Learning assembler

In Due Course

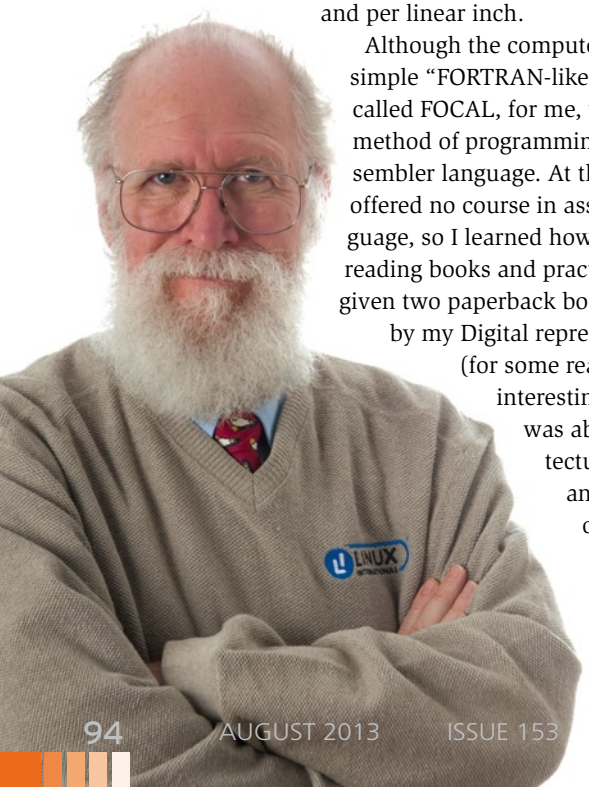
"maddog" explains why a knowledge of assembler, or other machine language, can be very useful. *By Jon "maddog" Hall*

I am working on a long-term project for Linaro, the association of ARM vendors that are working to make GNU/Linux work well on ARM processors. The project requires me to delve into assembly and machine language code – something I have not done for more than 20 years. I would like to tell you why I am so excited about this particular piece of work.

In 1969, I was an electrical engineering student at Drexel Institute of Technology (now called Drexel University) in Philadelphia. Drexel was a cooperative engineering school and I was lucky to get a "coop" assignment at the Western Electric Company in Baltimore.

While at Western Electric, I enrolled in a correspondence course called "How to Program the IBM 1130 Computer in FORTRAN." The course consisted of a book that described how to punch cards, write a FORTRAN program, and run it on the IBM 1130, which we had in the engineering department. That machine was so small and slow by today's standards that it only ran one job at a time, and you linked the device drivers into your program not the operating system. In effect, you "booted" your program to run it. This correspondence course was my first exposure to software.

When I returned to Drexel, I found the electrical engineering department had two Digital PDP-8 computers, along with two ASR-33 teletypes (paper output with a keyboard for input), that used paper tape for input and output for "storage." These machines punched paper tape at the rate of 10 bytes per second and per linear inch.



Although the computers did have a simple "FORTRAN-like" language called FOCAL, for me, the main method of programming them was assembler language. At that time, Drexel offered no course in assembler language, so I learned how to program by reading books and practicing. I was given two paperback books from Digital by my Digital representative who (for some reason) found me interesting. One book was about the architecture of the PDP-8, and the other was on binary arithmetic and how to program in assembler.

The PDP-8 was a very simple machine. It was designed before nano-electronics, and today the architecture would be called a "Reduced Instruction Set Computer" – very reduced. Every instruction was 12 bits long; the machine only had one general-purpose register (a 12-bit accumulator that also did all I/O) and was so simple it could only add. Subtraction was done by taking the two's complement of the subtrahend and adding it to the minuend. Although this machine was primitive by today's standards, it was a machine that I could touch and really get to understand, as opposed to the IBM and Burroughs mainframes, which were kept behind locked doors.

When I graduated a few years later, I won my first professional job because I had taught myself to program in assembler. My prospective employer asked me if I could program in IBM 360 assembler. My response was "Do you have a book?" I programmed in assembler for Aetna Life and Casualty during a four-year career. Later I went to teach at Hartford State Technical College. I had to teach myself PDP-11 assembler (a really nice CISC architecture), so I could teach it to my students.

My knowledge of assembler languages was also useful in building and programming early microcomputers, which were just coming out at that time, as well as in teaching operating systems courses and (my favorite course) compiler design. The knowledge of machine architecture and languages helped me explain to students the fine details of how the machine actually worked.

While working for Bell Laboratories, I taught courses at night for Merrimack College in Massachusetts. I tried to schedule a class on assembler language for some students and ran into some opposition from the Department Head of Computer Science in the day school. Finally, I went to speak with him, and he asked me why I thought the assembler course was necessary, because no one he knew programmed in assembler.

I told him about all the times that knowing assembler, or any machine language, had helped me debug errors that the compiler made or helped me write programs in high-level languages that executed faster because I knew how the object code might be generated. I also pointed out that, when I taught compiler theory or operating system design, a knowledge of machine code helped the class understand "reentrancy and recursion in high-level languages." I did not notice how the Department Head's eyes widened when I said those words.

After a few more minutes, he agreed to offer the course, and as I was leaving he asked me to make sure I invited him to the discussion of reentrancy and recursion. "I never really understood how those worked," he said. ■■■