HP-41 M-CODE DEBUGGER

User Guide

Mark Power

1986

Table of Contents

1. Introduction

The book 'HP41 M-code For Beginners' by Ken Emery has two debugging programs: BREAK, is used to interrupt a M-code routine and dumps the CPU registers into an area of main memory; LOOP, allows you to debug loops by executing them a specified number of times before dumping the CPU registers. Unfortunately, these programs are very restrictive, they can only be used to view CPU values, not to change them before or during execution.

The program 'DEBUG' gives the user more flexibility although it still has a few limitations. The program is arranged as a main menu with several editors and functions available from it.

The functions available in DEBUG allow:

- Execution of any M-code routine at a user specified address
- Editing of any of the five main 56 bit accumulators A, B, C, M, N
- Editing of the CPU status:
  - Pointers P and Q
  - The active pointer
  - Machine code flags 0-7
  - The G register
  - The CPU mode - hexadecimal or decimal
- Printing the content of the main CPU registers
- Editing of the state of the 56 FOCAL user flags
- Setting up of up to 10 breakpoints
- The ability to continue execution from a breakpoint stop

DEBUG cannot be used to manipulate:

- The carry flag
- The T (tone) register
- Machine code flags 8-13
- The Key register
- The Key flag
- The selected RAM chip or peripheral device

For further details of the limitations see the section on breakpoints (under BKED).

Additional functions are provided in the ROM to:

- Locate a buffer
- Test the existence of a buffer
- Make a buffer
- Clear a buffer
- Delete a buffer
- Increment or decrement M
- Give additional flexibility to the catalogue function

2. Buffer 9 Format

The DEBUG program relies on the existence of Buffer 9. This is automatically created by DEBUG and is used to hold all of the relevant CPU information.

Physically Buffer 9 is located in main user memory below the .END. and above the key assignments. It may be located by the function LCBUF for manipulation external to DEBUG (see additional function sections 11-16).

The format of Buffer 9 is:

Nybble

```
     13 12 11 10  9  8  7  6  5  4  3  2  1  0
    ------------------------------------------     R
    | 1| U|   STK 2   |HD|PT| P| Q|  G  |  ST |  6  e
    | <---------- N register -------------> |  5  g
    | <---------- M register -------------> |  4  i
    | <---------- B register -------------> |  3  s
    | <---------- A register -------------> |  2  t
    | <---------- C register -------------> |  1  e
    |  ID | SIZE| U| U|   STK 1   |   BKPT    |  0  r
    ------------------------------------------
```

Where	ID	=	Buffer identifier = 99hex
	SIZE	=	Buffer size = 07hex
	U	=	Unused nybbles
	BKPT	=	Address of last breakpoint stop
	STK 1	=	Bottom stack value at last stop
	STK 2	=	Second stack value at last stop
	A,B,C,M,N	=	Five main 56 bit accumulators
	HD	=	Hex (0) or decimal (1) mode
	PT	=	Active pointer (P=0 Q=1)
	P,Q	=	Position of pointers P and Q
	G	=	G register (1 byte)
	ST	=	Machine code flags 0-7 (1 byte)

It should be noted that the MP-ROM includes a preservation routine which ensures that the operating system does not delete Buffer 9 whilst it is plugged in.

This enables testing to be continued even if the machine is turned off. The buffer may be deleted manually with DELBUF if required and will be automatically deleted when the machine is turned on without MP-ROM present.

3. DEBUG – Main Entry Point

To use the DEBUG program, MP-ROM should be loaded into a Q-ROM page. Then all that is required is to XEQ'DEBUG.

The main menu prompt will then appear on the screen, 'DEBUG CMD ?'. The following keys then become active:

ON	Exit DEBUG and return to normal operational mode
B	Enter the breakpoint editor - BKED (section 9)
C	Continue from a breakpoint - CONTBK (section 10)
E	Enter code at specified address - START (section 4)
F	User flag editor - FLAGE (section 6)
R	Register editor - RED (section 8)
S	Status editor - STED (section 7)
ENTER	Print main CPU registers - PRCPU (section 5)

4. START - Begin Execution at Specified Address

To run a piece of M-code this routine is used. After pressing the E key in the main menu, the user is prompted for a hexadecimal address, and when this is confirmed by pressing R/S, execution begins at the specified address. Correction of entered values is possible with the delete key.

The address specified may be anywhere in the 64K machine code address space (0000-FFFFhex). The return address for DEBUG is placed on the stack and up to three levels of subroutine calls are possible in the code under test. DEBUG is re-entered when a RTN instruction is encountered and the value on the bottom of the stack is that of the DEBUG routine.

For example, if we wish to run the operating system routine which enables chip 00 (ENCP00):

ADDRESS	MNEMONIC	COMMENT
0952	C=0 S&X	Set C[0:2] to zero
0953	PRPH SLCT	Deselect peripherals
0954	RAM SLCT	Select RAM chip 00
0955	RTN	Return

Then the user would enter the address 0952 at the prompt ENTRY @ ____ and press R/S. DEBUG the loads up the CPU contents from Buffer 9 and performs an NC XQ 0952 instruction.

When the RTN at address 0955 is encountered the CPU re-enters DEBUG, stores all the CPU contents back in Buffer 9 and shows the main command prompt. The CPU contents in Buffer 9 may then be displayed and edited if required.

5. PRCPU - Print CPU Register A, B, C, M and N

To print the contents of the main accumulators (or use the print commands in any of the other functions) you will need either a HP82143A dedicated printer or an HP-IL interface with some printing device (such as the HP82162A printer, the HP82163B Video Interface or the ThinkJet).

These devices must be set to TRACE mode before printing can take place. With the HP82143A this simply means setting the switch on the front of the printer to TRACE, for HP-IL devices TRACE mode is set by setting user flag 15 (this can be done either before entering DEBUG or using the flag editor - see next section).

To print the values simply press ENTER at the DEBUG CMD ? prompt. The output for the default values of Buffer 9 will be:

```
CPU:
C=10000000000000
A=10000000000000
B=10000000000000
M=10000000000000
N=10000000000000
```

Each digit represents a single nybble in the register.

6. FLAGE - User Flag Editor

This function is called up by pressing the F key when the main menu prompt is shown. The display will then show FLAG __ .

The user may then enter a two digit flag number in the same way that flags are set or cleared using SF and CF. Any decimal flag number in the range 00-55 may be specified.

Once a flag number has been entered, the flags current state will be displayed with the words SET or CLR on the right hand side of the display. To toggle the state of the flag all that is required is to press the R/S key.

For example, suppose we have a HP-IL printer attached and we wish to set it to TRACE mode to print the contents of the CPU, the following keys would be pressed:

DISPLAY	KEY PRESSED
DEBUG CMD ?	F
FLAG __	1
FLAG 1_	5
FLAG 15 CLR	R/S
FLAG 15 SET	ON
DEBUG CMD ?	

Note that the ON key is used to return to the main menu and that FLAG 15 SET will appear automatically on the printer (as do all flag changes when a Tracing printer is connected).

To change the value of more than one flag, simply continue entering flag numbers. For example, suppose both flags 00 and 01 are clear and you wish to set them, then you would perform the following:

DISPLAY	KEY PRESSED
DEBUG CMD ?	F
FLAG __	0
FLAG 0_	0
FLAG 00 CLR	R/S
FLAG 00 SET	0
FLAG 0_	1
FLAG 01 CLR	R/S
FLAG 01 SET	ON
DEBUG CMD ?	

If a Tracing printer was connected during this operation, it would show FLAG 00 SET and FLAG 01 SET. The display annunciators are also automatically updated so 0 and 1 would appear in the bottom of the display.

For use of the user flags refer to the HP41 Handbooks.

7. STED - CPU Status Editor

This is entered from the main menu by pressing the S key. The status information consists of two screens, one shows the position of the pointers P and Q, and which of the two is active, and the other shows the value in the G register, the ST register (machine code flags 0-7) and the CPU mode (hexadecimal or decimal).

The main display set has the format **PT=x P=p Q=q** where x is the active pointer P or Q, p is the position of pointer P and q is the position of the pointer Q. Pointer positions are in the range 0-D hex (0-13 decimal).

The alternate display set has the format **ST=st G=gg m** where st are the contents of ST, gg are the contents of G and m is either H or D representing hex or decimal mode.

The following keys may then be used to change the information:

D	Sets the CPU mode to decimal, the display shows the alternate display set if it is not already shown, and decimal mode is indicated with a D on the far right of the display.
H	Sets the CPU to hexadecimal mode, shows the alternate display set and indicates hex mode with an H.
G	Selects the alternate display set and prompts the user for two hex digits for the G register. An underscore alternating with the old value indicates the users' progress.
P	Selects the main display set and prompts the user for a new position of the pointer P (0-D representing nybbles 0-13). The cursor alternates with the old value of P.
Q	As P above but for pointer Q.
R	Selects the main display set and prompts the user for the active pointer. The cursor alternates with the previous active pointer value. The user responds with either the P key or Q key as appropriate.
S	Selects the alternate display set and prompts the user for two hex digits for the ST register (machine code flags 0-7). Prompting is similar to G above.
R/S	Toggles the display between the main display set and the alternate display set.
ENTER	Prints the currently shown display set on any printer in TRACE mode.
ON	Returns the user to the main DEBUG prompt.

8. RED - Register Editor

The register editor is entered by pressing the R key at the main DEBUG CMD ? prompt. This editor allows the user to view and edit individual nybbles within any of the five main accumulators (A,B,C,M,N).

In RED the display format is:

R:NN hhh,d,lll

Where R is the current register being edited
NN is the current nybble number
hhh are the three nybbles above the current nybble
d is the current nybble which is being edited
lll are the three nybbles below the current nybble

The registers are arranged in the following order:

```
   N          M          B          A          C
13..00     13..00     13..00     13..00     13..00
LEFT <-                                   -> RIGHT
```

The starting position for editing is the nybble at C:00, thus to get to N you must move left and to get from N to A you must move right.

In RED the following keys are active:

ON	Return to main DEBUG menu
USER	Move left one digit or one register if preceded by SHFT
PRGM	Move right on digit or one register if preceded by SHFT
0..F	Enter that value at the current position

As each digit is entered RED will move to the next digit on the right. If the current position is C:00 then RED cannot move right and so it stays at C:00.

The following example shows how RED can be used to set up the registers. Suppose we wish to set the M register to 32000000000000 and C to 10000000000888. We will assume that the registers are all in their default state (i.e they contain 10000000000000).

```
DISPLAY                KEYS

DEBUG CMD ?            R
C:00  000,0,990        USER
C:01  000,0,099        USER
C:02  000,0,009        8
C:01  008,0,099        8
C:00  088,0,990        8
C:00  088,8,990        SHFT USER
A:00  000,0,100        SHFT USER
B:00  000,0,100        SHFT USER
M:00  000,0,100        SHFT USER
N:00  000,0,100        PRGM
M:13  000,1,000        3
M:12  003,0,000        2
M:11  032,0,000        ON
DEBUG CMD ?
```

If a Tracing printer is present, pressing ENTER would confirm these settings:

```
CPU:
C=10000000000888
A=10000000000000
B=10000000000000
M=32000000000000
N=10000000000000
```

To reset the registers to their default settings the function CLRBUF may be used from the normal operating mode (i.e from outside DEBUG). To do this place 9 in the X register and XEQ'CLRBUF (for more information on CLRBUF see section 13).

9. BKED - Breakpoint Editor

The breakpoint editor is entered by pressing the B key at the main DEBUG prompt. The facilities provided in BKED allow for the creation of up to ten breakpoints which enable the user to interrupt the execution of a program at fixed points and edit the CPU contents.

Once the results have been checked and possibly altered, execution can be made to continue from where it was interrupted. The mechanism to do this is described in section 10.

There are a few technical points on what breakpoints are and how they operate that you will need to know before being able to exploit these facilities fully.

Breakpoints are inserted by DEBUG into your code at the address specified. The actual instruction inserted is a class 3 'NC XQ BKPT' where BKPT is an address in DEBUG.

There are several implications with using this type of instruction. Firstly, the instruction overwrites two words of your code and so breakpoints can only exist in pseudo ROMs (this term will be used to refer to all types of MLDL and Q-ROM units). Breakpoints cannot exist in actual ROMs because DEBUG cannot write to them.

The code that is overwritten is transferred into a breakpoint table in the MP-ROM so that when execution continues all the correct instructions are executed. Thus the MP-ROM must also reside in pseudo ROM.

Secondly, the instructions that cause the breakpoint stop ('NC XQ BKPT') require that the carry flag is clear when they are executed.

This places restrictions on where breakpoints can be located. The size of the breakpoint instructions also means that they cannot be placed over instructions that are more than two words long.

Some places where breakpoints cannot be placed include: after class 0 and 2 instructions such as C=C-1 or ?A#C that may set the carry flag; where the second word of the breakpoint would over write one of these class 0 or 2 instructions; over GOSUB and GOLONG instructions (3 word page independent calls and jumps using the routines GOLONG, GOSUBH, GOSUB0 etc.); over MESSL type calls (i.e routines called like 'NC XQ MESSL' that are followed by a string of text); over class 3 local relative jumps. In addition, breakpoints should not overlap each other, that is they should not be assigned to consecutive addresses.

DEBUG does not check for valid breakpoint addresses due to the variety of instructions which may affect the carry and the fact that the programmer may know that the carry is never (or always) affected by certain instructions.

Generally, the safest place to locate breakpoints is after class 0 instructions which do not effect the carry and over two class 0 instructions the second of which always leaves the carry clear.

The example code below shows where breakpoints could and could not be located (David Assembler Mnemonics are used):

```
ADDRESS     MNEMONIC

xA00        N=C
xA01        C=C+A S&X
xA02        RAM SLCT
xA03        R=R+1
xA04        LD@R F
xA05        ?R= 0
xA06        JNC -1F
```

Breakpoints could be located at xA01 as the 'N=C' instruction always leaves the carry clear, however it would not be safe to place one at xA02 as the 'C=C+A S&X' may set the carry, thus avoiding the 'NC XQ BKPT'. xA03 would be a safe place as again the carry is guaranteed to be clear by the 'RAM SLCT'. xA04, xA05 and xA06 would not be suitable due to the use of the carry flag and local (class 3) jumps.

When a breakpoint is created, it and all the others held in the breakpoint table are enabled automatically. This means that if execution reaches a breakpoint instruction, either from inside DEBUG or from XEQuting a function under test, DEBUG will be re-entered.

Breakpoints may be disabled, so that execution is not affected. This involves DEBUG automatically placing the overwritten code back in its proper place.

Some or all breakpoints may be deleted, again the code overwritten by breakpoint instructions is restored.

It is very important to remember that all breakpoints must only be deleted by DEBUG (i.e not overwritten by an assembler or an editor) and that all breakpoints should be cleared before saving a ROM to or loading a ROM from a mass storage device.

If problems do arise by corruption of breakpoints then the easiest solution may be to re-load MP-ROM and then re-load the ROM under test (functions for ROM storage are not provided in MP-ROM but are available in ZENROM and the MLDL EPROM set).

The breakpoint editor has two distinct levels: the top level is entered by pressing the B key from the main DEBUG prompt or by pressing ON from the lower level of BKED; the lower level is entered by pressing one of the numeric keys, representing a breakpoint number, from the top level.

In the top level the display will show one of the following:

NO BKPTS	This indicates that there are no breakpoints set up in the system at present.
BKPTS OFF	This indicates that there are breakpoints set up but they are currently disabled.
BKPTS ON	Indicates breakpoints are present and enabled.

When the display shows one of these messages, the following keys become active:

ON	Returns the user to the DEBUG CMD ? prompt
ENTER	Prints the display on any Tracing printers
C	Disables and clears all breakpoints. The user is prompted 'SURE (Y/N) ?' to check that this is really required. If the Y key is pressed, then all breakpoints will be cleared and the display will return to showing 'NO BKPTS'. Otherwise the display will return to its previous state with all the breakpoints unchanged.
D	Disables all breakpoints. The overwritten code is restored but the breakpoint addresses are kept to allow them to be enabled at some later stage.
E	Enables all breakpoints. This is also done when a new breakpoint is entered or when an existing breakpoint is moved. If there are no breakpoints present the key has no effect.
0..9	Enter the low level editor for the breakpoint specified.

In the low level editor, the display shows:

BKPTn @ abcd

Where n is the breakpoint number and is in the range 0..9 inclusive and abcd is the breakpoint's address. If the breakpoint specified is undefined then the address field is displayed as four underscores (e.g **BKPT0 @ ____** indicates breakpoint 0 is undefined).

As mentioned earlier breakpoints may only be specified in pseudo ROM pages and so BKED automatically checks addresses as they are entered to ensure that they exist in a pseudo ROM page.

The lower level editor enables the following keys:

ON	Returns to the top level of BKED.
ENTER	Prints the currently shown breakpoint on any Tracing printer.
<-	Deletes the breakpoint shown or the last digit entered during breakpoint address entry.
R/S	Terminates address entry if four digits have been entered and sets up the breakpoint. Alternatively, if a set up breakpoint address or four underscores are shown then R/S moves to the next breakpoint in sequence. After breakpoint 9 has been shown R/S returns the user to the top level of BKED.
0..F	Are used for breakpoint address entry. Four digits are required for a breakpoint address. The first digit, the page number, is checked as outlined above to ensure it specifies a pseudo ROM page.

The example below shows how a breakpoint could be set up at address xA01 in the previous example code (we will assume that DEBUG is in its initial state and the code is located in page D):

DISPLAY	KEYS	COMMENTS
DEBUG CMD ?	B	Enter BKED
NO BKPTS	0	Assume default state
BKPT0 @ ____	D	Breakpoint 0 will be used. D is the page number.
BKPT0 @ D___	A	Rest of address
BKPT0 @ DA__	0	
BKPT0 @ DA0_	1	
BKPT0 @ DA01	R/S	Breakpoint set up
BKPT0 @ DA01	ENTER	Breakpoint printed
BKPT0 @ DA01	R/S	Go to next breakpoint
BKPT1 @ ____	ON	Empty, leave lower level
BKPTS ON	ON	Enabled automatically
DEBUG CMD ?		Back at top level

The breakpoint mechanism uses conventional CPU instructions and so must work within the restrictions of the four level stack. Thus although breakpoints may be placed in subroutines, these subroutines should not be nested more than two deep.

Note that this is in contrast to the normal operation of DEBUG where re-entry will only occur if a RTN instruction is encountered at the same level as the entry to the code.

The example below shows the worst case that BKED and CONTBK (described in the next section) can handle correctly:

```
LABEL      MNEMONIC

TST1:       :
            :
           GOSUB TST2
            :
           RTN
            :
TST2:       :
            :
           GOSUB TST3
            :
           RTN
            :
TST3:       :
            :
           BREAKPOINT
            :
           RTN
```

We assume that entry is at TST1 and that within TST1, TST2 is called. Similarly, it calls TST3 which contains a breakpoint. DEBUG is re-entered and execution may be continued using CONTBK. Execution recommences at the breakpoint in TST3, returns to TST2, then to TST1 and then back into DEBUG with the RTN at the end of TST1.

Any further levels of nesting (for example TST3 calling TST4 which contains a breakpoint), would cause a loss of data from the stack and a subsequent malfunction in the re-entry to BKED. In most cases this should not cause any difficulties provided that all the limitations are observed. In general it would be best to test TST3 separately rather from TST2 or TST1. This would make debugging much easier in most cases.

10. CONTBK - Continuation from a Breakpoint

When a piece of code is entered normally there is no definite way of telling whether or not a breakpoint will be executed, however if this is the case, then re-entry to DEBUG is different to the normal type of execution.

On a breakpoint re-entry, before the main prompt is shown, three other messages will appear (these are also sent to Tracing printer devices):

```
STK1 @ abcd
STK2 @ efgh
BKPT @ ijkl
```

The R/S and delete keys are used to move through these messages. They indicate the bottom two values on the stack when the breakpoint was executed and the address of the breakpoint encountered.

If the user wishes to continue execution from the breakpoint stop, possibly after viewing and altering the contents of the CPU registers, then the C key should be pressed at the DEBUG CMD ? prompt.

CONTBK will then allow the user to confirm the re-start address when the prompt CONT @ ijkl is shown (where ijkl is the same address as above). The R/S key is used to confirm this address, whilst any other key returns you to the main menu.

If the address is confirmed, then it is printed on any Tracing printers and the stack pre-loaded with the STK1 and STK2 values. All other CPU values are loaded normally. The overwritten code is run before the CONTBK jumps to the word after the NC XQ BKPT which interrupted execution.

If an attempt is made to execute CONTBK when no breakpoint stop has taken place or if the breakpoint which caused the stop has been deleted, then the command will be ignored.

11. LCBUF - Locate Buffer Function

This function may be used in the normal operating mode or in FOCAL programs to locate a buffer. It requires a buffer number in the X register and returns the buffer's main memory address to the M register.

The address is returned to the last two characters of the ALPHA register, which corresponds to the bottom four nybbles of the M register. This format is the same as that used by M+1, M-1 and the ZENROM functions RAMED, NSTOM and NRCLM.

The NONEXISTENT error message will be given if the buffer specified does not exist. Valid buffer numbers may be in the range 1-15.

For example, to locate buffer 9 for manipulation by RAMED: enter 9, XEQ'LCBUF and then XEQ'RAMED.

The FOCAL program given below also demonstrates the use of LCBUF to find the size of a buffer. It uses the ZENROM function NRCLM and the STO M synthetic instruction (for details of synthetic programming see Synthetic Programming Made Easy, the ZENROM Manual or Extend Your HP41).

Input to the program is simply a buffer number in X, the size of the specified buffer is left in X:

```
01 LBL'BUFSIZ
02 CLA
03 LCBUF
04 NRCLM
05 STO M
06 ATOX
07 ATOX
08 END
```

12. MKBUF - Make Buffer Function

This function creates a buffer from the header given in the M register (the last 7 characters of the ALPHA register). For details of the structure of buffer headers see pages 36-39 of the ZENROM manual.

In brief, the top byte of the header contains the buffer identity (which is usually two identical nybbles) and the next byte down contains the buffer size, in registers, including the header register.

So for example, to create a buffer with identity 4 and size of 32 registers (20hex), you would type:

```
CLA  'D{sp}ABCDE  XEQ'MKBUF
```

The decoded value of M in this case would be 44204142434445, giving ID=44, SIZE=20hex and additional header bytes 41 42 43 44 45hex.

Each register created in the buffer, except the header, is set to 10000000000000. This is done to avoid Bugs in the older Card Reader ROMs (for more details see page 39 of the ZENROM manual).

If there is not enough room for the buffer, then the machine will try PACKING and then suggest you TRY AGAIN. If the buffer size specified is 00 or if a buffer exists with the same ID but a different size, then the error message DATA ERROR is given.

13. CLRBUF - Clear Buffer Function

This function accepts a buffer number in X and if the buffer exists sets the contents of all the registers in the specified buffer, except the header, to 10000000000000. Errors are given for the same reasons as in LCBUF.

14. DELBUF - Delete Buffer Function

This function accepts a buffer number in X and if the specified buffer exists, deletes it and packs the I/O and key assignment area. Inputs and errors are the same as for LCBUF.

15. BUF? - Test Buffer Function

This function is used as a conditional test to determine the presence of a buffer, specified in the X register.

If executed from the keyboard it returns YES or NO depending on whether the buffer exists or not. If executed in a program, execution continues at the next line if the buffer exists or skips a line if it does not.

If the buffer specified is greater than 999 then the error NONEXISTENT is given.

16. M+1, M-1 - Increment and Decrement M Function

These functions simply add or subtract one from the value in the M register. This is a hexadecimal nybble operation on the whole word rather than a normal floating point add with the '+' function.

They are used when the M register contains a buffer address produced by LCBUF and it is necessary to move through the buffer one register at a time.

The following FOCAL program demonstrates the use of these functions. NRCLM and DECODE from ZENROM are used.

```
01 LBL'GETC
02 9
03 LCBUF
04 M+1
05 NRCLM
06 DECODE
07 AVIEW
08 END
```

This program locates buffer 9, moves the address pointer in M to the first register, decodes it and displays it. The register shown is the one containing the CPU C register value as used by DEBUG (see section 2 - Buffer 9).

As a further example of the use of these functions, the program below could be used to load up specific values into Buffer 9 for use by DEBUG.

```
01 LBL'LOAD9
02 CLA
03 9
04 LCBUF
05 M+1
06 ASTO L
07 '12345678901234          <- Value to go in C
08 XEQ 01
09 '23456789012345          <- Value to go in A
10 XEQ 01
11 '34567890123456          <- Value to go in B
12 XEQ 01
13 '45678901234567          <- Value to go in M
14 XEQ 01
15 '56789012345678          <- Value to go in N
16 LBL 01
17 CODE                     <- Code the value
18 CLA
19 ARCL L                   <- Get the address
20 NSTOM                    <- Store the value
21 M+1                      <- Point to next reg
22 ASTO L                   <- Save new address
23 END
```

This program can be changed to fit the individuals' requirements. To do this the values on lines 07, 09, 11, 13 and 15 should be changed. The basic structure should however remain the same.

If it is necessary to have several sets of set up values, then the program could be changed to get its values from Extended Memory ASCII files (this would also allow changes to be made very easily using the HP41CX ED function).

17. XCAT - Extended Catalogue Function

This function is simply an extension to the inbuilt CAT function. In addition to being able to perform the normal CATs 1,2 and 3, the ROM function catalogue (CAT 2) is extended so that it can be started at any page.

XCAT prompts for a catalogue number with:

XCAT (0-F) _

Pressing 0-4 will have the same effect as the normal CAT function. Pressing 5-F will start a CAT 2 at the specified page. If a ROM is not present in the page specified, then NONEXISTENT is given.

Operation of the catalogue, once it has started, is exactly the same as for the normal CAT function (this means BST will take you back to the previous entry etc, and the modified operation of the HP41CX is as normal).

18. DEBUG Example Use

The following is given as a very simple example of DEBUG in use.

Suppose we wish to test a piece of code that simply loads values into the main accumulator C. The contents of C will initially be unknown and when the code finishes will be C = 10000000001FFF.

The code to do this would be:

```
      ADDRESS     MNEMONIC          COMMENT

      EF00        C=0 ALL           Clear the whole register
      EF01        C=C+1 MS          Set mantissa sign to 1
                                    (C = 10000000000000)
      EF02        C=C+1 M           Set mantissa to 1
                                    (C = 10000000001000)
      EF03        SETHEX            Set CPU to hexadecimal mode
      EF04        C=C-1 S&X         Set exponent & sign to FFF
                                    (C = 10000000001FFF)
      EF05        RTN               End of routine
```

To test the code, we may want to begin execution at EF00, have a breakpoint at EF02 to check partial operation, complete execution and check the final register contents. To check that it does not matter what the contents of C are before we start we shall set it to all 2s.

The user would perform the following steps:

```
KEYS PRESSED           DISPLAY          COMMENT

XEQ'DEBUG'             DEBUG CMD ?      Enter the Debugger
F                      FLAG __          User flag editor
15                     FLAG 15 CLR      Turn on the printer
R/S                    FLAG 15 SET      Printer on
ON                     DEBUG CMD ?      Return to menu,
                                        prints "FLAG 15 SET"
B                      NO BKPTS         Breakpoint editor
0                      BKPT0 @ ____     Breakpoint 0 unknown
EF02 R/S               BKPT0 @ EF02     Breakpoint set up
ON                     BKPTS ON         Breakpoints are active
ON                     DEBUG CMD ?      Return to main menu
R                      C:00 000,0,990   Enter register editor
SHIFT USER PRGM        C:13 000,1,000   Load C with all 2s
22222222222222         C:00 222,2,990   Display and position
                                        change as 2s keyed in
ON                     DEBUG CMD ?      Return to main menu
ENTER                  DEBUG CMD ?      Print CPU contents,
                                        shows C=22222222222222
E                      ENTRY @ ____     Enter code at address
EF00                   ENTRY @ EF00     required
R/S                    STK2 @ 00F0      Runs to breakpoint
R/S                    STK1 @ DBF5      Displays the stack
R/S                    BKPT @ EF02      And breakpoint address
R/S                    DEBUG CMD ?      N.B Stopped prior to
                                        executing word at EF02
ENTER                  DEBUG CMD ?      Print CPU contents,
```

```
                                    shows C=10000000000000
S                   PT=P P=0 Q=0    Shows pointer data
R/S                 ST=00 G=00 H    Shows other status
D                   ST=00 G=00 D    Set decimal mode
ENTER               ST=00 G=00 D    Print screen
ON                  DEBUG CMD ?     Now continue execution
C                   CONT @ EF02     Confirms address
R/S                 DEBUG CMD ?     Prints Continue addr,
                                    execution finishes
ENTER               DEBUG CMD ?     Print CPU contents,
                                    shows C=10000000001FFF
S                   PT=P P=0 Q=0    Enter status editor
R/S                 ST=00 G=00 H    CPU changed to
                                    hexadecimal mode
ENTER               ST=00 G=00 H    Print screen
ON                  DEBUG CMD ?     Return to main menu
B                   BKPTS ON        In breakpoint editor
C                   SURE (Y/N) ?    Clear all breakpoints
Y                   NO BKPTS        All cleared
ON                  DEBUG CMD ?     Return to main menu
ON                  0.0000          Return to normal mode,
                                    printer shows .END.
```

This example although trivial in nature shows most of the features of DEBUG. To actually enter and correct M-code programs it is necessary to use some other tool such as ZENROM or the DAVID Assembler. It should be noted that DEBUG sessions can be continued even if the machine is turned off or used for some other purpose during debugging (all the relevant information is 'hidden' in a protected area of memory).

19. DEBUG High Level Design

In this section the high level design of DEBUG is described briefly. As was mentioned earlier DEBUG has a hierarchical structure. From the main menu it is possible to access several sub-menus.

The system was built up from the most necessary components first, this made testing easier and allowed the simpler parts to be used to test the more complicated parts.

The first routines written were: CPU>B9, used to dump the CPU contents into a buffer (buffer number 9) for editing; B9>CPU, used to load the CPU with the contents of buffer 9; START, to get an entry address, load the CPU using B9>CPU, enter the code at the address specified and dump the final CPU contents using CPU>B9.

These three main subroutines use many other smaller routines and operating system routines. The routine MLOCB was actually the first written, it finds the location of buffer 9 in user RAM and is used not only in the routines above but by most of the sub-menu functions.

Many other subroutines were necessary to implement START which were also useful in other sub-menu functions. These included routines for display and keyboard handling, input string handling, and pseudo ROM access.

The remainder of the system was coded and tested using these basic functions. When testing of each function was complete, the appropriate access call was added to the main menu.

The sub-menu functions are:

FLAGE	The FOCAL flag editor allows the user to toggle any of the 56 user flags from within DEBUG. All display annunciators are automatically updated as they would be in normal operation of the machine.
STED	The status editor enables the user to view and change the contents of the ST and G registers, the positions of the pointers P and Q, the active pointer and the arithmetic mode of the CPU.
PRCPU	Prints the contents of the five main accumulators.
RED	Allows the user to edit the five accumulators.
BKED	Enables up to ten breakpoints to be set up in any pseudo ROM page which will temporarily halt execution if they are encountered, thus allowing the user to view or edit CPU contents and then continue.
CONTKB	Allows execution to continue from a breakpoint.

These functions and START, are accessed from the main menu by typing the first letter of the routines name (with the exception of PRCPU which is called by the ENTER key to retain consistency with ZENROM).

This consistency of operation was maintained throughout DEBUG, so that for example once in the status editor to change the position of pointer P, you simply press the P key.

Printing of DEBUG information is achieved by pressing the ENTER key when the required information is shown on the display. The R/S key (Run/Stop) is used throughout to confirm inputs and toggle the display between alternatives. Similarly the delete key can be used in most circumstances to delete the last value entered. This philosophy makes DEBUG very easy to use.

20. DEBUG ROM Entry Points

The following table gives the addresses and a brief description of all the separate routines in the ROM. Routines are listed in address order.

NAME	LOCATION	ENTRY	FUNCTION
MARKS 1B	084-08C	08C	ROM header
-DEV FNS	08D-095	095	Header for DEBUG functions
M+1	096-09C	099	Increment RAM address in M
M-1	09D-0A3	0A0	Decrement RAM address in M
DSPRPC	400-467	400	RED Display register, position and contents
RED	470-4E4	470	RED Main program
XCAT	500-584	504	Extended Catalogue function
TKEYH	58C-5AF	58C	Get hex key (timeout 1sec)
MAIND	5B0-5EE	5B0	Main status display of STED
ALTD	5EF-644	5EF	Alternate display for STED
TKEY	645-655	645	Get any key (timeout 1sec)
KEYDBU	656-664	656	Key debounce & wait for up
SURE	665-67E	665	Ask user if SURE (Y/N)?
DABKPT	67F-6AD	67F	Disable All Breakpoints
ENBKPT	6AE-6D5	6AE	Enable All Breakpoints
CONTBK	6D6-74B	6D6	Continue from Breakpoint
DBI	74C-76D	74C	Display BKPTn Information
EDBN	76E-82E	76E	Edit BKPTn (low level edit)
PRCPU	82F-8A1	82F	Print main CPU registers
STED	8FF-A20	903	CPU Status editor main body
DEBUG	A27-A94	A30	Entry to DEBUG & main menu
BKED	A95-ADA	A98	Breakpoint editor top level
GETNYB	ADB-B00	ADB	Get nybble from buffer 9 for the register editor
OUTHEX	B14-B4C	B14	Output hex digits & '_'s
START	B4D-B99	B4D	Get address & run code
GETDRS	B9A-BA7	B9A	Get delete or R/S keys
B9>CPU	BA8-BFF	BA8	Transfer buffer 9 to CPU, enter code, RTN or BKPT
CPU>B9	C00-C74	C00	Transfer CPU to buffer 9
FINDF	C75-C8E	C75	Find a user flag for FLAGE
GETHXA	C8F-C96	C8F	Get a hex keypress in A S&X
MK9	C9A-CA3	C9A	Finds/makes/finds buffer 9
FIND9	CA4-CAB	CA4	Finds buffer 9 or gives NO BUFFER error
ERRNB	CAC-CC0	CAC	NO BUFFER error routine
WRC1	CC1-CD8	CC1	Write Class 1 instruction
ADDDIG	CD9-CE4	CD9	Add digit to input string
DELDIG	CE5-CF2	CE5	Delete digit from input
FPT	CF3-CFF	CF3	Find pointer position

NAME	LOCATION	ENTRY	FUNCTION
MLOCB	D00-D1E	D00	Locate buffer subroutine
LCBUF	D1F-D3F	D24	Locate buffer function
MKBUF	D40-D46	D45	Make buffer function
MMKBF	D47-D76	D47	Make buffer subroutine
BUF?	D77-D8A	D7B	Function to test buffer
CLRBUF	D8B-DA1	D91	Function to clear a buffer
DELBUF	DA2-DBA	DA8	Function to delete a buffer
FLAGE	DBB-E49	DBB	User flag editor
CE9	E4A-E4E	E4A	Check for BKPT or RTN
RBKPT	E4F-EE3	E4F	Return from BKPT
DBN	EE4-EF7	EE4	Display BKPTn @
B0-B9	EFF-F13	none	Breakpoint address table
RB0-RB9	F14-F3B	none	Breakpoint overwritten code
DBS	F3C-F8D	F3C	Display breakpoint status
DAFN	F8E-F94	F8E	Disable & find BKPTn
FINDBN	F95-FA0	F95	Find BKPTn in table Bn
-none-	FAE-FFF	o/s	Buffer preservation, sign on message, power down IL, ROM trailer