

HX-20 DeBug

BA	42	FF	FF	0B	90	42	41	53	59	43	2F
41	53	45	4D	42	4C	45	52	00	0F	36	CE
0B	D9	3C	FC	FF	DC	CE	0B	C0	BD	9E	?
32	7B	10	78	27	03	7E	0B	?	?	?	?
26	05	FE	FD	FC	AD	CC	?	?	?	?	?
CC	02	87	DD	6C	?	?	?	?	?	?	?
DD	6C	FC	?	?	?	?	?	?	?	?	?
2C	?	?	?	?	?	?	?	?	?	?	?

JA	DD										
DD	6E	C3									
FF	64	7E	0B	8A							
93	68	FD	01	36	93						
68	6C	0C	40	24	09	DE	6E	DF			
60	C6	DE	6E	A6	00	08	DF	8C	DE	60	
9C	66	27	76	49	24	0D	37	36	E7	00	83
60	00	D3	6E	ED	00	32	33	08	5A	26	D8
20	29	FE	01	34	08	08	08	A6	00	81	39
27	07	EE	01	09	09	09	20	F3	86	70	A7

REFERENCE MANUAL

HX-20 DeBug

REFERENCE MANUAL

DeBug Reference Manual:
First Edition © J.M. Wald January 1988

DeBug version:
1.0r © J.M. Wald January 1988

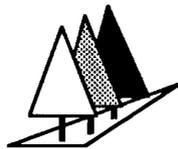
Assembler versions:
2.3r © J.M. Wald May 1987
2.4r © J.M. Wald January 1988

Epson is a trademark of Epson Corporation
Microcassette is a trademark of Olympus Optical Company

71 May Tree Close,
Badger Farm,
Winchester,
SO22 4JF
United Kingdom

National: Winchester (0962) 52644

International: +44 962 52644



Contents

Introduction	Chapter 1
Entering DeBug	1.1
Further information	1.2
Command interface	Chapter 2
Editing the contents of windows	2.1
General command keys	2.2
The Register Display	2.3
Changing the register values	2.3.1
The calculator facility	2.4
Using the HX-20 Monitor	2.5
Exiting from DeBug	2.6
Assembling and Disassembling code	Chapter 3
The Assembler/Disassembler window	3.1
Display modes	3.1.1
The display stack	3.1.2
Using the HX-20 Assembler interface	3.2
The Disassembler facility	3.3
Tracing and executing a program	Chapter 4
The Single Step and Trace facility	4.1
Setting and clearing break points	4.2
Executing a program	4.3

Using the Program Profiler facility	Chapter 5
Program Profiler commands	5.1
Hints on the use of the Program Profiler facility	5.2
Installing DeBug	Appendix 1
Syntax for <value>	Appendix 2
Error messages	Appendix 3
Index	

DeBug is a program development system that consists of the following five components:

Command Interface	This allows you to control the other modules. The command interface (see Chapter 2) allows you to type in commands, change register values, and exit from DeBug
Assembler	This allows you to assemble programs using the HX-20 Assembler (see Chapter 3)
Disassembler	This allows you to disassemble existing machine code programs (see Chapter 3). The generated source code is fully compatible with the HX-20 Assembler and includes labels
Single Step and Trace	This allows you to single step and trace the execution of programs during development (see Chapter 4). This facility allows you to set break points, vary the speed of execution and execute entire subroutines as if they are single instructions. You can also toggle between DeBug's screen and your program's screen
Program Profiler	This allows you to obtain a profile of the execution of a program (see Chapter 5). The profile shows you how much time is spent in each section of the program. You can therefore optimise the execution time of the program

In order to use DeBug it is essential that you have the HX-20 Assembler, also available from J.M. Wald. Note that this manual assumes that you are familiar with the contents of *HX-20 Assembler Reference Manual*.

1.1 Entering DeBug

Appendix 1 gives details of installing DeBug. There are five methods of entering DeBug:

- Select DeBug from the main HX-20 menu
- Use the *DEB* command in BASIC, either interactively or in a program
- Automatically on completion of a program profile (see Chapter 5). The Command and Status window displays the message: **Profiled**

- Automatically on execution of an illegal op-code or break point instead of entering the HX-20 Monitor. Note that this will only happen if you have already selected DeBug or HX-20 Assembler since turning on the HX-20. The Command and Status window displays one of the messages: **Trap!** or **Break**
- Pressing **CTRL + PAUSE** to escape from a program into DeBug. Note that this will only happen if you have already selected DeBug or HX-20 Assembler since turning on the HX-20. The Command and Status window displays the message: **Escape**

1.2 Further information

The following manuals are a useful source of information:

HX-20 Assembler Memory Map J.M. Wald

HX-20 Technical Reference Manual Epson, H8294018-0 Y202990006

A help and information service is available by writing to:

Julian Wald,
71 May Tree Close,
Badger Farm,
Winchester,
SO22 4JF
United Kingdom

Tel: Winchester (0962) 52644

enclosing a stamped addressed envelope.

On entry DeBug displays the following screen:

```

STX &HFFFF
X:140      A:10  B:C0
S:4AF      C:11HINZVC
P:0        >
    
```

There are two windows on the display that you can edit:

- The Assembler window on the top line (see Chapter 3)
- The Command and Status window on the right hand end of the bottom line

The rest of the screen contains the Register Display.

2.1 Editing the contents of windows

You can edit the contents of the Assembler window and the Command and Status window. To toggle between the two windows use the **SC ↓ RN** key. Both windows are windows to longer virtual lines. You can enter more than one command on a line using a colon (:) to separate each command. Within a window, you can edit using the following keys:

- ←** or **→** Moves the cursor to the left or right by one character
- CTRL + ←** Moves the cursor to the left by the width of the window
- CTRL + →** Moves the cursor to the right by the width of the window
- CTRL + A** Moves the cursor to the left end of the window
- CTRL + F** Moves the cursor to the right end of the window
- CTRL + E** Deletes to the end of the line
- CLR** Clears the entire line. The cursor moves to the left end of the window
- DEL** Deletes the character to the left of the cursor
- INS** Toggles between insert and overwrite modes. In insert mode the cursor flashes
- RETURN** Enters the line
- SC ↓ RN** Retrieves the last entered line for re-editing

2.2 General command keys

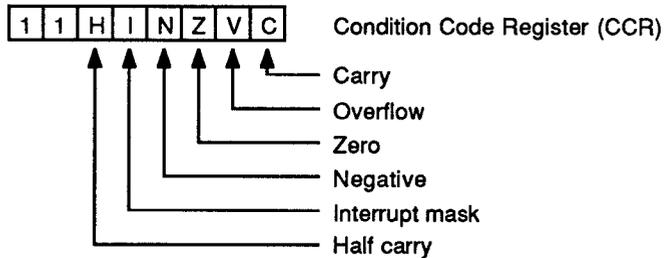
The following keys may be used at any time:

- BREAK** Aborts the current command. The Command and Status window displays the message: Abort. Control always reverts to the Command and Status window
- PAUSE** Temporarily suspends the current command. Pressing any key resumes the command. You can use a numeric key to alter the speed with which the command is executed. This key is of particular use when performing a trace (see Chapter 4)
- MENU** Returns to the main HX-20 menu

2.3 The Register Display

The Register Display consists of output-only windows that display the values of the registers in the current output base. The register display may be modified as follows:

- PF8** Toggles display of the Condition Code Register between displaying the flags and the numeric value. The flags are displayed in the order given in the following diagram:



If the flag is reset, the letter has a bar over the top. Otherwise the letter has no bar.

- PF9** Toggles the display of Register A between ASCII byte mode, numeric byte mode, and combined mode where Register A is joined to Register B to form Register D

- PF10** Toggles the display of Register B between ASCII byte mode, numeric byte mode, and combined mode where Register A is joined to Register B to form Register D

2.3.1 Changing the register values

You can change the value of any register by entering the following in the Command and Status window:

<register name>=<value>

where <register name> is one of A, B, C, D, P, S or X, and <value> is a numeric expression in the form given in Appendix 2.

Note that for D, P, S and X <value> may be optionally followed by an apostrophe (') to select Bank 1 ROMs. If the apostrophe is omitted, Bank 0 ROMs are selected. The current bank is indicated by the absence or presence of an apostrophe against the value displayed for the program counter.

2.4 The calculator facility

To use the Command and Status window as a calculator, type the following:

=<value>

where <value> is a numeric expression in the form given in Appendix 2. The answer is displayed in the Command and Status window in the current output base (see the *RAD* command in Chapter 3 of *HX-20 Assembler Reference Manual*) and in the form:

=<result>

In Symbolic mode (see section 3.1.1) the result may be displayed as a label. To force the result as a number, select Numeric mode first.

2.5 Using the HX-20 Monitor

To enter the HX-20 Monitor use the M command in the Command and Status window. You can now use the Monitor as normal. Note that the B command (Back) returns you to DeBug and not to the HX-20 main menu or BASIC.

2.6 Exiting from DeBug

The usual way to exit from DeBug is to use the Q command in the Command and Status window. Under certain circumstances it is possible to use either the C or G command (see Chapter 4).

Debug is intended to be used for program development in the following way:

- 1 Enter your source code using the Epson BASIC full-screen editor
- 2 Assemble the source code using HX-20 Assembler. This automatically creates a symbol table, which is the set of all labels currently defined. Debug uses the symbol table to display labels in Symbolic mode (see section 3.1.1)
- 3 Enter DeBug and use the Single Step and Trace facility (see Chapter 4) to verify that the code operates correctly
- 4 If the code does not work correctly you can do one of the following:
 - Use the Assembler/Disassembler window to make minor changes and re-assemble single lines
 - Use the Epson BASIC full-screen editor to make major alterations to the source code, and then re-assemble it using HX-20 Assembler
- 5 Repeat steps 3 and 4 until you have a working program
- 6 Use the Program Profiler facility (see Chapter 5) to identify the most frequently executed sections of your program. To make the program run more efficiently, continue from step 4

This chapter explains the commands required to perform the first part of step 4.

3.1 The Assembler/Disassembler window

The top line of the screen is used for the Assembler/Disassembler window. This window displays the disassembled equivalent of the instruction or data at the current value of the location counter (see section 2.3 of *HX-20 Assembler Reference Manual*) modified by the offset.

The disassembled line is displayed in one of four display modes: Instruction, Byte, Word or ASCII (see section 3.1.1). The disassembled line may contain numbers in the current output base (see the *RAD* command in Chapter 3 of *HX-20 Assembler Reference Manual*) and references to labels in Symbolic mode (see section 3.1.1).

3.1.1 Display modes

There are five display modes used to display the contents of memory:

Instruction	The byte at the address held in the location counter is displayed as an instruction mnemonic. If the instruction is a two or three byte instruction, the subsequent bytes are displayed as the data for the instruction. Note that DeBug scrolls backwards a byte at a time in instruction mode, so some spurious instructions may be displayed
Byte	The byte at the address held in the location counter is displayed as a data byte in an FCB statement
Word	The byte at the address held in the location counter and the subsequent byte are displayed as a data word in an FDB statement
ASCII	The byte at the address held in the location counter is displayed as a character in an FCB statement. Note that those byte values for which there are no corresponding ASCII characters, or which are Assembler metacharacters (see section 2.6 in <i>HX-20 Assembler Reference Manual</i>), are displayed in Byte mode
Symbolic	Any value that can be interpreted as a label is displayed as a label. This mode can be used in combination with the four modes above

You can change the display mode using the following keys:

- PF3** Toggles the display between Instruction mode (see above) and the most recently selected data mode (Byte, Word or ASCII)
- PF4** Toggles the display between the three data modes in the order: ASCII, Byte and Word
- PF5** Sets the location counter to the address held in the Program Counter and then selects Instruction mode
- CTRL + PF4** Toggles Symbolic mode on and off

You can also use the following commands in the Command and Status window to change the display mode:

- A** Selects ASCII mode
- B** Selects Byte mode
- DL** Turns on the labelling flag in Symbolic mode. This flag indicates that all labels are to be displayed, even if they occur in the middle of instructions
- DM** Turns on the direct flag. This flag indicates that all direct mode addresses are to be preceded by \$ when displayed

- I** Selects Instruction mode
- NA** Turns off address flag (see the PA command below)
- ND** Turns off direct mode flag (see the DM command above)
- NL** Turn off labelling flag (see the DL command above)
- NM** Turns off Symbolic mode (see the SM command below)
- PA** Turns on the address flag. This flag indicates that addresses that are not labelled are to be displayed as a numeric comment. This means that the address has a semi-colon (;) placed before it
- SM** Turns on Symbolic mode
- W** Selects Word mode

3.1.2 The display stack

The display stack is a last-in/first-out stack. You can push the current value for the location counter on to the display stack so that you can later pop the value to return to your current position. This facility allows you to examine a subroutine and then carry on examining the main routine.

In addition to popping a value from the display stack, you can also assign a new value to the location counter using commands and control keys.

You can use the following keys to operate on the display stack and location counter:

- PF1** Pushes the location counter counter on to the display stack
- PF2** This combines the effect of using **PF1** followed by **PF7**. DeBug pushes the location counter on to the display stack, and then moves to the address in the data field in the current instruction. This key is normally used to examine the subroutine referred to in a JSR instruction
- PF5** Moves to the address in the Program Counter and selects Instruction display mode
- PF6** Pops a value for the location counter from the display stack
- PF7** Moves to the address in the data field in the current instruction. This key is normally used to examine the routine referred to in a JMP or branch instruction. Note that the current value for the location counter is not pushed on to the display stack

HOME

Moves to the address in the Program Counter

↑

Moves back one byte

↓

Moves down according to current display mode. The location counter is advanced by one byte in ASCII and Byte modes, and two bytes in Word mode. In Instruction mode the location counter is advanced by the length of the current instruction

You can also use the following command in the Command and Status window to change the value of the location counter and offset:

[<origin>][,<offset>]

where <origin> and <offset> conform to the syntax for <value> (see Appendix 2). This command works in the same way as the ORG command in HX-20 Assembler (see the *ORG* command in Chapter 3 of *HX-20 Assembler Reference Manual*)

3.2 Using the HX-20 Assembler interface

The Assembler/Disassembler window provides an interactive interface to HX-20 Assembler so that you can assemble single lines of code as you make changes to them. You can do this simply by editing the line displayed in the window, or by entering a new line, and pressing **RETURN**. The HX-20 Assembler interface allows you to use all the HX-20 Assembler commands and instructions, with the exception of the *ASM* command.

For major changes you will still need to use HX-20 Assembler from BASIC, as described in *HX-20 Assembler Reference Manual*.

You can use the HX-20 Assembler interface for the following:

- Patching program code to correct minor errors
- Setting up and controlling listing files (see section 2.9 of *HX-20 Assembler Reference Manual*) for use by other DeBug facilities
- Defining new labels. Note that you may need to set up the RAM file area first if you want to define global labels and have not already used HX-20 Assembler
- Controlling the memory locations that can be changed. This prevents you from accidentally overwriting important locations, such as program code

The HX-20 Assembler interface automatically generates object code and optionally a listing file.

3.3 The Disassembler facility

The Disassembler facility allows you to do the following:

- Examine memory locations in mnemonic form
- Re-create the source code for a program if you have lost the original source code
- Set up the RAM file area for the symbol table so that you can use global labels in the HX-20 Assembler interface

The Disassembler facility allows the following commands to be used in the Command and Status window:

- D** Sets up the RAM file area for the global label symbol table and erases the definitions of all current global labels. The syntax is as follows:

D[<record length>][,<offset>]

Where <record length> is an integer in the range 1 to 255, and <offset> is an integer in the range 0 to the current RAM file size. A default RAM file of 256 bytes is automatically set up when you initialise the HX-20. The minimum size of the RAM file required is given by the following formula:

$(\text{<number of global labels>} * \text{<record length>} + \text{<offset>})$

where <number of global labels> is the maximum number of global labels you are likely to define.

Note that this command is similar to the DEFFIL command in BASIC (see section 5.1 in *HX-20 BASIC Reference Manual*)

- L** Generates global labels from the object code. The syntax is as follows:

L[<first address>][,<last address>]

where <first address> and <last address> conform to the syntax for <value> (see Appendix 2). If either or both are omitted, the previous value specified in a L, P, PL, or PN command is used. The labels are generated either on the basis of instructions in Instruction mode, or data in ASCII, Byte or Word mode. Note that it may take some time to generate the labels for a large block of memory

- P** Generates global labels from the object code and then produces a disassembly. The syntax is as follows:

P[<first address>][,<last address>]

where <first address> and <last address> conform to the syntax for <value> (see Appendix 2). If either or both are omitted, the previous value specified in a L, P, PL, or PN command is used.

The labels are generated either on the basis of instructions in Instruction mode, or data in ASCII, Byte or Word mode. Note that it may take some time to generate the labels for a large block of memory. The disassembly is produced in the current display mode, and is both displayed on the Assembler/Disassembler window and sent to the current listing file

- PN** Produces a disassembly. The syntax is as follows:

PN[<first address>][,<last address>]

where <first address> and <last address> conform to the syntax for <value> (see Appendix 2). If either or both are omitted, the previous value specified in a L, P, PL, or PN command is used. The disassembly is produced in the current display mode, and is both displayed on the Assembler/Disassembler window and sent to the current listing file

Single stepping and trace are two type of simulated program execution. In single step mode, a single instruction is executed each time you press the **TAB** key and then control returns to DeBug. In trace mode, when you press the **TAB** key DeBug automatically single steps through the program at a predetermined rate. You terminate the trace operation by pressing the **BREAK** key. You can also set a break point at a particular address so that DeBug traces the program until it reaches the break point and terminates the trace operation.

Before tracing, single stepping or excuting a program proceed as follows:

- 1 Set the following registers (see section 2.3.1):
 - The Program Counter to the address you want to start tracing or single stepping from
 - The Stack Pointer with the initial stack location
 - Any other registers
- 2 Set up the listing file using the Assembler/Disassembler window
- 3 Perform the single stepping, trace or execution

4.1 The Single Step and Trace facility

You can use the following keys:

CTRL + PF3 Toggles the screen flag. This flag indicates that your program's screen is displayed during single stepping and tracing instead of DeBug's screen

CTRL + PF5 Toggles the subroutine flag. This flag indicates that a subroutine called by a JSR or BSR instruction is executed as if the entire subroutine is a single instruction. This facility is useful for calling working subroutines, such as in the operating system. Note that when the subroutine is called, the return address on the stack will be to an address in DeBug and not in your program

You can use the following commands in the Command and Status window:

NG Turn off the subroutine flag (see the SG command below)

NS Turns off the screen flag (see the SS command below)

S Selects single step mode. Note that this command does not actually start the single stepping. To do this you press the **TAB** key (see section 4.3)

- SG** Turns on the subroutine flag
- SS** Turns on the screen flag
- T** Selects trace mode. Note that this command does not actually start the trace. To do this you press the **TAB** key (see section 4.3)

4.2 Setting and clearing break points

You can use the following commands in the Command and Status window:

- BC** Clear one or more break points. The syntax is as follows:

BC<break point number>[,<break point number>] ...

Where <break point number> is an integer in the range 1–4 representing the break point you want to clear

- BS** Sets a break point. The syntax is as follows:

BS[| < |]<address>[[[,<count>]],<flag>],<break point number>]
 | > |
 | @ |

Where:

<address> is the break point address. If you specify < or > the break point is set for all addresses below or above the specified address. If the Program Counter enters this range a break occurs, but only in single step or trace mode. In normal execution, no break occurs. If you do not specify < or > the break point is set at the specified address. The @ option sets the break point by replacing the op-code with an illegal op-code so that a break will always occur, even in normal program execution.

<count> is the number of times that the break point must be true before it is acted upon. The default value is one. @ option break points are always acted upon on the first occasion in addition to the specified count. For example, if <count> is 1000, a break will occur on the first execution and the thousandth execution

<flag> represents a branch condition, for example EQ or MI, that must be true before the break point is acted upon. The default condition is RA, that is the condition is always true

<break point number> is the number of the break point that you wish to set. If <break point number> is omitted, the next available break point is set. There are a maximum of four break points allowed, numbered 1-4

4.3 Executing a program

You can use the **TAB** key to execute a program in single step or trace mode.

You can use the following commands in the Command and Status window:

C Continues with your program. The syntax is as follows:

C[<execution address>][,<break point address>]

where:

<execution address> is the address you want to execute the program from normally. If <execution address> is omitted, the Program Counter value is used as the execution address.

<break point address> is the address at which you want control to return to DeBug. Note that <break point address>, if supplied, must be in RAM.

This command automatically clears the break point that caused your program to terminate

G Returns to your program. The syntax is as follows:

G[<execution address>][,<break point address>]

where:

<execution address> is the address you want to execute the program from normally. If <execution address> is omitted, the Program Counter value is used as the execution address.

<break point address> is the address at which you want control to return to DeBug. Note that <break point address>, if supplied, must be in RAM

The Program Profiler facility enables you to determine the most frequently executed sections of your program. This information helps you optimise the program code so that the program runs in the most efficient way.

To use the Program Profiler facility, you specify the memory range in which the profile is to be taken. The Program Profiler then subdivides this area into smaller partitions. When you run the program, the Program Profiler takes regular samples of the Program Counter (PC) register. On taking each sample, the Program Profiler checks whether the Program Counter is currently in one of the partitions in the chosen memory range, and if so updates the appropriate partition count.

When the Program Profiler has taken a predetermined number of samples, it stops sampling, enters DeBug if necessary, and displays the following message on the Command and Status window: **Profiled**. You can then print a listing of the profile. This indicates how many samples fell in each partition of the address range.

You can then select the partitions with the highest number of samples as the memory ranges for further profiles. These additional profiles may give a more accurate profile of your program.

5.1 Program Profiler commands

You can use the following commands in the Command and Status window:

PP Prints a listing of the profile in the same format as the following diagram:

E000:	0	=	0%
E1FF:	19	=	3%
E3FF:	0	=	0%
E5FE:	0	=	0%
E7FE:	0	=	0%
E9FD:	0	=	0%
EBFD:	0	=	0%
EDFC:	0	=	0%
EFFC:	0	=	0%
F1FB:	0	=	0%
F3FB:	102	=	20%
F5FA:	0	=	0%
F7FA:	359	=	71%
F9F9:	20	=	4%
FBF9:	0	=	0%
FDF8:	0	=	0%
FFF8			

First address in partition →
 Last address in last partition →
 Number of samples in partition →
 Percentage of total samples →

PS Starts or terminates a program profile. The syntax is as follows:

PS[<first address>,<last address>[,<samples>][,<frequency>]]

where:

<first address> is the lowest address in the memory range for the profile

<last address> is the highest address in the memory range for the profile

<samples> is the total number of samples required. If <samples> is omitted the default is 1024

<frequency> is the number of samples per second. If <frequency> is omitted the default value is 128. The minimum value for <frequency> is 2 and the maximum value is 256

If no parameters are given, any existing profile is cancelled.

5.2 Hints on the use of the Program Profiler facility

- The profile is only for the memory range specified, and gives no indication of how much processor spends elsewhere in memory. The processor could, for example, be spending 95% of the time in the operating system and only 5% in your program code. Therefore, a figure of 50% on the profile listing would represent only 2.5% of the total running time
- Choose the maximum number of samples per second so that the processor does not spend most of the time in Program Profiler. Each sample takes approximately one millisecond
- Choose enough samples to obtain reasonable accuracy bearing in mind the program size and the total running time
- To obtain more accurate results, carry out a profile on each subsection of the first profile
- Sections of code with masked interrupts will be listed as 0% on the profile listing, irrespective of the actual percentage time these sections may occupy, as the sampling interrupt will also be masked. Furthermore, the section of code executed after a section with masked interrupts will have its profile count artificially raised
- You cannot use the Program Profiler with any program that uses the real time clock chip interrupt as this is used to perform the sampling

Installing DeBug

Appendix 1

To install DeBug on ROM you should perform the following steps:

- 1 Switch the HX-20 off and install the supplied ROM or ROMs according to the instructions given in the document *Installing EPROMs* supplied with the product
- 2 Perform a cold start (see section 1.1.2 of *HX-20 BASIC Reference Manual*)
- 3 Select either DeBug or BASIC/ASSEMBLER from the main HX-20 menu. Selecting DeBug reserves memory and enters DeBug. If you select BASIC/ASSEMBLER, the following message is displayed:

```
Press TAB to reserve
memory for DeBug
otherwise press
RETURN to continue.
```

If you do not want to reserve memory for DeBug, press the **RETURN** key, otherwise press the **TAB** key. Note that if you do not reserve memory for DeBug, you will be unable to use any of its facilities. This facility is provided so that you can run the Assembler with the maximum amount of memory available.

You can now enter DeBug in any of the ways described in section 1.1.

Syntax for <value>

Appendix 2

This appendix gives the syntax for values used as parameters to commands in the Command and Status window. The syntax given here supplements the syntax in section 2.5 of *HX-20 Assembler Reference Manual*.

The syntax for <value> is:

```
[ |<| ]<assembler expression>  
|>|
```

where <assembler expression> is any expression that conforms to the syntax given in section 2.5 of *HX-20 Assembler Reference Manual*.

An <assembler expression> preceded by a left angle bracket (<) is unrelocated if you have specified a non-zero offset. An <assembler expression> preceded by a right angle bracket (>) is relocated if you have specified a non-zero offset.

Note that the syntax for <operand> given in section 2.5.1 of *HX-20 Assembler Reference Manual* is extended to allow the use of register names (A, B, C, D, P, S and X) as operands.

Error messages

Appendix 3

This appendix gives the error messages generated by DeBug. The error messages given here supplements the error messages listed in Appendix 3 of *HX-20 Assembler Reference Manual*.

- NB 132** No break points
There are no unused break points available
- NR 131** Not RAM
An operation that requires RAM has been attempted in a non-RAM area
- The user program's stack is in ROM or in the I/O area
 - You are trying to set an @ option break point in ROM
- OP 130** Bad op-code
The Single Step and Trace facility is unable to proceed
- The user program counter is below address &H80
 - The user program counter is pointing at an illegal op-code
- SU 129** Stack underflow
There are no more display addresses on the display stack
- SV 128** Stack overflow
The display stack is full

Index

Index entries refer to chapters or to sections within chapters. The main reference is listed first. Note that *Cn* refers to Chapter *n*, and *An* to Appendix *n*.

A command	3.1.1	L command	3.3
ASCII mode	3.1.1, 3.1	Labels	3.1.1, 3.1
Assembler	3.2, C1, C3	Listing files	3.2
Assembler/Disassembler window	3.1, 2.1	Location Counter	3.3
B command	3.1.1	M command	2.5
Bank selection	2.3.1	MENU key	2.2
BC command	4.2	Monitor	2.5, 1.1
Break key	2.2, C4	NA command	3.1.1
Break point	4.2, 1.1	ND command	3.1.1
BS command	4.2	NG command	4.1
Byte mode	3.1.1, 3.1	NL command	3.1.1
C command	4.3, 2.6	NM command	3.1.1
Calculator facility	2.4	NS command	4.1
Clearing register values	2.3.1	Offset	3.1.2
Command and Status window	C2, 1.1, 2.1, 3.1.2, 4.1,4.2, 4.3	Origin	3.1.2
Command interface	C2, C1	P command	3.3
Command keys	2.2	PA command	3.1.1
Condition Code Register	2.3	PAUSE key	2.2
CTRL key		PF1 key	3.1.2
with PAUSE key	1.1	PF2 key	3.1.2
with PF3	4.1	PF3 key	3.1.1
with PF4	3.1.1	PF4 key	3.1.1
with PF5	4.1	PF5 key	3.1.1, 3.1.2
D command	1.1	PF6 key	3.1.2
DEB command	1.1	PF7 key	3.1.2
DeBug		PF8 key	2.3
entering	1.1	PF9 key	2.3
exiting from	2.6	PF10 key	2.3
installing	A1	PN command	3.3
Disassembler	3.3, C1	PP command	5.1
Display stack	3.1.2	Program Profiler	C5, C1
Display modes	3.1.1, 3.1	PS command	5.1
DL command	3.1.1	Q command	2.6
DM command	3.1.1	RAM file area	3.3, 3.1
Entering DeBug	1.1	Register Display	2.3
Editing contents of a window	2.1, 2.6	Relocater	A2
Error messages	A3	S command	4.1
Exiting from DeBug	2.6	Screen flag	4.1
Flags		SG command	4.1
in the Condition Code Register	2.3	Single Step facility	C4, C1
Screen	4.1	SM command	3.1.1
Subroutine	4.1	SS command	4.1
G command	4.3, 2.6	Subroutine flag	4.1
HOME key	3.1.2	Symbolic mode	3.1.1, 3.1
Hx-20 Assembler	1.1	T command	4.1
HX-20 Monitor	2.5, 1.1	TAB key	4.3, C4
I command	3.1.1	Trace facility	C4, C1
Illegal op-code	1.1	Unrelocated expression	A2
Installing DeBug	A1	<Value> syntax	A2
Instruction mode	3.1		

© J.M. Wald 1988
71 May Tree Close, Winchester, SO22 4JF