Modern TEC Software Development

A twenty first century approach to writing TEC software

By Craig Hart

The Problem

Writing serious code for the TEC computer has always been difficult.

Key punching hex bytes, looking up opcodes and writing them out with pen and paper, calculating jump offsets, managing memory etc. is tedious at best, and really prevents a program of any real substance being created, especially when it comes to editing code.

As a result, the TEC is often viewed upon as a 'toy' and interest is lost quickly, since it becomes 'too hard' to work on software.

A better approach

The obvious solution is to develop code for the TEC on your PC – but how?

A number of tools are required:

- An Editor to write your code in
- An Assembler to compile the code into Z80 binary
- A means to transfer the code from PC to the TEC
- (optionally) A debugger and/or emulator to fault find or if you don't have a real TEC handy
- Somewhere to save and share your work with others

Once you have your PC development toolkit put together, the process becomes quite rapid and straight forward.

A Modern Workflow

Firstly, one writes code in Z80 assembly language using regular Z80 instructions. However, we also include hints to the assembler to let it know what we need done.

Consider the following simple block of Z80 code written for the TASM assembler:

```
         .ORG 0900h

COUNT    .EQU 08

         LD A, 7fh
         LD B, COUNT

LOOP:    DEC A
         OUT (C),A
         DJNZ LOOP

         .END
```

Let us look at the above code in more detail.

```
         .ORG 0900h
```

This is called an *assembler directive* and its purpose is to give the assembler a hint about where in memory the code is to be placed. In this case, 0900h.

```
COUNT    .EQU 08
```

This line assigns the *constant value* 08 to the word COUNT. If we use the word COUNT anywhere else in our code, the assembler will substitute the value 08 instead.

```
LOOP:    DEC A
```

This line uses a *label* (labels are suffixed by a colon). Labels are used as a marker or reference point that the code needs to return to or reference later.

Labels usually mark the target of a JMP or JR; the assembler will replace DJNZ LOOP instruction with DJNZ <memory address of the instruction labelled LOOP>.

```
         .END
```

Finally, this directive lets the assembler know it has reached the end of our code

Labels and constants are very powerful – for example if we alter the code such that the loop has to run 0F times, we simply change the EQU 08 to EQU 0F – now wherever COUNT is used, ALL references change from 08 to 0F. There is no need to go through and find and alter the relevant lines of code, which could be tedious and error-prone in complex code.

Same with the LOOP label – if we alter the loop to add or remove instructions, the assembler will automatically recalculate the new address offset for the DJNZ instruction for us.

As you can see, as assembler is very powerful and does a lot more than just 'convert opcodes to their byte values'. We have just scratched the surface here and most assemblers offer dozens of features to make writing code more efficient and less buggy.

Various assemblers tend to use slightly different formats for defining directives, constants, labels, and also macros, variables and much more in some cases. All assemblers are broadly similar and usually code can be easily adjusted to work on any desired assembler.

Assemblers will also highlight any faults in your code e.g., typo's, incorrect instructions, JR's past 128 bytes, wrong size operands etc. meaning you can't write bad code – no output is produced until your code is syntactically correct.

The assembler produces an output file which can be in various formats (binary, Intel HEX, source code listing, etc).

The source listing format shows the code, opcodes, resulting memory addresses etc. giving a quick view of what the assembler has produced.

Intel HEX is a standardized text format that includes the target memory address, binary data, checksum etc. to ensure reliable transfer.

The assembled code is transferred to the actual hardware via a serial link (this can be as easy as 2 or 3 mouse clicks) and then tested on the real machine.

A whole write, assemble, transfer, test cycle can be as short as a minute or two. Since your source code is in easy to manage text file, edits are a breeze.

Suddenly complex software development becomes far easier, once you are freed from the tedious chores of looking up opcodes, keypunching hex values, doing jump relative math, etc.

Editor

Text editors are numerous these days; Windows comes with Notepad built in, which works fine.

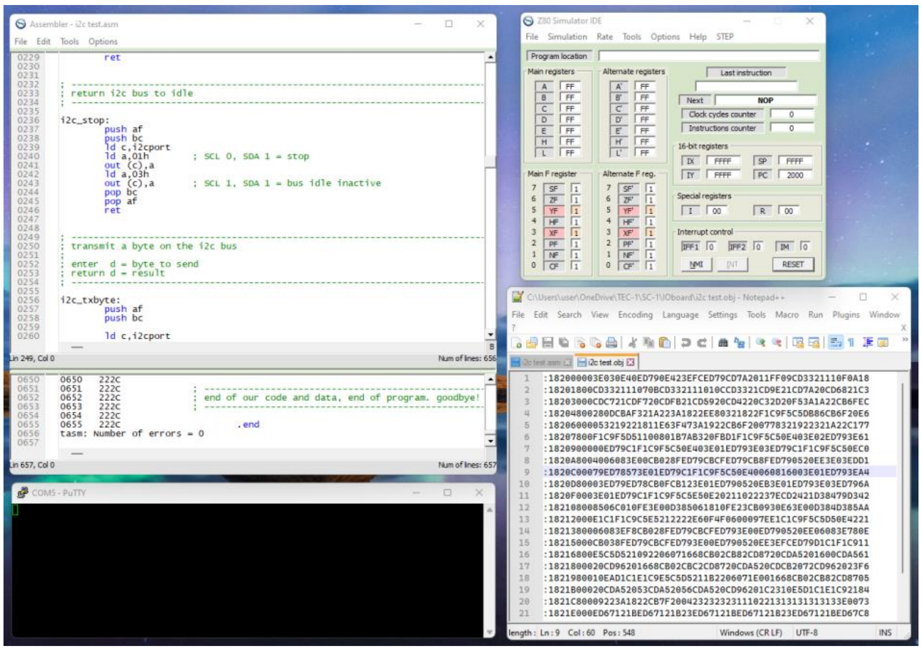
I personally prefer Notepad++, and there is also Microsoft's VS Code (Visual Studio), to name a few.

In short, any program that saves files as plain text will do, however the third-party tools can assist with formatting for readability, syntax checking for typos, code highlighting and colour coding for readability, etc.

Assembler

Once Again there are numerous assemblers out there, both free and paid.

I have previously used Oshonsoft's integrated Z80 development environment, available from



```
        ret


; ------------------------------------------------------------
; return i2c bus to idle
; ------------------------------------------------------------

i2c_stop:
        push af
        push bc
        ld c,i2cport
        ld a,01h        ; SCL 0, SDA 1 = stop
        out (c),a
        ld a,03h
        out (c),a       ; SCL 1, SDA 1 = bus idle inactive
        pop bc
        pop af
        ret


; ------------------------------------------------------------
; transmit a byte on the i2c bus
;
; enter  d = byte to send
; return d = result
; ------------------------------------------------------------

i2c_txbyte:
        push af
        push bc

        ld c,i2cport
```

Lin 249, Col 0 Num of lines: 656

```
0650   222C
0651   222C               ; ------------------------------------------
0652   222C               ; end of our code and data, end of program. goodbye!
0653   222C               ; ------------------------------------------
0654   222C
0655   222C                       .end
tasm: Number of errors = 0
```

Lin 657, Col 0 Num of lines: 657

```
:182000003E030E40ED790E423EFCED79CD7A2011FF09CD3321110F0A18
:18201800CD332111070BCD332111010CCD3321CD9E21CD7A20CD6821C3
:18203000CDC721CDF720CDFB21CD5920CD4220C32D20F53A1A22CB6FEC
:18204800280DCBAF321A223A1822EE80321822F1C9F5C5DB86CB6F20E6
:18206000053219221811E63F473A1922CB6F200778321922321A22C177
:18207800F1C9F5D51100801B7AB320FBD1F1C9F5C50E403E02ED793E61
:1820900000ED79C1F1C9F5C50E403E01ED793E03ED79C1F1C9F5C50EC0
:1820A8004006083E00CB028FED79CBCFED79CB8FED790520EE3E03EDD1
:1820C00079ED78573E01ED79C1F1C9F5C50E40060816003E01ED793EA4
:1820D80003ED79ED78CB0FCB123E01ED790520EB3E01ED793E03ED796A
:1820F0003E01ED79C1F1C9F5C5E50E20211022237ECD2421D38479D342
:182108008506C010FE3E00D385061810FE23CB0930E63E00D384D385AA
:18212000E1C1F1C9C5E5212222E60F4F0600097EE1C1C9F5C5D50E4221
:182138006083EF8CB028FED79CBCFED793E00ED790520EE06083E780E
:18215000CB038FED79CBCFED793E00ED790520EE3EFCED79D1C1F1C911
:18216800E5C5D521092206071668CB02CB82CD8720CDA5201600CDA561
:1821800020CD96201668CB02CBC2CD8720CDA520CDCB2072CD962023F6
:1821980010EAD1C1E1C9E5C5D5211B2206071E001668CB02CB82CD8705
:1821B00020CDA52053CDA52056CDA520CD96201C2310E5D1C1E1C92184
:1821C80009223A1822CB7F20042323232311102213131313133E0073
:1821E000ED67121BED67121B23ED67121BED67121B23ED67121BED67C8
```

length : Ln : 9 Col : 60 Pos : 548 Windows (CR LF) UTF-8 INS

Authors Development platform showing Editor, Assembler and Serial Transfer tools

oshonsoft.com – which offers an editor, assembler, disassembler, debugger and more all in one place.

Today I have settled on Telemark TASM which does the job nicely and has a good feature set. It is DOS based so driven from the command prompt however Oshonsoft can also call TASM directly, making assembly a case of just pressing F9.

I know that John Hardy prefers an all-online approach via asm80.com which is an online editor and assembler all in one.

Code Transfer

I have adopted the SC1 hardware as my development environment. The SC-1 includes a serial interface and ROM based transfer routine (Fn, 1) that accepts Intel HEX files.

Craig Jones has recently produced the TEC-1F PCB & CMON monitor which also supports a serial interface and Intel HEX transfer (as well as other modernizations); Brian Chiha's BMON for the TEC-1F has Intel HEX upload capacity also.

TASM generates the Intel HEX file when I assemble, so I simply have the HEX file open in Notepad++, start the Intel HEX transfer on the SC-1, and copy and paste the text to the serial port via the well-known serial program 'PuTTY'. 'Terra Term' works too.

Uploading new code proceeds as follows:

- SC: Reset, Fn, 1
- PC: Copy .HEX file from notepad++ & paste into PuTTY
- SC: Fn, 0 – code running.

FTDI serial adaptor

To connect the PC to the TEC I have purchased a cheap USB to Serial board on eBay for a few dollars. Don't pay $20+ at Jaycar!!

These boards are broadly known under the generic model name 'ft232' or 'FTDI' and are around $2 each. They use a standard USB to mini or micro-USB cable.

John Hardy has an online solution here too, with the Wicked TEC-1 emulator, that also accepts .HEX files.

Wicked TEC-1 Emulator

Debugging

Like Z80 assemblers, numerous debuggers exist on the market, but since I have it, I tend to use the Oshonsoft debugger if I want to debug code outside the TEC itself.

There is an amazing Z80 add on for VS code at Github which allows direct debugging within the editor.

Debugging is, of course, still a largely manual process but since one can now make code changes, re-assemble and re-download to the TEC with just a few clicks, it becomes trivial to test out code changes. No more tedious editing via the TEC keypad.

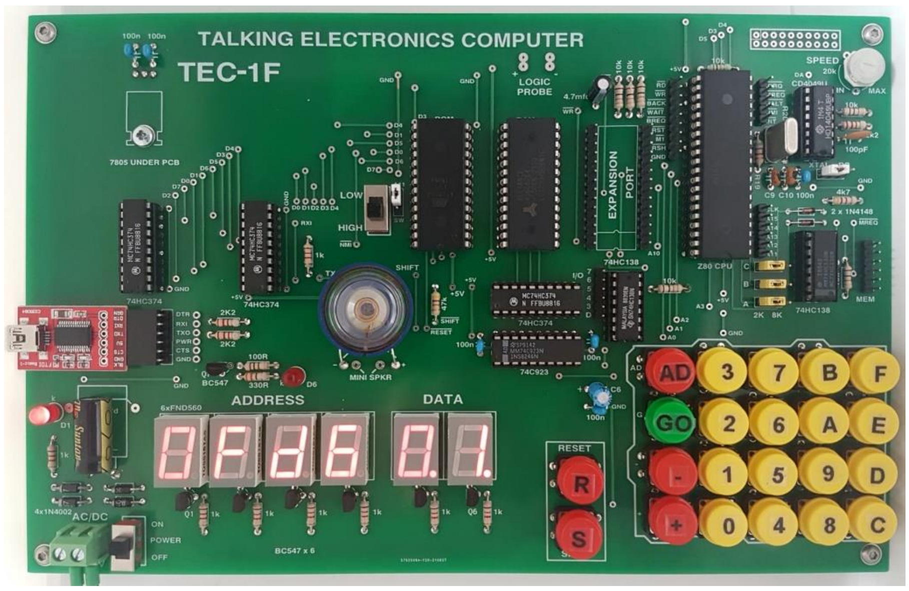
The best part of debugging is really the rapid write-test-revise cycle.

The serial port is of course two-way, so it can also act as debugging tool – data can be written to the serial port and viewed on the PC screen. In this way values of registers, memory etc. can be dumped out at various key points within your code, allowing a historical log of program behaviour to be created that is a lot easier to analyse compared to other methods.

GitHub? What's that?

Obviously one can simply save one's work to whatever storage device you prefer (e.g., your C: drive, USB etc.), in the traditional fashion.

One issue here is that of keeping different versions – with software development it's easy to make a change that doesn't work – but not always easy to remember how to get back to where you were.



TEC-1F with FTDI Serial Adaptor fitted

Another issue is – how to share your work and work collaboratively as a team?

Enter, GitHub.

GitHub is basically a website for storing, publishing sharing and (optionally) working with others on software projects.

Now is not the time to promote all of GitHub's features, but amongst the major ones, GitHub addresses these concerns and much more.

Please consider researching how GitHub can work for you in terms of sharing your work and keeping your code safe and manageable.

Much of the TEC's modern ongoing development as well as historical information, tech data, schematics, ROMs and more can be found on various TEC GitHub websites, located at

TEC https://github.com/tec1group/

Craig Jones (TEC-1F, CMON)
https://github.com/crsjones

Brian Chiha (BMON)
https://github.com/bchiha

Author
https://github.com/1971Merlin

Summary

Rapid, simple code development is now very achievable at basically no cost. The tools are available freely on the Internet and now, both the TEC-1 and the SC1 have the necessary interface and software to complete the picture.

Now is also the time to consider upgrading your old TEC-1B or earlier board to a modern TEC-1F or give the Southern Cross SC1 a go.

The 1F board offers significant improvements such as supporting 8k ROM and RAM, onboard crystal oscillator, Serial IO, as well as all the other mods and improvements the TEC has seen over the years. It can also be set back to 2K addressing mode & trimpot based clock speeds if required for legacy compatibility.

I strongly suggest you try it for yourself, as the 1F is sure to become the 'modern standard' for the future of the TEC family.

New that Both TEC-1F and SC1 PCBs are available via eBay – check seller
https://www.ebay.com.au/usr/craigrsj