Many facets of the system are configurable. If you have a file called `system.setup` in your current working directory, configuration parameters are read from it at startup. If you have no such file, `/usr/local/bin/system.setup` is used instead. That file is readable by everyone, so look inside it for examples. It contains five commands, the first two define the ranges of addresses for physical memory that will exist, the second gives the filename and size (in blocks) for the emulated disc drive, and the last two give the initial values for the PC and SP registers.

As you know, you start the emulator with the command "`computer`". If you wish any `.exe` files to be automatically loaded into memory at startup, there are two options. Either add a line like this "`load file.exe at 0x1400`" to `system.setup`, or include the file name on the command line like this: "`computer file`". In the latter case, `file.exe` will be loaded starting at memory location `0x400`.

If you wish the system to automatically start running, without any interaction or single stepping, just add "`-r`" to the command line, like this "`computer file -r`", and execution will start at PC=0x400. With the "`-r`" option, the emulator exits when the program it is running halts, thus saving you unnecessary interaction, but also depriving you of the opportunity to examine the contents of memory or registers.

If you wish the emulator to continue to run after the program halts, just use "`-rs`" instead of "`-r`".

The option "`-q`" means Be Quiet, don't report all the `system.setup` settings at startup, just start running.

To redirect input away from the keyboard, so that the `INCH` and `PERI` instructions read from a pre-prepared file instead, include "`-in=`*`filename`*" on the command line.

When the system starts up, some useful information is left in particular memory locations:

memory[0x100] = amount of memory actually installed and available continuously from location 0. In other words, if [0x100] is 12345, then all physical memory locations from 0 to 12344 exist.

memory[0x101] = first memory address that nothing (from an .exe file) has been loaded into - the beginning of free space.

memory[0x102] = the number of fake disc drives (as defined in system.setup) available for use.

If you do not use the "`-r`" command line option, the emulator starts in debug mode. It gives you the prompt "`>`" and waits for commands.

The commands are...

`ENTER`. Just press Enter and the system goes into single step mode. It shows the values of all the registers, and the next instruction to be executed, and lets you take control. While n single step mode, if you just press enter, it will execute just one instruction then wait for another command. If you type the command "`x`" it will leave single step mode and wait for another command. All other commands are equally available in or out of single step mode.

`STEP` or `S` followed by a number:

Execute that number of instructions, showing all debugging information after each. A BREAK instruction stops execution.

`RUN` or `R`:

Continue running the program without debugging. Just run normally as though the "-r" command line option had been used. In other words, behave like a real computer running normally.

`RUN` or `R` followed by a number:

The same as just "run" above, but after the given number of instructions have been executed, the system returns to single step mode and waits for another command.

`?`

Shows the contents of a register or some memory locations. If the "?" is followed by the name of a register or a special register, it simply shows the value of that register, e.g. "`?R3`" or "`?INTVEC`". Other options are best illustrated by example:

`?123`
: Show memory location 123

`?123:20`
: Show 20 memory locations starting from location 123

`?0x123`
: Show memory location 291 (the address is in hexadecimal)

`?R1+5`
: Show the contents of [R1+5]

`?FP-2`
: Show the second local variable (if you are using the stack correctly)

`?FP-2-8`
: Show all memory locations from [FP-2] to [FP-8]

`?R5-3+4`
: Show all memory locations from [R5-3] to [R5+4]

`?FP-15:10`
: Show all memory locations from [FP-15] to [FP-5]

`?FP`
: Show the value of the FP register

`?FP+0`
: Show the contents of memory location [FP+0]

of course, R1, FP, R5 etc can be replaced by any register name

`p?`

Exactly the same as "?", except that virtual memory is ignored. "`p? 123`" shows the contents of physical memory location 0x123 regardless of whether virtual memory is turned on or not.

`set`

Change the value of a register, special register, or memory location. Examples:

```
set r1=58
set r1=0x58
set intvec=0x1800
set 1800=12
```

the latter example stores the number 12 (decimal) in location 0x1800.

`pset`

Exactly the same as set, except that virtual memory is ignored. "`pset 100=3`" stores 3 in physical memory location 0x100.

`vm addr`

Shows every stage in the conversion of virtual address "addr" into a physical address. Used to check that page tables are correctly set up.

`x`

If single stepping, stop single stepping and go back to normal command mode.
If not single stepping, exit completely.

`exit`, `q`, or `quit`

Exit completely.

Control-C is properly trapped. If a program is running, control-C instantly stops it and puts the system in single step debugging mode. If no program is running, control-C is harmless. BUT two consecutive control-C's will exit the system. This is just in case I screwed up the control-c trapping. We don't want any unstoppable programs.

While a program is running (not in single step mode, but properly running), everything typed at the keyboard is instantly captured, without waiting for enter to be pressed. If interrupts are enabled (i.e. the $IP flag "interrupt being processed" is off), the IV$KEYBD interrupt is caused.