Using the VICE monitor

I am sure all of you are familiar with VICE, the **Versatile Commodore Emulator**. Bundled in this package is the nifty utility, “x64”, which lets you emulate a C64 on your modern machine. Yeah! But it's actually capable of much more, especially if you're a developer. Read on!

Invoking the monitor

This works differently depending on what target platform you are running VICE on. In Windows, you press **Alt+M**. In Linux, the keyboard shortcut is instead, for some elusive reason unbeknownst to any but the VICE developers, **Alt+H**.

Loading/dumping memory from/to a local file

```
  l "<path>" 0 [<start address>]
  s "<path>" 0 [<start address>] [<end address>]
```

For those of you who code in TASS, this is extremely handy, and it's quite useful for us others as well. The key here is using '0' as the device number, which tells VICE it should use the host's filesystem instead of an emulated device. *NOTE: The end address is* inclusive*, meaning that if you input '1000' as the end address, everything from the start address up until **and including** the value at $1000 will be saved. Also, all input files are assumed to be in .PRG format, so the first two bytes are always skipped even if you specify an explicit load address.*

Reading the status display

If you type `registers` in the monitor, or if you enable the register display window (WinVICE), you can see something like this:

```
  ADDR AC XR YR SP 00 01 NV-BDIZC LIN CYC
.;ded5 00 00 0a f1 2f 37 00100010 000 001
```

I will skip the normal C64 monitor stuff and move directly on to the two last and most interesting information posts: `LIN` and `CYC`. What do they mean? Could it be...? Yes, you are correct in your guess - these show the raster line and cycle currently executing! Extremely handy when debugging that nasty sprite ridden raster code.

Breakpoints and stepping

These mechanisms are the core of any serious debugging job.

```
  break [<address> [if <cond_expr>]]
```

This is the mother of code. You all probably know how to work breakpoints, but in short, it halts program execution and starts the monitor if the PC reaches <address>. Typing `break` without any arguments lists the currently active breakpoints. This is all basic stuff, but that optional `if` clause makes it interesting. For example, say you want to break at PC=$1000, but only if `Y = 1`. This is done as follows:

```
  break 1000 if .Y == 1
```

Note the double equality signs to denote comparison and not assignment! Anyway, now that we know how to set breakpoints, we might want to single-step through our code as well. This is done using two commands:

```
  step [<count>]
  next [<count>]
```

Where the optional count argument specifies how many instructions to execute. The difference between `step` and `next` is that `next` doesn't follow any subroutine calls (JSR) you make. These instructions can be shortened down to `z` and `n` respectively.

Import labels into the monitor

(This section was written by Frantic)

In case you are able to output the assembled labels from your assembler, then you can make some script to convert the info into the following form (note that you have to add a dot before the label names) and add that to your makefile before calling VICE:

```
al 1000 .player_init
al 1003 .player_driver
al d400 .SID_V1_FREQ_LO
al d401 .SID_V1_FREQ_HI
al d402 .SID_V1_PW_LO
al d403 .SID_V1_PW_HI
[etc…]
```

The command “al” stands for “add label”. Then call VICE with the following command line option:

```
x64 -moncommands yourstuff.txt -8 yourprogram.d64
```

If you do this, VICE will execute all the al commands in “yourstuff.txt” and you will see the labels from your assembler source when disassembling in the VICE monitor. That can be handy sometimes. If you add several labels pointing to the same adress, VICE will only show one of them. You can also add breakpoints to “yourstuff.txt” in the following form:

```
break 1000
al 1000 .player_init
```

```
al 1003 .player_driver
```

In this case, this would mean that the VICE monitor would pop up each time you call the “player_init” routine. In fact, I think you can add any monitor commands to the file called with -moncommands (but I am not entirely sure—some of them might not work).

Further reading

The VICE Manual has some info about the monitor, and don't forget you can use the `help` command from within the monitor itself, which can often reveal more info than the VICE Manual. Good luck!