Computer Engineering
30 Years After the IBM Model 91

The Model 91’s story shows just how far computer architecture has come since 1967. It may also reveal how practitioners must change to ensure continued innovation in computer engineering.

Often in our hurry to create the future in computer engineering, we ignore lessons from past projects. One project that offers interesting insights, the IBM System/360 Model 91, recently celebrated the 30th anniversary of its first shipment. The largest member of the series based on IBM’s most enduring and influential computer architecture, this machine owed much to its predecessors. Together, these machines show us what the state of the art in computer processors was like in the 1960s—when IBM announced the S/360—and illustrate how far computer engineering has come in 30 years. This leads me to some reflections on how this field has changed and needs to change for the future.

PREDECESSORS

In addition to the competition from the CDC 6600, there were three important influences on the Model 91 design.

The first was the IBM 7030 “Stretch” machine, the company’s first large, pipelined, high-speed machine. IBM began the project in the late 1950s and delivered it in the early 1960s. The 7030 embodied several imaginative machine technology and architectural concepts, including what we now know as emitter-coupled logic (ECL) circuits. A fully pipelined system with highly interleaved memory, it used speculation on branch as determined by a bit set in the branch instruction. Despite these innovations, the 7030 was a financial failure because of late delivery as well as disappointing performance, which partially stemmed from the long delay in recovering from a mispredicted branch.
Another important influence on the Model 91 was the IBM 7090 series. This series began in the early 1950s as IBM’s first large electronic computer, the IBM 701; it was partially influenced by John von Neumann and his Princeton machine. Among others, Gene Amdahl and John Cocke modified it into two versions—the IBM 704 and 709—by incorporating index registers, core memory, floating-point instructions, and so on.

As a 7090 designer, I have firsthand knowledge of that machine, an ECL version of the 709. First delivered in 1959, the 7090 was a nonpipelined machine that achieved about five times the performance of the earlier, vacuum-tube-implemented 709. IBM originally forecast making only 20 machines for government procurement; I believe the company eventually sold more than 300. So unlike the 7030, the 7090 was a profitable product. Both projects contributed to the Model 91 in terms of technology and market understanding.

The System 360 instruction set was the third important influence on the Model 91. IBM introduced this ISA in 1964 to unify its product line at the expense of turning its back on at least three successful and established product lines. These lines served the scientific (IBM 7090), small business (IBM 1401), and large business (IBM 7080) markets. The new ISA made the S/360 implementations incompatible with these older machines. So despite widespread emulation, the market niches carved out by these older machines were vulnerable to competition.

**WHAT THE MODEL 91 WAS LIKE**

When IBM introduced the S/360, the Model 90 (as the eventual Model 91 was then called) garnered only a footnote in the 1964 announcement. Unlike several other models that were further along in their engineering effort, the Model 90 consisted of two small study projects. It was not until the latter part of 1964—after CDC announced the 6600—that these two projects came together as a full development program for the Model 91. The machine’s market was clear enough: It consisted of the US Atomic Energy Commission, the National Aeronautics and Space Administration, and the US Department of Defense, as well as DoD contractors and university research institutes. Such a focused group of customers stands in stark contrast to today’s varied market for computer processors.

**ECL circuits**

Its design included ECL circuits, using multitransistor chips mounted on a ceramic substrate. Passive devices were implemented as thin-film printed components on the substrate. Two small substrates were stacked one on top of each other, forming a module about the size of a sugar cube, 1/2 in (1 cm) on each side. The processor’s density was roughly comparable to what we now think of as small-scale integration (two to three circuits per package). We could mount approximately 60 such modules on a daughterboard and place 20 daughterboards on a motherboard that measured about 1 ft². Twenty motherboards formed a frame (about 6 x 6 x 1 ft³), and four frames formed the system’s basic CPU.

The ECL circuits were not slow: Their average delay was a little over a nanosecond. Added to that was a 1.5-ns transit delay because the average distance between logic gates was more than a foot. All interconnections were made by terminated transmission lines (except for a small number of stubbed transmission lines). Designers took great care in developing the signal transmission system, which used a dual impedance of 50 and 90 ohms. The basic 50-ohm line width could be reduced to create a 90-ohm line in the vicinity of loads. In this way, the effective impedance of the loaded 90-ohm line would appear as a 50-ohm line.

The processor had about 120,000 gates—certainly small by today’s standard of many millions. The total delay per gate—including transmission time, loading effects, and transit time—was approximately 5 ns. With 12 stages of logic as the definition for cycle time, this led to 60 ns as the basic CPU cycle. The multiply-divide unit had a 20-ns subcycle to execute an iteration that reduced the 12 partial products to two.

**Cooling and power**

Water-cooled heat exchangers between the motherboards dissipated heat. A motor-driven generator powered the system and isolated it from short power disruptions. Total power consumption was probably a significant fraction of a megawatt! For other features, see the sidebar “Significant Features of the IBM Model 91.”

**Problem with physics**

Sometimes what earns physicists a prize earns engineers nothing but pain. The Model 91’s engineers suffered at least one such pain as we discovered the electromigration effect, which causes aluminum to disintegrate into silicon at very high electric-field densities (over 300,000 amperes/cm²). Fixing this required IBM to redesign the basic logic devices, contend with a significant schedule slip, and write off the value of already fabricated inventory.

To its credit, IBM stuck with the Model 91 despite such setbacks, and the processor had an unexpectedly (in my view) long lifetime. For example, the more evolutionary IBM mainframes did not substantially exceed Model 91’s floating-point performance for more than 15 years after its initial delivery.
REFLECTIONS

In view of the discussion on transmission lines, I could call what follows simply reflections down the line; but I offer these thoughts as computer engineer-}

Business strategy

Until the mid-1980s, IBM based its success on management foresight: the ability to look beyond profit and see future opportunity. It certainly did this in the S/360 era, when it effectively terminated three or four very profitable computer lines to provide a better overall solution for its customers. Managemen
t failed to make similar bold moves in the 1980s; after all, IBM was then enjoying the high profit margin from mainframes. It failed to foresee the shift to client-server architectures and the new pricing model that accompanied them. A lesson to be learned is that a company is in great dan-

Excellence

Companies can have many reasons for building systems to extend the state of the art. It can do so simply }

IBM produced about 12 of the Model 91. The company also sold perhaps twice that number of the Model 195, which was based on the Model 91 design, but had a faster cycle (54 ns) and incorporated a cache. It came on the market in about 1971.

References


Significant Features of the IBM Model 91

Other interesting architectural or organizational features of the Model 91 system included the following:

- It was a deeply pipelined system, as it had no cache. The overall pipeline length was probably 20 stages. As a comparison, the Intel Pentium II pipeline has 11 stages, the Sun UltraSparc III, 14.
- Memory had a 10-cycle access, was fully buffered, and interleaved 32 ways. (The processor could have up to 32 memory accesses pending to reduce latency.)
- The execution units shared a common data bus; it was designed by Bob Tomasulo to allow out-of-order execution of operations. This implementation represented the first use of a dataflow approach to the control of concurrent operations.
- Each of the floating-point arithmetic units were innovative. The floating-point adder had a two-cycle latency and was internally pipelined so that it could accept an operand every cycle. The floating-point multiplier had a three-cycle latency, but it was implemented using a 20-ns subcycle that reduced 12 partial products each subcycle. The divide unit was designed by Bob Goldschmidt using what we now refer to as the Goldschmidt algorithm, a multiplicative series approximation to the reciprocal. When multiplied by the numerator, it gives the quotient. Executing a divide required 11 cycles.
- Model 91 implemented a speculative branch using a branch-loop mode. If the branch was at a target within approximately the last eight instructions, the processor speculated that the target was taken.
- Maintenance features included checking of all data transfers, residue checking for all arithmetic operations, full scan of all registers, full state display of register state and error logging. We devoted probably 15 to 20 percent of the processor system to supporting these maintenance and reliability features.
- Overall performance approaches one cycle per instruction on scientific, loop-oriented code. On heavily branched nonscientific code, the long pipeline takes its toll and performance slips to three or more cycles per instruction.
Computer engineering is a profession in which we apply scientific and mathematical principles to social needs.

Technical management
A processor has a long gestation period—something between one-and-a-half to perhaps three or four years. This requires consistent management commitment and focus on the project. Significant changes in objectives, schedules, and/or staffing almost always bring about disastrous results.

Managing a development project requires several types of understanding and support. It requires understanding of both the technology and the management process itself. Designers require tools—for test support, validation, and CAD—that are integrated and work together seamlessly. Designers also need a process that supports communication and documentation, and sets well-defined project targets. This is simply the essence of good management.

What actually occurs in industry is often the opposite. In particular, I am horrified by two developments especially prevalent in Silicon Valley. One is the concept of what I call "macho hours"—the more hours designers work, the better jobs they must be doing. Indeed, designers are expected to bid against one another in putting in more hours; this supposedly shows that they are better designers.

This problem is coupled to and exacerbated by the fact that technical managers are simply not trained to manage. The unskilled manager, unable to articulate specific project targets, measures projects by hours rather than efficient work. For instance, designers working long hours to make up for, say, a lack of tools, expose projects to errors because they are fatigued. What's the best management here—showing how many hours you're working to get around a poor situation (the lack of tools) or solving the problem?

Management is clearly mistaken if it expects more truly creative ideas by simply extending a designer's hours in the office. I personally have come up with better ideas while pruning roses than I ever would have sitting in some sterile office. Of course, any project requires overtime at some point—perhaps for one or two, even three, months. But to expect designers to endlessly operate at peak efficiency for 15 to 18 hours a day is patently wrong. This is clearly indicative of the lack of maturity that our field has in managing itself.

Reliability
Users want speed and improved cost-performance, but only if it comes with reliable computation. I am dismayed that some PC-based operating systems let user applications crash the system. This is simply a failure to use hardware features such as system state registers and memory protection. I can't imagine why software vendors do this. An errant application ought never bring down either the system or any other applications. Failure to use obvious hardware features to prevent this is just simply a dereliction on the part of the software designer.

Hardware designers are equally to blame when they fail to use simple techniques to ensure reliable computation. Techniques such as scan, design for testability, and error checking (detection and/or correction of the computation) ought to be universally used. These techniques have been well known for more than 30 years. Perhaps the lack of sophistication on the part of today's mass-market users allow shoddy, crashable implementations to survive. Current advertising stresses megahertz and SPECmarks; how about an emphasis on reliability and system robustness (RELmarks, maybe)?

Technology changes
Technology changes unevenly—the speed of light remains constant, but cycle times have decreased perhaps 20 times over 30 years. Memory costs decreased by 10,000 as densities increased more than a million times. For disks, advances in magnetic storage have reduced cost per byte by a factor of a million, yet disk access has probably changed by less than a factor of 10.

The net result of this uneven change in technology parameters means that approaches that seemed "outrageous" or a negative result in one context, may become quite reasonable in another. I recall working with Gary Tjaden in 1970 to show that multiple-instruction issue (superscalar) machines would probably be limited to performing at about 1.8 instructions per cycle. At that time, we took this—given the implementation complexity—as a negative result. Today, multiple-instruction issue (of now up to eight instructions per cycle) is an obvious approach to processor implementation.

Computer engineering
Engineering is a profession in which we apply scientific and mathematical principles to social needs. Engineering disciplines accumulate understanding so that advances of systems or structures can provide a long-term social benefit.

Computer engineering is in a peculiar situation in which the underlying technology and user behavior are changing at a rapid and uneven rate. This has made computer engineering a vibrant, fast-paced field, full of opportunity for dynamic entrepreneurs. On the other hand, the same characteristics make it easy to forget important lessons we've already learned. I was greatly amused a few years ago—when
companies were introducing pipelined microprocessors—to learn that RISC technology enabled pipelining. However, we define RISC, the term had only been introduced a few years earlier. That this could be responsible for pipelining, which has existed for more than 20 years, illustrates the amnesia present in computer engineering.

I am also fearful that we have unlearned several important lessons in reliability, maintainability, testability, diagnosis, and overall integrity of a commercial processor design. As our user base becomes increasingly sophisticated, I believe that it will be important to relearn the lessons of the past decades. But do we have to relearn them from scratch?

Communicating ideas

When civil engineers complete a new structure, they publish almost all the interesting details of that structure. The same is true of most mechanical innovations in aeronautic and automotive design. When it comes to commercial microprocessors, however, there is a noticeable reticence about publishing details. One of the Model 91's great contributions was simply that its designers published all the relevant details in a timely way. This wasn't easy; I remember having to meet with IBM's chairman of the board to obtain permission. In the same spirit, Jim Thornton also published a detailed description of the CDC 6600.1

In today's environment, I applaud the efforts of Microprocessor Reports and IEEE Micro to at least begin to present some semblance of information on new commercial processors. But the larger lesson is that computer engineers ought to consider it their responsibility to publish the details of a new product in a reasonably timely way. It is a source of recognition for both the designers and the company. Of course, I am not advocating that a company lose its intellectual property; timely publication would have to allow for satisfaction of patent requirements.

Another need is for organized access to information. Some technical societies have overly refined their information forums by publishing too many specialized transactions and journals. In this case, however, technology is coming to our aid as we move to electronic distribution of technical information.

A major goal of computer engineering in the next 30 years should be to accumulate a comprehensive collection of computer engineering literature. Next, we need to ensure that every computer engineer uses the existing publications and source material. Finally—perhaps most importantly—computer engineers need to acknowledge the use of prior references. It is depressing to see engineers waste work rediscovering an old idea. It is even more discouraging to see them use old ideas without acknowledgment when they are obviously aware of the source.

I can sum up the goal for computer engineering in one word: integrity. Integrity in the products we build and in the way we build products—integrity in the design process itself. 

-----------------------------------------------------------------------------------

Acknowledgments

I dedicate these remarks to the designers of the IBM S/360 Model 91 machine and its derivative, the S/360 Model 195, and also to the designers of the CDC 6600 and its derivative, the CDC 7600.

References


Michael J. Flynn is a professor of electrical engineering at Stanford University. Earlier, he worked at IBM as a designer and design manager for the IBM 7090, 7094/II, and S/360 Model 91 CPU. Flynn received a PhD from Purdue University. He is a member of the IEEE Computer Society and the ACM.

Contact the author at the Computer Architecture and Arithmetic Group, Stanford University, Gates Computer Science Bldg., Rm. 334, 353 Serra Mall, Stanford, CA 94305; flynn@ee.stanford.edu.

April 1998