How It Was: Programming (and debugging) microprocessors

Clive (Max) Maxfield - November 18, 2011

Editor’s Note: This “How it Was” story is told by Aubrey Kagan, who is a professional engineer with a BSEE from the Technion-Israel Institute of Technology and an MBA from the University of the Witwatersrand. Aubrey is engineering manager at Emphatec, a Toronto-based design house of industrial control interfaces and switch-mode power supplies. In addition to writing several articles for Circuit Cellar and having ideas published in EDN and Electronic Design, Aubrey is the author of Excel by Example: A Microsoft Excel Cookbook for Electronics Engineers (Newnes, 2004).

I cut my teeth on the RCA COSMAC 1802 microprocessor. In 1977 I was working for Racal (a British telecommunications manufacturer) and we needed a CMOS processor for a hand held radio set. There was only one other alternative – the Intersil IM6100 12-bit micro built around the DEC PDP8 minicomputer instruction set. 8 bits was much easier than 12 from the perspective of byte wide memories and peripherals. Actually there were precious few of either, but those that existed were either 4 or 8 bits. At the time there was an article in Popular Electronics to build the “Elf” micro development board. Using this as a base I created a “development system” designed around several single Eurocards in a 19” rack. The front panel had the luxury of an address bus and data display in hexadecimal, rather than a binary display with many LEDs. The keyboard had toggle switches rather than a keypad. It was possible to dissociate the micro from the bus and enter an address and data directly into memory. Aside from this, the only debug feature was the ability to halt and single step through the program. There was no breakpoint capability – the equivalent was only possible by inserting a “GOTO HERE” loop in the program. I don’t recall how I did it, but I did manage to implement a 1K program that included reading and sending code over a 150-baud modem, interfacing to a Baudot teletype, programming a 32x8 PROM with a unit identity and activating several outputs.

The author "Then" (left) and "Now" (right)
I had a blank programming sheet template with allocated spaces for the mnemonics, op-codes and comments. After the flow chart was created, I would write the program in mnemonics and then translate it into op-codes, which was particularly difficult when working with relative jumps. Once coded into hex format, I would then enter the code via the toggle switches into RAM and debug. When a bug presented itself, you didn’t want to edit and re-enter the code, so you inserted a “GOTO” at the appropriate place and then inserted the corrected version of the code somewhere else. I developed the habit of leaving gaps between subroutines and allocating fixed addresses to the subroutines to allow me to tack the corrected part later without having to recalculate all the jumps. The RAM was not battery backed up, so any power fluctuation would result in keying in the data afresh. Fortunately, the electricity supply in South Africa back then was a lot more consistent than it is now, and I think that only happened once.

At that time there were no CMOS PROMS (erasable or otherwise) and so the design used a 1K x 8 bipolar PROM for the program storage and loaded bits of the program into a pair of four-bit 256 byte CMOS RAMs. Power to the PROM was only applied when data was fetched. I don’t know why we decided to make our own programmer, but we did. There was no connection to allow me to download data from the development system to the programmer – this had to be done manually as well.

Testing the product had to be performed at elevated temperatures. After a while the unit started exhibiting some very unusual behavior. It took me some time to try and understand the problem. At some point I had read a report that the fuses on the PROM could regrow, but dismissed that as one of those things that happen once in a million years. In frustration I read back the PROM and compared it byte by byte with the handwritten listing. Sure enough the fuse in one byte had regrown. And they say that those were the good old days!

In 1979 I succumbed to the entrepreneurial bug and started my own business with a partner. Our idea was to develop a desktop computer running the CP/M operating system. My partner (marketing and mechanical design) was already planning for providing the whole package – hardware and enterprise software. I started designing around an Intel SDK85 (see image below), which provided a monitor in addition to the LED display and was thus a significantly improvement over my original 1802 design. When working with CMOS the easiest test for correct operation was simply to put your finger on the IC. If there was any detectable temperature there was a problem. The 8085 was an N-MOS device, and I remember thinking as I sucked the burn on my finger that something was definitely wrong. The SDK85 still required program entry by keypad and I needed something better.
At the time, the Z80 was all the rage and so I bought a single board development kit from SGS-Ates (an Italian outfit now subsumed into ST... which was a French manufacturer Thomson... which acquired Mostek as well). The kit had a monitor and drivers to connect to a teletype and a high speed paper tape reader. This was living. The monitor would allow you to load the editor from paper tape. You would then enter your program, edit it and list it if so desired. When complete, you could dump your program out onto paper tape. Since the teletype worked at 110 baud, the benefits of a high speed reader quickly became obvious and I acquired and figured out how to connect one.

Documentation wasn’t great in those days either! Although direct telephone dialing was possible, it was terribly expensive and I was thoroughly disenchanted when tech support in England was unable to give any help at all.

Once the program was on paper tape it was time to assemble it (this was in assembler only). Firstly you had to read the assembler program from paper tape into the memory. Assemblers in those days were referred to as two-pass or three-pass. This meant that your program had to be read two or three times to arrive at the desired object code. This was necessary to create a cross address table
in order to calculate the jumps etc. As I recall mine was a two-pass assembler. So, at the prompt of the assembler, you would run your assembler code through the reader, rewind the paper tape, and then wait for the second prompt and do it all again.

The output would be a program listing and a paper tape containing the object code. There was no possibility for module development to reduce the length of the paper tape, although it was possible to enable and disable portions of the listing through in-line switches. A twenty-page listing at 110 baud can be very time-consuming. The paper used for the listing was friction feed and came as a continuous non-perforated roll. Although the listing was formatted, you still had to cut each sheet with a ruler and then bind it either with a staple or with a clasp.

Debugging involved loading the tape of the object code and then using the monitor to run or step through the program. At least it allowed breakpoints and even a one line assembler. Some of the more sophisticated monitors allowed addressing by name rather than absolute address, but I can’t recall how this one worked. Once again, corrections in the short term were implemented using patches so as to avoid having to re-assemble. The same techniques of GOTOs and leaving gaps between subroutines applied here.

Once the development was over, the object code was loaded into a PROM programmer and then the EPROM was inserted in to the target hardware. Next was the eternal challenge – how do you know that the system is working or – more importantly – why it is not working? Some people wrote their own monitors to address this, but I quickly became a confirmed believer in In-Circuit Emulators.

As you can imagine some of these tapes were quite long. It was possible to get the tape in fan-fold format, but I never manage to find a supplier. Organizing the length of paper tape could be quite a challenge. They could form a roll 5” or 6” in diameter. Reading them would result in many feet of tape spewed out over the floor. Rewinding them was not only tedious but could also lead to damage of the tape. My solution (although not my original idea) was to wind them on to a plastic bobbin derived from a sewing cotton reel. My mom was a sewer and there were many “empties”. I see they are still available ([Click Here](#)). I would cut a slot and feed the tape into it. Then I had to find a method to wind it on. I acquired a manual grinding wheel (see photo below), removed the grindstone, and wound tape around the shaft to increase the diameter. I then wedged the bobbin onto the shaft and although it was a two handed operation, rolling the tape became a breeze.
Most of the CP/M computers at this time (around 1979) were designed to work with a teletype input, which quickly morphed into “glass teletypes“ as dumb terminals were called. In the middle of our desktop computer development, Lear Siegler (one of the glass teletype manufacturers) brought out a desktop system that was a clone of the DEC (the largest minicomputer manufacturer of the day) PDP12. We figured there was no way we could compete and so I was left rudderless. It did not matter that DEC successfully sued Lear Siegler and their product never made it to market.

I found my way into some industrial design and decided to revert to Intel, largely because of the quality of the support, both hardware and personnel, of their distributor in South Africa. I broke down and financed an Intel MDS236 development system with ICE for the 8085 and 8048 families. The equipment cost more than my house! The paper tape approach had been replaced by 3 8” floppy drives- 1 single density (720KB) and 2 double density drives (1.2KB). I also had a high level language: PL/M. It was now possible to develop software in modules and use libraries. Although the processes were similar (three-pass assembly), they were transparent to the user and there was mostly enough disk space to do this, although sometimes you had to shuffle floppies. It mainly supported Intel products, although there was a plug in ICE for a Z80 (see photo below).
I got a project designing a calorimeter and, for cost reasons, the customer opted for an 8080. I could produce my code on the development system since 8080 and 8085 code was identical, but I could not debug since I did not have (and couldn’t afford) and new ICE. I managed to get a used Intel development tool called a μScope which was essentially a reduced feature emulator in an attaché case (see photo below). The user interface was a bit clumsy (especially as I never received a user manual), but still usable.

I also acquired an Osborne 1 for the express purpose of producing data manuals (using Wordstar) so that I would look professional. I also ordered Supercalc, which was my first introduction to the world of spreadsheets. (Supercalc was for CP/M-based computers; Visicalc was for Apple machines.) The Osborne 1 was a “luggable” computer and had a 5” screen that was like a magnifying glass view of your document. You could see about 24 characters (of an 80 character wide document) and 10 lines at a time. Storage was limited to two 360KB 5¼ inch floppy disks, one of which had the application you were running. Changing floppies was not a simple task of opening the drive latch, removing one, and inserting another disk. On CP/M machines there was some initialization required every time disk was inserted, all of which further complicated mass storage. We’ve come a long way!
The IBM PC finally became accepted as the industry standard about 1983 (in SA at least) and I started moving towards using it for documentation and PC layout. Some of my efforts are described in my earlier article *How It Was: PCB Layout from Rubylith to Dot and Tape to CAD*.

More than that though, Intel started accepting it as the development base for all their hardware and introduced the ICE5100 emulator for the 8051 hosted on a PC via an RS232 connection. Intel even created emulators to run all the existing software compilers, assemblers, editors under PC-DOS. By 1986 the development approach was not much different to what it is today, with the major exception of the user interface.

Unlike the single chip microcomputers that we use today, back then the microprocessor had to connect to RAM, ROM and peripherals externally. That meant there were up to 8 data lines, 16 address lines, and 3 control signals (27 lines) snaking their way around a PCB. The probability for a manufacturing fault increased dramatically and Hewlett Packard believed they had a technique to aid debugging when a product failed test in manufacturing. They created an instrument called a Signature Analyzer (see photo below) to capitalize on the idea. It also provided much amusement when seen by the uninitiated who assumed it applied to one’s John Hancock.

In circuits with repetitive waveforms, diagnostic manuals had pictures of the waveforms identified with nodes and the settings that produced these waveforms. Of course microprocessor busses are non-repetitive and it is quite an art to debug them. HP’s idea was to provide some “waveform” at

(Click Here to see a larger, more-detailed version)
each node to prove that the system was working. The concept was to force the micro to run a set of
instructions that would repeat a bit pattern through a particular node. This pattern is fed into a shift
register (in the Signature Analyzer) with some feedback loops similar to CRC calculation to generate
a 16 bit signature that is displayed as a 4-digit hexadecimal word on the instrument. That could only
be done when the basic system was working. To start up there had to be some method of opening
the data bus and forcing the micro to execute a single instruction over and over to allow the exercise
of the address bus and ROM read signal. This was fairly easy in Intel processors because the NOP
instruction was 00hex and so all that was needed was an 8 way DIP switch to open the bus and 8
diodes connected in a common cathode arrangement with a switch to ground. You could then
establish a signature on each address line and EPROM data output, and slowly enable the memories
etc. from there.

Click Here to see other articles in this "How it was..." series...

Editor's Note: It would be great if you took the time to write down short stories of your own. I can
help in the copy editing department, so you don't need to worry about being “word perfect”. All you
have to do is to email your offering to me at max@CliveMaxfield.com with “How it was” in the
subject line.

I can post your article as “anonymous” if you wish. On the other hand, what would be really cool
would be if you wanted to add a few words about yourself - and maybe even provide a couple of
“Then and Now” pictures showing yourself as a young engineer ("Then") and as the hero you've
grown into ("Now").

If you found this article to be of interest, visit EDA Designline where - in addition to blogs on all
sorts of "stuff" - you will find the latest and greatest design, technology, product, and news articles
with regard to all aspects of Electronic Design Automation (EDA).

Also, you can obtain a highlights update delivered directly to your inbox by signing up for the EDA
Designline weekly newsletter - just Click Here to request this newsletter using the Manage
Newsletters tab (if you aren't already a member you'll be asked to register, but it's free and painless
so don't let that stop you [grin]).