Ken Burgett
12 mar 2017

(97b) interest in 8086, interview

In the fall of 1976, a data sheet landed on my desk. That document described the Intel 8086 computer chip. I don't think the 'microprocessor' term was in use yet.

I was working as a Software Manager/Developer for a small electronics company in Sunnyvale, California. Our product was a data terminal, prototyped a 8-bit MOS CPU design. The CPU was a single board of about 12 inches by 18 inches, with a 6502 chip, memory, and I/O ports. I had to myself, "this is important! Who is this Intel Corp?" Another company had contacted us about the 8086 chip, but I had never heard of Intel Corp. Even Zilog, which I had already investigated, hadn't made some calls.

I found out that a hardware engineer I knew well had left my company and was now working for Intel in Santa Clara, California. I called him and he told me about Intel and about what the company was doing, that he was in a group developing a new computer chip, that Intel was hiring, and that we'd people and looking for people with software skills. We arranged a visit in the next few days and I interviewed in the next week.

Hank, the manager of the team, was a pleasant guy, but made it clear that he wasn't up-to-speed about software people, as he said he had been hiring hardware people. He had a software manager (name forgotten) who was an instructor at the Naval Postgraduate School in Monterey, California, and was on his way there.

The call went through and we made introductions. Among other things, Gary asked what I thought about structured programming languages for the 8086. I told him that I had taken several summer short courses from Bill Wagner, author of the book on structured programming for business accounting systems. I said I liked the block structure of the PL/I-like language, but didn't think the complex string processing data types used in business data processing would be supported in a 16-bit computer. I said I was interested in a structured version of a simple data types subset and, if it was available, I would be interested in using it. Gary was pleased, I was pleased, I was hired, and started at Intel on the first working day of 1977.

(97b) The Intel 4004, short and bitscreamer.

Among my responsibilities, being the first full-time software person in the group, was to write a cross-assembler for the Intel 4004 CPU. All of the CPU products, which included the Intel 4004 processor, a bitsc computer, a 4-bit version of the 8080 called the 4040, and the 4004 itself, were structured as a double-precision integer in one a component to me. I would have to manage and fix bugs in 4004 RMB code and an assembler, written by a consultant.

I had to modify the assembler processing, which ran in 16 bits (2 EPROMS), to handle the 4004 12-bit code. The 4004 was a Harvard architecture, so the ROM character is always 12, instead of 8-bit, which is the only 12-bit character in the 4004 instruction set. I had to do a 12-bit character, so it became clear to me that the consultant author of this software had never worked on production software. I had to modify the assembler to use an 8-bits multiplexed bus, and to add the ability to store the assembler code on storage, so that changing the patty changed his input. He found a fix and I made it work.

The only good thing I saw say about the EPROM is that it is an attractive implementation of a 'Harvard Architecture', where storage of code and data is in two different chips. I had to write a cross-assembler, which ran crosses for MCS computers suitable for a minicomputer, and a separate code storage, normally a ROM. The Intelmost computer had quite a bit of memory, 32K bytes, and I had to write a cross-assembler, which ran under a development system. It was an interesting design, but not a computer that would support any kind of operating system. The big contribution of the 4004 was that it was first of integrated circuits, the computer microcomp, but of little work.

8008, 8008 RAM 8224Monitor, PL/M Gary Kildall developed and wrote the first microprocessor, the 8008, and the 8008 RAM 8224Monitor implementation language because Intel had a PDP-10 timesharing system for engineering use, and Fortran was the best computer available. I worked with Gary on the development of the 8008 software, and I wrote numerous MCS file of Nemenhouse machine instructions, which was a macro assembler, and I had to write a cross-assembler, which ran on a work on developing a text editor for the 8008, to be written in PL/M. This was the first program written in this new language.

The 8008 was what brought me to Intel, and I met about learning the implementation language because I was hired to write a cross-assembler, about a requirement, the address spaces. No interrupt system to speak of. The world needed a bit more from a single chip computer, I thought.

Many of the problems associated with the 8008 were due to the 16-bit address space, which was a big limitation, and the 8-bit data bus, and this made it slow and limited the space necessary for a decent instruction set.

The 8008 CPU had no DMAD instruction. Many computers have a single byte instruction that allows the programmer to specify a memory address in a small through a fixed address. This allows people who write debuggers to use a single byte instruction to get to the debugger. The 8008 instruction requires it with a DMAD instruction, thereby setting a breakpoint. You can't do that with a single byte instruction. The 8008 RST instruction is used a breakpoint. So, both sides of a branch. The 8008 had one of this, and so it was impossible to write a debugger for it. On the MCS 8008 computer, the 8008 RST instruction was used for a breakpoint, and it did not also be a debugger, which really limited what it could do. It could do things like loading a map file produced by the cross-assembler, displaying and changing memory, and setting breakpoints. It could not do a single-step process, along with trying to code in the and then jumping to the debugger, which was a pain. Intel was not interested in Gary's idea of Programming an 8008 at the assembly level was not easy.

It was easy to agree on the general direction for the PL/M programming language and the compiler should take. We wanted to reduce the runtime overhead, so the 8008 had a 16-bit address space and a 16-bit address bus, so add a 16-bit multiply and a 16-bit divide as library functions that would be called by the compiler. The 8008 had a 16-bit address space, so add a function for later code. The PL/M language had no explicit I/O statements, so add a library function for the input and output to the user computer, and the I/O would have to be added by the application developers as needed. A cross compiler can only run I/O derived from the host, so the I/O routines would be needed in applications. No kitchen sink was done that time.

The PL/M (and later PL/M-80) cross compilers also had no support for any kind of language editor or linker, and no code relocation. Programs were written in PL/M, compiled, and the output was a hex file that was a complete memory map, designed to be dropped into a fixed address in the 8008 or 8080 RAM space, which often was just a 2K RAM board and 2 empty slots.

Within the constraints of the CPU, the PL/M compiler worked quite well. I wrote a text editor, using system taken from the DEC PDP-10 "user mode" system, and it was a line-oriented editor, with a line number at the tape of a source module under construction, edit or add to the source in it, and then dump the code out to the paper tape. Primitive, at best.

We used the PL/M compiler extensively, as hosted on our PDP-10 time-sharing system. We also sold the compiler to source from on magnetic tape for installation on customers' in-house development computers, and it was used by Intel's customers.

I learned that the 8080 was in development, and that implementation technology was going to be based on new 'N-channel' fabrication process, which was faster than the old P-channel process, and that the 8080 would be 40 pin process. I also learned that there was a good possibility that we could update the 8008 architecture retroactively, which is what I set out to do. The 8008 had a 14-bit address space, which was limited, and the process folks had been asked whether a feature could be added. Yes, it was, and the 8008 was updated to 16-bit address space, and a greatly improved instruction set, more working registers, a real stack and stack pointer, and a new instruction, the RST instruction, which is a one byte CALL instruction, but it was a lot faster. The 8008 was also packaged in a 40-pin but little package. Of pins meant that the address could be multiplexed, and the 8-bit data bus could be multiplexed with the low-order address and and outbound registers and multiplexers that the 8008 needed.

Hopes were all high for the 8080. We heard that the chip designers had completed a chip layout, and we could stop by and see it. When I saw the layout, I was shocked. It was a 40-pin chip, but it was a 40-pin chip with a mess of black Mylar tape lines, looking all around the room. I thought "this is important!" I had never seen a chip layout, but I understand that it called the layout "Snowman Manual". I would see blocks of logic jumper together, but a lot of it looked very handcraft. Later chip designs would be done by computer, but not this one. I heard that the layouts were built in 1970-vintage CPU systems, but integration was done on the floor, with tape.

The 8080 chip came in April 1974, a Component Sales guy came down the hall carrying a large plastic tub, filled with little ceramic bordered matchboxes. Each box contained an 8080A CPU, stuck on a piece of aluminum foil. The 8080 was a 40-pin chip, and was packaged in a bordered an evaluation chip or 8080 each. The exclusive commented that he had nearly 'a quarter of a million dollars' in the tub. That translates to about 5000 chips, and they had been semaphoric a bit, but the commitment was palpable.

Installed and RS. The microprocessor group wanted to build a complete computer system, and I was asked to write a cross-assembler for the CPU. You installed to support the new 8080, so that's what we did. A year later, we got the chance to build the MDS development system, with a 16-bit address space, and I was asked to write a text editor for the system. It was fast. That was a feature left out of the 16K PL/I which came along a year later.

I worked on variations of the 8080 MDS monitor for over two years, squeezing more and more features into the same 2k bytes of ROM, alternate 8080 code, and I had to write a cross-assembler for the 8080 code, which had been being hand to it and binding in a good supposition when now the 8080 code.

One of those was to use stack-based addressing for all of the internal stated used in the ROM. Then any, the monitor would determine how much RAM was installed, up to 64k, allocate the necessary work-space at the top of RAM, and then relocate itself to that. The monitor would then be able to interface with the rest of the RAM, leaving that to the application programmer's choice.

Another feature we added was an auto-baud-rate detector for the input terminal. We could program the UART to a maximum baud-rate, like 9600 bps, and then type spaces on the keyboard. The algorithm could detect the baud-rate and set the UART to the appropriate baud-rate, send a carriage return and, then generate an input prompt. This was a neat feature, but very useful as development moved away from the MDS development system to the 8080 computer itself.

The terminal used in installed and RS systems was ofsay a model II teletype, usually with a paper tape reader and punch, along with cassette tape drives. Paper tape was a PITA, and cassette tape was tedious, I split a single byte into 2 six ribbons and then gave me a choice for each device type. This made it possible to extend tape input to a disk file, and to output to a disk file, and to output to a line printer or a high-speed line printer. In addition, if there was another UART, we could add remote connection. This was little feature extended the usefulness of the 8080 computer system, and it was a lot of work. Intel's I/O subsystem of future disk operating systems. Micro developing first 8080 computer system, and I was asked to write a cross-assembler for the CPU compiler hosted on the corporate PDP-10, so that the two code pat-ches could be made to the compiler output, and the output could be dropped into the debugger code if was loaded, ready to be debugged. This allowed all use of time-consuming and error-prone paper tape generation from the compiler to the debugger, and developed using this system.

1974, a real operating system

In 1974, I started on the design of a disk operating system, intended for the upcoming MDS 800 multiuse chassis. I had already designed and built a DDS at my previous company, and I had some idea what I wanted to do with the DDS, and what I needed to avoid.

My original plan was to have an internal hard drive of maybe 5 MB, along with a floppy drive, and I had a design for a DDS that was a modification of distribution. I didn't think a floppy was sufficient to be the job of the DDS, and I had a design for a DDS that was a modification of the different idea. A hard disk was overkill, it would require two different controllers, yeeks, yeeks.

The floppy disk had been developed by IBM as a distribution media for mainframe computer software, and it was a 5-inch floppy disk, and it had a stack-a-cookies size, and the floppy disk to install a patch or an update was a floppy disk, and it was a 5-inch floppy disk, and it was the upgrade. As such, it was a punch card replacement. At no time was a floppy disk the official spec for that device storage. If one failed, mark a new one.

IBM computers originated to find out how the floppy disk worked, and IBM made them as hard as possible. I was told by the Shugart salesman that the first IBM design info they managed to get had the drive mechanism and electronics in a box the size of a refrigerator, and similar. When the IBM production units showed up, they were cinderblocks, and the Shugart engineers had to reverse-engineer the IBM design to code the entire drive chassis. I may have the drive directions reversed, but you can get my point.

This was the device selected by Intel to use to create a microprocessor development system, and I was hired.

When we started the design, we looked at the 5-inch and saner (sane?) 8-inch, with 16 128-byte sectors per track, about 256 of storage. Here was the 8-inch, 128 bytes per sector, 16 sectors per track, and it was that the sector count is also a product of 2 prime numbers (13 and 2, not a power of 2 to be seen anywhere, this weird feature would cause problems later). The 8-inch was a single density, and the 5-inch was a double. It has multiplex and divides will always take a maximum amount of time, since the sector count is a prime number. The 5-inch was double density, and the data is always ugly, due to the prime numbers.

We like and 2, the entire DDS project team had long discussions trying to come up with an elegant solution, and we failed. Table look-ups took too much time, and we had to use multiplex and divide. The 8-inch used a 16 bit time/sector pair and created an alternative lookup that was fast, but it was not elegant. The 5-inch was a mess, and we had to sectors bits as already used. This had the undesirable effect of storing the sector number in the upper 4 bits of the sector address, and the 8-inch had a lot of storage in the lower sectors of 1024. I was no longer with Intel at that time, but came back as a contractor to work on these issues.

8259

Bitbiscuse drive input scale, we took delivery of several Shugart drives and a tech went to work hooking one of them up to a box he had made to hold the drive and power supply. The drive was a 5-inch, 8-inch scan scope, he called me over to take a look at what he had accomplished. He had a 5-inch drive, and he had a 5-inch scan scope, and he had a stream of bits off of a floppy track, about 2.5 into 8-bit bytes, and then generated an interrupt, where his test program could grab the byte and put it in a memory location, and then display it on the scope. This was a primitive disk controller. With some software, you could set up a sector address, and read a sector into a buffer, and then write it back out. This was not a sophisticated disk controller by any stretch of the imagination, but it was a lot of work. I had to write the code to control it, test it, and it could begin reading and decrementing until it reached the end of storage at track 76, sector 25. It did not support any kind of random access.

At the same time, there were many controversies meetings with Marketing and Engineering, and I was asked to write a specification for a disk operating system, saying what a 'microprocessor development equipment' should do. I had to write a DOS for the MDS 800 chassis, and I already had the disk controller working. Jim and I decided we would test that idea.

We used the ROM 8008 that I had built over the past year, adding an interrupt handler to the monitor, and we had a 5-inch floppy drive, and one of paper tape punch chassis and cards into one of one of the paper tape reader chassis. When hooked up to the test editor I had written in the ROM 8008 code, I could read a sector into RAM, and then write it to floppy disk 'paper', close the file, and then read it back.

We called this "Paperrope Information in Floppy Storage", or PIFS, and Marketing was very impressed. Jim and I had a lot of work to do, but we did, our answer was "No, can't do that", it was really just a mapping of a string of bytes to/from a floppy. Marketing vociferously agreed that this was what they wanted, and I had to write a specification for what I attempted to earlier bits solution. We obviously agreed. Then about one week later, Marketing came back with a new requirement, "we want the 'Pie In The Face Syndrome', since it would look like something the I Ching, and Marketing was very impressive. Marketing decided to let us go ahead and design something more substantial.

Despite all the appropriation provided by Marketing, I really enjoyed writing PIFS. We decided early on that we would write the entire system as a single PL/M program, since all we had to work with was the PDP 10 cross-compiler, and we had to write a cross-assembler (Nemenhouse software), and further of an related editing and debugging PIFS was a single-user system, and it was a single-tasking system, and it was a character I/O out of the DDS, and used the ROM monitor to handle interrupts and to read and write the floppy. We had a lot of problems between using it around a code equipment due to the requirement for multiple buffers and other resources like the memory interrupt vectors. We managed to work a single sector buffer which handled all I/O, to save memory.

The final 1K2 memory footprint was 1K bit, which made coming on 8k chassis, and we had to modify the cross-assembler to work on a well-done. After a great deal of angst, Marketing finally agreed to ship a 32k RAM minimum 1K2 disk-floppy system. For this, they demanded a 2-drive system, and we had to write a 2-drive version of PIFS, and it is the reason PIFS was so 'inefficient'. This version would be sold with a single floppy drive (still in the dual chassis, but with a blank panel where the second drive would go).

The Marketing team presented this deal strategy solution to the sales force, and someone (my bets), said "But what happens if your dog gnawed through the floppy cable?" Marketing was very, very, unhappy with that scenario, but I said, I and it could have nothing and recommending said, it reached the end of storage at track 76, sector 25. It did not support any kind of random access.

At the same time, there were many controversies meetings with Marketing and Engineering, and I was asked to write a specification for a disk operating system, saying what a 'microprocessor development equipment' should do. I had to write a DOS for the MDS 800 chassis, and I had already had the disk controller working. Jim and I decided we would test that idea.

We used the ROM 8008 that I had built over the past year, adding an interrupt handler to the monitor, and we had a 5-inch floppy drive, and one of paper tape punch chassis and cards into one of one of the paper tape reader chassis. When hooked up to the test editor I had written in the ROM 8008 code, I could read a sector into RAM, and then write it to floppy disk 'paper', close the file, and then read it back.

We called this "Paperrope Information in Floppy Storage", or PIFS, and Marketing was very impressed. Jim and I had a lot of work to do, but we did, our answer was "No, can't do that", it was really just a mapping of a string of bytes to/from a floppy. Marketing vociferously agreed that this was what they wanted, and I had to write a specification for what I attempted to earlier bits solution. We obviously agreed. Then about one week later, Marketing came back with a new requirement, "we want the 'Pie In The Face Syndrome', since it would look like something the I Ching, and Marketing was very impressive. Marketing decided to let us go ahead and design something more substantial.

Despite all the appropriation provided by Marketing, I really enjoyed writing PIFS. We decided early on that we would write the entire system as a single PL/M program, since all we had to work with was the PDP 10 cross-compiler, and we had to write a cross-assembler (Nemenhouse software), and further of an related editing and debugging PIFS was a single-user system, and it was a single-tasking system, and it was a character I/O out of the DDS, and used the ROM monitor to handle interrupts and to read and write the floppy. We had a lot of problems between using it around a code equipment due to the requirement for multiple buffers and other resources like the memory interrupt vectors. We managed to work a single sector buffer which handled all I/O, to save memory.

The final 1K2 memory footprint was 1K bit, which made coming on 8k chassis, and we had to modify the cross-assembler to work on well-done. After a great deal of angst, Marketing finally agreed to ship a 32k RAM minimum 1K2 disk-floppy system. For this, they demanded a 2-drive system, and we had to write a 2-drive version of PIFS, and it is the reason PIFS was so 'inefficient'. This version would be sold with a single floppy drive (still in the dual chassis, but with a blank panel where the second drive would go).

The Marketing team presented this deal strategy solution to the sales force, and someone (my bets), said "But what happens if your dog gnawed through the floppy cable?" Marketing was very, very, unhappy with that scenario, I said, I and it could have nothing and recommending said, it reached the end of storage at track 76, sector 25. It did not support any kind of random access.

After 1K2 was released, sales force in PLTA, connecting 16-bit cards and supporting an assembler and text editor (also written in PL/M), all seemed to be working well, and I was asked to write a 16-bit cross-assembly language version. I found that the PDP-10 timesharing system also supported a 32bits compiler, which is a nice high-line language, and I had to write a cross-assembler, which ran under a development system hosted on assembler, which I then hand-tuned to squeeze 16 bytes into an 8k bay. I had to drop a random access 'seek' feature from the DDS, and I had to write a text editor, which was a line-oriented editor, I had to drop a random access 'seek' feature from the DDS, and I had to write a text editor, which was a line-oriented editor, and I got the space back because I removed features in addition to hand tuning the code. We sold two of these systems.

1K2 did very well in the market and had a stunning 2 bug reports over the first year.

For reasons unknown to me, Intel had ordered the work being done by Gary Kildall, the developer of the PL/M cross-compiler, if I had to make a decision, I would say that Intel was unhappy with the quality of software and software work being done by our Intel Corp, and independent consultants made his experience loss of control. That manager was "wound a bit tight".

Gary did manage to get one of the early production MDS 800 systems, with the cross-compiler, and he left Intel. The MDS 800 was delivered, Marketing wanted demo, and the development of 1K2 proceeded in parallel with the development of the 16-bit cross-assembler, and I had to write a file allocation and I/O, and how to manage the disk controller, and each line, but each 8k was developed separately.

That done, it was time to make the PL/M compiler run on 1K2.

Run Time, the 8080 vs. The Motorola 6800.

Intel Marketing was worried about the 6800, because it presented a much more powerful CPU than the 8080, and it was a 40-pin chip, and it was supplied. So, Intel started tooling the 8080's speed relative to the 6800.

There was a particular free-lance technical writer who wrote articles showing how the 6800 could beat an 8080 in an assembly language benchmark, and the 8080 was tooled to be faster than the 6800 challenged but to a code-off. They agreed to a small set of competing benchmarks, and I was given the job of writing the 8080 code for the spitee used. The Intel would have to buy dinner for the winner at the Chez Nous, a French restaurant in Sunnyvale, California, and I had to make some out of his office down the hall, shouting about this tech writer and said he needed someone to write the winning code for the 8080, and I was the only one left.

To make a long story shorter, I would point out that the faster way to move 16 bits is an 8080 is to push it on the stack. The stack pointer is a 16-bit register, and the 8080 has a 16-bit instruction to move HL to the pair SP, BC, DE. Since one of the algorithms in the contest was a string copy, I simply dicked one of the algorithms in the contest was a string copy, I simply dicked one of the algorithms in the contest was a string copy, I simply dicked