PART 6 DEBUG

The debugging package DEBUG is used to help in testing programs written in Assembler.

The user may define breakpoints where the package will suspend execution of the user program. At a breakpoint one or more DEBUG commands may be executed, allowing the user to examine or modify registers or memory locations within his program. This allows the user to check the program, section by section.

DEBUG makes temporary changes to the program under test, which must therefore be declared either memory-resident or swappable. A re-entrant program cannot be used and debugged simultaneously. The DEBUG package itself is re-entrant; it creates its workspace in the dynamic area of the program to be tested, so several users (up to a maximum of 16) may debug their programs simultaneously. However, it is not possible to debug a main program and its scheduled label routines at the same time, because the workspace may be corrupted between queueing a scheduled label and calling it for execution, and this will confuse the DEBUG package. It is in any case recommended not to use breakpoints within scheduled labels, as DEBUG performs all its input/output with wait, thus holding up execution of the user program.

STARTING DEBUG

DEBUG is called by the FCL command:

```
DEB <program-name>
```

which is described further in Chapter 11 of the MAS Manual.

DEBUG occupies 6K words in memory-resident area; it allocates a 2K word buffer in the dynamic area of each program to be debugged.

GENERAL

DEBUG establishes control of the user program by means of scheduled labels which handle any interrupts produced by it. The user may set up to eight breakpoints within the boundaries of the program under test. A breakpoint is a location within the program where execution will be suspended, allowing the user to perform DEBUG operations.

DEBUG's main functions are:

- Examine memory contents (DM command);
- Examine register contents (DR command);
- Modify memory contents (WM command);
- Modify register contents (WR command);
- Read data from an external device (RE command);
- Trace the execution of user program instructions (TR command);
- Restart the user program at a given address (GO command).

These functions may be executed either unconditionally or conditionally, by use of the IF command. Connecting the user program to a level is optional; if it is not done by the user it will be done by DEBUG.

ADDRESSING

Memory location addresses may be specified in one of three ways:

- Absolute
- Relative
- Symbolic.

Absolute Addressing

An absolute address is indicated by a slash (/), followed by up to four hexadecimal digits giving the address, e.g. /3A16 .

Relative Addressing

A relative address is specified as a displacement from the start address of the user program under test. It is indicated by an at sign (@) followed by up to four hexadecimal digits giving the address, e.g. @1B2E . Note that the first address of the program is @8.

Symbolic Adressing

A symbolic address may be specified in one of two ways, either as an entry point or as a symbol table name followed by a symbol (a label). In either case an optional decimal offset may also be specified.

An entry point address may only be specified if, when the program under test was Link-Edited, the parameter DBUG=ENTR was supplied to the OPT control statement. The syntax is:

    $<entry-point>[{+|-}<offset>]

For example:
$START
$ENT1+64
$ENT2-2

A label symbolic address may only be specified if the assembly directive STAB was specified for one or more modules at assembly time and the parameter DBUG=STAB, or the parameter DBUG=ENTR, was supplied to the OPT statement when the program under test was Link-Edited. The syntax is:

$<table-name>&<label>[{+|-}<offset>]

(in which <table-name> is the name specified in the END assembly directive of the module in which <label> is defined). For example:
$MOD1&LAB023
$MOD1&LAB031+64
$MOD2&LAB60-2

BREAKPOINTS

The user may specify up to eight breakpoints within his program. For each breakpoint one or more DEBUG commands may be specified; these commands will only be executed when the breakpoint is reached. Thus there are two modes of operation; "online mode", in which DEBUG commands are executed immediately, and "offline mode", in which DEBUG commands are stored for later execution.

Online Mode

This is the mode in which DEBUG starts operating. In this mode any DEBUG commands, except the IF command, may be entered and will be executed immediately the command is terminated by (CR)(LF). Entering the AT command switches DEBUG to offline mode.

Offline Mode

In this mode all DEBUG commands are stored for later execution when the breakpoint, with which they are associated, is reached. Entering any of the GO, RT or // commands causes the command to be stored, but also switches DEBUG back to online mode.

INPUT/OUTPUT

At initial entry, DEBUG reads all commands and data from, and outputs all its messages to, filecode /01. Commands entered from filecode /01 are echoed to filecode /02. If the system aborts the program under test, and the abort address is not one of the breakpoint addresses, DEBUG outputs the following information to filecodes /01 and /02:

- program status word;
- abort address, relative to the program start address;
- the contents of user registers A1 to A14;
- (to the line-printer only) a memory dump of the user area.

The DEBUG commands CI (Change Input Device) and CO (Change Output Device) may be used if it is required to input from, or output to, a device other than filecode /01. The filecodes given in these commands must have been assigned before the debugging package is called.

If DEBUG reads a command from a device other than the operator's console, and finds an error in it, it prints the command together with an error message on the console and then prompts the user to enter the correct command. The remaining commands are then read from the other input device.

DEBUG may read data from a specified device through the RE command. If the device is not the operator's console, its filecode must have been assigned before DEBUG is called.

Prompts

When DEBUG first takes control of a program, it assigns a two-character identifier to it. From this point onwards, all messages associated with this program are prefixed with this identifier; in this way, when DEBUG is being used on more than one test program, the user knows to which program each message belongs.

Filecodes

The DEBUG package uses filecodes /01 for the operator's console and /02 for the line-printer; these filecodes do not need to be assigned explicitly before DEBUG is started. Any other filecodes may be used, for alternative input or output devices, but they must be assigned before use.

PROGRAM ABORT

If the system aborts the program under test, a check is made that the abort address is not a checkpoint address. If it is, the command string associated with that checkpoint is executed. If it is not, the information given above (see "INPUT/OUTPUT") is printed. In either case DEBUG is put into online mode, so that the user may continue debugging.

MONITOR CALLS

The DEBUG package makes use of the following LKM instructions:

```
LKM  Special   (/2884 - planted at each breakpoint)
LKM  1         (Input/Output)
LKM  3         (Exit)
LKM  4         (Get Buffer)
LKM  5         (Release Buffer)
LKM  7         (Keep Control on Abort Condition)
LKM  25        (Read Unsolicited Operator Message)
LKM  35        (Get Name and Load Address of Test Program).
```

COMMAND SYNTAX

Each command consists of a two-character mnemonic, which may be followed by a space and one or more parameters. Each command with its parameters must be contained within a single line or input record; a continuation line is not possible. A continuation character (full stop) immediately after a command allows another command to be entered on the same line; this may be repeated until the line is full.

A command mnemonic must start at the first character position on a line, or the first character after a continuation character. Spaces after a command (and its parameters, if any) are not significant.

PARAMETER SYNTAX

If a command has more than one parameter, the second and succeeding parameters are each separated from the one preceding by one comma.

The following definitions are adopted in the description of the DEBUG commands:

<memref> ::= {<absolute-address> | <relative-address> | <symbolic-address>}

For the formats of these, see "ADDRESSING" in Chapter 2 of this Part. The IF command has a special format for absolute addresses; see the command description for details.

<register> ::= {R1 | R2 | ... | R14}

These refer to the programmer's general purpose registers, R1 to R14 inclusive. R0 and R15 may not be used.

<constant> ::= /<hexa-digit>... (up to four digits)

<filecode> ::= /<hexa-digit>[<hexa-digit>] (one or two digits)

If the same syntactic item occurs more than once within the syntax of a command, the occurrences are distinguished by appending a decimal digit to the description within the syntax brackets, for example:

DM <memref1>,<memref2>

AT COMMAND (Define Breakpoint)

Syntax: AT <memref>

The AT command is used to suspend temporarily the execution of the user program at the memory reference specified as a parameter, thus defining a breakpoint.

Once the AT command has been entered, DEBUG switches to "offline mode"; this permits the user to enter one or more DEBUG commands, which will be executed when the breakpoint at the specified location is reached. The final or only command must be one of GO, // or RT; depending on which of these is used, DEBUG will perform one of the following functions:

a) Last command is GO: the user program will resume execution, after DEBUG has executed the command(s) associated with that string.
b) Last command is //: the user program will restart from its execution start address.
c) Last command is RT: any pre-defined instructions associated with that breakpoint are executed and then the online (interactive) mode is entered. A prompt is output on the operator's console, informing the user that he may enter any DEBUG commands to be executed immediately, other than the IF command. The user may resume execution of his program by entering the GO command.

When a breakpoint is encountered, its absolute address is printed out in the format: BP: <absolute-address>

Breakpoints are held in a table, along with their associated command strings. The maximum number of breakpoints that may be defined at any one time is eight. However, if eight breakpoints already exist, the user may delete unwanted breakpoints by use of the DB command, thus allowing himself to create alternative definitions if required. Address specifications for breakpoints need not be defined in ascending order.

Restrictions

The breakpoints defined may not:
- be modified by the user program;
- refer to memory areas within the user program defined as DATA (the breakpoint is not executed);
- contain commands to define a new breakpoint.

It is not allowed to test a main program and its scheduled label routines simultaneously.

IF COMMAND (Conditional Execution of DEBUG Command)

Syntax:

IF {<memref1> | <register1>}<operation>{<memref2> | <register2> | <constant>}

<operation> ::= {> | = | <}

The IF command is used in conjunction with the AT command. It allows conditional execution of the command string attached to the breakpoint being executed. It need not immediately follow the AT command; any intervening commands will be executed unconditionally.

The contents of one general purpose register or memory location are compared with a constant or with the contents of another register or memory location. If the condition is TRUE the command string associated with this breakpoint is executed, otherwise an implicit GO command is generated to continue execution of the program from the breakpoint.

If a memory location is specified as an absolute address, then it must have the format: M nnnn

i.e. a letter M followed by a blank, followed by a hexa-decimal number of up to four characters without a preceding slash.

GO COMMAND (Restart User Program)

Syntax: GO [<memref>]

This command may be used in two ways:

a) In offline mode, during definition of a breakpoint. It will be the last command entered in the stored string of commands, to be executed when the breakpoint is reached; it also terminates the breakpoint definition and returns DEBUG to the online mode.
b) In online mode, entered when the last command executed in assocation with a breakpoint was the RT command. The user program is restarted at the point specified by <memref> or, if <memref> is not specified, at the breakpoint. If the user program has not yet been started by the // command, the GO command is rejected with the message:

   REFUSED IN ON-LINE MODE

The specified memory address must lie within the boundaries of the test program, and not within an area defined as DATA.

DB COMMAND (Delete Breakpoint)

Syntax: DB <memref>

The DB command is used to delete a previously defined breakpoint, together with the command string associated with it. A breakpoint may be deleted from the breakpoint table at any time; if the command is used to delete a currently executing breakpoint, deletion is postponed until a GO command is executed.

DM COMMAND (Dump Memory)

Syntax: DM <memref1>[,<memref2>]

The DM command allows the user to examine any memory area within the boundaries of the program under test. The dump is output on the operator's console, unless an alternative output device has been defined through the CO command.

The dump is presented eight words to a line. Each line is preceded by an absolute address as a multiple of /10. The dump begins at <memref1>, rounded down if necessary to an exact multiple of /10, and ends at <memref2>, rounded up if necessary to an exact multiple of /10 plus /E, thus filling the last line of the dump.

If <memref2> is omitted, one line is output including the contents of <memref1>.

The dump is in hexadecimal, and to the right of each line its character contents are also presented. To the left of each line the program name and its two-character code (in brackets) are given.

DR COMMAND (Dump Registers)

Syntax: DR [<register1>[,<register2>]]

The DR command allows the user to examine the contents of a range of the general purpose registers A1 to A14. Registers A0 and A15 cannot be examined.

If no parameters are specified, the contents of registers A1 to A14 inclusive will be dumped.

If only <register1> is specified, only the contents of that register will be dumped.

If both parameters are specified, the contents of <register1> to <register2> inclusive will be dumped. Note that <register1> must be less than <register2>, but not less than R1; <register2> must not be greater than R14.

The dump is in hexadecimal format and, unless an alternative output device has been previously defined by the CO command, will be output to the operator's console typewriter.

WM COMMAND (Write into Memory)

Syntax: WM <memref>,<constant1>[,<constant2>]...

The WM command allows the user to place the values of one or more constants into any memory area located within the test program's boundaries. The constants are moved into contiguous memory locations, starting at the point specified by <memref>.

WR COMMAND (Write into Registers)

Syntax: WR <register>,<constant1>[,<constant2>]...

The WR command allows the user to move constant values into any of the general purpose registers A1 to A14 inclusive. Registers A0 and A15 may not be altered.

The contents of <register> will be set to the value of <constant1>; if more than one constant is specified, their values will be moved into consecutive registers, starting from <register>.

RE COMMAND (Read from a Device)

Syntax: RE <filecode>,<memref>,<constant>

The RE command allows the user to read a number of characters into a buffer from an external device. The number of characters specified by <constant> is read from the specified <filecode> into the buffer whose start address is specified as <memref>. The filecode must have been assigned before DEBUG was invoked.

When this command is executed, a standard read is sent to the Monitor with the specified filecode, buffer address and number of characters. If the user requests data input from the console, a prompt 'READ' is printed on the console log, informing the user that DEBUG is ready to accept the specified number of characters.

The <constant> specifying the number of characters must be specified in hexadecimal format.

RT COMMAND (Return to Online Mode (Interactive))

Syntax: RT

The RT command may be used in two ways:

a) In offline mode, during definition of a breakpoint. It will be the last command entered in the string of commands, to be executed when the breakpoint is reached; it also terminates the breakpoint definition and returns DEBUG to the online mode.
b) When the commands associated with a breakpoint are executed, this command returns DEBUG to the online mode, allowing the user to specify more DEBUG commands.

CO COMMAND (Change Output Device)

Syntax: CO <filecode>

The CO command directs the output from DEBUG to the device with the specified filecode. This filecode must have been assigned before DEBUG was invoked.

CI COMMAND (Change Input Device)

Syntax: CI [<filecode>]

The CI command causes further DEBUG input commands to be read from the device with the specified <filecode>. The filecode must have been specified before DEBUG was called. If no parameter is specified, the filecode of the operator's console is assumed.

TR COMMAND (Trace)

Syntax: TR <ch1>[<ch2>]

where <ch1> and <ch2> are ASCII characters.

The TR command allows the user to associate an ASCII identifier of one or two characters with a breakpoint. The defined ASCII identifier is written to the defined output device every time the associated breakpoint is encountered. This is particularly useful for checking a branch instruction which may place the test program into a loop.

The ASCII identifier may not be two spaces, nor contain a full stop (the command continuation character).

RX COMMAND (Exit)

Syntax: RX

This command causes execution of DEBUG, and of the program under test, to be abandoned. Execution of the RX command takes place immediately, whatever mode DEBUG is in at the time. Control is returned to the Control Command Interpreter.

UNKNOWN BP

The debugging package has been asked to delete a breakpoint which either has not been defined or has already been deleted.

BP DOUBLE DEFINED

An attempt has been made to define a breakpoint which already exists; to re-specify the breakpoint, it must first be deleted (see 'DB' command).

REFUSED IN ON-LINE MODE

The 'IF' command was entered outside a breakpoint definition, or the 'GO' command was entered before the program was started by the '//' command.

REFUSED IN OFF-LINE MODE

An attempt was made to define a new breakpoint without terminating the previous one with the 'RT', 'GO' or '//' commands.

BP TABLE OVERFLOW

An attempt was made to define a new breakpoint when the maximum number, eight, had already been defined. The 'DB' command may be used to delete an unwanted breakpoint, to make room for the new breakpoint. In this way more than eight breakpoints may be defined during the run of the test program.

PARAMETER ERROR

An illegal parameter was specified.

SYNTAX ERROR

Incorrect syntax specified for a command.

FILE CODE NOT ASSIGNED

An attempt was made to use a filecode which was not assigned before DEBUG was called.

COMMAND UNKNOWN

The previously entered command is unknown to this release of DEBUG.

SYMBOLIC REF. ERROR

The specified symbolic address does not exist within the specified reference table, or the table itself does not exist.

NO START ADDRESS

The load module to be debugged has no start address.

COMMAND TABLE OVERFLOW

There is not enough space in the command table to record the command string being entered. The breakpoint currently being defined will be deleted from the table, together with all its stored commands. DEBUG is switched back to online mode.

ERROR CQ. END OF COMMAND INPUT STREAM

An error has occurred on the input device, or the file containing the input commands has been completely read. DEBUG exits the user program.

SP. CH. UNKNOWN

The special characters, used to identify input for a program being debugged, are not recognised for any current program.

The following example shows the use of DEBUG under MAS. The program EX1 is to be debugged in a foreground machine; filecode /01 is assigned to a terminal and filecode /02 to the line printer.

The program EX1 is very simple; it reads a card image, counts the number of occurrences of the letter 'A' in the first 20 characters and outputs this number. A symbol table, TABTAB, was generated at Assembly time and kept at Link-Edit time by the OPT option DBUG=STAB.

The following commands set up and start the foreground machine:

```
MACH-ID : SYSTEM          DATE :
DCF WB,1
CMA 5,7000
SEG 1,1
FCD 1,DY18
FCD 2,LP
FCD /E0,DY18
FCD /C0
FCD /C3
FCD /F0,/C0,SUPERV
FCD /FA,/C3,WB200
DEN
BYE WB
```

The following commands and messages appear on filecode /01 of the foreground machine:

```
MACH-ID : WB              DATE :
LOD 1,EX1,/FA       (or SWP EX1,/FA)
CNL EX1,44
DEB EX1
RUN EX1

DEBUG : EX1     AT /AF46, COMM: %A
```

These messages appear on the operator's console:

```
MAC:WB     ,PROG:EX1    ,KEY-IN,SP.CH:%A
M:KI WB,EX1,%A,AT /AF5C.DM @8,@58.GO
MAC:WB     ,PROG:EX1    ,KEY-IN,SP.CH:%A
M:KI WB,EX1,%A,AT $TABTAB&NEXT.DR R1,R2.GO
MAC:WB     ,PROG:EX1    ,KEY-IN,SP.CH:%A
M:KI WB,EX1,%A,//
```

The following is output on the line printer:

```
DEBUG : EX1     AT /AEE4,COMM: %A
EX1   (%A)  AT /AF5C.DM @8,@58.GO
EX1   (%A)  AT $TABTAB&NEXT.DR R1,R2.GO
EX1   (%A)  //EX1   (%A)  BP: AF5C
EX1   (%A)  AEE0 0000 A001 AF5C 0001 0118 AFF0 4142 4142      \      ABAB
EX1   (%A)  AEF0 4142 4142 4142 4142 4142 4141 4241 4241ABABABABABABAABABA
EX1   (%A)  AF00 4241 4241 4241 4241 4241 4241 4241 2020BABABABABABABA
EX1   (%A)  AF10 2020 2020 2020 2020 2020 2020 2020 2020
EX1   (%A)  AF20 2020 2020 2020 2020 2020 2020 2020 2020
EX1   (%A)  AF30 2020 2020 2020 2020 2020 20200000 0000
EX1   (%A)  BP: AF76
EX1   (%A)  A1 =FFEC A2 =0001
```

(These last two messages are repeated 19 times more, with different values for the register contents.)