Souping-up Grant's Simple Z80 **Phil Green, Oct 2021, revised Mar 23**

Grant Searle's "Simple Z80" in its 32k format is a great computer, reminding me very much of my old Nascom-1 back in 1978. Its true 'Retro' and quick, easy and cheap to make using not-too-hard-to-find parts, and Jeff Tranter has kindly made his PCB available for it via easyeda.

As designed, its a BASIC-only machine which is brilliant for playing Star Trek and Sargon Chess, but I always like to have raw access with a machine-code monitor - such as the T2 in my old Nascom and NAS-SYS in later models.

The 8k BASIC has had a few redundant routines removed, such as cassette and screen handling, so its actually slightly smaller than its original 8k. In fact within the EEPROM, the interpreter finishes at address 1DB7 hex and 1DB8 to 1FFF are unused - nearly 600 ROM bytes free!

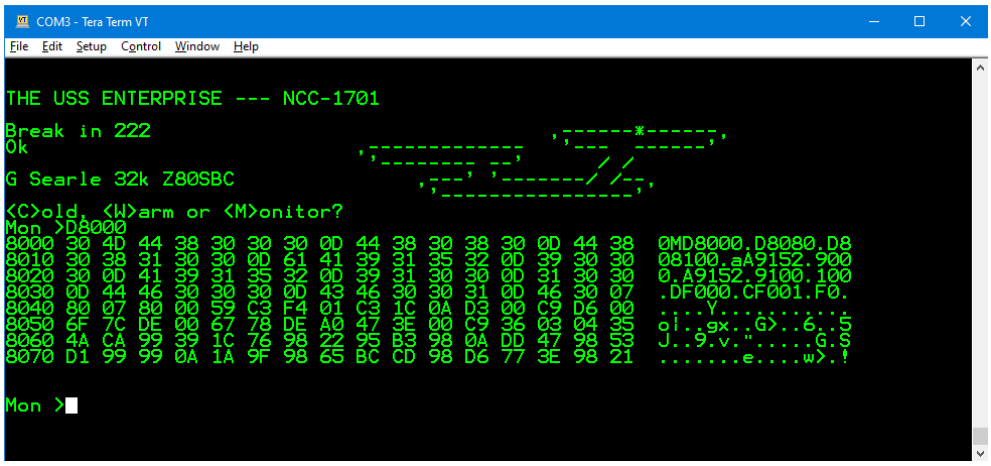
I decided that within the confines of the 8k EEPROM there would be room for a modest monitor if I kept it simple. On the other hand it had to have all the essential commands to be of any genuine benefit - in fact its all worked out very nicely although it is an 'only just' fit! The monitor code actually starts at 1DC0 hex, thats also its entry point.

Do please bear in mind that this is a half-a-kilobyte monitor, which conveniently fits under the ceiling of the standard 8k BASIC ROM. As such it cant be expected to compete with the full-8k monitors such as RomWBW or SCM.
It does however give you all the essentials for entry and examination of machine code, utilising ROM space that is otherwise wasted :-)

The modification is in 3 parts:

1. the INT32K.ASM file has the "Warm or Cold?" signon message extended to "Warm, Cold or Monitor?"
2. the file GS32KMON.ASM is my monitor, which lives at the top of ROM
3. the BASIC source has had two bytes changed so that the MONITOR command actually puts you into the monitor rather than causing a reset as it does in its published form..

For ease, all three files have been combined into one 8k hex image for the EEPROM, just blow it in the TL866 or whatever, plug it in & away you go.



Monitor Commands:

Selecting '**M**' from the signon message will place you in the monitor and you will see the "Mon >" prompt, ready to accept commands. There is no 'backspace' correction, all arguments are taken as the last four digits typed before pressing <return>. If you make a mistake, just continue typing with the four correct digits, ie 99579958 registers as 9958.

A is the hex arithmetic function. Given two arguments x & y it will display the sum x+y, the difference x-y, and the relative jump offset necessary to jump from address 'x' to address 'y". Note that I reversed the subtract parameters with respect to the Nas-Sys arithmetic command as y-x always seemed the wrong way around to me!
Syntax:

```
Mon >A
3456 <return>
3434 <return>
SUM  DIFF RJ
688A 0022 DC
Mon >
```

B is the 'Return to BASIC' command. It takes no arguments, just type B and the signon message will appear, just as though you'd pressed a hardware reset.

C is the copy function. Its intelligent so it doesnt matter if the source and destination areas overlap. The syntax is:

```
Mon >C
9000 <return> "copy from address 9000"
9800 <return> "...to address 9800"
80 <return> "...for 80 hex bytes, ie 9000-9080 will be copied to 9800-9880"
```

D is the display command. It takes one argument which is the start address, and displays in hex and ASCII eight lines of sixteen bytes of code. The syntax is:

```
Mon >D9000 <return>
```

G is the "Go to" command which takes one argument as the start address and will run a user program from that address. Syntax is:

```
Mon >G9000 <return> "Go to 9000 hex and start executing code there"
```

If you unintentionally enter “G” the goto can be aborted with a full-stop “.”
Programs can return to the monitor using either a RET instruction or a jump to 1DC0H.

I is the Intel-Hexloader command. It doesnt take an argument as the load address is specified within the hex file. Once 'I ' has been pressed, begin sending the file (at fast baud rates you may need a line-delay of maybe 100ms or so). For every line received, a dot is displayed to show progress. Unlike many implementations, this is a fully error-checked loader - non-hex characters and bad checksums will be identified with "~Hex!" and "Csum!" messages. You can see loading progress on the CH340G LED.

M is the modify memory function. It takes one argument as a start address, and displays the current contents of that memory address. You may press <return> or <space> to step through without making changes, or type a new hex byte of two characters followed b <return>. The command is terminated with a '.' full-stop which will bring you back to the "Mon >" prompt. Syntax is:

```
Mon >M9000 <return>
9000 54 <pressing return leaves contents as 54>
9001 76 77 <keying '77' & return will change the contents of location 9001 to 76
9002 45 . <a full-stop finishes the modify session>
Mon >
```

All arguments are taken as the last four digits typed before pressing <return>. If you make a mistake, just continue typing with the four correct digits. For example, 93457281 is actually taken as "7281". If you are entering a byte (as when using modify) the same applies - if you typed 32 when you actually meant 23, just continue with the '23' because '3223' is taken as '23' in this context.

Single arguments as in Display and Modify are entered against the command letter, on the same line.
For clarity, multiple-argument commands like Copy and Arithmetic place each argument on a separate line.

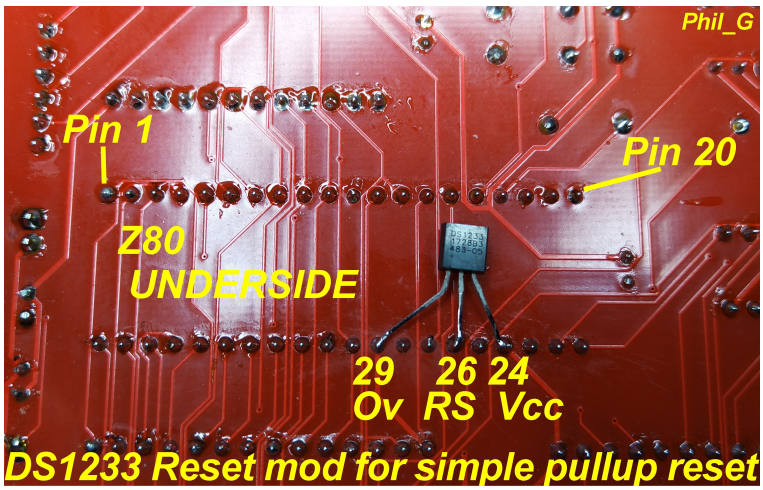
Hardware mod - a proper 'reset'

The circuit as published uses a simple reset with a pull-up resistor and a push-button to ground. There is no power-on reset and the button has to be pushed. This is easily upgraded using a DS1233 'reset chip' which looks like a small signal transistor but is actually a voltage sensor and time delay. As well as holding the reset active until the power rails are settled, it also debounces the manual button. It costs about a pound (RS) and is very simple to add underneath the Z80 on Jeff's PCB layout:

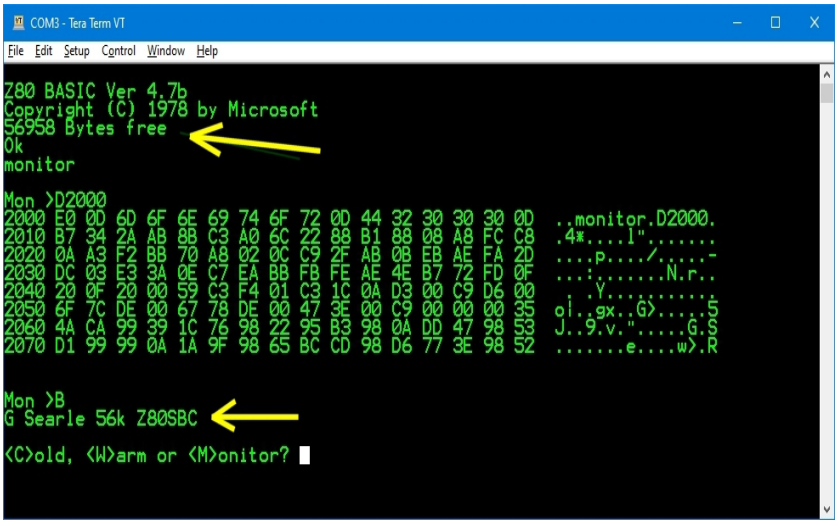
Improved reset with a DS1233:

z80 pin 29 neg
z80 pin 24, 25 or 17 (all positive 5v on Grants schematic)
z80 pin 26 reset

```
 1233 pins
3  2  1
|  |  |
24 26 29    Z80 pins
+  R  -
```



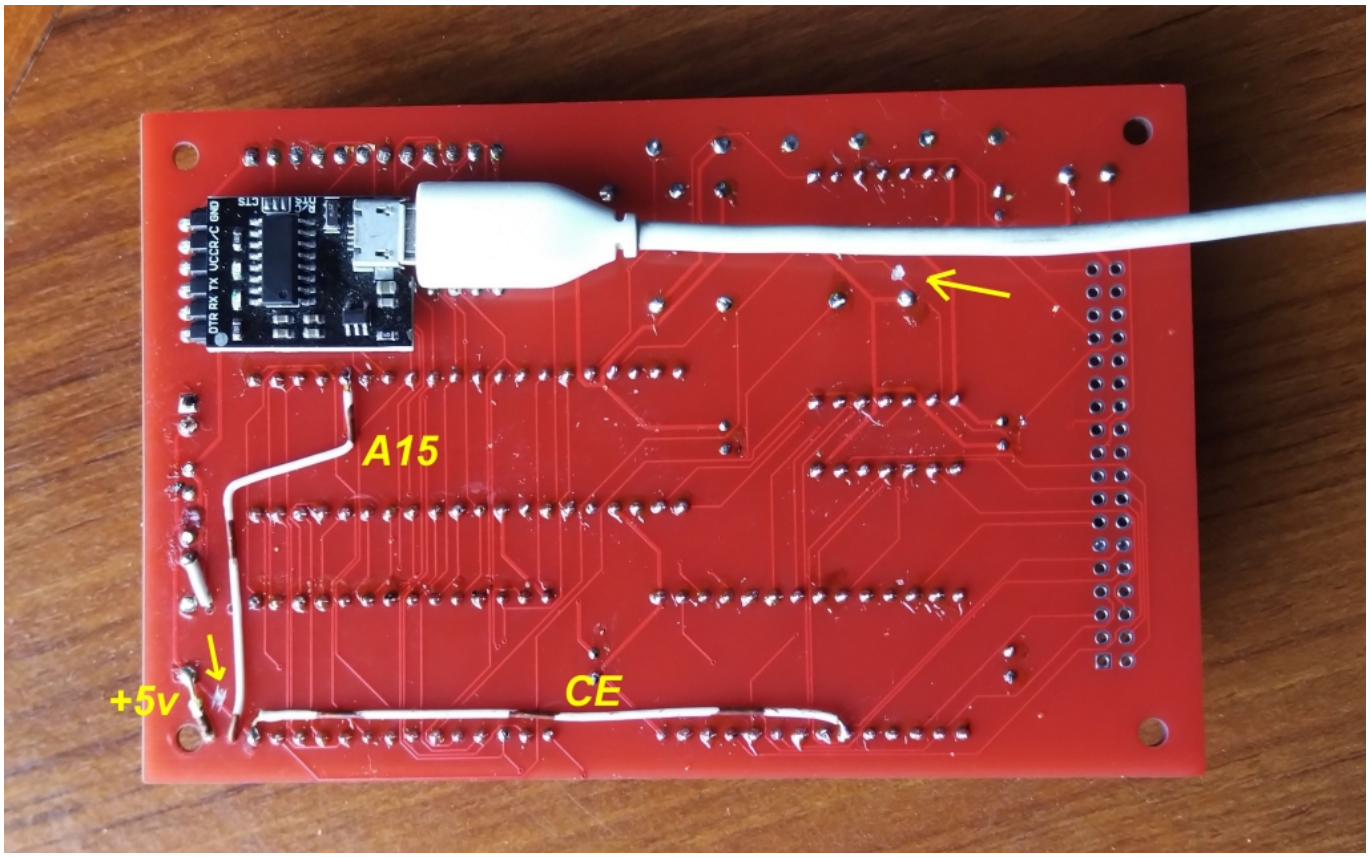
Hardware mod - building the 56k version using Jeff's 32k PCB:

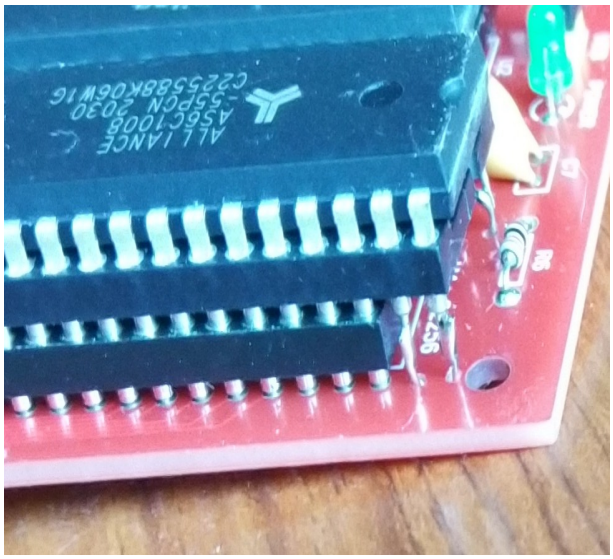


I initially used an **AS6C1008-55** 128k x 8 bytewide static RAM from RS, but later chose a **DS1245Y** 128k NVRAM.

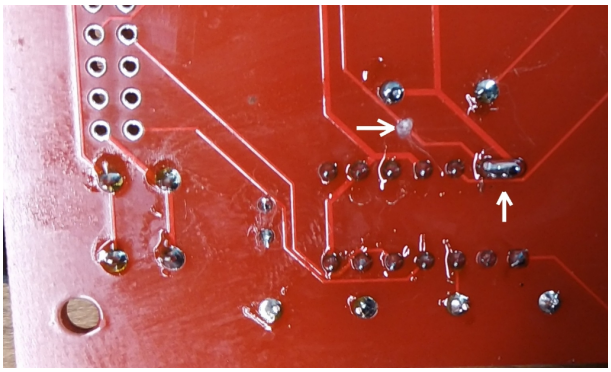
Drill extra 2 holes for the 128k 32-pin RAM using 0.1" stripboard as a guide
Cut 62256 pin 28 to R6 45-degree track
connect 128k pin 32 to R6 (+5)
connect 128k pin 31 to A15 (Z80 pin 5)
ground 128k pin 1 & 2 (I used C7 neg) [you could manually switch this for two banks of RAM]
cut track to isolate U5 pin 13
connect U5 pin 13 to pin U5 pin 14
connect 128k pin 30 to U6 pin 20
optionally do the DS1233 reset mod as above, cheap, simple & well worth doing

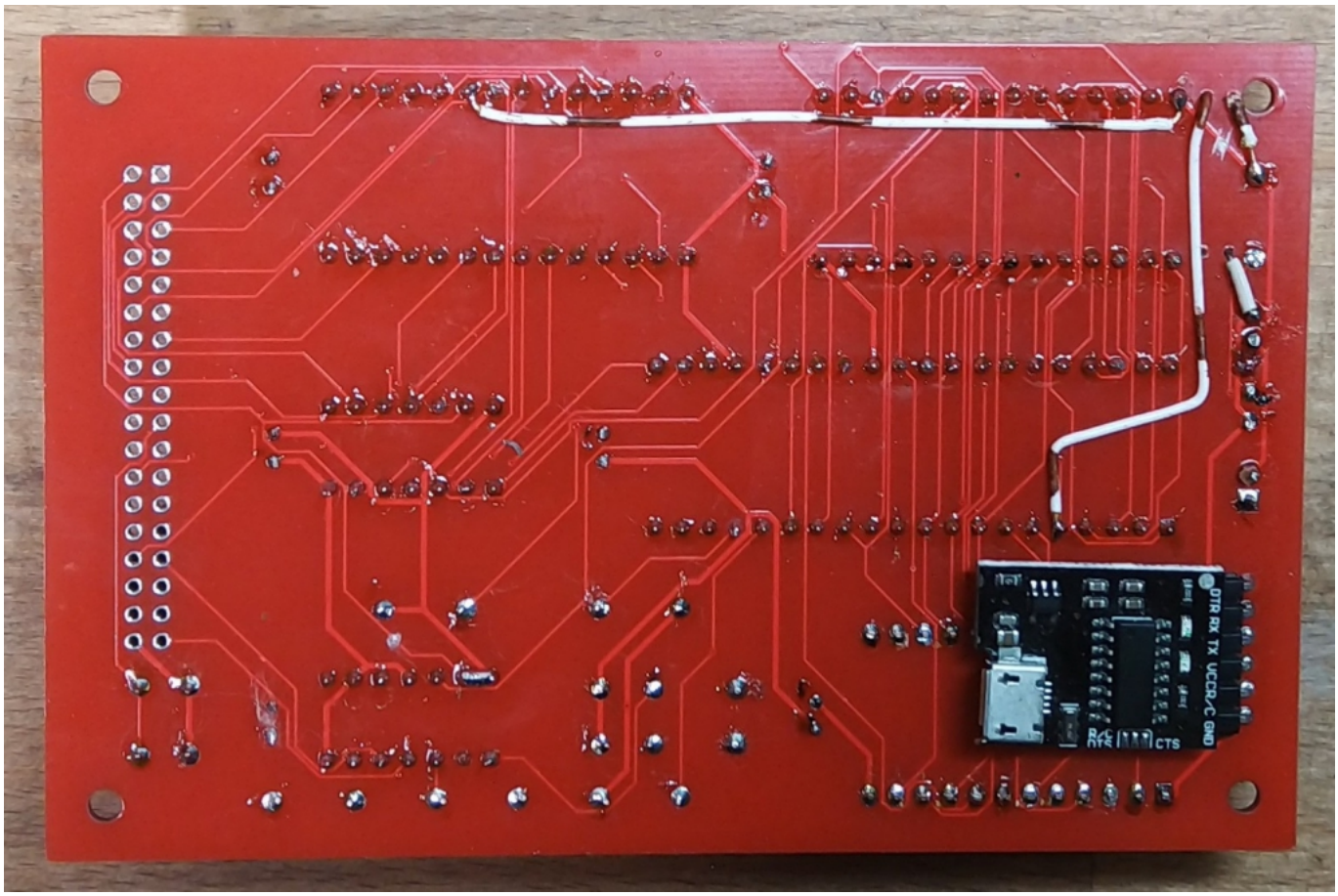
Refer to the 32k document but substitute the 56k version of the modified ROM (with the monitor) from philg.uk





Pins 1 & 2 are grounded. Pins 31 and 32 are extended through the board to A15 and Vcc respectively.



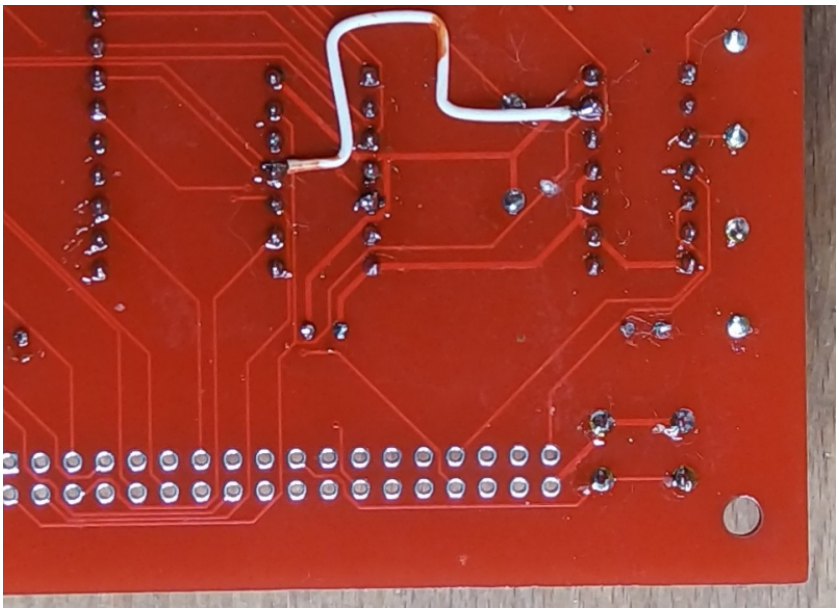


Please note that the 56k and 32k ROM images are different and must match the hardware. The differences are trivial, just memory addresses. Use the 56k mod in conjunction with the 32k doc above but note that RAM starts at 0x2000 on the 56k version so bear that in mind where I refer to any absolute RAM addresses in the document.
Refer to the 32k zip for the manual but use the 56k ROM image: gs56kz80rom.hex

Hardware mod - a 56k version using a 128k x 8 NVRAM:

128k x 8 NVRAMs are available but expensive – the DS1245Y from RS is 35 quid and at the time of writing they have 81 in stock. Mouser UK are a tad more expensive and have 1700+ in stock.
Will "silicon-shortage" prices ever return to normal? :-)
Actually, when I think about how much my Nascom 32k RAM-B and Buffer Board cost in 1978, maybe 35 quid is a bargain!

To install a DS1245Y NVRAM another small mod is necessary as it doenst have a 'true' chip-select on pin 30.
The isolated pin 13 of U5 is connected to U7 pin 11 (instead of U5/14 as it was in the pics above):



Since the NVRAM is 128k, we can manually switch A16 (pull-up & a switch to ground) to give two non-volatile pages!

(The shiny things that look like solder-splashes are just flux reflecting on the camera)

How BASIC programs are stored in memory:

The relocated Nascom BASIC source starts at address 813E as follows, this can be verified with the monitor:

```
813E 00 eol?
813F 66 \
8140 81 8166 is next line location
8141 32 \
8142 00 0032 is line number 50
8143 8E REM token  (for example)
8144 2A ***********etc
8165 00 eol
8166 8D
8167 81 818D is next line location
8168 37 \
8169 00 0037 is line number 55

The program ends with the 00 eol then 00 00 as the 'next line address'
```

MONITOR command within BASIC:

This is a simple 3-byte jump, ie hex C3 followed by what should be the address of the monitor. The command however as published is disabled, with a jump to 0000,
C3 00 00 which will reset the computer when 'MONITOR' is used:

```
1D99 C3
1D9A 00 <--- change to actual monitor address (little endian, ie LSB first then MSB)
1D9B 00
```

Now we actually have a monitor I've changed this to:

```
1D99 C3
1D9A C0 <--- change to actual monitor address (little endian, ie LSB first then MSB)
1D9B 1D
```

These changes are implemented in the supplied ROM update.

Calling a machine-code routine from BASIC, and passing parameters

First the USR() vector must be provided, the jump is at hex address 8048:

```
8048 C3  (jump instruction)
8049 yy  (lsb of routine address)    (little-endian!)
804A xx  (msb of routine address)
```

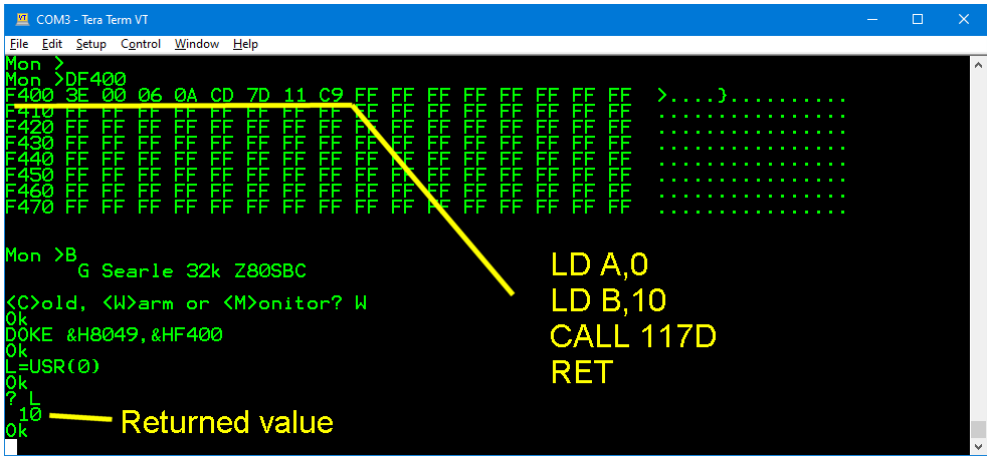
To call a USR m/c routine at (say) hex FD00, use `DOKE &H8049,&HFD00` followed by `X=USR(0)`
DOKE of course accounts for it being little-endian so its arguments are typed logically as above.
Of course the M/C routine to be called must first be present in memory!!!

The following is 'translated' from the Nascom ROM BASIC document, appendix D, which should be referred to for a really detailed explanation of calling machine-code routines from BASIC. Ignore any absolute addresses given in the Nascom document though – our BASIC has been relocated!

"When USR is called, the stack pointer is set up for 8 levels (16 bytes) of stack storage. Memory locations and registers can be changed by the M/C routine. USR is called with a single argument (the value in the brackets) , which can be retrieved by the M/C routine by calling address hex 0A07 (CD 07 0A) which puts the argument into the DE register pair as a signed 16-bit integer."

In the opposite direction, the M/C routine can return a value to BASIC by loading a signed 16-bit integer value into the register pair A, B and calling hex address 117D (CD 7D 11). Then for example "L = USR(0) " will put the A,B value into BASIC variable 'L'. Note that A is the MSB and B the LSB:

Here is an example, a short machine-code routine is entered into free memory using the monitor 'M' command.
In BASIC, the address of this routine is DOKE'd into the USR vector at &H8049.
The machine-code routine is a simple demonstration where the value decimal 10 (signed 16-bit hex 000A) is loaded into the A,B register pair, the transfer subroutine is called, and the result is returned in the BASIC variable 'L' :



Links:

The updated and complete 8k EEPROM file (in Intel hex format) is available on http://philg.uk
With the addition of a monitor or 'bug' as they were known, the 32k "Simple Z80" becomes a really handy little hobby computer with much more of a "Nascommy" feel, and it presents the opportunity to try a few machine-code routines, hand-coded or cross-assembled elsewhere. I've had great fun with mine, thank you Grant!

Grant Searle's site: http://searle.wales and specifically the 32k RAM version:
http://searle.x10host.com/z80/SimpleZ80_32K.html

Jeff Tranter's PCB gerbers: https://easyeda.com/tranter/z80-single-board-computer

My stash where you can find the updated ROM files & a short document: http://philg.uk
(in the 'Retro' section)

The video: https://www.youtube.com/watch?v=mvZ_L-rdgBQ

For a really nice Z80 cross-assembler running in a PC 'CMD' window, try **AZ80.**
https://www.retrotechnology.com/restore/az80.html (just under the A18 History header)
This monitor and all the mods presented here were done with AZ80.

Cheers
Phil_G
philg@talk21.com

Revised Mar 23: **G**oto cmd - monitor return address pushed onto stack & added **G**oto abort '.'