Micro to mainframe

Although Intel’s first microprocessor, the 4004, is usually associated with the Busicom calculator for which it was originally created, a connection to early mainframes actually exists that’s not widely known. One of these connections involved Ted Hoff and I, when we did the processor’s architecture at Intel in late 1969.

As computer architects, our job was to design it to be efficient for the proposed application tasks. We chose some features based on our previous computer experiences because most computers share similar features such as registers, a program counter, and common instructions. In other cases, we invented some new mechanisms. In this article, I describe some of the 4004 processor’s inheritance from mainframes and some other features.

Early experience and education

While studying mathematics at San Francisco State University in 1962, I learned to program the school’s IBM 1620 computer, a machine that was popular with educational institutions at that time. My professor assigned me the task of writing an interpreter program to let the 1620 emulate another computer called IPL-V. In other words, while running my interpreter we could accept and run programs written in IPL-V machine code. (A few years later, I used emulation in Intel’s first microcomputer system.)

At Stanford University, Ted Hoff customized his IBM 1620 with hardware added for speech recognition. He had programmed his 1620 to recognize single spoken utterances such as the numbers 0 to 9. Because of our shared experiences, I arranged to meet Ted in 1963 to see his computer in action. (This would be a prophetic meeting because I later worked for him at Intel.)

The IBM 1620 was especially designed for scientific calculations, and it worked in decimal arithmetic rather than binary. In the 1620, each addressable memory location holds a single digit, but the computer’s instructions handled multiple digits, and as a result, a data field could be lengthy. A “flag” marked the end of the data, so a single instruction could add two 10-digit numbers or even 200-digit numbers. This category of machines is called a variable-length data machine, as opposed to the more common fixed-word-length machines that operate on binary data 32 or 64 bits at a time. Because some computer applications (such as in astrophysics) involve calculating large numbers, the ability of the 1620’s computer hardware to handle long numbers was advantageous. The first application of Intel’s 4004 was performing decimal arithmetic for a calculator, and given the similarities of the tasks tackled by these two computer architectures, it’s understandable that some of the 1620’s features leaked into our 4004 design.

In 1964, I joined Fairchild Semiconductor in Mountain View, California, as a business programmer—first on the 1620, then the IBM 1401, which was an extremely successful commercial computer. The 1401 was designed for good character handling capability and was outfitted with fast printers and magnetic tape drives. A couple of years later, Fairchild bought its first IBM 360/30. It had firmware microcode stored in ROM that let it emulate both the 1620 and 1401. Interpreting another computer’s instruction set permitted software compatibility and made it easier to migrate from one computer generation to another. At this point, I was introduced to the concept of ROM-based control programs—another feature we used later in the first microcomputer.

Work at Intel

In 1968, Robert Noyce and Gordon Moore (of Moore’s law) left Fairchild, the company they had founded, to start Intel, a company focused on the new semiconductor memory market. Although the first Intel memory chips were successfully produced, no market existed for this new type of product and commercial viability didn’t occur for a few more years.

Because Intel was ahead of the market, Noyce looked for ways to get commercial production volume for his idle memory factory. An acquaintance of his in Japan needed some custom chips for a new desktop calculator design (Busicom). Calculators were being sold in unit volumes of the hundreds of thousands, and each calculator needed six or seven custom integrated circuit (IC) chips. Busicom promised substantial sales volume if Intel could design and build the custom chips.

At that time, Ted Hoff was the Intel manager of applications engineering in charge of evaluating Busicom’s custom chip designs. In 1969, I joined him at Intel from Fairchild R&D, where I had been working on the design of a serial decimal computer in the digital system research group. Our Busicom customer, Shima Masatoshi, had done the design partitioning and overall logic design for the calculator’s custom chip set. Shima’s design called for a processor that performed operations on 16-bit numbers—similar to the IBM 1620. However, Shima had no experience with large-scale IC design, and each of his chips needed a large amount of on-chip random wiring that’s expensive and difficult to implement. More pressing, how-
ever, was that Intel had only a couple of chip designers and wasn’t at all equipped to tackle six to seven difficult chip designs.

At the time, Hoff and I were working on some other projects at Intel that used an inexpensive 12-bit minicomputer, the PDP-8, made by Digital Equipment Company.9,10 Hoff knew that the PDP-8 was a simple machine but could be programmed (in Focal) to do complicated decimal floating-point arithmetic calculations. Hoff reckoned that if we could build a small CPU chip, we could program it to do most of the Busicom’s calculator arithmetic and control functions, including getting information in from the keyboard and out to the printer. Furthermore, the control programs we would write to do the calculator functions could be stored in ROM chips. We’d be converting a custom chip project into a memory chip project—which after all was Intel’s primary business.

Shima had done quite a bit of work on his design, so naturally, he resisted our new line of reasoning. (Later, in Shima’s book about his experiences,11 he would liken our rejection of his original design to “having his ship crashing on the rocks.”) Also, all his flow charts for floating-point arithmetic assumed that the computer engine was capable of 16-digit numeric operations, but our 4004 CPU could only operate on a single digit.12 It seemed that we would have to scrap most of his software design if we pursued our design approach for the CPU chip. I was the principal liaison on the Busicom project, and I had an unhappy customer. How could I make our CPU look more like Shima’s original CPU architecture?

Hoff’s 4004 CPU architecture was a simple 4-bit computer, similar to the PDP-8, and operated on a single (4-bit) digit at a time.13 I programmed loops to operate on a field of digits, similar to that of the IBM 1620, but instead of using flags to mark the end of the data, the loops counted to zero. By writing subroutines corresponding to each of Shima’s original CPU instructions, we salvaged most of his floating-point arithmetic flowcharts, using subroutines written with 4004 instructions, rather than executing his code directly.

It was my job to demonstrate how we could achieve various desktop calculator features by programming rather than in hardware. For example, Shima asked me how the printer would print in red when a negative result is calculated. I wrote sample program pieces for scanning the keyboard, displaying data in lights, and running the printer to resolve his doubts.14

Unlike most computers of the day, our calculator control program had to be permanently stored in ROM. The idea of having a control program in ROM wasn’t new. The IBM 360 had many of its advanced features microprogrammed in ROM, but most minicomputers did not use microprogramming or have a ROM for program memory. Typical minicomputer programs ran from core memory, and subroutine return addresses were temporarily stored in core memory. But because you can’t write into ROM, we needed a way of remembering the subroutine return addresses outside the main program memory.

At the same time, the computer maker Burroughs had a line of large computer mainframes designed for programming in ALGOL, which influenced its architecture. The Burroughs machine had a stack (useful for implementing software subroutines), which is different than machines organized with many data registers. Subroutines are important in programming, and one software design style demands partitioning a program into a structure using a large number of subroutines. The IBM 1620 also had a little single-word stack for remembering the return address after a subroutine call instruction.

Hoff and I recalled the stack feature of the IBM 1620 (and the Burroughs computer), and we implemented a four-level stack within the CPU chip, which allowed for nested subroutines (subroutine called from a subroutine) in the 4004.

As we implemented the calculator functions by programming, we looked for ways to reduce the amount of program storage (number of ROM chips) to lower the costs. Based on my experience with the IBM 360, I embellished the 4004 architecture to support virtual machine emulation. I pretended that we had Shima’s original computer instructions for 16-digit arithmetic but interpreted his pseudo-instructions. (As a minor technical detail, the normal subroutine call instruction in the 4004 requires 2 bytes of instruction memory, but the interpretive scheme cut the memory requirement in half, using only a single-byte pseudo-instruction.)

To build an efficient interpreter routine, you need a way to fetch the pseudocode data and branch to the correct subroutine. As a consequence of wanting to support interpreters or emulators, I added two new general-purpose instructions to the 4004 CPU chip: the ability to fetch data from ROM (fetch indirect) and jump to a subroutine (jump indirect).12 With these two new instructions added to the 4004 CPU instruction set, we enhanced the architecture to extend this computer’s applications and programming capability.
Shima did the final programming of the Busicom calculator using his pseudo-instructions while my interpreter directed the program to the correct subroutines. About half of the ROM code implemented the calculator’s control and peripheral functions and the other half did floating-point arithmetic. Most of Hoff’s original architecture remained intact. Intel’s Federico Faggin did all the chip design, circuit design, and layout. Busicom introduced its line of calculators using the custom chips with options such as square root provided by adding another ROM chip onto their printed circuit board. Intel refunded Busicom’s development fee, in exchange for allowing the chips to be sold as a standard product for noncalculator applications. Marketing announced the new MCS-4 chip set consisting of a CPU chip, a ROM, and RAM chip in 1971.15,16

Collaborative innovation

As this account demonstrates, most inventions and developments evolve out of past experiences and are often the product of a unique set of circumstances and needs. In the case of the 4004, Intel was in the memory chip business but needed to take on production of a set of custom chips, and the lack of in-house resources required Ted Hoff to implement the calculator project with a micro-programmed solution using memory chips. For compatibility with my customer’s original design, I programmed a virtual machine and augmented our 4004 CPU design with a couple of instructions for implementing an interpreter. This interpreter scheme reduced the amount of program space needed for the calculator program and eased the customer’s job in using our architecture. Lessons learned from the IBM 1620, IBM 360, and PDP-8 helped us build a successful product for our customer.

The 4004 laid the groundwork for a new type of component—the microprocessor—and paved the way for other Intel processors, such as the 8088 and ultimately the PC. Today, Intel is known as a microprocessor company, not as a memory component company.

Acknowledgments

I thank the Intel staff Marcin Hoff, Masatoshi Shima, Federico Faggin, Les Vadasz, Hank Smith, Robert Noyce, Gordon Moore, and Andrew Grove for their help.

References


Stanley Mazor
stannazor@sbcglobal.net