Some Reflections on Computer Engineering: 30 Years after the IBM System 360 Model 91

Michael J. Flynn

30th International Symposium on Microarchitecture
Durham, NC

Invited Talk, 2 December 1997
(Following is an annotated and edited version of the presented talk.)

I am very pleased to address the MICRO conference on the occasion of its 30th anniversary, and to speak to you about the IBM System 360 (S/360) Model 91, which also celebrates the 30th anniversary of its first shipment. I will describe first the background to the project, then the machine itself; in the second half of my talk I will share some reflections on the changes in computer engineering over this time.

Before going further, I would like to dedicate these remarks to the designers of the 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. The CDC 6600 machine was for the S/360 Model 91 a very important competitive machine. I believe this competition pushed both projects to achieve excellence within the context of the technology of the day.

The Background

In addition to the CDC 6600 [4], there were three important precursors to the Model 91. The IBM 7030 [3], or the Stretch machine, was IBM's first large pipelined high speed machine. It was begun in the late 1950's and delivered in the early 1960's. It developed a number of imaginative machine technology and architectural concepts, including what we now know as ECL circuit technology. It was a fully pipelined system with highly interleaved memory. It used speculation on branch, as determined by a bit set in the branch instruction. However, to IBM and the field it was regarded as a failure because of late delivery and disappointing performance. The long delay required to recover from a misguessed branch prediction was at least in part to blame for the performance shortfall.

Another important predecessor to the S/360 Model 91 was the IBM 7090 series. This series began in the early 1950's as IBM's first computer, the IBM 701. It was influenced in part by John von Neumann and the Princeton machine which he had successfully designed. It was modified (by Gene

Amdahl and John Cocke) in two versions: the IBM 704 and IBM 709, as they incorporated index registers, core memory, floating point instructions, etc. I saw this machine firsthand as a designer of the IBM 7090, which was an ECL transistorized version of the 709. Forst delivered in 1959, the 7090 was a nonpipelined machine that achieved about five times the performance of the earlier, tube-implemented 709. The 7090 was an extraordinary success for IBM. It was originally estimated that only 20 copies would be made as part of a government procurement. Before it ended production, I believe more than 300 copies were made.

The System 360 instruction set [2] was the third important influence on the Model 91. This instruction set architecture was introduced in 1963, to unify IBM's product line. In this integrated approach, IBM turned its back on at least three very successful and profitable lines of computers: the scientific line (IBM 7090), the small business machine line (IBM 1401), and the large business machine line (IBM 7080). Since the new System 360 implementations were not compatible with these older machines (although emulation was widely used), they were vulnerable to competition.

The S/360 Model 91 [1]

When the System 360 line was announced, the S/360 Model 90 (as it was called)[1] was simply a footnote in the announcement in early 1963. Indeed, unlike a number of the other models,[2] which had been well along in the engineering effort, the "Model 90" consisted of two small study projects. It was not until the latter part of 1963, when CDC announced the 6600, that these two projects were brought together as a full development program to realize the Model 91. The market for the machine was clear enough; it consisted of the AEC, NASA and DOD labs, as well as DOD contractors and several university based research institutes. The machine itself was implemented in ECL circuits, using multitransistor chips mounted on an aluminum 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 which was about the size of a sugar cube 1/2 inch (1 cm) on a side [1h]. The density was roughly comparable to what we now think of as SSI (two to three circuits per package). Approximately 60 such modules could be mounted on a daughterboard and twenty daughterboards could be plugged in to a motherboard (about 1 sq. ft. or 0.1 sq. meter). Twenty motherboards formed a frame (about $6 \times 6 \times 1$ feet, or $2 \times 2 \times .3$ meters), and four frames formed the basic CPU for the system [1a].

---

[1] The nomenclature for the "Model 90" is as follows: Model 90—a concept term only; Model 91—used 0.75 μsec memory; Model 92—planned to use 0.5 μsec memory, never built; Model 95—used 0.18 μsec thin film memory, 2 machines built; Model 195—a later machine (c. 1971) with cache.

[2] The other S/360 models were smaller, and ranged from model 20 through models 30, 40, 50, 65, and the model 75.

The ECL circuits were not slow; average delay was little over a nanosecond [1f], but the average distance between logic gates was more than a foot. All interconnections were made by terminated transmission line (except for a small number of stubbed transmission lines). A great deal of care was paid in developing the signal transmission system; for example, a dual impedance system of 50 and 90 ohms was used—the width of the basic 50 ohm line could be reduced and create a 90 ohm line in the vicinity of loads, so that the effective impedance of the loaded 90 ohm line would appear as a 50 ohm line.

The processor had in total about 120,000 gates, certainly small by today's standards. Since each gate had an associated interconnection delay (transmit time plus loading effects) of about three and one half nanoseconds, the total delay per gate was approximately five nanoseconds. With twelve stages of logic as the definition for cycle time, this led to a 60 nanosecond cycle as the basic CPU cycle. The multiply–divide unit had a subcycle of 20 nanoseconds for an iteration for reducing 12 partial products to two.

The processor used water cooled heat exchangers between motherboards for cooling. A motor generator set powered the system (also isolating the system from short power disruptions and allowing for state preservation on power failure). The total power consumption was probably a significant fraction of a megawatt!

Some of the architectural or organizational features of the system[3] that are most interesting include the following:

1. It was a deeply pipelined system, as it had no cache. The overall pipeline length was probably 20 stages [1b].

2. Memory had a 10 cycle access, was fully buffered and interleaved 32 ways [1e].

3. The execution units shared a common data bus, designed by Bob Tomasulo, which allowed out of order execution of operations and represented the first use of a dataflow approach to the control of concurrent operations [1c].

4. Each of the floating-point arithmetic units were innovative [1d]. 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 20ns subcycle, which reduced 12 partial products each subcycle. The divide was designed by Bob Goldschmidt, using what we now refer to as the Goldschmidt algorithm. This is a multiplicative series approximation

---

[3]The CDC 6600 was also quite innovative. It introduced a load–store register based instruction set (3 register specifiers in each instruction), a "scoreboard" to control inter-instrcution dependences and a ten processor multi-threaded peripheral processor ensemble which time shared the ALU [4].

to the reciprocal which when multiplied by the numerator gives the quotient. Divide required 11 cycles.

5. Model 91 implemented a speculative branch using a branch loop mode—if the branch was at a target within approximately the last 8 instructions, the processor speculates that the target is taken [1b].

6. The maintenance features included full checking of all data transfers, residue checking on all arithmetic operations, full scan of all registers, full state display of all register state, together with error logging. Probably 15–20% of the processor system was devoted to the support of these maintenance and reliability features [1a].

The performance overall approaches one CPI on scientific loop oriented code. On heavily branched nonscientific code, the long pipeline takes its toll and performance slips to 3 CPI.

About twenty Model 91s were made,[4] and perhaps an equivalent number of the Model 195. The Model 195 used the basic Model 91 design, but the implementation had a faster cycle (54 ns) and the 195 incorporated a cache. It was available in about 1971.

Sometimes what earns physicists a prize earn an engineer nothing but pain. And indeed, the Model 91 suffered at least one such pain as we discovered the electromigration effect, whereby aluminum starts to disintegrate into silicon at very high electric field densities (over 300,000 amperes per square cm). This required a significant schedule slippage, a redesign of the basic devices, and a writeoff of the value of inventory.

Despite this, the processor had an unexpectedly (in my view) long lifetime, as the more evolutionary IBM mainframes did not substantially exceed its floating point performance for more than 15 years after the initial delivery.

Reflections

I guess in view of the discussion on transmission lines, I could call these simply reflections down the line; but I offer these thoughts as computer engineering continues to evolve.

Business Strategy Model

IBM success, up until the mid-1980's, was based upon management foresight in being able to look beyond profit and see future opportunity. It certainly did this in the era of the System 360, when it effectively terminated 3 or 4 very profitable computer lines to provide a better overall solution for its customers. Management failed to take similar bold moves in the 1980's when

---

[4]I was told that this number included two machines used primarily for COBOL job processing! This was a major feat of salesmanship, as we hadn't even implemented the decimal instructions (they were interpreted).

it enjoyed a high profit margin from the mainframes. It seems to me that a company is in great danger when it is very profitable yet lets this profit blind itself to future opportunities.

Excellence

Building systems to extend the state of the art can be done from many reasons. It can be done for simply market prestige, or it can be done to continually stretch our limits of understanding of technology, reliability, and architecture. "Excellent" projects must sometimes fail. Otherwise, they would not be stretching the limits. Management that fears failure, fears excellence. I notice that IBM abandoned the Model 91 approach to concentrate on evolutionary and safer approaches to processor implementation. I think this was a mistake, akin to its business strategy mistake mentioned previously.

The Technical Management Process

A processor has a long gestation period—something between one and a half, and perhaps three or four years. This requires consistent management commitment and focus to the project. Changes in objectives, in schedules, in 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—CAD tools, test support, validation tools, provided in a integrated, seamless way. Designers equally require process support. This includes discipline in communications, documentation, and especially in well defined specific project targets. This is simply the essence of good management. I am horrified by two developments, especially prevelant in the Silicon Valley. One is the concept of what I call "macho hours"—the more hours a designer works, the better a job he or she must be doing. Indeed, designers are expected to bid against one another in being able to put in more hours than their collegues, to show that they are in fact better designers. The second problem is that technical managers are simply not trained in management. These two items are related. The unskilled manager, unable to articulate specific project targets and sub-targets, measures projects by hours of effort. If a designer is working long hours simply because there is not enough tools, then the project runs the exposure of errors because, as I believe we have learned from the 19th century, errors occur when the mind is fatigued. If management expects more truly creative ideas by simply being present in the office for longer hours, they are clearly mistaken. I personally feel that I have had better ideas pruning roses than I got in some sterile office. Of course, in any project, a moment comes when extra effort is required, and for a period of time (perhaps one or two, even three months) overtime is required; but to imagine that a designer is expected to operate at peak efficiency for 15–18 hours a

day for endless periods of time is patently wrong and clearly indicative of the lack of maturity that our field has in managing itself.

Reliability

A user wants speed and cost–performance, but only if it comes with reliable computations. I am horrified that some PC-based operating systems let the 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 do 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, error checking (detection and/or correction of the computation), and data transfers ought to be universally used. All of the above techniques have been well known for now more than 30 years. Perhaps it is the lack of sophistication[5] on the part of today's mass market user that allows shoddy implementations.

Technology Changes

Technology changes in uneven ways—the speed of light remains unchanged, but cycle times have decreased perhaps 20 times over 30 years. On the other hand, memory costs have decreased by 10,000 as densities have increased by more than a million times. For disks, these costs per byte have been reduced by a factor of a million, yet disk access has probably changed by less than a factor of ten. The net result of this uneven change in technology parameters means that approaches which seem "outrageous" at one time, or as a negative result in one technology context, may be quite reasonable in another. I recall in 1970 working with Gary Tjaden [5] and showing that multiple instruction issue machines (superscalar) would probably be limited in performance to about 1.8 IPC. At that time, we took this—given the implementation complexity—as a negative result. Today, multiple instruction issue is the obvious approach to processor implementation.

Computer Engineering

Engineering is a profession whereby we apply scientific and mathematical principles to social need. Engineering disciplines accumulate understanding so that advances of systems or structures can provide a long term additional social benefit.

---

[5]Advertisement and product literature stress Mhz and SPECMARKS—how about an emphasis on reliability and system robustness? —RELMARKS?

Computer engineering is in a peculiar situation, where the underlying technology and user behavior are changing at a rapid and uneven rate. It becomes easy to forget important lessons that have already been learned in the past. I was greatly amused a few years ago, when pipelined microprocessors were being introduced, to learn that pipelining was enabled by RISC technology. However one defines RISC technology, it had only been introduced a few years earlier. That this could be responsible for pipelining which had existed for more than 20 years earlier, illustrates the amnesia that is present in the computer engineering field. I am also fearful that we have unlearned a number of important lessons relating to reliability, maintainability, testability, diagnosis, and overall integrity of a processor product 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 really have to relearn them from scratch?

Communicating Ideas

In civil engineering, when a new structure is complete, almost all interesting details of that structure are published. The same thing is true of most mechanical innovation (aircraft, automobiles). When it comes to processor products, however, there is a noticable reticence to publish details. One of the great contributions of the Model 91 was simply that all relevant details were published in a timely way. I can't say the this was easy; I remember in order to achieve this I had to go down to world headquarters and have a meeting with the chairman of the board of IBM. Subsequently, Jim Tornton published a very interesting book describing the CDC 6600 [4].

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 processor products. But the larger lesson is that it ought to be the responsibility of each computer engineer to at least see that the details of a new product are published in a reasonably timely way. It is a source of recognition for both the individuals involved, the designers, and the company. Of course I am not advocating the loss of intellectual property for the company; I believe that timely publication is simply publication after the necessary patent requirements have been satisfied.

There is another need, and it is a need for organized access to information. I believe that some of our technical societies have overly refined the forums by publishing too many specialized transactions and journals. In this case, however, the technology is coming to our aid as we move to an electronic distribution of technical information.

Overall, looking forward to the next 30 years, we must achieve excellence within the computer database. Next, we need to ensure that every computer engineer makes every effort to use the database; to use the existing publications and source material. Finally, perhaps most importantly, is the need to acknowledge the use of prior references. It is very depressing to

see engineers rediscover an old idea, but it is even more discouraging to see engineers use old ideas when obviously they are aware of the source and fail to acknowledge it.

Integrity

I think 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.

References

[1] The IBM System/360 Model 91 (issue). *IBM Journal of Research and Development* 11(1), January 1967.

(a) M. Flynn and P. Low; Some Remarks on System Development.

(b) D. Anderson, F. Sparacio and R. Tomasulo; Machine Philosophy and Instruction Handling.

(c) R. Tomasulo; An Efficient Algorithm for Exploiting Multiple Arithmetic Units.

(d) S. Anderson, J. Earle, R. Goldschmidt, and D. Powers; Floating-Point Execution Unit.

(e) L. Borland, G. Granito, A. Marcotte, B. Messina, and J. Smith; Storage System.

(f) J. Langdon and E. Van Derveer; Design of a High-Speed Transistor for the ASLT Current Switch.

(g) R. Sechler, A. Strube and J. Turnbull; ASLT Circuit Design.

(h) R. Lloyd; ASLT: An Extension of Hybrid Miniaturization Techniques.

[2] G. M. Amdahl, G. A. Blaauw, and F. P. Brooks, Jr. Architecture of the IBM System/360. *IBM Journal of Research and Development* 8(2):87–101, April 1964.

[3] W. Buchholz. *Planning a Computer System*. McGraw-Hill, New York, 1962.

[4] J. E. Thornton. *Design of a Computer: The Control Data 6600*. Scott, Foresman and Co., Glenview, IL, 1970.

[5] G. S. Tjaden and M. J. Flynn. Detection and parallel execution of independent instructions. *IEEE Transactions on Computers*, C-19:889–895, October 1970.