Caltech10 CPU

Simulator

User's Manual

version 1.0

.

Introduction

The simulator is a program that simulates a given microprocessor and supplies the user with a convenient command interface for testing and debugging programs. This command interface consists of a number of one and two keystroke commands. With these commands the user can run or single step their program, examine and modify simulator memory, and manipulate breakpoints.

Throughout the manual, characters enclosed in angle brackets ('<' and '>') indicate a single keystroke. For example, '<ESC>' indicates the key labeled ESC on the terminal. Also, '<CR>' indicates a carriage return (Enter key) and '<LF>' a linefeed.

To run the simulator type simCaltech10 in response to the system prompt. The simulator will return with its identifying message and will then wait for a command. Note that the simulator has no prompt. To exit the simulator a 'Q' may be used at any time. While a program is running on the simulator a <CTRL/C> may be used to suspend program execution and return control to the user.

Commands

As previously mentioned, most simulator commands are single characters. This is done to allow the user to use auto-repeat to do continuous or repetitive testing.

Most commands take a numeric argument which is typed before the command. This argument is always in hexadecimal and therefore will consist of the characters 0 through 9 and A through F. Those commands which take a word argument (four hex digits) will use only the last four digits typed and ignore all preceding digits. For example, if '123456' is typed as a word argument, 3456 will actually be used as the argument. Those commands which take a byte argument will only use the last two digits typed (*i.e.* 56 in the previous example). If not enough digits are supplied for a command the number is padded on the left with zeros (*i.e.* for a word argument '123' becomes 0123). Those commands which take no arguments simply ignore all digits preceding them. Beware, however, that some commands act differently depending on whether they are given an argument or not (register and breakpoint commands for example).

Many commands which take arguments also have a default argument which is used if no argument is supplied. The defaults are chosen to facilitate the use of auto-repeat. For example, the default location for a deposit instruction is the last location examined. Whenever an argument is supplied for a command the default value for that command is changed to the supplied value. Notice too, that similar sets of commands use the same default values. There are four defaults used by the simulator. They are:

Default	Descripton
FILL	the last value deposited to memory
LOC	the last memory location examined
PC	the current PC
START	the last address supplied for starting execution

The character '*' is used by the simulator as shorthand for the current contents of the program counter (PC). This is analogous to the meaning given '*' by many assemblers. The asterisk may be given as an argument at any time and is equivalent to having typed the value of the PC. When '*' is typed the value of the PC will be echoed. The following examples will illustrate this. If '*' is input and the PC contains 3456, '*3456' will appear on the screen and 3456 will be used as the argument to any command now typed. Since typing '*' is equivalent to entering the contents of the PC by hand, typing '*' then '78' will echo as '*345678' and 5678 will be used as the argument to a subsequent command.

The simulator has five sets of related commands. These are: examine and deposit commands, register commands, execution commands, breakpoint commands, and miscellaneous commands. The simulator error message is a question mark ('?') followed by a beep. Any command error will cause this message to be output. Some errors will also cause further explanatory information to be output.

Following is a description of all of the simulator's commands by command type. Throughout these descriptions "**n**" represents a hexadecimal number entered by the user. All numbers in the examples are, of course, in hexadecimal. In these examples the computer response is italicized.

Examine and Deposit Commands

Only an open memory location may be deposited to. If a memory location is not open an error message will be output. A location is opened only if it has just been examined, otherwise it is closed. The only command which just opens a memory location is the '/' command (this also examines it). Some of the deposit commands also open memory locations after depositing to memory. Any command which does not examine a memory location closes any open memory location. The only time a memory location will be opened without using an examine or deposit command is after a trace ('T' or 'X') command when the memory location pointed to by the PC is opened. Below are the examine and deposit commands.

n/ Open and examine the contents of memory location **n** (word). If not specified **n** defaults to LOC, else LOC is set to **n**. If the current memory space being accessed is program memory, addresses are given as thirteen bit values (4 hex digits). Otherwise addresses are given as eight bit values (2 hex digits) to access data memory or I/O space (whichever is currently being used).

Examples:

```
34/03  /03  37/17
```

Location 34 is examined, LOC is set to 34 and the contents of 34 are displayed (03). Location LOC (34) is then examined again and found to contain 03, of course. Finally, location 37 is examined (contains 17), opening it and setting LOC to 37. At this point, LOC is 37, FILL is unchanged, and location 37 is open.

n<CR> Deposit **n** to the presently open memory location and close it. If **n** was specified FILL is set to **n**. If no **n** was given both the location and FILL are left unchanged and the locations is closed. If the current memory space being accessed is program memory a word is written to memory by the command. Otherwise a byte is written to data memory or I/O space (whichever is currently being used).

Examples:

```
35/05  <CR>
38/18  25<CR>
```

Location 35 is opened and examined by the '/' command, but not changed and closed by the '<CR>'. Location 38 is opened, examined (contains 18), and its contents are changed to 25. Now LOC is 38, FILL is 25, and no memory location is open.

n+ Deposit **n** to the presently open memory location and examine the next location. If **n** is not given the location is left unchanged. If **n** was specified the contents of the opened location are changed to **n** and FILL is set to **n**. After depositing **n**, LOC is incremented and this new memory location is examined (opening it). If the current memory space being accessed is program memory, addresses are output as thirteen bit values (4 hex digits) and words are written to program memory by the command. Otherwise addresses are output as eight bit values (2 hex digits) and bytes are written to data memory or I/O space (whichever is currently being used).

Examples:

```
43/37  +
44/38  42+
45/39  +
46/40
```

Location 43 is opened and examined, but not changed. Location 44 is then opened and examined (due to the '+' command) and its contents are changed to 42. Locations 45 and 46 are then opened and examined, but not changed, through the '+' command. LOC is now 46, FILL is 42, and location 46 is open.

n^ Deposit **n** to the presently open memory location and examine the previous location. If **n** is not given the location is left unchanged. If **n** was specified FILL is set to **n** and **n** is stored at the open memory location. After depositing **n**, LOC is decremented and this new memory location is examined (opening it). If the current memory space being accessed is program memory, addresses are output as thirteen bit values (4 hex digits) and words are written to program memory by the

command. Otherwise addresses are output as eight bit values (2 hex digits) and bytes are written to data memory or I/O space (whichever is currently being used).
Examples:

```
    E0/50  23^
    DF/4F  22^
    DE/4E
```

Location E0 is opened and examined and changed to 23. Next, location DF is opened and examined through the action of the '^' command. It is then changed to 22 and location DE is opened and examined. At the end of this sequence LOC is DE, FILL is 22, and location DE is open.

n; Deposit **n** to the presently open memory location and re-examine it. If **n** is not given the location is left unchanged. If **n** was specified, **n** is stored at LOC (if open) and FILL is set to **n**. LOC will always be left unchanged and open. If the current memory space being accessed is program memory, addresses are output as thirteen bit values (4 hex digits) and 16-bit words are written to program memory by the command. Otherwise addresses are output as eight bit values (2 hex digits) and 8-bit bytes are written to data memory or I/O space (whichever is currently being used).
Examples:

```
    C0/60  ;
    C0/60  8E;
    C0/8E
```

Location C0 is opened and examined. It is then re-opened and examined with the ';' command. Then 8E is stored at C0 and it is opened and examined yet another time, again using the ';' command. Note that when it is examined this time its contents are 8E not 60. LOC is now C0, FILL is now 8E, and location C0 is open.

n. Deposit **n** to the presently open memory location and examine the next location. If **n** is not given it defaults to FILL. If **n** was specified FILL is set to **n**. After depositing to memory, LOC is incremented and this new memory location is examined (opening it). Unlike the '<CR>', '+', '^', and ';' commands, this command always changes the presently open location. If the current memory space being accessed is program memory, addresses are output as thirteen bit values (4 hex digits) and words are written to memory by the command. Otherwise addresses are output as eight bit values (2 hex digits) and bytes are written to data memory or I/O space (whichever is currently being used).
Examples:

```
    B1/61  32.
    B2/62  .
    B3/63  33.
    B4/64
```

Location B1 is opened and examined. It is then changed to 32 and location B2 is examined with the '.' command. At this point FILL is 32. Next, location B2 is changed to 32 and location B3 is opened and examined. Finally, location B3 is changed to 33 and location B4 is opened. LOC is now B4, FILL is 33, locations B1 and B2 contain 32, location B3 contains 33, and locations B4 is open.

nZ Fill the next eight (8) or sixteen (16) memory locations with **n** (word or byte). If the current memory space being accessed is program memory, eight words are filled. Otherwise sixteen bytes are filled. If **n** is not specified it defaults to FILL, else FILL is set to **n**. The eight words (program memory) or sixteen bytes (data memory or I/O space) following those changed are then displayed in the format of the 'M' command. Additionally, the memory location following the changed values (the first of the eight or sixteen displayed) is opened and examined. LOC is updated to this memory location.
Examples:

```
    50/70  0Z
    60  80 81 82 83 84 85 86 87  88 89 8A 8B 8C 8D 8E 8F  60/80  Z
    70  90 91 92 93 94 95 96 97  98 99 9A 9B 9C 9D 9E 9F  70/90
```

Location 50 is opened and examined. Locations 50 to 5F are then changed to 0 with the 'Z' command. This also causes locations 60 to 6F to be displayed and location 60 to be opened and examined. The next 'Z' command stores a 0 at locations 60 through 6F (FILL is 0) and displays locations 70 to 7F and examines and opens location 70. Now LOC is 70, FILL is 0, and location 70 is open.

nM Display the next eight (8) or sixteen (16) memory locations starting at **n**. If **n** is not given it defaults to LOC. After displaying memory LOC is set to the address of the word or byte following the last word or byte displayed. This closes any open memory location. If the current memory space being accessed is program memory, eight words are output by the command. Otherwise sixteen bytes from either data memory or I/O space (whichever is currently being used) are output.

Examples:

```
30M
30  A0 A1 A2 A3 A4 A5 A6 A7  A8 A9 AA AB AC AD AE AF  M
40  B0 B1 B2 B3 B4 B5 B6 B7  B8 B9 BA BB BC BD BE BF  80M
80  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F
```

Locations 30 to 3F are displayed. Then locations 40 to 4F are displayed (LOC was 40 after first 'M' command). Finally, locations 80 thru 8F are displayed. At the end of all this, LOC is 90, FILL is unchanged, and no memory location is open.

nN Display the previous eight (8) or sixteen (16) memory locations from **n**. **n** is the address of the location following the last location to be displayed. That is, locations (**n** - 9) or (**n** - 17) to (**n** - 1) will be displayed. If **n** is not given it defaults to LOC. After displaying memory LOC is set to the address of the first of the word or byte displayed. The locations are still displayed with increasing addresses as you move to the right across the output. This command closes any open memory location. If the current memory space being accessed is program memory, eight words are output by the command. Otherwise sixteen bytes from either data memory or I/O space (whichever is currently being used) is output.

Examples:

```
40N
30  F0 F1 F2 F3 F4 F5 F6 F7  F8 F9 FA FB FC FD FE FF  N
20  E0 E1 E2 E3 E4 E5 E6 E7  E8 E9 EA EB EC ED EE EF  80N
70  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F
```

Locations 30 to 3F are displayed. Next, locations 20 thru 2F are displayed (LOC was 30 after the first 'N' command). Finally, locations 70 to 7F are examined. When completed, LOC is 70, FILL is unchanged, and no memory locations are open.

NOTE: Deposit commands work only immediately following a command which opens and examines a memory location (ie. '/', '+', '^|', ';', '.', 'T', 'X', or 'Z'). At any other time (no open memory location) an error message will be output when an attempt is made to deposit to memory.

Register Commands

The register commands are used to display and alter the contents of the CPU registers. They will always close any open memory location.

R Display the contents of the CPU registers.

Examples:

```
R
A:00  F:C0  X:00  S:00  P:0000
```

The registers are displayed.

nR**r** Deposit **n** (byte or word, depending on the register) to register **r**. Note that if **n** is not specified this becomes the register display command and **r** (a register name) should not be input. **r** is a letter ('A', 'F', 'P', 'S', or 'X') which specifies the register to change.

Examples:

```
35RA
20RS
1300RP
R
A:35  F:C0  X:00  S:20  P:1300
```

Stores a 35 in register A. Next, a 20 is stored in the stack pointer (register S) and then 1300 is stored in the program counter (register P). Finally, the registers are displayed with the 'R' command, reflecting the new values. Note that the simulator only outputs a newline in response to this command.

Execution commands

The execution commands are for executing user software. Breakpoints are active (if enabled) while a program is running in response to a 'G', 'S', or 'P' command, and for certain cases an 'X' command. Interrupts can also be active while running under these commands.

nG Start execution at location **n**. If not given, **n** defaults to START, else START is set to **n**. The breakpoints will be active if enabled by the user and interrupts can occur if they are enabled by the program.

Examples:

```
1500G
```

Begins program execution at location 1500. A newline is output to position the cursor on the next line when program execution begins. This command sets START to 1500 to subsequent 'G' commands without an argument will begin execution at location 1500.

nS Call the subroutine at location **n**. If not given, **n** defaults to START, else START is set to **n**. If enabled by the user the breakpoints will be active. The program can also be interrupted if it has enabled interrupts.

Examples:

```
1321S
RETURN   A:35  F:A5  X:46  S:FE  P:1322
```

The subroutine beginning at location 1321 is executed. On return from the subroutine the registers are output. START is set to 1321, so any subsequent 'S' command without an argument will execute this same subroutine again.

nP Proceed with execution at location **n**. If not given, **n** defaults to LOC if there is an open program memory location and the user PC if not. The breakponts and interrupts will be active if enabled by the user and program respectively. This is identical to the 'G' command except for the default argument value.

Examples:

```
P
TRAP    A:0A  F:03  X:0F  S:AB  P:1FE3
```

The program is executed beginning at location 1400. A newline is output to position the cursor on the next line. In this case the program execution began at location 1400 because there was no argument to the 'P' command, no open memory locations, and the program counter contained 1400.

nT Trace at location **n**. Executes the instruction at location **n**. If not input, **n** defaults to LOC if there is an open program memory location and to the user PC otherwise. The CPU will execute only one

instruction and then return control to the user with the current PC opened. LOC is set to the current PC in the program memory space. Breakpoints and interrupts are always disabled while tracing.
Examples:

```
1000T
TRACE   A:0A  F:06  X:10  S:27  P:1001 / FFE9   T
TRACE   A:0A  F:06  X:10  S:25  P:1FE9 / 1C00   T
TRACE   A:0A  F:06  X:10  S:27  P:1002 / 6015
```

First the instruction at location 1000 is executed and the registers displayed. Next the instruction at 1001 is executed (a CALL to location 1FE9) and the new register values displayed. Finally the instruction at location 1FE9 (an RTS instruction) is executed and the register values displayed.

nX Top level trace at location **n**. If not given, **n** defaults to LOC if there is an open program memory location and to the user PC otherwise. If the instruction at **n** is not a subroutine call this command is exactly like trace, executing only one instruction, with breakpoints and interrupts disabled. If, however, the instruction at **n** is a subroutine call then this command acts like S and calls the routine. In this case, breakpoints are active (if enabled through the '$E' command) and so might be interrupts (if enabled by the program). After the instruction or subroutine returns LOC is set to the current PC in the program memory space.
Examples:

```
1A00X
TRACE   A:0A  F:02  X:10  S:27  P:1A01 / E500   X
RETURN  A:6E  F:03  X:10  S:27  P:1A02 / 6116   X
TRACE   A:84  F:06  X:10  S:27  P:1A03 / 8112
```

First, the instruction at location 1A00 is not a subroutine call so it is single-stepped and after it is executed the registers are displayed. Next the instruction at location 1A01 is a subroutine call so the subroutine is executed and the registers are displayed on return. Finally, the instruction at location 1A02 is not a subroutine call so, again, it is traced and the registers are displayed after executing the instruction.

Breakpoint Commands

The breakpoint commands are used to set or reset specific breakpoints, enable or disable all set breakpoints, or clear or display all set breakpoints. The "$" is the actual character "$" or the key <ESC>. For more information see the section on breakpoints.

$B Display all set breakpoints and the enabled/disabled status of the breakpoints.
Examples:

```
$B
Breakpoints (enabled):  1200  1237  1274
```

The breakpoints are displayed along with the enable/disable status of the breakpoints.

n$C Clear one or all breakpoints. If **n** (word) is given the breakpoint at location **n** is removed. If there is no breakpoint at location **n** it is an error. If **n** is not given all of the breakpoints are cleared (removed). In this case if there are no breakpoints set, it is an error.
Examples:

```
1200$C
$C
```

First the breakpoint at location 1200 is reset. The next command removes all of the breakpoints. Note that nothing is output by the simulator other than a newline.

$D Disable all set breakpoints.
Examples:

```
$D
```
The set breakpoints are disabled. Nothing is output by the simulator except a newline to position the cursor on the next line.

$E Enable all set breakpoints.
Examples:
```
$E
```
The set breakpoints are enabled. Nothing is output by the simulator except a newline.

n$S Set a breakpoint at location **n**. If **n** is not specified it is an error.
Examples:
```
1200$S
1250$S
```
A breakpoint is set at location 1200. Next a breakpoint is set at location 1250. Note that nothing is output by the simulator in response to this command except a newline to position the cursor on the next line.

Miscellaneous Commands

I The simulator displays an identifying message on the screen. The message contains the simulator processor type and the version number.
Examples:
```
I
Caltech10 Simulator v1.0.0
```
The simulator ID message is output.

Q Exits the simulator.
Examples:
```
Q
"system prompt"
```
The simulator is exited and control returned to the system. The system will then output its standard system prompt (for example '>' or '$').

L Load an object file. The simulator will prompt for the name of the file. The simulator expects object files to be in the format specified in Appendix A. If the file doesn't exist, is in the wrong format, or has a data error in it, an error message will be output.
Examples:
```
L
filename: GCD.COD
```
The file gcd.cod is loaded into memory. It is assumed that gcd.cod is the proper format.

U Upload a memory file. The simulator will prompt for the name of the file. The simulator will then output all of the currently selected memory (data memory, I/O space, or program memory) to the file in ASCII hex with eight (for program memory) or sixteen (for data memory and I/O space) locations per line. If there us an error writing to the file, an error message will be output.
Examples:
```
U
filename: DUMP.OUT
```
The file dump.out is created and filled with a full dump of the currently selected memory space.

!D Change all future memory accesses to access data memory. Note that this means all addresses are eight (8) bits (2 hex digits) and the data is also eight (8) bits (2 hex digits).
Examples:

```
!D
```

After this command all future memory commands will refer to data memory. Note that nothing is output by the simulator in response to this command except a newline.

!I Change all future memory accesses to access I/O space. Note that this means all addresses are eight (8) bits (2 hex digits) and the data is also eight (8) bits (2 hex digits).
Examples:

```
!I
```

After this command all future memory commands will refer to I/O space. Note that nothing is output by the simulator in response to this command except a newline.

!P Change all future memory accesses to access program memory. Note that this means all addresses are thirteen (13) bits (4 hex digits) and the data is sixteen (16) bits (4 hex digits).
Examples:

```
!P
```

After this command all future memory commands will refer to program memory. Note that nothing is output by the simulator in response to this command except a newline.

H Output the help message, a brief summary of the simulator commands.
Examples:

```
H
```

Outputs the help message.

Breakpoints

Breakpoints allow the suspension of a program just before executing a specific instruction. This is very useful in a variety of circumstances. It is often enlightening to see if the program gets to a certain point, or examine the registers at the termination of a loop of subroutine.

To suspend execution of a program at a specific instruction set a breakpoint at that instruction and enable the breakpoints. This breakpoint will return control to the user before executing the instruction. From there the proceed ('P') command may be used to continue the program, or the trace ('T') command could be used to step the program, or some memory locations could be examined, or ...

Breakpoints may be disabled and enabled with the '$D' and '$E' commands respectively. The effect is to disable or enable all of the set breakpoints. A breakpoint will only stop program execution if it is both set and breakpoints are enabled. This feature makes it easy to set strategic breakpoints, turn them off to allow the program to run to check if the bugs are eliminated, and then re-enable them if there are still bugs without necessitating again looking up the locations at which to set the breakpoints.

The simulator allows eight (8) breakpoints to be set. Initially, all breakpoints will be cleared and disabled. These breakpoints may be individually set and reset with the '$S' and '$C' commands respectively.

Note that breakpoints have no effect when the execution is being traced with the 'T' or 'X' commands.

Software Debugging

The simulator is very rich in commands for debugging software. With the execution, register, and memory examine/deposit commands one can very thoroughly test any piece of code.

The most obvious command to use when debugging software is the trace ('T') command. This executes a program one step at a time, displaying the registers after each step. To start tracing the 'T' command is used, usually with an argument (the starting trace address). To continue tracing the 'T' command is used without an argument (assuming memory isn't opened and important registers aren't changed) and the program will execute one step at a time.

A somewhat more useful command when some of the program has already been debugged is the top-level trace command ('X'). As described before, if the instruction to be executed is a subroutine call that routine will be called. Otherwise the single instruction will be executed (as for trace). Any called subroutine will be executed with breakpoints active if the user has enabled them and interrupts possible if the program has enabled them. By using this command wasting time tracing through routines which have already been debugged can be avoided.

Another useful command is the subroutine call command ('S'). This command is like go ('G') execept it calls the routine at the address given to it rather than jumping to it. When the routine ends and executes a return from subroutine instruction, control is passed to the user and the registers are displayed. This is most handy when checking routines that do something different on every call (such as some I/O drivers and counting routines).

When program development is well along the way, breakpoints are often the most useful tool for debugging. When a breakpoint is reached the program is stopped, the register contents displayed, and control returned to the user. The instruction at the breakpoint has not been executed. The program can then be continued with the proceed ('P') command, or stepped with the trace ('T') command, or any other command may be used. This can be a great help in tracking down an error in a program which loops may times, or when the approximate location of the error is known and is deep within the program. The procedure is to set a breakpoint where the error is thought to be and check memory and the registers, as appropriate, for incorrect values. This procedure is continued (proceed to next breakpoint, trace, etc.) until the error has been found. Note that the typical way to start the program running again after tracing is with the proceed ('P') command.

Appendix A
File Formats

The simulator has two types of files associated with it. Load files contain memory values to be loaded into program memory using the 'L' command. Upload files are output by the simulator and contain the complete contents of the currently selected memory space and are generated by the 'U' command.

Load File Format

The load file is assumed to contain hexadecimal characters giving the values to load into program memory. Each line of memory data consists of a program memory address (typically 4 hex digits) and program memory data (16 bits or 4 hex digits) separated by white space. All other white space in the file is ignored, as is the remainder of a line after a semicolon (';').
Example:

```
; this line is just a comment
0ABA  89FE    ;LDI  $FE
```

Upload File Format

The upload file is written by the simulator and is a complete dump of whichever memory space is currently selected (256 bytes for data memory and I/O space and 8192 words for program memory). The data is written in ASCII hexadecimal characters. If data memory or I/O space is being dumped each line of the upload file will contain sixteen (16) two digit hex numbers (bytes). If program memory is being dumped each line of the file will contain eight (8) four digit hex numbers (words).
Example:

```
0E00 B735 1B90 0E00 B771 137F A010 11A7
0E00 B211 10A1 0E00 B241 12B0 0E00 1122
```

Appendix B
Command Summary

Memory Commands

/	examine memory location
<CR>	deposit argument (if given) to memory and close memory
+	deposit argument (if given) to memory and examine next location
^	deposit argument (if given) to memory and examine previous location
;	deposit argument (if given) to memory and examine same location
.	deposit argument (or default) to memory and examine next location
M	examine next 8 or 16 memory locations
N	examine previous 8 or 16 memory locations
Z	deposit argument (or default) in next 8 or 16 memory locations

Register Commands

R	display the registers
nRr	deposit n in register r

Execution and Breakpoint Commands

G	begin program execution
P	proceed from present location
S	execute subroutine
T	trace (or single-step)
X	top-level trace
$B	display breakpoints
$C	clear breakpoints
$D	disable breakpoints
$E	enable breakpoints
$S	set a breakpoint

Miscellaneous Commands

I	display ID message
Q	exit simulator
L	load an object file
U	upload (dump) memory to file
!D	set memory space to Data
!I	set memory space to I/O
!P	set memory space to Program
H	display this message