# COMPUTER-1 -- A Modern Simple Computer to Introduce Computer Organization and Assembler Language Programming

Donald S. Miller
Computer Science Department
Arizona State University
Tempe, Arizona

## ABSTRACT

COMPUTER-1 is an interactive editor/assembler simulator-debugger and program evaluator to be used as an instructional tool for an introductory course in computer organization and assembler language programming. COMPUTER-1's organization, assembler language and interactive facilities are designed to introduce basic concepts of computer architecture and assembly language programming while minimizing the amount of computer system dependent details present during this learning period. COMPUTER-1 is a decimal machine with a small modern single address instruction repertoire. A run-time view into COMPUTER-1's memory and registers is provided to help in program understanding and debugging. COMPUTER-1 provides a flexible instructor-oriented method for specifying and evaluating programming assignments and a way for students to determine whether and how well their programs have worked. COMPUTER-1 runs under UNIX and presumes the availability of a CRT with full-screen cursor addressability. COMPUTER-1 is a modern descendent of BASIC1 [1] in that it simulates a more contemporary architecture and possesses interactive features which are not tied to the capabilities of hard copy terminals or card readers.

## I. Introduction and background

It is well known that a major problem in teaching an introductory course in computer organization and assembler language programming such as ACM Curriculum '78 CS 3 "Introduction to Computer Systems," [2] is the amount of detailed information about the hardware and software of the computer system being used which must be concurrently explained. This includes computer system command string interpreter or JCL, complex computer architecture and assembler language syntax, text editor and terminal usage, binary and hexadecimal or octal number systems, and rules for performing the assembly/link/load/execute steps, and input and output on the given system. The obfuscation provided by these details makes teaching and learning the basic concepts far more difficult for two reasons:

1. The amount of material which has to be absorbed initially is huge.
2. It is difficult to separate the basic concepts from computer system dependent details.

There have been many attempts over the years to narrow the "environment", simplify the machine and/or its assembler language, simplify I/O and reduce or simplify the steps between source code composition and program execution that the student must perform. These include MIX [3], ASSIST [4] and BASIC1[1] used by the author at Washington State University and many others. COMPUTER-1 was developed as a direct result of experience using BASIC1.

BASIC1 is an interactive assembler/loader/interpreter which includes the following characteristics [1]:

1. a simple hardware structure--a uniform, word organized memory, a minimum of programmable registers and basic instruction and data fetch sequencing
2. a small instruction repertoire containing the fundamental operations and addressing modes--all instructions are of the same length and format
3. decimal numbers used for all addresses, data, op-codes and arithmetic
4. a straightforward and simple method for performing input/output
5. an assembler possessing a minimum of syntactic rules, only essential pseudo-ops and which accepts free-format source input and provides diagnostics in English
6. a loader which requires no user control information
7. a run-time executive which

a. can execute the loaded program and its data with no further user control information
b. provides substantial offline debugging aids--a dump, trace, and intelligible diagnostics
c. has runtime interactive data and command input/output capability including the ability to set breakpoints, examine and set memory and to step or run through program segments with or without tracing.

A co-program, another assembler/loader/interpreter named TESTER provides a method to determine whether programs really work and in some sense how "good" they are. BASIC1 was designed to incrementally introduce students to the complexities of assembler language programming. A "basic" instruction set consisting of add, subtract, store, branch if less or equal, input, output, halt and no operation, and only one addressing mode for each instruction was presented first. Later this was expanded to "partial" or "full" instruction sets which included loading, immediate data, indirect addresses, conditional and unconditional branching, digit manipulations, transfer of control, loop control and stack manipulation. Experience with BASIC1 indicated by student and instructor reactions over a long period of time was very favorable. During the conversion of the BASIC1 source code from FORTRAN to C and the simultaneous change to a more terminal-oriented and interactive version it became very clear that BASIC1 had two fundamental shortcomings [1].

1. It represents a machine and a method of studying machines which comes out of the 60's. Two examples -- BASIC1 has no index register and the assembler syntax does not include a label field.

2. It's runtime interactive features are tied to the technology of hard copy terminals and punched cards. Very little use was made of the full screen cursor addressability feature present in many terminals these days nor of the features available from a modern hierarchical file system and command string interpreter such as those found in UNIX. As examples -- a multiple logical area screen display part of which displays memory and registers, another part of which displays computer messages with another part reserved for echoing user input, any or all of which could be updated independently could provide a lot more flexibility than BASIC1's line-at-a-time scrolled input. As a second example, the UNIX command string interpreter (the shell) and its file system make editing and running a user program on his own test data and on the instructor's test data as well as directing hard copy to the printer all possible from within an interactive computer simulation.

Observations such as these led to the design of COMPUTER-1 [5].

## II. COMPUTER-1 Organization and Programming

The organization of COMPUTER-1 is shown in Figure 1. It has a 999 word memory and three

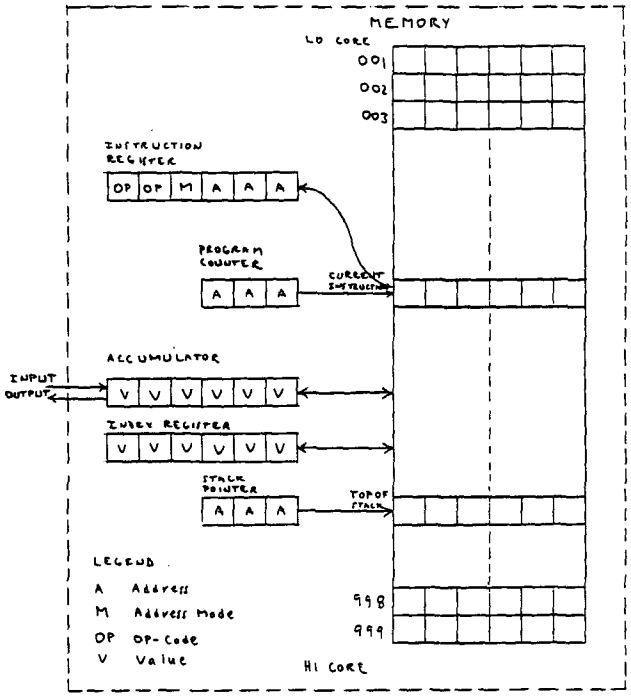


FIGURE 1. COMPUTER-1 COMPUTER ORGANIZATION

programmable registers, an accumulator, an index register and a stack pointer. Memory words, the accumulator and index register hold six decimal digits and sign. The stack pointer and program counter hold three-digit memory addresses. The program counter is incremented once per instruction cycle and its contents may be altered with program control type instructions. The stack pointer points to the top of the current stack. Instructions consist of a two digit op-code, an address mode digit and a three digit operand. The total instruction repertoire consists of 23 instructions which can have from one to six addressing modes. Initially only the "basic" instruction set of add, subtract, store, branch less equal, input, output, exit and nop using direct or implied addressing modes is presented. Students code mnemonics for op-codes, labels or absolute numbers for addresses and decimal signed magnitude representation for data. The COMPUTER-1 assembler has only three pseudo-ops, dc, save and end. Each instruction in the basic set, its op-code, operand, corresponding machine instruction, a Pascal like notation description and an instruction description are given in Figure 2a, as they would be presented to students. Input is to the accumulator from an external data set and output is to an external data set from the accumulator. COMPUTER-1 assembler pseudo-ops are given in Figure 2b. COMPUTER-1 source code is nearly free format with spaces as a delimiter. As shown in Figure 3, each source code

| Instruction Name | Assembler Mnemonic | Machine Instruction | Operation | Description |
|---|---|---|---|---|
| add | add loc | 01lLOC | acc := acc + loc | add the contents of loc to acc; leave loc unchanged |
| subtract | sub loc | 02lLOC | acc := acc - loc | subtract the contents of loc from acc; leave loc unchanged |
| store | store loc | 03lLOC | loc := acc | store contents of acc at loc; leave acc unchanged |
| branch if ≤ 0 | ble place | 08lLOC | if (acc < = 0) go to place | branch to place if contents of acc is ≤ 0 else take next instruction in sequence |
| input | in | 040000 | acc =: input number | read next input value into acc |
| output | out | 050000 | output (acc) | write contents of acc to output; acc is unchanged |
| exit | exit | 060000 | stop | stop COMPUTER-1 execution and return to command mode |
| no operation | nop | 070000 | none | do nothing |

(a)

| Name | Mnemonic | Description |
|---|---|---|
| define constant | dc value | put value into memory at current location |
| save space | save number | set the next number locations to zero |
| end delimiter | end <loc> | signals end of source program; execution starts at l or loc |

(b)

Figure 2. COMPUTER-1 "basic" Instruction Set (a) General Instructions, (b) Pseudo-operations

line contains one or more of the following: a label, a machine instruction or pseudo-op, an operand and a comment. Instructions consist of the full instruction name or a mnemonic for it. The operand may be a label defined in the program or a decimal number representing either an address or data. Simple address addition is permitted. In addition, as described below, the operand field may contain a symbol indicating addressing mode.

```
label:          instruction operand      ; comment
```

label: a character string starting with a letter and ending with ":"

instruction: a character string specifying an instruction

operand: specifies memory location, register or literal data

comment: preceded by ";" and ignored by COMPUTER-1

Figure 3. COMPUTER-1 Instruction Format

Figure 5 contains the source text of the initial design of a simple program to input a number, multiply it by ten, output it and loop back for another number. Suitable programming assignments using only the basic 8 instruction subset described so far would be serially reusable programs which perform a left shift or a right shift of up to 5 places determined by an input parameter. The complete set of addressing modes and the full instruction set given in direct or implied modes are presented in Appendices I and II respectively. Typical programming assignments are to implement a fixed point multiply or divide, perform a bubble sort or play the game of REVERSE.

## III. COMPUTER-1 Control and Runtime Interaction

COMPUTER-1 has a set of commands to assist in program composition, assembling running, debugging, evaluation and understanding. These are listed in Appendix III. COMPUTER-1 commands fall into three general categories: control of COMPUTER-1 execution, debugging aids and general utility. Program control commands cause assembly and initiate execution in various modes. A program can be run to termination or a breakpoint is reached, it can be manually single stepped or automatically stepped at a user defined rate. Input data can be reused. Debugging commands provide breakpoint control, memory window displays and core dumps, memory setting, flexible program patching, program testing using the instructor's test cases and listing of program plus changes. General utilities provide an ability to enter the editor to compose or modify text, list at the terminal or the line printer, define a source file, specify terminal type, get help on how to use COMPUTER-1, execute any UNIX command and exit COMPUTER-1. The example described below and in Figures 4-10 presents some of COMPUTER-1's available interactive features.

```
%c1

Greetings student - the right shift basic set program is due by
sundown tomorrow!



--> edit by10
```

Figure 4. Initiating a COMPUTER-1 session

After logon, the user responds to the UNIX shell "%" prompt with

```
c1
```

to execute COMPUTER-1. COMPUTER-1 outputs any messages from the instructor that may be present followed by a "-->" prompt at the lower left hand corner of the screen as shown in Figure 4. The programmer inputs

```
edit by10
```

to enter the UNIX text editor vi [6] to compose a program which multiplies an input number by ten and loops back to input another number. The source code is to be stored on a file named by10. The initial text of the program is given in Figure 5.

```
;
;       Program to multiply the input by 10
;
more:
        in              ; acc := input
        add a           ; acc := acc + acc
        store t         ; t := 2*acc
        add t           ; acc := 4*acc
        store m         ; m := 4*acc
        add m           ; acc := 8*acc
        add t           ; acc := 10*acc
        out             ; print(10*acc)
back:   ble more        ; input another number
        exit            ; that's all
t:      dc 0
m:      dc 0
        end
```

Figure 5. Initial source text of multiply by10 program

When the programmer quits the editor, he is returned to COMPUTER-1 as indicated by the "-->" in lower left corner of screen. The user types

```
go
```

COMPUTER-1 responds with

```
Input file to be assembled
```

The user enters

```
by10
```

which causes COMPUTER-1 to assemble by10, load it into COMPUTER-1 (i.e., into COMPUTER-1's interpreter), initiate execution and generate the display shown in Figure 6 when it reaches a point where user interaction is required - in this case to provide the input requested by by10's first instruction. The five logical areas in the display are indicated by the numbers in the left hand column (which are not present in the actual

```
1:      Last instruction:
        Current instruction:  in
        Next instruction:     add a


2:


3:      PC          Accum           Index           Stack Pointer
        1             0               0               999
4:  *   Input:
5: ->
```

Figure 6. COMPUTER-1 display during execution of first instruction of by10

display). Area 1 contains the previous, current and next instruction. Area 2 is for windows into memory (see below). Area 3 displays register contents. Area 4 displays messages from COMPUTER-1. The most recent message is indicated by an "*". Area 5 contains the command prompt and error messages. COMPUTER-1 is prompting for input. The user responds with

```
10
```

and COMPUTER-1 executes the remainder of the program after which the display is updated as shown in Figure 7. After checking the operation of by10 on other numbers using the commands "go 1"

```
Last instruction:    ble 1
Current instruction: exit
Next instruction:



PC          Accum           Index           Stack Pointer
10            100             0               999

 Input: [10]
 Output (loc 8): 100
* Execution stopped at: 10

->
```

Figure 7. COMPUTER-1 display after completion of last instruction of by10

or "go more" the programmer tries by10 on the instructor's test data by entering

```
tester mult
```

COMPUTER-1 responds taciturnly

```
Program did not work
```

The programmer realizing that his program failed because it will not loop back for products greater than zero hits the "DEL" or "BREAK" key causing an interrupt message to appear and COMPUTER-1 to enter the command mode. Then he patches his program with

```
replace back
```

COMPUTER-1 responds

```
Input new instruction
:
```

and the programmer enters

```
br more
```

and then

```
dump more back
```

to verify that the change was made. The resultant COMPUTER-1 display is shown in Figure 8. To further check operation the user enters

```
display t
go more
```

and in response to the input request at location 1

```
10
```

```
--> dump more back
  1 in
  2 add a
  3 store 11
  4 add 11
  5 store 12
  6 add 12
  7 add 11
  8 out
  9 br 1
-->
```

Figure 8. COMPUTER-1 dump after replacing ble more with br more

The resulting display, given in Figure 9, indicates that the program did in fact loop back for more data, an output of 100 occurred and the appropriate intermediate values were stored in t and m as indicated in the eight memory location window

```
          Last instruction:     br 1
          Current instruction:  in
          Next instruction:     add a

11:   40      80      0       0       0       0       0

          PC          Accum           Index       Stack Pointer
          10           100             0            999

  Input: [10]
  Output (loc 8): 100
* Input:
```

Figure 9. COMPUTER-1 display after processing an input of 10 with patched version of by10

display. The programmer tries the instructor's test data again with

```
tester mult
```

and COMPUTER-1 responds with

```
Program worked for test data: mult
In 44 steps, using 12 core.  Figure of Merit: 528
```

telling the programmer that his program worked and had a figure of merit of 528 the product of memory used and instructions executed. The programmer must now enter his change on his source text file by10 using the editor. After returning to COMPUTER-1 he enters

```
submit
```

to get the listing shown in Figure 10 containing the source text preceded by three columns containing source code line number, memory location and the COMPUTER-1 machine language instruction. COMPUTER-1 has additional useful debugging aids such as breakpoint control and the ability to step through a program at varying rates which are not demonstrated in the above example.

```
Computer-1 listing of: by10      Page 1

  1              ;
  2              ;       Program to multiply the input by 10
  3              ;
  4    1                 more:
  5    1   040000              in              ; acc := input
  6    2   012000              add a           ; acc := acc + acc
  7    3   031011              store t         ; t := 2*acc
  8    4   011011              add t           ; acc := 4*acc
  9    5   031012              store m         ; m := 4*acc
 10    6   011012              add m           ; acc := 8*acc
 11    7   011011              add t           ; acc := 10*acc
 12    8   050000              out             ; print(10*acc)
 13    9   141001        back: br more         ; loop back for more
 14   10   060000              exit            ; that's all
 15   11        0        t:    dc 0
 16   12        0        m:    dc 0
                              end

Program worked for test data: mult
In 44 steps, using 12 core. Figure of Merit: 528
```

Figure 10. COMPUTER-1 line printer listing of final version of by10 with instructor test data results

## IV. Instructor Features

COMPUTER-1 has been designed to provide maximum assistance to the instructor. He may specify a file for his messages to be displayed whenever COMPUTER-1 is executed. Test inputs are prepared by writing to a file in the instructor's directory a header line consisting of the allowed instruction set, total number of inputs and total number of outputs, followed by the inputs and outputs as they occur during testing. For example writing

```
full  5  5
1
10
-1
-10
0
0
99999
999999
-99999
-999999
```

to the file mult would have provided a reasonably good set of test data for the example in the previous section. It is also possible to set things up so that the students execute COMPUTER-1 directly instead of a UNIX shell when they log on by modifying the password file for their accounts thus creating an almost total COMPUTER-1 environment. As noted COMPUTER-1 evaluates the students programs.

## V. Summary

COMPUTER-1 was designed as a modern replacement for BASIC1. The objective was to keep as many of the instructional features which have worked so well over the years while

1. including architectural and software features commonly found in contemporary computer systems

2. more fully utilizing the full screen addressability feature commonly found in today's CRTs and the software features found in modern process-oriented operating systems such as UNIX.

To achieve the first objective an index register, a more general set of addressing modes, modern subroutine jump and return and in-memory increment and decrement instructions were added to the hardware and labels and very simple address arithmetic were added to the assembler syntax. A new set of commands was written to achieve the second objective. These feature a five field display presenting windows into the current instruction area, memory and all registers and message and prompt areas. Each field is independently updatable. These windows can be examined as a program is stepped at user determined rates through its instructions. Such initially difficult concepts as the distinctions between addresses and the values stored at addresses, loading and storing, indexing, indirection and immediate data are all displayed in an obvious manner. Dumps are seldom needed if one can view several sections of memory while a program is being executed. The flow of more complicated algorithms, e.g., a bubble sort can be viewed at a convenient rate to search for logical errors. The UNIX file and shell features were utilized to implement runtime patching, testing and printing and the instructor oriented test program and message generation. These features were also utilized to implement the file and entry/exit linkages between COMPUTER-1 and the text editor.

Many needed less important enhancements to BASIC1 were incorporated into COMPUTER-1. A few of these follow. The message and prompt-error message display areas enable a more comprehensive and timely set of error diagnostics and user information to be provided. Breakpoint setting is more fine grained depending on the type of access to a given location. User data memory and execution time are both reduced as a result of the decrease in memory size to 999 words. And lastly, adding a tag byte to every memory location has finally put an end to attempts to get low evaluation numbers by using memory areas not defined in the source program.

COMPUTER-1 still requires that the student learn how to use a text editor concurrently with the basic concepts of assembler language programming. Fortunately, vi can be invoked in a rather easy to learn and use line-oriented editor "ed", for which an excellent tutorial exists [7]. It is not necessary to use the more complicated "visual" features of vi to compose source programs of the size that are required in COMPUTER-1 programming assignments. A more serious criticism of COMPUTER-1 is that the efforts to modernize have added some complexity to COMPUTER-1's architecture and its commands. This could have considerable impact on the primary objective of COMPUTER-1 contained in the title of this paper. The author has no classroom experience with COMPUTER-1 at this time on which to base a judgement. Two desirable features which are not in the current version of COMPUTER-1 are the ability to scroll through COMPUTER-1 memory and to single cycle through the fetch, increment and execute parts of an instruction. Neither of these is particularly difficult to implement.

## VI. Current Status

COMPUTER-1 is written in C and ran on the Computer Science Department PDP 11/60 under UNIX Level 6 at Washington State University. It since has been ported to the Computer Science Department VAX 11/780 at Arizona State University where it runs under the Virtual VAX-11 version of Berkeley UNIX 4.1 bsd. COMPUTER-1 has a 15 K Byte shareable text segment and a 15 K Byte data segment per user. Current CS 3 class size of 250 students per semester necessitates that CS 3 programming be done on the ASU Academic Computing Services IBM 3081 or PDP-11/70's neither of which currently support UNIX. Inquiries by potential users of COMPUTER-1 should be addressed to the author.

## Acknowledgements

COMPUTER-1 is the MS project of Stephen R. Zeck. Many of the innovative ideas contained within it are completely his. COMPUTER-1 has been ported to the ASU VAX 11/780 by the combined efforts of Cheng Ta Yu, Shih-Shan Tan and Bruce R. Millard. The author would also like to thank Jody Dean for her excellent typing of the draft and Mildred Fort for her excellent typing of the camera ready copy.

## References

[1] D. S. Miller and B. R. Millard, "BASIC1 -- a Simple Computer to Introduce Computer Organization and Assembler Language Programming," ACM SIGCSE Bulletin, February 1982, Volume 14, Number 1, pp. 71-77.

[2] R. H. Austing, B. H. Barnes, D. T. Bonnette, G. L. Engel, G. Stokes, "Curriculum '78: Recommendations for the Undergraduate Program in Computer Science--A Report of the ACM Curriculum Committee on Computer Science," Communications of the ACM, Volume 22, pp. 147-166, March 1979.

[3] James L. Peterson, Computer Organization and Assembly Language Programming, Academic Press, New York, N. Y., 1978.

[4] R. A. Overbeek and W. E. Singletary, Assembly Language with Assist, Science Research Associatates, Chicago, IL, 1976.

[5] S. R. Zeck, "A Modern Simple Computer to Introduce Assembler Language and Computer Organization," Master of Science Project, Computer Science Department, Washington State University, Pullman, Washington, January, 1981

[6] W. Joy and M. Horton, "An Introduction to Display Editing with VI," Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1979.

[7] B. W. Kernighan, "A Tutorial Introduction to the UNIX Text Editor," Bell Laboratories, Murray Hill, N. J., 1975.

## Appendix I. COMPUTER-1 Addressing Modes

| Mode | Operand Format Assembler Syntax | Machine Code[1] | Address Description |
|---|---|---|---|
| Implied | | OP0000 | Implied by instruction |
| Direct | loc[2] | OP1LOC | MEM[loc] |
| Register | a | OP2000 | Accumulator |
| | i | OP3000 | Index Register |
| | sp | OP4000 | Stack Pointer |
| Indexed | loc(i) | OP5LOC | MEM[loc + index] |
| | loc(sp) | OP6LOC | MEM[loc + stack pointer] |
| Immediate | #value[2] | OP7LIT | |
| Indirect | (loc) | OP8LOC | MEM[MEM[loc]] |

Notes:

1. OP is 2 digit op-code
   LOC is 3 digit location
   LIT is 3 digit constant

2. loc and value are a number or a label or the sum of two of these

## Appendix II. COMPUTER-1 Full Instruction Set

Arithmetic:

| Instruction | Opcode | Addressing Modes | Description[1] |
|---|---|---|---|
| add | 1 | All | acc := acc + source; |
| sub | 2 | All | acc := acc - source; |
| inc | 16 | All but immediate | des := des + 1 |
| dec | 17 | All but immediate | des := des - 1 |
| sl | 20 | Albsolute, Indirect Indexed | acc := acc*10**source; |
| sr | 21 | Absolute, Indirect, Indexed | acc := acc/10**source; |

Data Movement:

| | | | |
|---|---|---|---|
| load | 15 | All | acc := source; |
| store | 3 | All but immediate | des := acc; |
| in | 4 | Implied | acc := input number; |
| out | 5 | Implied | output(acc); |
| push | 22 | All | sp:=sp - 1;<br>memory[sp] := source; |
| pop | 23 | All but immediate | des := memory[sp];<br>sp := sp + 1; |

Program Control:

| | | | |
|---|---|---|---|
| br | 14 | (All branches | goto place |
| bgt | 13 | Absolute or | if (acc>0) goto place; |
| blt | 12 | Indirect or | if (acc<0) goto place; |
| bge | 11 | Indexed) | if (acc>=0) goto place; |
| ble | 8 | | if (acc<=0) goto place; |
| bne | 10 | | if (acc≠0) goto place; |
| bz | 9 | | if (acc=0) goto place; |
| jsr | 18 | Absolute or<br>Indirect or<br>Indexed | sp := sp - 1;<br>memory[sp] := pc;<br>goto place; |
| return | 19 | Implied | pc := memory[sp];<br>sp := sp + 1; |
| exit | 6 | Implied | stop execution |
| nop | 7 | Implied | do nothing |

Note:

1. source is the source operand value
   destination is the destination memory location or register
   place is the operand effective address

## Appendix III. COMPUTER-1 Command Summary

Program Control[1]

| | |
|---|---|
| go <loc> | starts program at current pc or loc |
| single <loc> | executes one instruction at current pc or loc |
| speed <value> | sets delay to zero or value seconds |
| continue <count> <loc> | executes count instructions starting at current pc or at loc with delay seconds between cycles |
| rewind | reset input to point to first data value |

Debugging

| | |
|---|---|
| break <write> list | sets a list of breakpoints; if loc is an instruction, breakpoint is after instruction execution; for data accesses breakpoint is after access; write specifies only for writes |
| display list | display 8 memory locations starting at loc |
| show | list breakpoints, memory windows and delay |
| clear (break/display) (list/ALL) | clear the indicated breakpoints or display windows |
| dump <loc 1> <loc 2> | dump from loc 1 to loc 2 |
| set loc value | set loc to value |
| delete loc | replace instruction at loc with nop |
| replace loc | replace instruction at loc with the one prompted for |
| insert loc | like replace but makes room for the prompted instruction before the one at loc |
| tester case prog | tests prog with case data |
| submit case prog | like tester but listing to line printer |

General Utility

| | |
|---|---|
| help <name> | lists brief description of all commands or name |
| list <name> | lists current program or name at terminal<br>assemble name if not already assembled |
| print <name> | like list except output to line printer |
| quit | terminate COMPUTER-1 |
| file <prog> | display current file source or set it to prog |
| tty name | tell COMPUTER-1 your terminal type |
| edit | call vi with current file if defined |

Note:

1. parameters in < > are optional
   loc is a memory location indicated by a number or a symbol or the sum of two of them
   list is a series of locs separated by blanks
   one parameter of each pair inside ( ) must be selected