TRCALL command acts like a TRACE command.

2.23 Verify Memory Command

The VERIFY command provides an easy means of comparing two blocks of memory for equality. The format of the command is:

VERIFY <start1-exp>,<end-exp>,<start2-exp>

Where:

start1-exp Indicates the starting address of the block of memory to compare against.

end-exp Indicates the ending address of the block of memory to compare against.

start2-exp Indicates the starting address of the block of memory to be compared.

When the VERIFY command is executed the Debugger compares the first byte of each block to the other. If the two bytes equal each other the address pointers for the blocks are incremented and the test continues. When the two bytes do not equal each other their respective addresses and contents are displayed on the console before the address pointers are incremented.

For example:

```
=V 3900,3904,4900
3900: 18, 4900: B5
3901: 33, 4901: C3
3902: 44, 4902: B3
3903: 45, 4903: B2
3904: 42, 4904: FD

=
```

The above example indicates a total mismatch between the locations 3900H and 4900H for 5 bytes.

2.24 Write Disk Command

The WRITE command allows the user to write a sector or sectors of data from memory onto disk. The format of the WRITE command is:

WRITE <sect-exp> <addr-exp> [<count-exp>]

Where:

sect-exp Indicates the relative sector number that the data is to be written to. The disk drive is selected with the SEL command.

addr-exp Indicates the first address of data that is to be written to disk.

count-exp Indicates the number of contiguous sectors to be written. If this field is not specified then a value of one is assumed.

Caution: The integrity of the disk is the responsibility of the user when this command is used.

2.25 Immediate Instruction Command

The OASIS Debugger allows any valid Z80 instruction to be executed in immediate mode. To execute an immediate instruction you type a period followed by the Z80 mneumonic of the instruction to be executed. Use a space to separate the opcode from the operand. Any synonyms described in the MACRO Assembler Reference Manual are not valid as immediate instructions (i.e. ADD D is okay, not ADD A,D).

If the instruction is valid it will be executed, the primary registers will be displayed and the Debug prompt character will be displayed again.

For example:

```
=REG
E     AF=0000 BC=0000 DE=0000 HL=0000 PC=1100 JP   02393
 I=10 AF'0000 BC'0000 DE'0000 HL'0000 SP=A2DB IX=0000 IY=0000
(BC): 2100113E C3180403 C30811BE 20151803 '!..>........ ...'
(DE): 2100113E C3180403 C30811BE 20151803 '!..>........ ...'
(HL): 2100113E C3180403 C30811BE 20151803 '!..>........ ...'
(SP): EC31
=.LD A,FF
Invalid Command
=.LD A,0FF
E     AF=FF00 BC=00000 DE=0000 HL=0000 PC=1100 JP   02393
=
```

(This page intentionally left blank)

APPENDIX A

USING BASE REGISTERS

The OASIS MACRO Assembler supports the generation of relocatable programs. This is a very convenient and powerful feature but causes difficulties during the development and debugging of the relocatable programs as the listing generated by the assembler displays addresses relative to the origin of the program and not the addresses that the program will be executing at.

There are two features of the OASIS dynamic debugging program that greatly ease this problem. One is the GO command used with a breakpoint of * (i.e., GO,*). This causes a breakpoint to be set at the origin of the next command to be executed. Whenever a breakpoint is encountered by the debugger the break address is displayed on the console. At this point you will know the origin address of the program to be debugged and you could use this address to debug your program by always adding it (plus 3 for the jump instruction placed at the beginning of your program by the linkage editor) to the addresses listed by the assembler. This is not too difficult to do and most systems that support relocatable programs require you to do so.

However, a second feature in the OASIS debugger alleviates this requirement. The BASE command gives you access to nine internal base registers with eight of them settable by the user (base register 0 is always defined with an address of 0000H).

Say that the GO,* command was used followed by the execution of the program to be debugged. Assuming that the breakpoint address displayed was 02E00H you would then enter the command: BASE 1 2E03. From this point on, until you specified another base register to be used, all instructions listed by the LIST or TRACE command would be displayed with the same addresses as those listed in the assembly listing of your program. Any time that you wanted to know the absolute or execution addresses merely enter the command: BASE 0. This resets the base register to zero which has an offset address of zero.

To determine what the base registers are currently set to and which base register is currently being used enter the command: BASE.

For example:

```
=BASE 1 2E03
=BASE 7 3F00
=BASE 8 2FF0
=BASE
1  2E03
7  3F00
8* 2FF0
=
```

The * in the above example indicates that base register 8 is the current base register.

The availability of multiple base registers provides the capability of debugging programs with multiple program address blocks (PABs). In that situation a base register would be defined for each PAB.

When a base register other than zero is defined all addresses displayed by the Debugger that have been adjusted to reflect that base register will be followed by a single quote character (') or a period character (.). Similarly, when you specify an address that is to be treated as the address specified plus the base register offset, you must follow the address with a single quote or period character.

For example:

```
=BASE 0
=REG
E ZP  AF=0044 BC=3F3E DE=4D97 HL=0017 PC=15D2 JP   15D2
=BASE 1 1100
=REG
E ZP  AF=0044 BC=3F3E DE=4F97 HL=0017 PC=04D2' JP    04D2'
=REG HL 17'
=REG
E ZP  AF=0044 BC=3F3E DE=4F97 HL=1117 PC=04D2' JP    04D2'
=
```

The contents of the registers, with the exception of the PC register, are not offset by the base register as it is not determinable whether the contents are data or addresses.

(This page intentionally left blank)

APPENDIX B

DEBUGGING DEVICE DRIVER ROUTINES

Debugging device drivers presents difficulties for two reasons: the driver is relocated by the ATTACH command into high memory and is protected from access by the debugger.

To properly debug a user written device driver you should link it together with your test program and debug it as a command, calling the entry points as subroutines instead of using the system calls.

After a user written device driver has been tested in the "stand alone" mode described above you can re-link it as a system device driver (using the SYSTEM option of the LINK command) and it will be treated by the operating system as a standard device driver.

APPENDIX C

DEBUG COMMAND SUMMARY

ASM <addr>	Assemble Z80 mnemonics
BASE [<n>[,<addr>]]	Set/display base registers
CALC [<nnnn>][<op>]<nnnn>	Calculate values
DUMP [<addr>]	Display memory
FILL <start>,<end>,<value>	Fill memory with constant
GO [<addr>][,<brk>]...	Execute with optional breakpoints
INPUT <port>[,<start>,<end>]	Input data from I/O port
LIST [<addr>[,<lines>]]	Dis-assemble memory
LOAD <fd>,[<addr>]	Load program into memory
MOVE <start>,<end>,<to>	Move data in memory
OUTPUT <port>,<value>|<<start>,<end>>	Output data to I/O port
PAGE	Dis-assemble one screen of memory
PRINT	Output debug info on printer
QUIT	Exit and unload Debugger
READ <sect>,<addr>[,<count>]	Read data from disk
REG [<reg>,<value>]	Set/display Z80 registers
SAVE <fd>,<start>,<end>	Save program from memory
SEARCH <start>,<end>,<value>...	Search memory for data
SELECT <drv>	Select disk drive to be used
SET <addr>[,<value>]...	Set memory to value
TRACE [<addr>|*][,<count>]	Trace execution of program
TRCALL	Trace through subroutine call
VERIFY <start>,<end>,<with>	Compare two regions of memory
WRITE <sect>,<addr>[,<count>]	Write data to disk
.<inst>	Execute immediate mnemonic instruction