IBM-ACS: Reminiscences and Lessons Learned From a 1960's Supercomputer Project*

Lynn Conway

Professor of Electrical Engineering and Computer Science, Emerita
University of Michigan, Ann Arbor, MI 48109-2110
lynn@ieee.org

Abstract. This paper contains reminiscences of my work as a young engineer at IBM-Advanced Computing Systems. I met my colleague Brian Randell during a particularly exciting time there – a time that shaped our later careers in very interesting ways. This paper reflects on those long-ago experiences and the many lessons learned back then. I'm hoping that other ACS veterans will share their memories with us too, and that together we can build ever-clearer images of those heady days.

Keywords: IBM, Advanced Computing Systems, supercomputer, computer architecture, system design, project dynamics, design process design, multi-level simulation, superscalar, instruction level parallelism, multiple out-of-order dynamic instruction scheduling, Xerox Palo Alto Research Center, VLSI design.

1 Introduction

I was hired by IBM Research right out of graduate school, and soon joined what would become the IBM Advanced Computing Systems project just as it was forming in 1965. In these reflections, I'd like to share glimpses of that amazing project from my perspective as a young impressionable engineer at the time.

It was a golden era in computer research, a time when fundamental breakthroughs were being made across a wide front. The well-distilled and highly codified results of that and subsequent work, as contained in today's modern textbooks, give no clue as to how they came to be. Lost in those texts is all the excitement, the challenge, the confusion, the camaraderie, the chaos and the fun – the feeling of what it was really like to be there – at that frontier, at that time.

This is my first foray into writing down this story, in hopes of bringing some of that thrill back to life. In doing so, I have made use of the extensive ACS historical files compiled by Prof. Mark Smotherman of Clemson University [1], along with my own archive of ACS documents [2]. Of necessity, much relies on my memory. Some reflections also involve insights gained from recently uncovered documents, and will need revision and expansion as further evidence arises.

One of my goals is to encourage other ACS vets to also share their reminiscences, so that we can better illuminate and widen our collective understanding of the large-scale *project dynamics* of that era. Toward that end, and even though very much a work in progress, I'd like to convey the gist of the story to you now – as best I can.

* L. Conway, "IBM-ACS: Reminiscences and Lessons Learned from a 1960's Supercomputer Project"; in: *C. B. Jones, J. L. Lloyd, (Eds.), Dependable and Historic Computing: Essays Dedicated to Brian Randell on the Occasion of his 75th Birthday*, Springer-Verlag, 2011, pp.185-224. Original publication available at: http://www.springerlink.com/content/u8952k538h128472/

As often happens in life, my most exciting times at ACS were triggered by fateful accidents. For example, while building an architectural timing simulator for the new machine, I overheard what I thought was a design challenge. Unaware that it was actually an open research question, I went off and solved it by inventing a hardware subsystem that enabled the dynamic issuance of multiple out-of-order (OOO) instructions during each machine cycle. We called it “dynamic instruction scheduling” (DIS).

That invention led to a roller-coaster ride of ensuing events. In a rather confrontational internal memorandum entitled “Clean Machine Design”, my ACS colleague Brian Randell proposed that my invention, along with a related innovation he had made in branching methods, become the basis for a major machine redesign [3]. Brian’s advice went unheeded, however, and given an IBM design culture that tended towards energetic but ad-hoc local optimizations, the DIS scheme was instead inserted as an incremental add-on within the existing architecture.

Even so, we now know that the resulting ACS-1 machine would likely have been the premier supercomputer of its day, had it been successfully built [4]. Full of pioneering innovations in machine architecture, it would have been implemented in the most advanced integrated circuitry of the day with 1.6ns gate delays and an approximate 12ns cycle time. However, not being S/360 compatible, it was killed-off in the inevitable political shootout – a type of showdown long common in technology firms.

When the project was eventually canceled, IBM threw out “the baby with the bathwater”, at least in part I now surmise, because many principal players were unaware of the significance of DIS. This is not too surprising given the way the innovation had been exploited in the first place, i.e., as a minor add-on.

It wasn’t long before the project was declared a “failure” by B. O. Evans, the incoming President of IBM’s System Development Division [5], and there was little follow-up on what had happened. Outside of IBM, multi-OOO DIS eventually emerged as a foundational concept in superscalar computer architecture [6, 7], but that is another story.

As these rather stimulating events unfolded, I learned many lessons. I learned about the need for targeted computer design tools to cope with design complexity, about carefully defining levels of design abstraction, about the need to cohere design efforts across all levels of abstraction, about ranging widely across levels of abstraction when seeking innovative system optimizations, about principled versus ad-hoc design and the need for principled design of the design process itself. The collective significance of these multi-faceted lessons became clear to me ten years later, when I worked on another hard research challenge: the design of VLSI design methods.

What follows is a very personal account of those lessons learned, so long ago at ACS.

2 The Adventure Begins

As a young student, I found myself at the beginning of a great technological revolution: digital computing in the early 1960s. I was also at the right place, Columbia University’s School of Engineering and Applied Science in New York City, with its close ties to IBM, then a leading force in the new industry.

Along with delving into every relevant course in mathematics, physics, electrical engineering and computing at Columbia, I also did an independent study with a new faculty member, Herb Schorr, just prior to his joining IBM Research.

Herb suggested that I study some recent research papers by M. V. Wilkes of Cambridge University. The topic: Wilkes’ list processing language "WISP" and his explorations into self-compiling compilers and machine-independent compilation using that language [8].

To test my understanding, I decided to implement a self-compiling compiler for a basic version of WISP on the IBM 1620 computer at Columbia's engineering school. You had to have known the 1620 up-close to realize what an arcane effort this was. However, working late into the evenings I quickly got the little compiler working and explored its use to bootstrap more sophisticated versions of the language.

Fortunately, Leeson's and Dimitry's marvelous book on the 1620 had just come out [9]. Using it along the way, I learned the 1620 inside and out, hardware, software, I/O, everything. I was thrilled by the intellectual sensation this experience provided, becoming totally hooked on creating and making things work in the world of computers, and at any and all levels of design.

3 Joining IBM and the ACS Project

My explorations at Columbia must have made a good impression: I was quickly recruited by IBM Research in Yorktown Heights, N.Y. Right from the start I felt great pride and empowerment upon joining the staff there. The stunning new facility had been designed by famed architect Eero Saarinen. It had a sweeping curved wall of glass at the front, and walkways along it gave close connection with the greenery outdoors. It was unlike any building I'd ever seen. The whole atmosphere made employees feel truly special.

T. J. Watson, Jr., then President and CEO of IBM, had long been interested in supercomputers. He'd started a project called "Stretch" [10] back in the late 1950's, hoping it would be a sort of flagship effort for the company. But when the project rolled out in 1961, it ran at only 60 percent of its promised performance. The quick-tempered Watson became enraged [11]. Scolding the engineers involved, he called a very public press conference and reduced the machine's price from $13.5 million to $8 million to reflect the lesser performance, effectively dooming the project [11].

Watson's temper flared again, as he himself later told the story, when he learned of the CDC 6600:

> *In August 1963 came their bombshell: a machine called the 6600 that everyone recognized as a triumph of engineering. For seven million dollars it delivered three times the power of Stretch. Word of the machine made me furious, because I thought the distinction of building the world's fastest computer should belong to IBM – T. J. Watson, Jr., 1990 [11]*

In early 1965, unaware of the exciting events about to swirl around the laboratory, I was assigned to "Project Y" – an exploratory project in supercomputer architecture in Jack Bertram's Experimental Computers and Programming department. Herb Schorr and John Cocke (of Stretch fame) were guiding much of the exploration; my assignment was to work on a timing simulator to enable performance studies of various design alternatives.

In a dramatic decision in May 1965, T. J. Watson, Jr., charged the Project Y group to "go for broke" to create the world's most powerful scientific computer [1] – exploiting advanced computer architecture and compilation methods and the latest in high-performance ECL integrated circuitry. It was his personal, powerful reaction to the CDC6600. Thus was launched a highly proprietary supercomputer project staffed with pre-eminent IBM computing experts of the time. Named "Advanced Computing Systems" (ACS), the project was so secretive that it remained largely unknown, even within the company.

Herb Schorr led the new ACS architecture department, where I reported to Don Rozenberg who reported to Herb. I was tasked with building an architectural simulation model of the design. Robert Riekert and Don Rosenberg had recently developed a simulation technique, similar to but far faster than SIMSCRIPT, by crafting FORTRAN utility routines that managed cycle-to-cycle machine state changes as specified by the hardware controls [12].

I helped tune-up the simulation methods [12] to create the software environment for the "main processing module" (MPM) timing simulator, as it became known [13]. As part of that work, I attended many exciting architecture team meetings: As ideas swirled around and agreements were made on various details, I captured the evolving design and began simulation runs to debug and study the MPM's behavior.

Given the relative complexity of the machine, especially the control logic, it wasn't long before the architecture team began using the MPM timing simulator as our evolving detailed description of critical portions of the machine's micro-architecture. By confirming that each design update ran correctly, we also avoided nuisance-type system-level design bugs (those in state-machine controllers, for example).

The initial architecture for what was christened the ACS-1 was presented at a pivotal high-level meeting at Arden House in August 1965 [1] (which I didn't attend). By then the design included cache memory, novel instruction pre-fetching and decoding hardware, an innovative instruction set and branch hardware for anticipating and minimizing branch disruptions in instruction flow, and multiple pipelined functional units for performing arithmetic-logic-unit (ALU) operations.

However, there was a bottleneck in instruction issuance, for ALU functional units could stand idle as stalled instructions awaited results. The problem had been somewhat alleviated by splitting the incoming instructions into two separate, parallel streams, one of index instructions and the other of floating point instructions, and re-synchronizing the two streams at branch points – a scheme whose origins lay in the earlier "Stretch" supercomputer [10].

Each of these two streams had its own instruction register and secondary register, thus more than one instruction could sometimes be issued during a machine cycle if operands were available, including the issuance of a secondary-register instruction around a stalled primary-register instruction if no source-destination register-conflicts were detected.

Such an issuance scheme was somewhat ad-hoc, being non-extensible, and also made interrupt handling problematical. In effect, it replaced one machine with two, each running only one type of instruction, but was not extensible to more streams, and thus not extensible to higher-level parallelism. Nor were there apparent ways to extend the secondary instruction registers to tertiary ones within the required seven levels of logic. Even so, the resulting two-stream machine was well suited for scientific computing where the two streams only indirectly interacted with each other – as when re-synchonizing upon branching. But although speedup was provided by the hardware over a single stream machine, long stalls could still occur during critical floating point operations.

Gene Amdahl, already famous inside IBM for his work on System 360, along with other prominent computer architects of the day, presumed that no single-stream architecture could be found that issued (on average) more than one instruction per machine cycle. John Cocke questioned this presumption [14], but no way had been found around that bottleneck within single streams. The two-stream approach provided an ad-hoc way to push against this boundary, but not step through it.

4 The Invention of DIS

During one of the architecture meetings at Yorktown Heights, John Cocke raised a question along the line of "why can't we issue more than one instruction per cycle, on average?" John often spoke his mind that way, but this time he really got my attention.

Being right out of graduate school, I had the benefit of lots and lots of recent coursework. However, I had no knowledge at all of the earlier IBM work on Stretch, of the ongoing supercomputer work at CDC, or of the work on the IBM Mod 91. Unaware of the ACS-1's origins in Stretch, I assumed that the Arden House architecture had arisen from scratch during the early Project Y architecture meetings. I also did not know about Stretch's failure to meet IBM's expectations [11], or that John Cocke, who had contributed to Stretch's architecture, might have felt residual concerns from that experience. However, I did pick up on a concern that the machine's issuance rate on real code might not meet expectations.

Clueless that the instruction-issuance problem was an open research question, I though of it as a really cool design challenge and obsessed over it for more than a month. I explored various ways to represent and issue instructions, mentally juggling all aspects of the problem simultaneously – everything from mathematical abstractions, to architectural structures, to circuit-level implementations, but to no avail.

I had a good background for doing this sort of exploration, for I had recently studied and read widely in number theory, probability theory, queuing theory, switching theory, logic design, and pulse and digital electronic circuit design. I had also followed recent work in computer arithmetic-logic-unit (ALU) design, work in which rather broad-gauge innovations had been made that cut-across abstraction levels from algorithms to architecture to logic design to circuitry. Although such ALU work for pipelined systems was by then becoming somewhat classical and well-understood, it left meta-level clues on how multi-level system design might be done in other architectural areas.

I was particularly fortunate to have studied Caldwell's book on *Switching Theory and Logic Design* [15], which gave extensive coverage to the earlier contact-relay switching circuits – including topics such as symmetric functions and their implementation in arrays of contact-relay switches. Those readings had opened my eyes to a long-lost art in which important 'spatially-arrayed logic-algorithms' (such as those that implemented tally functions, for example) could be directly mapped almost by inspection into simple, elegant switch arrays.

Right in the middle of all this, I got the news that the ACS project was transferring to the System Development Division (SDD) and moving out west to a building on Kifer Road in Sunnyvale California – and that I was invited to go along as a member of the architecture team. This was totally thrilling news, and the excitement surrounding the move further stimulated my thinking.

Shortly after the move to California in September 1965, it suddenly beamed down to me: by holding pending instructions in a queue, and representing the identifying numbers of source and destination registers and functional units in unary form rather than in binary, I determined that it would be possible to scan the queue, resolve dependencies, and *dynamically issue multiple out-of-order (OOO) instructions* each machine cycle, even when various entries were stalled awaiting results. (In today's computer architecture terminology, such a queue would be known a *centralized* instruction *window* [6].) It was so simple and elegant that I'd converged on the idea while struggling to visualize and simulate alternative functionalities in my head.

One can visualize and mentally model such a subsystem as taking a first-in-first-out (FIFO) queue and making it into a "first-in dynamic multiple-out-of-order out" (FIDMOOOO) queue that holds the pending instructions – decoding each instruction's source and destination register numbers into unary, instead of binary. Multiple cross-comparisons to detect source-destination conflicts among instructions could then implemented by using simple logical ORs up and down the columns of decoded register-position bits of instructions in the queue.

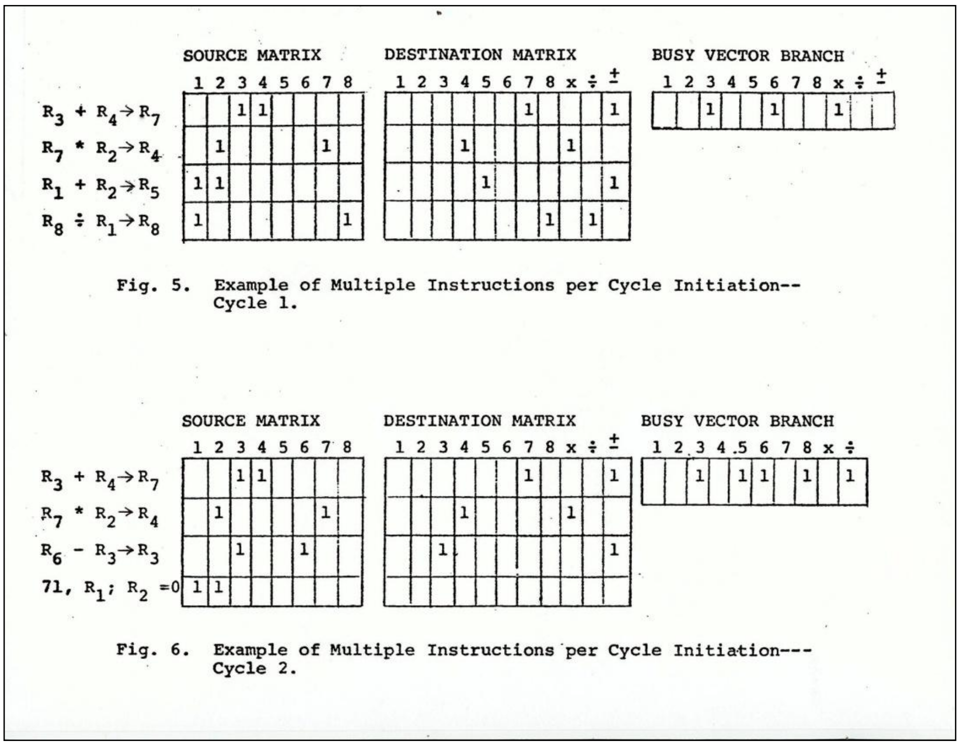
Thus the scheme involved not only mathematical and micro-architectural conceptions, but also utilized tricks at the logic and circuit levels by using arrays of ACS high-speed integrated circuits – exploiting ECL 'wired-OR' connections to enable the queue-column scans to be completed within the seven levels of logic-circuitry machine-cycle time constraint.

At first I could hardly believe it all worked, so I carefully diagrammed it and checked it over and over and over again. Once convinced that it might really work, I showed it to Herb Schorr and Don Rozenberg. Herb thought it looked pretty interesting, and suggested that I give a presentation about it to ACS staff members.

I'm not sure who all attended my presentation, but the audience included a number of the key folks at Kifer Road at the time (this was in late October or early November 1965). Although I was quite junior and very shy at the time, my presentation seemed to go well. Apparently neither John Cocke nor Ed Sussenguth had moved out to California by then, and in any event were not in attendance. Unfortunately, I never had a chance to explain DIS to them, either then or later.

Brian Randell of the original Project Y group had come west with ACS, and he attended my initial presentation. Brian got very excited about the idea and coined a perfect name for the scheme: *Dynamic Instruction Scheduling* (DIS). Before long, Herb Schorr tasked me and Brian to write a tutorial on the functionality of the new method – with help from Don Rozenberg and Don Senzig – so as to enable others to better understand it.

The DIS tutorial included examples of how to handle branching, indirect addressing and interleaved memory accesses during dynamic multi-OOO scheduling operations, in order to reveal the generality of the DIS subsystem and its ease of insertion in among various micro-architectures. We completed the tutorial in February 1966 and circulated it widely in the ACS architecture and engineering departments. It has survived to this day [16].



SOURCE MATRIX

	1	2	3	4	5	6	7	8
$R_3 + R_4 \rightarrow R_7$			1	1				
$R_7 * R_2 \rightarrow R_4$		1					1	
$R_1 + R_2 \rightarrow R_5$	1	1						
$R_8 \div R_1 \rightarrow R_8$	1							1

DESTINATION MATRIX

1	2	3	4	5	6	7	8	x	÷	±
						1				1
			1					1		
				1						1
							1		1	

BUSY VECTOR BRANCH

1	2	3	4	5	6	7	8	x	÷	±
		1			1			1		

Fig. 5. Example of Multiple Instructions per Cycle Initiation-- Cycle 1.

SOURCE MATRIX

	1	2	3	4	5	6	7	8
$R_3 + R_4 \rightarrow R_7$			1	1				
$R_7 * R_2 \rightarrow R_4$		1					1	
$R_6 - R_3 \rightarrow R_3$			1			1		
71, R_1; R_2 =0	1	1						

DESTINATION MATRIX

1	2	3	4	5	6	7	8	x	÷	±
						1				1
			1					1		
		1								1

BUSY VECTOR BRANCH

1	2	3	4	5	6	7	8	x	÷
		1		1	1		1		1

Fig. 6. Example of Multiple Instructions per Cycle Initiation--- Cycle 2.

Fig. 1. Figs. 5 and 6 from the DIS tutorial of February 23, 1966 [16]

During that work, Brian innovated a clever and rather counter-intuitive branching method (which we now call *Prioritised Conditional Branching*) that dovetailed well with multi-OOO DIS instruction-issuance. In architectures where "branch anticipation" is not supported there will be no instructions in the queues below a branch instruction. Brian proposed in such architectures to simply give priority to the latest executable instruction in the queue, despite the strange (though valid) effects this would have on instruction sequencing. (This strategy would of course need an extension if branch outcomes were to be anticipated.)

In parallel with all that, I began interacting with Charlie Freiman in architecture (an expert in pipelined ALU architecture and logic design) and with Merle Homan in engineering (a top logic designer) on the logic design of the "contender stacks" (as the engineering group began calling the two DIS queues). After some effort, Merle and I finally confirmed that everything would work within the seven levels of ACS logic circuitry, up to floating point issuance rates of 3 instructions per cycle out of an 8-deep instruction queue.

5 An Unspoken Challenge, Unspokenly Resolved

When DIS first gelled in my mind, I saw it as a very general instruction-issuance method that might possibly have wide applicability. I even hoped that it might somehow be used at ACS. That hope quickly proved naïve, however, for the idea presented an unspoken challenge to the team: it threatened to destabilize the current machine architecture. After all, why use the somewhat ad-hoc, two-stream, Stretch-like 'Arden House' architecture, if it could be replaced by a simple, elegant, parametrically variable, higher-performance single-stream architecture? Immersed in a sea of powerful personalities, I began to worry about how the top architects would react to my proposing DIS in the first place.

That concern evaporated when Herb tasked me to put two DIS queues into the machine, one in each instruction stream, and to run simulations to measure the performance improvement. A man of few words, he mentioned the request in passing in the hallway, just outside my office. Herb's approach compounded the performance boosts provided by both methods, although at a cost of complexity. In any event, DIS was in the machine! This really made my day, and I was in the clouds for a long time afterwards.

Thus it was that we 'kept' the Arden House architecture: by replacing the two instruction-register pairs with two separate DIS multi-OOO instruction-issuance queues. This ad-hoc incremental update did not impact the ACS instruction set architecture (ISA) [17] and it also kept John Cocke's Stretch-like two instruction streams in place, along with a pioneering branching scheme that Herb Schorr, Ed Sussenguth and Dick Arnold had innovated (using a *branch history table*) to well exploit the instruction set's separate Branch and Exit instructions. Any subliminal 'NIH' problems were thus also finessed.

I assumed that Herb had, over time, fully coordinated this decision with other the key folks ACS, including John Cocke, Ed Sussenguth and Dick Arnold, and I began the intensive process of putting the new hardware into the simulator and making runs to evaluate its performance.

The result: DIS provided a sort of 'turbocharger' for pushing more instructions through a processor during each machine cycle than would otherwise have been possible. Although huge regular arrays of ECL circuits were required to implement the two 'turbochargers' for the ACS-1's two instructions streams (i.e., a significant fraction of the main processor's total circuitry), the DIS scheme was simple and elegant in both function and structure; later simulation runs showed that it roughly doubled the machine's performance over the Arden House version.

6 Brian Randell's Challenge: "Clean Machine Design"

In early summer 1966, Brian revisited Herb's DIS-insertion decision by writing a blockbuster memorandum entitled "Clean Machine Design" and sending it to all the key people at ACS [3]. In it he argued against the additive accumulation of ad-hoc sub-optimizations in the ACS-1, and for a more principled approach to its design. In particular, my recollection is that he argued that the two-stream Stretch-based Arden House design and its branching mechanisms should be shelved, and replaced with a single-stream DIS-based design and his associated branching mechanism. (Unfortunately, it appears that no copies of this memorandum have survived.)

My concern about the NIH aspects of DIS re-emerged, but to my amazement there were no open discussions about Brian's memo, no reactions, nothing. Everything went on as if nothing had happened.

Instead, it wasn't long before Brian left ACS and returned to IBM Research at Yorktown Heights, leaving his colleagues with the predictions that (i) the project would never be able to convince IBM's main East Coast Laboratories that there is a worthwhile advantage to having a non-S/360-compatible word length and instruction set, so would be forced to switch to producing a S/360-compatible design, and (ii) the project would then be killed.

I began to wonder whether the key folks had understood what Brian was talking about. Did they think his "Clean Machine Design" memo was just an academic argument on how to run a project, rather than about our real situation? Furthermore, how many of the key architects had even studied the DIS tutorial? Was it possible that some didn't even know what DIS was? Or, if they did know what it was, did they perhaps think "Well, it's already in the machine, isn't it?" There was no way to answer such questions back then, and they have mostly remained a mystery to me until now.

I stumbled into another inexplicable situation while exploring the patenting of DIS with an IBM attorney named Vincent Cleary, shortly after we filed an invention disclosure entitled "Instruction Sequencing" on the idea on January 27, 1966 [18]. Unfortunately, DIS was such a novel high-level system architectural concept, and its logic design implementations were so regular in structure, that the attorney didn't see any way to protect it. I vividly recall him one morning telling me that DIS was a "software invention" and thus was not patentable.

I now suspect that Cleary had confused the DIS queues either with abstract mathematical matrices, or with 'programmable logic' of the sort IBM had used for years in its tabulating equipment. If the latter, he might have thought that the entries in the queues were preprogrammed control codes rather than transient instruction-constraint data used for controlling instruction issuance. Whatever the reason, he couldn't "find" a hardware invention and the file was closed on September 27, 1966. This was an extremely disturbing experience, for I was convinced that DIS was a fundamental architectural invention that would eventually have real impact.

By the end of 1966, the work on the MPM simulator began demonstrating the power of DIS, and the engineering group had made considerable progress on its implementation. I believe it was then Charlie Freiman who recommended that we file another invention disclosure – and this time title it "Multiple Instruction Issuing System." It was witnessed by Charlie, and filed with IBM San Jose Patent Operations on January 20, 1967 [18]. I don't recall having any interactions with the attorneys after that filing. We simply waited and waited to hear back, finally getting the news on November 22, 1967:

> *At a meeting of November 17, 1967, it was decided to close the subject disclosure. The Thornton et al patent, pertinent to the CDC6600, appeared to meet the basic points of novelty in your disclosure.*
>
> – Bernard A. Meany, Patent Attorney

It turns out that DIS's most relevant predecessor was the CDC 6600 scoreboard, which issued instructions out-of-order but only at a sustained rate of one per machine cycle. I knew nothing about the 6600 at the time. Brian had learned about the 6600 when the first attorney mentioned it to him; he'd referenced it in the tutorial, but we never discussed it.

Meany sent us copies of the Thornton patent in December, 1967 [19]. Before then I had no clue what a "scoreboard" was, and from the patent I couldn't understand how it issued multiple instructions per cycle. In hindsight, it now seems pretty clear that Meany did not understand the *dynamic-multiple-out-of-order* capability of DIS – which went well beyond what scoreboarding could provide. However, the deed was done, and as we'll see later, I had bigger problems to worry about by the end of 1967.

7 Lessons Learned from These Early Experiences

Reflecting on all this, one might ask how a shy, naïve, freshly-minted MSEE could be the one to invent multiple-OOO DIS? The problem had apparently been clear to others; so why hadn't they found a solution?

The belief that it couldn't be done undoubtedly held back progress, and ethnographic observations reveal further problems: By the mid-1960s, chasms had developed between the various specialist subgroups working on computer architecture, logic design, circuit design, packaging and manufacturing. Each specialty optimized the work at their level of abstraction, and then passed it on to the next.

Those working in the separate specialties almost never asked those in other specialties how they did what they did, much less looked over their shoulders to follow their detailed work. One had the feeling back then that such behavior would have been deemed impolite, perhaps even perceived as questioning whether others knew what they were doing. Remember, this was IBM in the 1960's. Think *Mad Men* and you'll get some of the culture.

As a result, most computer architects lacked knowledge about the details of the machine's logic design, much less the rapidly advancing ECL integrated circuitry, and thus could not envision how to reach down into and more fully exploit that circuitry. Neither could many top logic designers grasp the arcane architectural tradeoffs being made at the level above – except to provide exact analyses of the levels of logic to implement particular slices down through those architectures. Nor could top ECL circuit designers easily collaborate with the architects to provide inventive circuit-level hooks that resolved intractable computer architecture problems.

Furthermore, although DIS was embedded into the ACS-1 machine, it now seems clear (as discussed later) that many of the architects involved in the Arden House version apparently didn't realize that DIS had been put in there – while others didn't fully grasp what it was. Such was the effect of incrementally adding compartmentalized "improvements" to a large hardware system in an ad-hoc, unsystematic manner.

For me, DIS revealed a big lesson, namely that only a rethinking of the basics across all levels of abstraction could break the instruction-issuance logjam. Such work required a different sort of 'expert', namely someone who understood the basic gist of each and every specialty and who could then innovate across all those compartments of knowledge. This lesson deeply affected my later work in VLSI, when creating innovative methods for working across multiple levels abstraction in digital-design.

8 Uncovering the Subculture of Logic Design at IBM

When Herb Schorr asked me to work with Merle Homan and other logic designers to confirm that DIS would fit within the machine cycle, I no idea what I was getting myself into. After all, it seemed straightforward: I would explain the DIS queuing structures and control sequencing to the logic designers, and they would explore alternative implementations in ACS logic circuitry.

I knew we'd have to cleverly exploit the ECL DOT-OR capability, and thus some tricky design work might be involved. We'd also have to take into account the fan-in/fan-out rules and circuit delays given in the ACS Circuit Manual, as well as other wiring rules and constraints given in the ACS Packaging Manual.

If and when we were successful, the resulting logic designs would be entered into ACS's computerized logic-design data base, the Design Record Keeping System (DRKS). During that entry, we'd also define the layout of logic circuit chips onto multi-chip-modules (MCM's) and the placement of the MCM's onto boards, along with the associated wiring for the design amongst chips, modules and boards.

Formal design diagrams called ALD sheets of moderate blue-print size could then be printed out from the DRKS files, showing the details of the logic, including chip placement and wiring. Further work to generate wiring sequences and other manufacturing information could then proceed from there.

However, no sooner had work begun when I discovered I couldn't precisely follow the designers' logic equations and design sketches. I kept stumbling into apparent inconsistencies and couldn't figure out why. And remember: when discussing logic designs, even one tiny misunderstanding – about signal inversions for example – could become costly.

At first I thought I was 'losing it'. Were there circuit types I didn't know about? No, that couldn't be it. The ACS ECL current switch circuits provided familiar NOR-NOR and NOR-OR logic functions. So what was going on? Why couldn't I get my head around their sketches?

Although embarrassed by my confusions, I began making a pest of myself by asking Merle and others to slowly walk me through their logic equations and logic-circuitry sketches. DIS was my baby, and I was determined to follow design alternatives down to the final circuitry. I had to be able to ask serious questions about these alternatives, even be able to do much of the work myself, to be sure it all worked. Fortunately I was junior enough in rank that I could get away with asking such questions without appearing to threaten anyone's expertise – and the designers opened up to me.

And then, Boom! I figured it out: the breakthrough came when several designers admitted they could not initially follow other designers' work either, and it had taken them quite a while to become 'experts.' It turned out that different designers were using different ad-hoc symbolic representations for the same things. Thus all logic equations and logic circuit sketches were in the particular 'shorthand' of the originating designer, right up until entry into DRKS. Thus newcomers to ACS engineering could easily become confused while attempting to decipher other designers' work.

Those who had recently studied logic design in college and who thought in terms of AND-OR-NOT or OR-AND-NOT logic, would use various methods to transform their equations into ACS NOR-NOR or NOR-OR form. In contrast, those who had learned logic design on the job used various methods for directly writing down NOR-NOR or NOR-OR equations, along with using several different sets of logic notation for doing so – with the variations depending on where within IBM they'd previously worked.

Not only did designers use different conventions and notations for writing logic equations, they also used different heuristic methods to create logic diagrams from those equations. Some even mixed different representations together in the same design sketches.

Unfortunately, the designers' shorthand methods were undocumented and could only be learned by debriefing each individual. Even though the material was very basic, I found such debriefings surprisingly time-consuming, and many designers said they too had similar difficulties in learning each others' methods. As a result, there were surprisingly many ways for designers to subtly miscommunicate about design intent and design function, especially at the interfaces between the work of various team members.

As this situation became increasingly clear to me, I decided to do something about it: I took notes as I went along, and gradually built a 'covering set' of representations that enabled me to follow any designer's work after asking a set of basic questions about their methods. These notes proved such a useful tool for communication among designers that I polished and eventually published them, in November 1967. The ACS Memorandum was rather slyly entitled "ACS Logic Design Conventions: A Guide for the Novice" [20].

9 Lessons Learned from the Logic Design Observations

It is no wonder that walls had built up between ACS architecture and logic design. An architect would have to invest a lot of time in learning a number of different logic design shorthands in order to follow that work, and then know how to ask focused questions about it. This also explained why I never saw any architects sketching logic diagrams (for example, to determine the levels of logic required for proposed subsystems). Instead, architects would toss block diagrams and state machine descriptions over the wall to the logic designers, and wait for a response.

The same 'expertise-limitation effect' worked in reverse: other than in my work with Merle Homan, I don't recall logic designers asking detailed questions about the architectural level either. I suspect they too felt insufficiently 'expert' to follow the work at that level, because of the lack of transparency of that level to 'non-architects'.

The fact that different logic designers used different methods for representing and interpreting their designs also tended to compartmentalize the work horizontally *within* that single level of expertise. I noticed a similar effect within the architecture group – namely that different architectural experts tended to focus on either the ISA, the pipelined ALU functional units, or the instruction caching, prefetching, branching and issuance hardware – while failing to entertain in-depth discussions about the other lateral areas.

It gradually dawned on me that becoming an 'expert' within a design specialty mainly involved learning the shorthand representation schemes informally used within that specialty, so as to be able to communicate impressively and maintain one's reputation, i.e., "pass" as an expert within that specialty. Being able to step back and reveal underlying basic principles, and then simplify things so that others (including outsiders) could easily grasp what one was doing, would only serve to undermine the impressiveness of a specialty – and thus was seldom engaged in.

These meta-level insights derived largely from my studies and readings in anthropology while at Columbia, and especially from the recent appearance of a foundational work in "ethnomethodology" by Harold Garfinkel at U.C.L.A [21].

From an ethnomethodological point of view, novice designers struggled daily to appear to be accountable members of the 'architecture clan' or 'logic design clan', etc., while avoiding tells of not understanding what was going on. They gradually built confidence and thus 'expertise' by figuring how others did their work, mostly by a process of observation and convergent imitation – motivated by their deep needs for clan-acceptance. Upon finally becoming experts, such architects, logic designers and circuit designers would feel uncomfortable when straying outside their specialties, lest they be seen as ‘novices’ by folks in those other areas. Such is the power of clan initiation rites.

In these subcultures, both at IBM and elsewhere all across the industry, designers naturally constrained their creative explorations to fall within the confines of their expertise. The resulting compartmentalization of innovations often led to the accumulation of additive suboptimizations, rather than to broad-gauge system-level breakthroughs.

Worse yet, truly creative innovations were often intelligible to only a few experts within the relevant specialty. Innovations that cut across levels of abstraction might be invisible to all – requiring entrepreneurial breakouts to see the light of day. As a result, major innovations in large organizations might often go unrecognized for what they really were, and even fall through the cracks when projects were cancelled. (Think back on the trajectory of the DIS patent disclosures!)

10 The ACS Logic Simulation System (LSS) – and the Challenge It Presented

The sheer scale of the ACS-1 machine, and the possibility of subtle miscommunications during early stages of logic design, led to an urgent need for logic simulation.

The concept of such simulation seemed straightforward enough. Given an initial state and sequence of inputs, a designer could predict the internal state sequences and output sequences expected from their design. By comparing those expectations with the input-output behavior of a logic simulation, designers could confirm whether their designs behaved correctly – at least for the given I/O sequences.

A group at System Development Division (SDD) in Poughkeepsie, N.Y., proposed development of a package of programs for performing simulations on logic partitions that had been formally specified and entered into the DRKS data base. One of the programs would enable users to specify a logic partition along with its input and output lines, and extract that partition’s logic from the DRKS data base. Users could then specify the initial states and a sequence of inputs and simulate that partition by using a second program – the TALES logic simulator (not sure what that acronym stood for) – which would compare the simulated and expected output sequences at the partition interface, marking any mismatches.

Although designers could “sanity-check” their designs this way, the difficulty of generating the I/O signal sequences at the partition interfaces was a major obstacle to practical application of the proposed system. It did not appear at all feasible for the designers to hand generate *all* the correct test patterns necessary for even a modest level of debugging of all partitions of a machine of this scale.

During some early brainstorming in August 1966, Gerry Paul suggested that perhaps we could use the architectural timing simulator to generate some of the partition I/O signal sequences. What a great idea! We’d already begun using the timing simulator to document and simulate portions of the high-level design. And from our experiences so far, it seemed that we might indeed be able to automatically generate large numbers of signal sequences at specified machine interfaces while simply running real programs on the timing simulator.

For such a scheme to work, we would need to coordinate the work in architecture and engineering carefully, in order to standardize, document and maintain the integrity of all the partition interfaces. However, the added investment in interface selection, standardization and automatic signal generation promised to pay back handsomely during design validation.

During the following months, I was tasked with defining and planning an ACS "Logic Simulation System" to tie all this stuff together. The overall system included the partition extraction and TALES programs to be written at Poughkeepsie, and an "interface signal generator" written at ACS. The interface signal generator was to produce I/O sequences for logic simulation runs by running machine code on an augmented ACS timing simulator, and capturing I/O signals crossing specified partition interfaces within the simulated machine.

Many questions had to be resolved along the way: How large or small should/could the partitions be? How should/could the interfaces be best selected? How should we describe partitions at the logic level of design, for subsequent extraction from DRKS? How could we also craft dummy subsystem partitions at the timing simulation level, so that the whole MPM could run at that level even though the internals of some subsystems were not yet fully specified? How could we best coordinate the debugging of system level design partitions as we iterated on debugging the logic designs, with all the issues of version control?

We also needed a way to manually insert interface signals to debug isolated sections of logic for which such insertions were adequate – for example for functional units such as adders, multipliers, dividers, etc. There were also questions on how to best present simulation results to the architects and logic designers so they could jointly debug the architectural/logic partitions.

Finally, there were questions of feasibility and resources. How large a partition could we run on an IBM Model 75 and at what rate would it run? And, how much time and how many people would be required to develop and maintain the system?

Prem Shivdasani made a clever analysis showing that TALES could handle partitions as large as 56K ACS circuits, and that it would take only a few seconds of Mod 75 time to simulate one machine cycle for partitions of that size. (By comparison, the MPM timing simulator ran at the rate of 10 to 15 machine cycles per second on the Mod 75.) These run times were quite reasonable, being within the range where the time and effort to debug the results was likely a stronger limitation than machine time.

By the summer 1967, most of these questions had been resolved, and I submitted the resulting plan for ACS in a memorandum entitled "A Proposed ACS Logic Simulation System (LSS)" [22].

11 Lessons Learned from the Logic Simulation Planning

One day while reflecting on my observations of logic designers, a vital meta-level benefit of logic simulation came into view: In addition to its primary purpose of debugging the logic, such simulations promised to help designers better visualize and understand each other's work. By jointly observing and discussing output traces, different expert logic designers and architects could join in and communicate less ambiguously about what was going on. Meanwhile, less experienced participants could stand around and listen to such discussions, and gradually reduce their inner confusions about what was going on, without having to ask embarrassing 'ignorance-revealing' questions. Assimilation of newcomers into the specialist clans would thus be eased. At least that was the promise.

However, the new tools also brought new difficulties. Remember, back in those days we used punch card input, courier-based batch processing and line-printer outputs. Programmers today can hardly conceive of the limitations these imposed at the time, which was even before time-sharing had come along. It was incredibly difficult to turn-around enough runs at a fast-enough rate to support thorough debugging. The planning and management of such simulation runs involved many messy logistical and project-coordination issues, especially since the software itself was often a moving target.

Furthermore, hardware designers at all levels now had to interact with yet another set of "experts", namely the programmers who were building the new design automation software (i.e., electronic design automation, or *EDA,* as it is now called). Designer involvement in EDA tool development was essential in order to help specify, learn how to use, update and maintain those tools. But that involvement often came at a considerable cost of time and energy.

Meanwhile, as might be expected, different EDA programmers had very different notions of what they, and we, the designers, were doing – depending upon how they'd learned their craft and how much or little they knew about digital design at the various levels. As a result, EDA spawned a whole new arena of ad-hoc software, some of which constrained designers in ways not immediately understood. For example, most of the new design tools directly supported a particular specialty level. Such tools were the easiest to specify, for they only involved one narrow clan of users. The new EDA tools thus tended to heighten and solidify the walls between specialties, rather than breaking them down.

12 Ongoing Work on the MPM Timing Simulator

We made many simulation trials during 1967 to tune up the MPM architecture, much of the work being done in coordination with compiler research by John Cocke, Fran Allen and Jim Beatty. By mid-67, the ACS-1 MPM architecture had stabilized, and had been captured in the MPM timing simulator along with my design notes in support of the simulator.

That summer I completed an ACS memorandum entitled "MPM Timing Simulation", [13] detailing how the simulator worked, how to prepare its input, and how to interpret its output. The report contained a number of tutorial examples, using output listings to reveal how instructions could be followed as they were dynamically issued out-of-order from the DIS queues.

All of these documents still remain and are now posted on the internet, including my design notes [23] and the full listings of the Fortran H source code [24] for the simulator.

13 Compiler Explorations Using the MPM Timing Simulator

Compiler development was a vital dimension of the ACS project, and was a special focus of much of John Cocke's work while there.

> *My interests have been in achieving high performance for a broad range of scientific calculations. Therefore, I have focused mainly on optimizations involving the compiler and the underlying machine architecture . . .*
>
> – John Cocke, Turing Award Lecture, 1988 [25]

John teamed up with Fran Allen and Jim Beatty on this effort, and they continued on with it well beyond the ACS project. Fran later described what this work was about, and its ultimate impact, as follows:

> *From the beginning of the project it was recognized that the only way to realistically realize the performance goals and make them accessible to the user was to design the compiler and the computer at the same time. In this way features would not be put in the hardware that the software could not use or which the software could support more effectively.*
>
> *In order to isolate the effects of changes in the CPU design and to modularize the experiment, the ACS compiler classified the optimizations as machine independent or machine dependent . . .*

Out of this project . . . came numerous advances in compiler technology, and, more importantly, the foundations of the theory of program analysis and optimization.

– Fran Allen, "The history of language processor technology in IBM", 1981 [26]

The MPM timing simulator provided John, Fran and Jim the means to study the machine's performance on code generated by their experimental compilers, as they made successive iterations between theory and experiment. They mostly divided up this work, with John and Fran working on machine independent compilation while Jim Beatty did the machine-dependent explorations.

Throughout much of 1967 John, Fran and Jim often met with me as a group to go over simulation runs. John was very excited about these explorations, and he and Fran often carried on about their work during the meetings. They seemed to be getting a lot out of the simulation runs, and it was great fun to interact with them. Seems that wherever John went he spread excitement and enthusiasm, causing everyone and everything in his path to crank up the volume and get into action!

Oddly though, I don't recall John or Fran ever asking for explanations about the detailed MPM simulation output traces, nor did they ask any questions about the hardware architecture. Instead they seemed mainly interested in the overall number of machine cycles required to run loops of various codes – as a function of variations in compilation methods that generated those lines of code.

The architectural functionality of DIS had been well described in the DIS tutorial. It had been implemented in detail in the MPM timing simulator, and its behavior made quite visible in the simulation outputs. Thus I assumed that John and Fran fully understood the machine's micro-architecture including the dynamic multiple out-of-order instruction issuance provided by each DIS queue – and that they could visualize those details of MPM functioning by looking at the output traces.

However, as we'll see later, it now appears to me that this might not have been the case. Although John had by then heard something about the inclusion of the 'contender stacks', as we called the DIS queues, I now suspect that he might have thought those stacks were some sort of FIFO instruction buffers that simply fed the original Arden House instruction registers – or served as some part of the pre-fetch/decode hardware.

An incident during one of our 1967 meetings shines light on this issue. John arrived at the meeting after having met with some folks in engineering, and he was very excited about *something*. It turns out that he had just learned that "they" had figured out the details of how to handle interrupts in the "contender stacks" (my best recollection of the words he used), which had been one remaining piece of the puzzle.

I vividly recall the incident, because it was an embarrassing jolt to hear John use the word "they". It was instantly clear to me that he wasn't aware of my role in DIS, although I assumed he knew what it was and how it worked. I guessed that John assumed that I was simply the support programmer who ran the simulator – without realizing that I'd been involved in the engineering work he had just raved about.

Neither then, nor later, did John ask me how the 'contender stacks' could issue so many instructions per machine cycle. Of course even if John harbored such questions, he wouldn't have asked me about them if he assumed I was in a support role.

Jim Beatty was familiar with the hardware architecture, for he was doing the explorations in machine dependent compilation, and John could have learned the details of the DIS functionality by listening to Jim talk about his work. However, Jim was a quiet and studious guy; he might have kept mostly to himself and shared his work through his technical reports. Jim has passed, and unfortunately it seems that none of his internal reports on his important work have survived – as with so much that happened at ACS at the time.

This, it would appear, is another example of how, in the rush of events, lacks of awareness and various misunderstandings might unknowingly develop among designers in adjacent specialties – especially if everyone is highly-involved in their own area of work and too polite to offer, or seek, clarifying information over the walls of specialization. It will be interesting to go back and explore these issues further.

14 The Design of the ACS Design Process

As 1967 drew to a close, all the lessons from the logic design and logic simulation work gradually combined in my mind, and I began to *go meta* on all of it. (Remember your maths: when you see or do something interesting, ask yourself: "What set is this thing a member of?")

I began to see the same kind of difficulties existing in the communications between architects, engineers and EDA tool builders as they all struggled to understand what others were talking about as we jointly engaged in intense improvisational work within our different fields and across different partitions – while at the same time attempting to make coherent sense of it by working together at the interfaces. While making these observations, I began drawing sketches that at first helped me – and later helped others – visualize what I was thinking and talking about.

Before making these diagrams, my head would spin whenever I tried to visualize it all – especially when doing so from an *ethnomethodological* point of view that maintained the observable inconsistencies in different specialists' personal viewpoints at each level of design. You know the old saying, "consistency is for small minds". Well let me tell you, this level of inconsistency-management was *way out there*. I desperately needed something besides mental simulations to keep track of it all!

The first challenge was to make sure we each knew what *stage* or *level* of *design abstraction* we were talking about. The second was to precisely identify what *partition* of the design we were referring to at that level. I solved these problems by slipping a diagram such as in Figure 2 into discussions (see below), and pointing to various aspects of the illustration when referring to them.

A completely different kind of diagram was needed, however, to help us all visualize the *tools* and *processes* being used by particular designers as they evolved particular partitions at each particular level – so that these separate activities could be done in a manner coordinated with the levels above, below and sideways.

This was a different problem than project management using the by-then-familiar PERT chart. Instead, the process involved the actual doing of the things being done, the building of the thing itself. This was something a PERT chart could be wrapped around later when assigning people to a project, making budget decisions and building project time lines – but not substituted for.

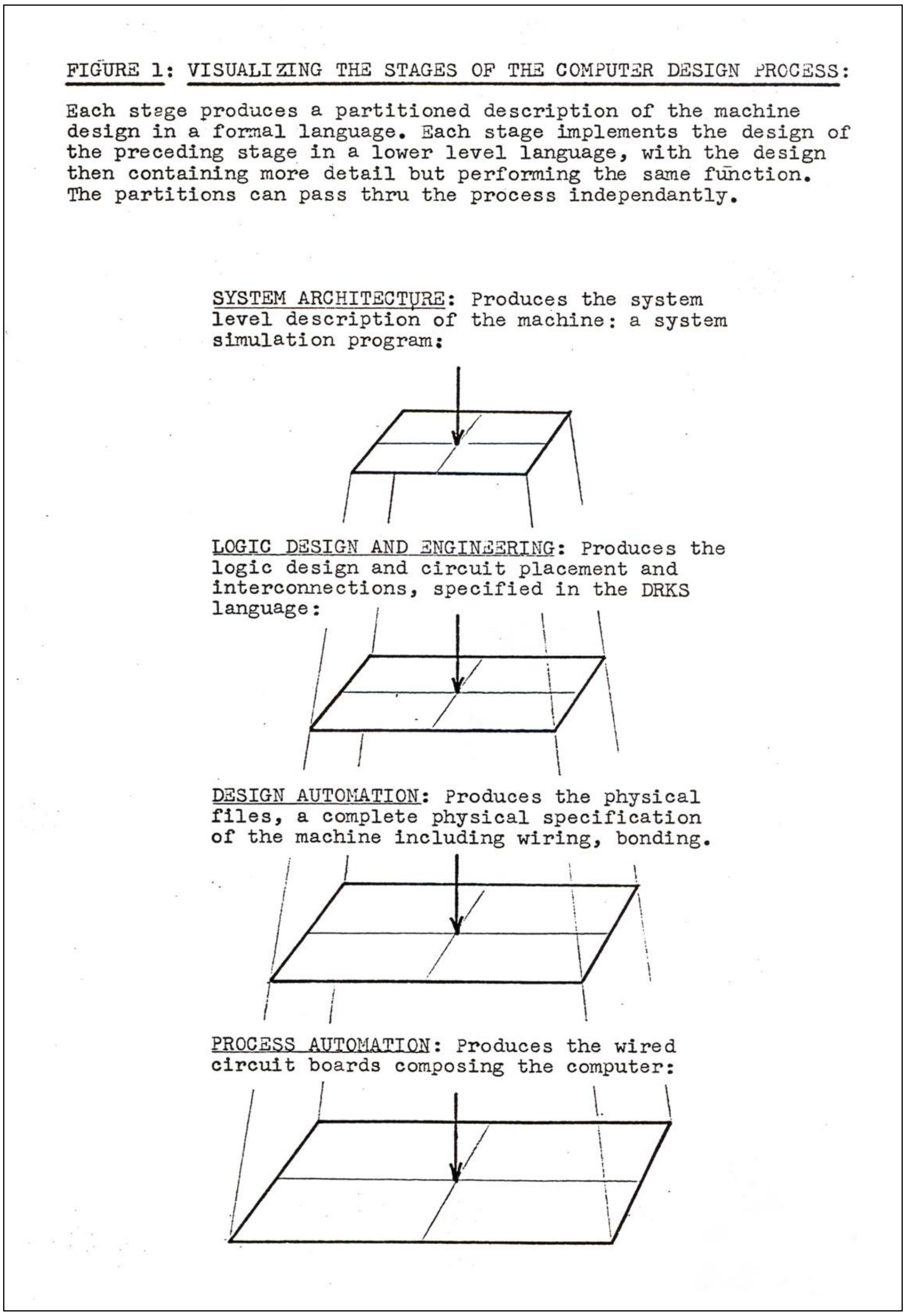


FIGURE 1: VISUALIZING THE STAGES OF THE COMPUTER DESIGN PROCESS:

Each stage produces a partitioned description of the machine design in a formal language. Each stage implements the design of the preceding stage in a lower level language, with the design then containing more detail but performing the same function. The partitions can pass thru the process independantly.

SYSTEM ARCHITECTURE: Produces the system level description of the machine: a system simulation program:

LOGIC DESIGN AND ENGINEERING: Produces the logic design and circuit placement and interconnections, specified in the DRKS language:

DESIGN AUTOMATION: Produces the physical files, a complete physical specification of the machine including wiring, bonding.

PROCESS AUTOMATION: Produces the wired circuit boards composing the computer:

Fig. 2. Figure 1 from "The Computer Design Process: A Proposed Plan for ACS" [27]

In late 1967, it suddenly came to me that I was making a conscious effort to *design the ACS design process*, and make it into a visible entity just as real as the design of the processor itself. Only in this case, the entity being designed was the detailed structuring of the groups of people and tools and all the interactions that over time produced a working machine design. I called it a "flowchart" of the computer design process, and it eventually came to look like Figure 3.

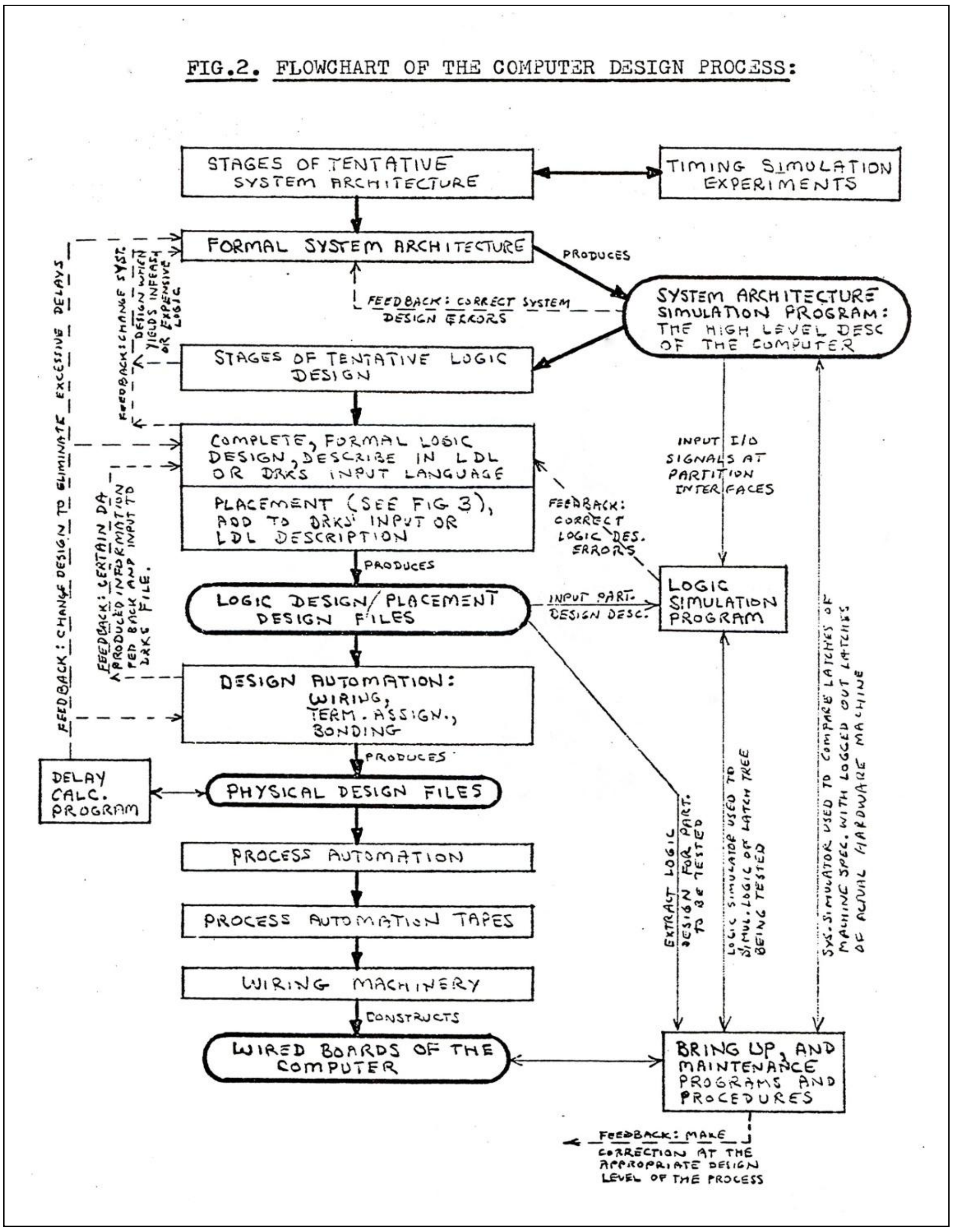


FIG.2. FLOWCHART OF THE COMPUTER DESIGN PROCESS:
STAGES OF TENTATIVE SYSTEM ARCHITECTURE
TIMING SIMULATION EXPERIMENTS
FORMAL SYSTEM ARCHITECTURE
PRODUCES
SYSTEM ARCHITECTURE SIMULATION PROGRAM: THE HIGH LEVEL DESC OF THE COMPUTER
FEEDBACK: CORRECT SYSTEM DESIGN ERRORS
STAGES OF TENTATIVE LOGIC DESIGN
FEEDBACK: CHANGE SYST. DESIGN WHICH YIELDS INFEAS. OR EXPENSIVE LOGIC
FEEDBACK: CHANGE DESIGN TO ELIMINATE EXCESSIVE DELAYS
COMPLETE, FORMAL LOGIC DESIGN, DESCRIBE IN LDL OR DRK'S INPUT LANGUAGE
PLACEMENT (SEE FIG 3), ADD TO DRKS' INPUT OR LDL DESCRIPTION
FEEDBACK: CERTAIN DA PRODUCED INFORMATION FED BACK AND INPUT TO DRKS FILE.
INPUT I/O SIGNALS AT PARTITION INTERFACES
FEEDBACK: CORRECT LOGIC DES. ERRORS
PRODUCES
LOGIC DESIGN/PLACEMENT DESIGN FILES
INPUT PART. DESIGN DESC.
LOGIC SIMULATION PROGRAM
DESIGN AUTOMATION: WIRING, TERM. ASSIGN., BONDING
PRODUCES
DELAY CALC. PROGRAM
PHYSICAL DESIGN FILES
PROCESS AUTOMATION
PROCESS AUTOMATION TAPES
WIRING MACHINERY
CONSTRUCTS
WIRED BOARDS OF THE COMPUTER
EXTRACT LOGIC DESIGN FOR PART. TO BE TESTED
LOGIC SIMULATOR USED TO SIMUL. LOGIC OF LATCH TREE BEING TESTED
SYS. SIMULATOR USED TO COMPARE LATCHES OF MACHINE SPEC. WITH LOGGED OUT LATCHES OF ACTUAL HARDWARE MACHINE
BRING UP, AND MAINTENANCE PROGRAMS AND PROCEDURES
FEEDBACK: MAKE CORRECTION AT THE APPROPRIATE DESIGN LEVEL OF THE PROCESS

Fig. 3. Figure 2 from "The Computer Design Process: A Proposed Plan for ACS" [27]

The two diagrams evolved over time so as to capture and cohere more and more of our overall design effort. We had already planned to use the high-level architectural simulator (which formally captured the evolving architectural design fragments) to generate interface signals for logic simulation at the next level down (where similar improvisational logic designs were firmed up when they were entered into DRKS).

We now planned to continue this type of process on down to the physical specification of the machine (partitioning onto MCM's, wiring, bonding) and to the final level: the process automation that produced the wired boards that composed the machine – where the interface signals generated by the MPM simulator could be used during the bring-up and maintenance of the machine. In an eerie way, the 'design process design' by analogy mimicked the sort of pipelined, multiple-OOO-instruction issuance, multiple functional unit hardware of the design itself!

In general, the work of the architects preceded that of the logic designers on such large computer projects, and the work of the logic designers preceded that on circuit design, wiring and packaging. There was a time-sequencing to such work, with the architects 'completing' most of their work first, the others seen as simply implementing that architecture. In fact, by the time the later engineering work was being done, the architects were often off onto other projects. At ACS, for example, Herb Schorr was by now working flat-out on the ACS-1 software effort and John Cocke was working on his compiler research.

A subtle but profound side-effect of this apparently *top-down* process is that architects were presumed to be of some kind of higher-order intellect, even though few people understood what they actually did. The job of engineering groups was in turn often visualized as implementing architectures handed down from *above*. Within the architecture group there was a similar *Upstairs-Downstairs* effect: there were the "real" architects, while the rest were merely "support" staff.

These status and clan distinctions further exacerbated communication problems, and tended to deflect the participants from seeing the machine as an evolving entity growing under the constant collaborative interactions of *all of the participants*. After all, any participant might make an important innovation or discover a fatal bug at any time, and at any level. However, that was not the way folks usually thought of things at that time.

Once everyone's work was coordinated under a well-understood design process, the high-level simulator could help maintain consistency amongst all the levels by generating the needed interface signals to test and validate them – even as they progressed, each at a different pace. It could also be used to re-involve the architects at any point, such as when a later stage of design revealed problems that needed iteration at the architectural level. Thus, the playing field could be leveled, enabling fuller participation by all staff members.

In a cool coincidence, I later learned that during the same time I was working on the ACS Design Process, Brian, now back at IBM Research, was working on a method of jointly simulating the design of a computer hardware configuration and its operating system, so that the interactions between the hardware and software designs could be evaluated as an integral part of the design process. He and colleague Frank Zurcher similarly incorporated the concept of the concurrent existence of several representations of the overall system being modeled, at differing levels of abstraction, in their Iterative Multi-Level Modelling System [28].

It seems that these issues were suddenly in the air, as people struggled to cope with the newfound complexity of very ambitious hardware and software projects back then – as seen also in the works of Nicklaus Wirth, Edsger Dijkstra, Ole-Johan Dahl and Tony Hoare [29, 30, 31]. Complexity management in system design was brought to even wider attention by Fred Brooks, when he published his famous book *The Mythical Man-Month* [32], about his experiences in managing the development of the S/360 operating system.

I completed the design process report in the summer of 1968, and it was distributed as an ACS Memorandum entitled: “The Computer Design Process: A Proposed Plan for ACS,” August 6, 1968 [27]. Figures 1 and 2 above were taken from that report.

These many lessons deeply affected my later work in VLSI, for that later work involved not just a restructuring of the levels of abstraction in digital design, but also a breaking down of rigid technical, clan and class boundaries between the various levels, as well as the use of ‘unusual methods’ to propagate the results. But that’s another story!

15 The Demise of ACS and Ending of My Early Career

Just as I was making great progress in my design process work, everything came crashing down – on all fronts.

In hindsight, it is now recognized that had the ACS-1 been successfully built, it would have been *the* premier supercomputer of the era [4], eclipsing both the CDC 7600 and the IBM Model 91. But, that was not to be. Gene Amdahl had long advocated that the ACS efforts be focused on a S/360-compatible supercomputer. In late 1967 his views were brought more strongly to the attention of SDD management, and the ACS-1 soon became victim of the resulting techno-political confrontation.

The major troubles began shortly after Herb Schorr and others presented the ACS-1 design and schedule to a review committee in November 1967. The first customer ship of ACS-1, which included an OS and a complete set of compilers and utilities, was not projected until four years in the future – and a rough estimate of software expenses was given of $15M. As a result, Harwood Kolsky was sent to meet with Amdahl in December to investigate Amdahl's ideas for a S/360-compatible alternate version of ACS [1].

At that very same time, I was sidelined and pushed out of the loop, for reasons clarified below. I sensed something terrible was happening at ACS, but had no knowledge of the details. One thing I do know: no one ever asked me for help in defending the ACS-1. No one ever asked for explanations regarding the inner workings of the machine architecture, or for revelations of its powerful performance via MPM simulation runs. Instead, out of the blue, it happened: in May 1968 the ACS-1 project was canceled and replaced with a project to build an “ACS360.”

I was devastated. How could this have happened, I wondered? And what fate might befall the cool things accomplished there – including my own work on DIS and on the design of the computer design process? Such questions began to nag at me, and they’ve stayed with me down through the years.

However, I now had even bigger problems; I was pioneering down another path, and things had run off the rails. In early 1968 I alerted Wally Koch, IBM-ACS personnel manager, that I was undertaking a gender transition to resolve a terrible existential situation I had faced since childhood. I was hoping to quietly maintain my employment during that difficult period, in order to survive the ordeal ahead.

Wally was very supportive, as was Gene Amdahl, when (as the new Director of ACS) he later learned about the situation. For some months everything had seemed to be going well, even after Wally talked about it with Don Miller, Director of SDD Personnel. I was even given assurances that when I was ready to socially transition, the company would find a way for me to quietly transfer to another division with a new identity.

However, that summer Miller checked with Walt Peddicord, IBM Vice President and Corporate Director of Personnel, about how to handle my situation. Before long, I learned that the news had reached IBM’s Medical Director, John Duffy. He was not at all supportive.

Duffy’s reaction really hurt, but should not have been a surprise. Many psychiatrists back then presumed that gender transitioners were profoundly mentally ill, and often recommended forced institutionalization of such people. Duffy never talked to my medical caregiver, Harry Benjamin, M.D., to learn about his pioneering new methods for compassionate assistance of transitioners, nor did Duffy ever meet and talk with me. Soon afterwards, Wally Koch told me that the case had finally escalated to the Corporate Executive Committee – thus to CEO T.J. Watson, Jr.

One can only imagine the tantrum that T. J. Watson, Jr. likely threw when he heard the news that some junior-level IBM employee was undergoing a “sex change,” as transitions were rather grossly referred to those days. It must have been something to behold.

You can get the picture by reading Watson’s own words, in his book *Father, Son & Co.*, about his uncontrollable temper, and about his vendetta against an outed gay employee at the Greenwich Country Day School where Watson served as president of the board of trustees [11].

Watson’s tyrannical behavior, especially towards the School’s headmaster who didn’t act swiftly and harshly enough against that gay employee, so disturbed the other trustees that Watson was compelled to resign from that board. Additionally, the incident so marred his local image, he was never again asked to serve on any Greenwich board [11].

Clearly, in retrospect, I didn’t stand a chance. I was called to Gene Amdahl’s office on August 29, 1968, and he gave me the news: I was fired.

I learned later from Wally that the executives feared scandalous publicity if my story ever got out, while Duffy raised concerns that IBM employees who ever knowingly interacted with me might suffer major emotional problems (so frightening was the typical mental model of gender transitioners back then).

In the months leading up to that my firing, I’d attempted to maintain a reduced profile at ACS. I began working part of the time at home to complete the design process memorandum, which I was by then also turning into a research paper. (Don Rozenberg later presented the paper for me at an internal IBM symposium; he reported that it was quite well-received.)

During that time I kept most of my files at home, including my detailed ACS-1 hardware architecture notes and the FORTRAN source code for the MPM simulator. When I left IBM that September, no one ever asked for my design notes, papers or files. The project was in turmoil by then and the company seemed frantic to get rid of me. In the chaos, they somehow overlooked the established procedure of an exit interview.

I couldn’t bear to part with my work materials, so I carefully stored everything and kept it all down through the decades (Subsequently, I persuaded IBM to declassify all of these documents in August 2000 [33], so that I could publish them on my website [2].)

Finding myself unemployed and in the midst of transition, I visualized all of my contributions at ACS going down the tubes as the failed project simultaneously imploded. I grieved over this double misfortune, but there was nothing I could do about it.

It’s the nature of war and politics that the winners write the history. Few good things were ever said about the ACS-1, because of its stained image as a cancelled project, and little curiosity ever arose within IBM about what might have occurred there. Brian’s twin predictions about the fate of ACS came true, and before long it was almost as if ACS had never happened.

16 On to a New Life

I was now on to a new life, however, and had no time to look back. After completing my transition in late 1968, I started my career all over again in the spring of 1969, working in a new identity at the bottom of the career ladder, as a contract programmer.

A gritty survivor, my adjustment to that life went completely against the dire prejudgments of the IBM executives who'd fired me. Living in stealth and attracting no publicity, I became so happy and full of life that my career took off like a rocket. Seems it's always best to be underestimated, then outperform people's expectations. Moving upwards through a series of companies, I landed a computer architecture job at Memorex in 1971. In 1973, I was recruited to work at the exciting new Xerox Palo Alto Research Center, just as it was forming.

Building on many deep lessons I'd learned at ACS and then later at Memorex and Xerox PARC, I seized an opportunity between 1976 and 1979 to radically restructure the levels of abstraction, and of specialization, that had become institutionalized in digital design. Working with Carver Mead of Caltech, this research led to my invention of the scalable MOS chip design rules and to a simplification of the overall methods for VLSI chip design.

While thinking about how to present the results, I conceived of and began working with Mead on the textbook *Introduction to VLSI Systems* [34]. I then went on to pioneer an intensive VLSI design course at M.I.T., teaching the new methods to a class full of enthusiastic students [35]. Before long, universities all over the world were adopting the text for similar courses, and DARPA launched a major new VLSI research program to build on the "Mead-Conway" methods [36].

During an effort to scale up university participation in the new methods, I invented a new form of internet-based rapid-chip-prototyping infrastructure [37] later institutionalized by DARPA as the *MOSIS* system. MOSIS has supported the rapid development of thousands of chip designs down over the years [38]. The clean interface between VLSI design and fabrication exhibited by MOSIS triggered the modern 'fabless' model of chip design, in which creative designers are no longer captive employees of a handful of semiconductor firms. Scores of startup companies sprang up to commercialize the knowledge, and all of this happened without people catching on to my "past."

In 1981 I reflected on these events in paper entitled "The MPC Adventures" [39]. A concise history of the events is also given in the book *Funding a Revolution* [40], published by the National Academy Press, which reveals the impact in academia and industry of the Mead-Conway design methods, the textbook, the VLSI design courses and the MOSIS infrastructure.

Along with the thrusts in personal computing at PARC and the vigorous entrepreneurial culture that emerged there and in the Valley beyond, these collective events had by 1990 spelled doom for the paternalistic, autocratic, monopolistic IBM of old (which stood in contrast with the later IBM, as rebuilt by Lou Gerstner) [41, 42]. What a dramatic reversal of our mutual fortunes since that terrible time in 1968 when IBM fired me, a firing that could have been a death sentence in those dark times.

17 The Resurrection of ACS

As the years passed by, I often wondered about the early work at ACS and what might have happened with DIS. I saw no evidence whatsoever that either IBM or Amdahl Corporation had ever figured out how to exploit it; I began having serious doubts as to whether or not all but a tiny few in either company had ever really understood it.

During the late 1970's at PARC I made copies of the old DIS tutorial [16] (after removing the cover sheet) and occasionally released it to key visitors from the architecture community. It made for a nice example of the sort of regularly-structured micro-architectural subsystems that might be exploited using VLSI, and it fitted well in among other handouts I was using at the time. The tutorial had "IBM Confidential" on the cover sheet, along with the title and the authors' names. Of that, the inside pages only retained the title: "Dynamic Instruction Scheduling."

I thought hard before doing that, but in the end had no moral qualms in releasing the tutorial. IBM had broken every promise to help me, and had almost doomed me to life on the streets, or worse. In the process they'd apparently killed my cool invention too: ten years had passed and no signs of it had ever emerged. My hope on releasing the tutorial was that dynamic multiple out-of-order DIS might be used someday, even though my name would never be associated with it. I had nothing to gain. I just wanted to get the idea out there.

By the mid 80's, DIS began to appear as an apparent reinvention by Torng [43] at Cornell, among others, and by the 90's was being implemented in a variety of ways in many high-performance VLSI microprocessors, such as the Intel Pentiums, et al. [7]. DIS even became the subject of 100^+$ million patent suits by Cornell University against Intel and Hewlett-Packard [44, 45].

Although happy to see DIS out there, I felt growing angst about its early origins being unknown, especially since academic papers and textbooks were using the same name for the concept: Dynamic Instruction Scheduling [43, 7]. The name originally emerged from Brian Randell's usage of British English (pronounced "shed′-du-ling"); to me, the odds seemed pretty long that someone else would independently coin the term for the same invention, as had Torng [43]. Tutorials on the invention's functionality also bore what I viewed as considerable resemblances to the old ACS tutorial [7, 43].

I had no way to probe what was going on, back in those days before modern search technology. I was also living in *stealth mode* and dared not contact anyone about my concerns, lest I reveal my past connection to work that had by now become important. I tried to forget the past and avoid that angst, and more years went by.

Fate intervened in late-1998, some thirty years after my firing by IBM: I casually typed the word "superscalar" into an internet search engine, and up popped: "ACS--The first superscalar computer?" It turns out that Professor Mark Smotherman at Clemson University had stumbled onto information about the old ACS project, and theorized in his website [1] that ACS was indeed the first such supercomputer.

Just a few years before, Jim Smith and Guri Sohi of the University of Wisconsin had published an invited article in the Proceedings of the IEEE entitled "The Microarchitecture of Superscalar Processors," describing, and bringing to wider attention, the type of architecture being used in the latest high-performance microprocessors [46]. Smith and Sohi summarized the 'new' type of architecture as follows:

> *A typical superscalar processor fetches and decodes the incoming instruction stream several instructions at a time. As part of the instruction fetching process, the outcomes of conditional branch instructions are usually predicted in advance to ensure an uninterrupted stream of instructions. The incoming instruction stream is then analyzed for data dependences, and instructions are distributed to functional units, often according to instruction type. Next, instructions are initiated for execution in parallel, based primarily on the availability of operand data, rather than their original program sequence. This important feature, present in many superscalar implementations, is referred to as dynamic instruction scheduling.*
>
> – J. E. Smith and G. S. Sohi, 1995.

Because of the growing success of the Intel Pentiums and other powerful superscalar microprocessors, Mark's theory that ACS was the *first* had now become a question of interest. (Interestingly, John Cocke himself had coined the name *superscalar* for this general class of machines back in 1987 [47].) Stunned, I realized it was inevitable now that the story would come out, and that I needed to get out in front of it. I was also sick and tired of living like a Cold War spy, as if somehow ashamed of my past – which I was not.

I contacted Mark and gradually revealed my role in the ACS project. I also visited Jim Smith and Guri Sohi at Wisconsin in early 1999, and alerted them too. Fortunately, I had saved my old notes including the DIS tutorial [16] and since then I've posted them all in an online archive [2], where you can read them in full detail. I shared all of this with Mark and pointed him to other ACS veterans likely to have additional documents. I also began posting information in my website, *lynnconway.com*, to quietly explain my transition to more of my colleagues, hoping times had changed and at least some would understand.

Mark arranged for a reunion of ACS veterans the following summer to further the process of gathering more documents and doing reconstructions. It was held on July 29, 1999 at IBM's Research Center at Yorktown Heights, where Herb Schorr, Fran Allen, Mark Smotherman and I met with John Cocke and Peter Capek. Ed Sussenguth also participated by speakerphone, from North Carolina.

Mark went through the ACS reconstruction effort and discussed his website. I brought along a copy of my ACS archives for John, and discussed how it might be used to reconstruct the hardware architecture. Fran Allen brought a number of other key ACS documents, including a critical missing link: the ACS Instruction Set Manual. Herb Schorr contributed lots of reminiscences, and I'd never seen him so animated in conversation before. Many of us went on to a wonderful dinner and enjoyable conversations at Kittle House in Chappaqua, N.Y., with Peter's wife Mikki joining us too.

I had kept in touch with Herb, Charlie, Don, Fran and a few other ACS vets down over the years. They'd known about my transition (and kept it in confidence), and some had even helped me with references when I restarted my career. However, this was the first time I'd seen John since 1968. It would also be the last: he passed on July 16, 2002.

I later learned that John had heard quite a bit about 'Lynn Conway', having been familiar with my research in VLSI system design. However, John apparently had no clue that this woman had formerly been the boy he'd worked with briefly at ACS. Peter Capek quoted John as saying shortly before our reunion, "Holy Shit! That's amazing!", when he learned how the boy had gone on to become Lynn Conway.

Subsequently, during the 2000s, Mark compiled a comprehensive history of IBM-ACS in his website with the help of many ACS veterans [1]. His work led to a growing appreciation of the many transformative innovations made there in the areas of compiler development, computer architecture, electronic design automation, high-speed ECL integrated circuitry, system cooling and system packaging.

At the close of the decade, on February 18, 2010, the Computer History Museum (CHM) hosted a special event in honor of the many ACS-related accomplishments and to honor the surviving veterans of the long forgotten project [48]. It was a splendid occasion, bringing many of us together for the first time in four decades. Around that same time I also received the IEEE Computer Society's Computer Pioneer Award [49], based in part on my work on dynamic multiple out-of-order instruction scheduling. It felt wonderful to see that work, done and then lost so long ago, finally acknowledged.

Fig. 4. John Cocke, Fran Allen, Herb Schorr and Lynn Conway;
IBM T. J. Watson Research Center, July 29, 1999. Photo by Mark Smotherman.

18 Long-Lingering Questions Remain

As ACS became recognized and my role there became known, I began to feel some career closure, at last. It was a good feeling, but at the same time I still wondered about all those long-lingering, unanswered questions. Something seemed to be missing, some strangeness was still there, but I couldn't quite put my finger on it. Always the curious mind, I couldn't stop questions from arising.

For example, why had Jack Bertram, Herb Schorr, John Cocke and others been unable to defend the ACS-1 against the ACS360? Why had neither IBM nor Amdahl Corp. (Gene Amdahl's later company) exploited multi-OOO DIS in subsequent years? In later publications both Herb and John mentioned *high issuance* rates for each of the ACS-1's two instruction streams, but neither described how it worked. Neither did they mention the advance in functionality provided by DIS, namely the *dynamic out-of-order issuance of multiple instructions per machine cycle*. Good detectives always look for "what's missing" when searching for clues, and in this case those missing words and missing descriptors seemed telling.

Scattered documents also held clues. For example, U.S. Patent 3,718,912 [50], by Hasbrouck, Madden, Rew, Sussenguth, and Wierzbicki, which focuses on register renaming innovations, is apparently also in part related to the logic design of DIS contender stacks, although it does not describe the fundamental functionality of DIS. The patent includes some informal diagrams (Figs.

6a and 6b) that hold the key to teaching part of the functionality of DIS, but they seem thrown in as an afterthought and are not described in detail.

When I first came across this patent in 2000, I quickly recognized those figures. They were the same kind of conceptual diagrams I had used when teaching and reminding the logic designers about the *three types of source and destination register cross-comparisons* required to see if a subsequent instruction could be issued in front of, or at the same time as, any of a set of other instructions.

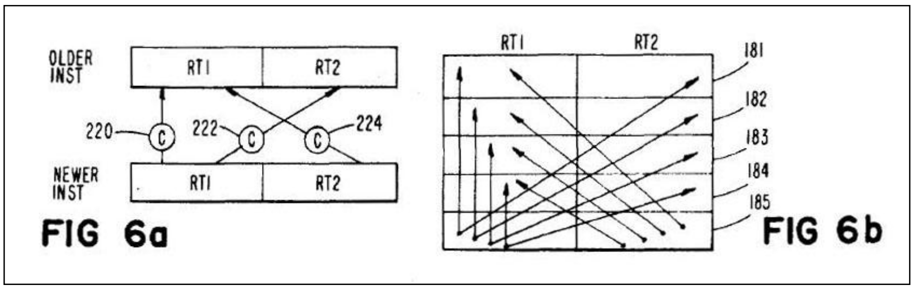


Fig. 5. From USP 3,718,912: Sketches of the source/destination interlocks in the Contender Stacks [50]

As suggested by the arrows in Figure 6, (i) such an instruction cannot use as a source register the destination register of any earlier instruction; (ii) it cannot have as a destination register the source register of any earlier instruction, and (iii) it cannot have as a destination register the destination register of any earlier instruction. To do otherwise could lead to a different, and thus incorrect, result.

Those interlocks thus protect against three types of *hazards* during the re-ordering of instructions: *read-after-write* (RAW), *write-after-read* (WAR) and *write-after-write* (WAW), respectively (as they are now called) [46]. These ideas were simple in a way, however *we didn't have words for them* at the time in the project. Thus the engineers had to look at those diagrams over and over again, in order to remember what the ideas meant, and thus firm-up their mental models of what the thing did.

So what happened with 3,718,912? Did the authors fully understood DIS and try to slyly patent it without revealing its functionality? I don't think so. Such conspiracy theories seldom pan out, and are poor bets when doing investigations. Unexplained group behavior far more often results from people 'bumbling along' in a collective set of incomplete understandings.

Given all that, I suspect that 3,718,912 was a 'clean-up' patent, i.e., an attempt to extract IP regarding register renaming and other architectural innovations out of fragments of work lying around after the demise of ACS. Some of the logic designers knew that Merle and I had worked hard on contender stack logic design and had possibly made breakthroughs in that work. Could some perhaps have remembered functional diagrams like those shown in Figure 6, and thought it would be good to include them?

But then too, why hadn't any ACS veterans ever asked me anything about dynamic multi-OOO instruction issuance down though the years? Had they known that it was a significant innovation in the machine?

Even more recently, I noticed that none of the ACS veterans mentioned DIS to me during the CHM event, even though quite a few had heard about the IEEE Award. Furthermore, during the lead up to the meeting, and during the meeting itself, I encountered a lot of side-comments about "multiple" instruction issuance having been invented by John Cocke at Stretch. What was that all about, I wondered anxiously.

19 The Dawn Comes

Soon after the CHM event I began triangulating on these many open questions, now fully motivated to get to the bottom of things. It wasn't long before the convergence began, but as I began making sense of things I was surprised at where the evidence was leading. How could I have had it so wrong all of these years?

The dawn came with the internet publication of B. O. Evans' condensed memoir in June 2010 [5], and my subsequent realization that Gene Amdahl and B. O. Evans probably had no clue about DIS in the ACS-1. Let's go back and relive the drama surrounding the downfall of the ACS-1, as described by Evans in the section in his memoir entitled "The Ill-Fated ACS Project" [5]. As often happens in such memoirs, the dates are off by a year – but we can certainly get a feeling for the flow of events and emotions nonetheless (multiply the following "circuit" counts by five, for approximate counts of transistors):

> *In December 1968 [sic], still President of the Federal Systems Division, I was at a neighborhood Christmas party in Bethesda, Md. when I received a call from Gene Amdahl. The conversation started with: "I have the son-of-a-bitch," language not characteristic of the gentle Gene and showing how deeply resentful he was of Dr. Bertram's rejection of his ideas . . . In this conversation Gene related that John Earle and he, using the same technologies as that planned for ACS, and using 360 architecture, had designed a supercomputer that was plus or minus 5% of the speed of the Bertram ACS design. Moreover, the Amdahl-Earle design required only 90,000 semiconductor circuits vs the 270,000 semiconductor circuits required for the Bertram ACS design. This was profound! The news broke quickly and the System Development Division headquarters was in turmoil. Erich Bloch, the VP responsible for engineering in SDD assigned a trusted colleague to investigate. In the meantime, the Research contingent driving the ACS project, having clearly overdesigned the system, were now under the pressure of Amdahl's claim and were frantically striving to reduce the semiconductor circuit count in their design. They brought the count down to 200,000 semiconductor circuits without losing much performance, blunt testimony to overdesign. In March 1969 [sic] Bloch's man reported: his count was that Amdahl's design was 108,000, not the 90,000 Amdahl claimed, however still a remarkable difference. In short order, the Research contingent departed ACS, and at my suggestion to Vin Learson, Gene Amdahl became "CEO" of ACS . . .*
>
> – B. O. Evans, *The Genesis of the Mainframe* (condensed memoir), W. Spruth, ed., June 2010 [5].

How on earth did this happen? Here's a summary, as best I can piece together right now: A relative latecomer, Earle joined the ACS engineering group as an expert in ALU logic design, but he never seemed to fit in either there or among the architects he sometimes interacted with; something about his personality rubbed folks the wrong way. At some point Earle began interacting with Gene Amdahl, and in Evans' words, "retreated into obscurity to pursue the 360 architecture approach" [5]. By then Earle had became *persona non grata* among the architects, and was kept completely in the dark about the ongoing architectural work on the ACS-1.

We learn more about these events, and what happened next, from Gene Amdahl in an interview on September 24, 2000 [51]: Gene was a first-hand observer, thus we must in the absence of other evidence lend more credence to the details in his report than to Evan's earlier one (noting, however, that Evan's report is particularly significant since it contains the viewpoint that apparently became operational within SDD).

> *Their top project designer, a really gifted designer, was not very manageable, and he'd gotten into some kind of trouble in Philadelphia and had been badly beaten up. He got out of the hospital, came back, and was recovering from it. But he was still unmanageable, and so*

they decided the best way to get rid of him was to move him over to me. Well, I figured at least I got somebody to talk to. So I began telling him what I had in mind doing, and it took probably more than a month to get him to understand what it was that I had in mind to do. And he did. He decided, "Looks like pretty fertile territory." So he began to do the logic and testing the number levels to do the controls, and we were able to show that we could do it, go faster than the 48-bit machine, and be cheaper. And that was sort of the death knell for the other computer, because we clearly had an advantage in not having to do our own operating system or application programming.

So with the logic designer I inherited, he was the one who had done the timing for the 48-bit machine and designed the critical paths for it. So here he designed the critical paths for our alterative version and did the timing on that to show the improved performance and the lower cost as well. So this was really the basis for a shoot out, which we held. Management from the East coast came out; some six people I believe and we worked for about three days. The 48-bit people had two days and we had one and they had no way of really challenging our work since the work that they were using to evaluate it was done by the same person. So my inheriting that designer turned out to be a very valuable thing. Any way, we won the shoot out.

– Gene Amdahl, September 24, 2000 [51]

But wait a minute: How could Gene Amdahl have thought that John Earle, a non-architect logic designer, could take numbers from critical path analyses and make inferences about the relative overall performances of two radically different machines? Such analyses do not yield mental models of the micro-architectures from which those logic-slices are extracted, and we must wonder what mental-models of the ACS-1 and ACS360 Earle had in his head when he compared the two.

One thing for sure: Earle never talked to me about anything, nor had he ever looked at any of the detailed MPM simulation output traces (which I kept in my office). Earle might have played some late role in confirming the ACS-1's ALU critical paths, but that would not have given him any insights into the overall micro-architecture of the machine – especially of the DIS queues.

This leads us into interesting speculations for further investigation: For example, how did Gene Amdahl get the impression that Earle had played a major role in the ACS-1? Did Earle make that claim when 'given' to Gene? Did the ACS-1 architects and engineers talk up Earle's role in the ACS-1 to help get rid of him? It seems unlikely that Earle knew about DIS and how it worked. And then, by not taking the DIS architectural enhancements into account, Earle would have been off by something on the order of a factor of two in his performance comparisons. Could this oversight have been what led Earle, Amdahl and Evans to jump to the erroneous conclusion that the revised ACS-1 was merely an over-designed version of the earlier Arden House machine – and thus to inflicting a mortal blow on what would have been a great machine?

In any event, from Evans we learn that Earle's numbers hit like a cruise missile out of the blue, and were apparently accepted by senior SDD management right at the outset. Thus when the formal SDD challenge came in the spring of 1968, it must have felt like an ambush to the ACS-1 leaders.

One might think that those leaders could have explained that the increased circuit count was more than justified by a large increase in performance – as demonstrated during detailed simulation runs. Did they attempt that defense? If so, why wasn't it successful?

According to Evans, instead of defending the machine's performance the designers went on a crash re-design effort to reduce the circuit count [5]. That effort must have been chaotic indeed. For example, what did they think they were doing, and why? Did some take Earle's and Amdahl's numbers seriously? Did this cause them to lose confidence in the ACS-1 revisions beyond the Arden House version? Did they attempt to redesign the machine back towards that earlier version? If so, why weren't they aware of the power-boost provided by the DIS 'turbochargers'? All these and further speculative questions are open to investigation.

A brief round of final, formal performance comparisons were held by SDD in April 1968, apparently with minimal ACS-1 team involvement. It seems that the ACS-1 didn't stand a chance. The end came in May 1968. ACS staff members were called into a big meeting in the cafeteria and told that the project had been cancelled. For many, it was the first time they'd become fully aware of the drama swirling around them [1].

On reflection, I now suspect that some of the problems in defending the ACS-1 were triggered early in the project, when we began relying on the timing simulator to document the critical portions of the ACS-1's data path and controls. In the process, we failed to create and maintain high-level architectural diagrams and clear tutorial explanations to provide widely-understood visualizations of what was in the machine and how it worked. Thus, when the confrontation developed, the ACS-1 team must have been hard-pressed to describe the advanced features of the machine and defend its performance. Gads, this was exactly the sort of design process problem that I'd worried about in the first place. How could I have missed it? I was too busy designing the design process!

In any event, the ACS-1 was doomed the moment Earle's numbers hit the fan at SDD. Major controversies over S/360 incompatibility and potentially large costs for ACS-1 software development had already threatened to derail the project – and those alone were serious challenges indeed. Earle's numbers were probably simply the last political straw.

Declared "a failure" by B. O. Evans as he became President of SDD, it wasn't long before the entire ACS project was disbanded, including the ACS360 [1, 5]. On top of everything else, T. J. Watson, Jr., finally decided to get out of the supercomputer market – this time on account of growing concerns about anti-trust suits [52]. Meanwhile, the DIS invention, apparently never fully appreciated by many key players, was shelved away in dusty technical reports and long lost within IBM.

20 Revisiting Later Writings

While trying to sort out these events, I went back and reread later writings about ACS, including Herb Schorr's 1971 reflections on his experiences there (based on a 1968 internal IBM symposium paper):

> *Multiple decoding was a new function examined by this project. By multiple decoding it is possible to initiate the execution of three index-type operations and three of the next eight arithmetic or logical-type instructions in every cycle. Eight arithmetic-type instructions are examined in each cycle to determine three executable instructions so that an instruction held up due to logical dependencies, register conflicts, memory interference, etc., does not impede the execution of any logically-independent instructions that follow it . . .*
>
> – Herb Schorr, 1971 [53]

On seeing that report years ago, I wondered why Herb hadn't spelled out the multiple out-of-order nature of the issuance mechanism, and the words left me with a vague sense of unease down through the years. For a while I thought Herb perhaps hadn't wanted to give away details of the still-confidential DIS idea, but I never discussed the matter with him. Given the many fascinating questions now in the air, I'm hoping very much to follow up with him on all that – especially about the ACS-1 vs ACS360 shootout.

John Cocke’s post-ACS reflections reveal similar glimmers of what had been accomplished, but again there’s no mention of *dynamic multiple out-of-order* issuance. John presumably knew these details, and he would have been in an excellent position to build upon them in his future work. As in all cases of such written communications, we have to wonder what visualization, what *mental model*, John had in mind as he penned the following words:

> . . . *I would like to tell you a little bit about the Advanced Computer System (ACS). This was a project we undertook between 1964 and 1968. It had a simple yet irresistible goal: to design and build the fastest scientific computer feasible. Under the late Jack Bertram's leadership, we designed a computer with many organizational features that even today are not well known.*
>
> . . . *It would have had a ten nanosecond cycle time, on the order of today's multimillion dollar computers, so it would have been very fast hardware. This is only where machine organization begins. At each cycle we dispatched seven operations, one to the branch unit, three to the fixed point unit, and three to the floating point unit. The fixed point unit could initiate three instructions on each cycle. The floating point unit had a buffer of eight operands and logic to pick the first three out of the eight that were ready . . .*
>
> – John Cocke, Turing Award Lecture, 1988 [25]

In modern day discussions of instruction level parallelism in superscalar machine architecture [6, 54], the words “dynamic” and “multiple” usually together modify “out-of-order”, as in “dynamic multiple out-of-order” instruction scheduling.

However, those unfamiliar with DIS and superscalar architecture might not be aware of that usage and think that “multiple” instead refers to any ad-hoc method for issuing more than one instruction per machine cycle – such as in various vector processors or in the Arden House version of the ACS-1. Those thus confused might then think that John Cocke had invented “multiple” instruction issuance (in all its forms) at Stretch, even though “multiple” takes on very different meanings in different architectural inventions.

Meanwhile, in recent years I have stressed the word “multiple” when referring to DIS for a very particular reason: namely to *contrast DIS* with ‘single’ out of order dispatch and issue as in the CDC 6600 and IBM Model 91. However, that distinction may have been lost on those who had other usages in mind. In any event, apparent confusion has existed down over the years among ACS veterans about the meaning of that word, something that became apparent to me at the Computer History Museum event.

Thus it all becomes clear: it seems that many of the confusions all along were the result of differences in the interpretation and mental modeling of the word “multiple” by different participants in the ACS saga. I’m kicking myself now for not noticing these usage-inconsistencies years ago, for these are just the kind of issues that were at the core of concerns in design process design. If I had noticed them, and had done a better job of teaching about them, things might have turned out quite differently.

In any event, the sudden cancellation of ACS back in 1968 led to a rapid scattering of key staff members and a massive dumping of documentation. As a result, it seems that a nagging sense of loss developed as time went by – a collective realization amongst the architects and designers still within IBM that they couldn’t fully explain the ACS-1’s detailed design. Worse yet, no one seemed able to “put it back together again,” especially given the incompleteness of, and variances among, different participants’ recollections.

Years later John Cocke echoed these sad feelings, in his 1988 Turing Award Lecture:

ACS never made it out of the laboratory; I suppose it was too big and too expensive, but for me it was probably the most exciting project I have ever been involved in. In reflecting on this, I believe that what made it particularly exciting was that we were a small team, mostly hand-picked by Jack Bertram, and we pioneered every aspect of the project . . .

My only regret about ACS is that unlike compiler ideas, we did not take the time to publish our ideas on hardware so others could build on them . . .

– John Cocke, Turing Award Lecture, 1988 [25]

Given the tragic loss of most of the ACS documentation, and the historical confusions about what happened there, there is no easy way to unravel what impact the many ACS-1 innovations, including DIS, had on later work within IBM and through whose hands that work might have flowed. For example, others within IBM may have later seen the DIS tutorial, or perhaps grasped the principles of DIS after talking with IBM engineers who had worked at ACS. Some may have heard John and others talking about "three out of eight" issuance at ACS, and wondered how that could have been done. Some might have heard about the work of Torng and others outside IBM, work on what became known in the textbooks as 'dynamic instruction scheduling'. For example, consider Mark Smotherman's observations about Torng's interactions with IBM:

It is ironic that Torng had IBM sponsorship for his research, but none of his sponsors at IBM knew of the ACS work. Rather, Torng remembers being asked repeatedly during presentations to IBM groups at Yorktown, Poughkeepsie, and Kingston, as to how his work related to the Tomasulo method of the Model 91 [personal communication]. Additionally, Harry Dwyer was an engineer from IBM Endicott who went to Cornell for his Ph.D. under Professor Torng. Dwyer worked on and later patented improvements to the dispatch stack. Dwyer never encountered information about ACS during his work at IBM Endicott or at Cornell, or even later at IBM in Austin [personal communication].

– Mark Smotherman, ACS Legacy Page [1]

Yet more questions to explore!

21 What Ifs, and What Might Have Been

I've really been enjoying this hunt, this process of generating theories and trying to "make common sense" of long-ago events. There's still a way to go, of course, and more fun to be had! One never knows what we might all turn up next.

Many of these insights and answers to old questions echo and illuminate the same concerns revealed in my early work on logic design, logic simulation, and the design of the computer design process: No matter how brilliant, we human beings have limited capabilities for visualizing what others might be thinking. We all need tangible models, diagrams, simulations, artifacts, machinery, and so forth, in order to facilitate our communications and converge upon a collective understanding of what we are talking about – especially when designing complex systems.

As a corollary, you can make good bets that the majority of people on occasion, even those on tight-knit top-rank teams, have no clue what their colleagues are talking about. Such 'not knowing' of important things is not a failure on anyone's part. Instead, it is merely a common and natural side-effect of human behavior in groups. It is why we need good tools to help us do our collective work. Meanwhile, a big part of the excitement and fun we have when doing creative, improvisational work as a team is seeing what beautiful, real things can take form out of that chaos.

Unfortunately, ‘not knowing’ can sometime have profoundly negative effects if it goes undetected, as perhaps when Earle’s lack of awareness of DIS contributed to the ACS-1’s demise.

But sometimes ‘not knowing’ can work in other, strange ways. While writing these reminiscences, I suddenly realized that there were significant things that *I didn’t know* about at ACS. And that ignorance has had major effects on my life down through the years – as revealed in this series of ‘what ifs’:

What if I had known, as I now believe, that few at ACS had grasped the significance of the DIS innovation or were aware of my role in that innovation? *What if* I had realized that most of my colleagues thought of me simply as a support programmer, rather than a research contributor?

Imagine the emotional blow, the sense of having deluded myself, and the impact on my confidence after the firing. Imagine the added emotional angst I would have felt when watching DIS being reinvented by others, knowing there’d be no way to ever straighten out its history.

On the other hand, *what if* the reverse had happened, and more or all of the key people at ACS *had* understood DIS and seen it as an important architectural principle? The team might then have been better able to defend the ACS-1, and reveal that Earle’s numbers were flawed. The machine might have been built and become the most famous machine of its day. Or, *what if* Brian’s advice had been taken instead? DIS might have been used in a S/360 compatible machine with similar results.

Ah, but look what would have happened then: In either of those cases my birth name would have become associated with an enabling invention in a successful machine, as had Tomasulo’s with the IBM Model 91. Having a well-known name in computer architecture, I would have faced extreme difficulties when ‘disappearing’ and then restarting my career all over again in the same field and same geographical area. It’s likely that I would have been *outed* and lost everything, all over again.

Fortunately, I dodged those bullets – completely by accident – and confidently went forward into my new career. I knew that I had done important work and that it had become part of a critical machine design. Out of ignorance, I thought the top ACS folks knew that, as well. My deep respect and admiration for my brilliant colleagues at ACS further boosted my confidence, for I thought they had actually seen me as “one of them.”

That confidence propelled me forward in my struggle to build a new career and get back into a top-ranked R&D organization. In the end that struggle led me to Xerox PARC, just in time to help launch the VLSI revolution. That revolution, in turn, enabled DIS to finally come to life in modern high-performance VLSI microprocessors – while I was still around to see it happen [55, 56].

Once in a while, ignorance is indeed bliss!

Acknowledgements

Many thanks go to Mark Smotherman for his outstanding collection of historical information about computer architecture. Mark’s penetrating investigations provide amazing amounts of detail and are a tremendous resource for gaining understandings of project dynamics in earlier times. Thanks too to Brian Randell for sharing his memories of ACS with me, and for the many fun-filled Skype-Video sessions we enjoyed during the preparation of this paper. I’d also like to acknowledge my many ACS colleagues: it was a thrill to join you on the adventure that was ACS, and I hope to connect with more of you in the future. I most especially wish to acknowledge and thank Herb Schorr: Herb took a chance on me years ago, and thus launched my career. I’m forever grateful to him.

References

1. Smotherman, M., “IBM Advanced Computing Systems (ACS) -- 1961 – 1969”, historical reconstruction website, Clemson University.
http://www.cs.clemson.edu/~mark/acs.html

2. Conway, L., Ed., “IBM-ACS Archive”, *lynnconway.com*.
http://ai.eecs.umich.edu/people/conway/ACS/Archive/ACSarchive.html

3. Randell, B., “Reminiscences of Project Y and the ACS Project”, *Technical Report Series CS-TR-891*, School of Computing Science,University of Newcastle upon Tyne, February 2005.
http://www.cs.ncl.ac.uk/publications/trs/papers/891.pdf

4. Smotherman, M., and Spicer, D., “Historical Reflections: IBM’s Single-Processor Supercomputer Efforts – Insights on the pioneering IBM Stretch and ACS projects”, *Communications of the ACM*, Vol. 53, No. 12, Dec. 2010, pp.28-30.
http://dl.acm.org/citation.cfm?id=1859216

5. Evans, B. O., “The Ill Fated ACS Project”: pages 27-28 in Evans’ memoir *The Genesis of the Mainframe*, Wilhelm G. Spruth, ed., University of Leipzig, Department of computer science, June 2010.
http://www.informatik.uni-leipzig.de/cs/Literature/History/boevans.pdf

6. Johnson, M., *Superscalar Microprocessor Design*, Prentice-Hall, 1990.

7. Hennessy, J. L., and Patterson, D. A., *Computer Architecture: A Quantitative Approach*, *2nd Ed.*, Morgan-Kaufman, 1996, Figure 4.60: Recent high-performance processors and their characteristics, p.359.

8. Wilkes, M.V.,"An Experiment with a self-compiling compiler for a simple list-processing language", Tech. Memorandum No.63/1, University Mathematical Laboratory, Cambridge University, Feb. 1963.
http://ai.eecs.umich.edu/people/conway/CSE/M.V.Wilkes/M.V.Wilkes-Tech.Memo.63.1.pdf

9. Leeson, D. N., and Dimitry, D.L., *Basic Programming Concepts and the IBM 1620 Computer*, Holt, Rinehart and Winston, 1962.

10. Smotherman, M., “IBM Stretch (7030) -- Aggressive Uniprocessor Parallelism”, historical reconstruction website, Clemson University.
http://www.cs.clemson.edu/~mark/stretch.html

11. Watson, T. J., Jr., *Father, Son & Co.: My Life at IBM and Beyond*, Bantam [paperback], 1990, pp. 282-283; 315-316.

12. Rozenberg, D., Conway, L., Riekert, R., "ACS Simulation Technique", IBM-ACS, Mar. 15, 1966.
http://ai.eecs.umich.edu/people/conway/ACS/SimTech/SimTech.pdf

13. Conway, L., “MPM Timing Simulation”, IBM-ACS AP #67-115, August 25, 1967.
http://ai.eecs.umich.edu/people/conway/ACS/MPMSim/MPMSim.pdf

14. Shriver, B., and Capek, P., “Just Curious: An Interview with John Cocke,” Computer Magazine, Vol. 32, No.11, November 1999, pp. 34-41.
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00803638

15. Caldwell, S. H., *Switching Circuits and Logical Design*, John Wiley & Sons, 1958; See especially: Chapter 7: Symmetric Functions, p. 236-274; Chapter 8: Synthesis of non-series-parallel contact networks, pp. 274-318.

16. Conway, L., Randell, B., Rozenberg, D., Senzig, D., "Dynamic Instruction Scheduling", IBM-ACS, February 23, 1966.
http://ai.eecs.umich.edu/people/conway/ACS/DIS/DIS.pdf

17. ACS-1 MPM Instruction Manual, IBM-ACS, January 8, 1968.
http://ai.eecs.umich.edu/people/conway/ACS/Archive/Instructions/ACS-1%20Instruction%20Manual.pdf
http://www.cs.clemson.edu/~mark/acs_inst_set.html

18. Smotherman, M.: *File of DIS patent disclosures and correspondence*, personal electronic communications, April-May, 2011.

19. Thornton, J. E., and Cray, S. R., “Simultaneous Multiprocessing Computer System”, U. S. Patent 3,346,851, October 10, 1967.
http://www.freepatentsonline.com/3346851.pdf

20. Conway, L., "ACS Logic Design Conventions: A Guide for the Novice", IBM-ACS, November 29, 1967.
http://ai.eecs.umich.edu/people/conway/ACS/LogDes/LogDes.pdf

21. Garfinkel, H., *Studies in Ethnomethodology*, Prentice Hall, Englewood Cliffs, N.J., June 1967.
http://en.wikipedia.org/wiki/Ethnomethodology
http://www.amazon.com/Studies-Ethnomethodology-Social-political-theory/dp/0745600050

22. Conway, L., "A Proposed ACS Logic Simulation System", IBM-ACS, October 31, 1967.
http://ai.eecs.umich.edu/people/conway/ACS/LSS/LSS.pdf

23. Conway, L., “Timing Simulator Source Code Listings”, IBM-ACS, August 1967.
http://ai.eecs.umich.edu/people/conway/ACS/Archive/ACS211-327.pdf

24. Conway, L., “MPM Architecture and Simulator Notebook”, IBM-ACS, August 1967.
http://ai.eecs.umich.edu/people/conway/ACS/Archive/ACS93-210.pdf

25. Cocke, J., “The Search for Performance in Scientific Processors,” Turing Award Lecture, CACM, Vol. 31, No. 3, pp. 250-253, March 1988.
http://dl.acm.org/citation.cfm?id=42392.42394

26. Allen, F., “The history of language processor technology in IBM,” *IBM Journal of Research and Development*, Vol. 25, Issue 5, September 1981, pp. 535-548.
http://dl.acm.org/citation.cfm?id=1664867

27. Conway, L., "The Computer Design Process: A Proposed Plan for ACS", IBM-ACS, August 6, 1968.
http://ai.eecs.umich.edu/people/conway/ACS/DesProc/DesignProcess.pdf

28. Zurcher, F.W., and Randell, B., “Iterative Multi-Level modelling: A methodology for computer system design,” *Proc. IFIP Congress 68*, Edinburgh, 1968, pp. D138-D142.
http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

29. Wirth, N., “Program development by stepwise refinement”, *CACM*, Vol. 14, No. 4, April 1971, pp. 221-227.

30. Dijkstra, E. W., “GOTO statement considered harmful”, CACM, Vol. 11, No. 3, May 1966, pp. 366-371.

31. Dahl, O. J., Dijkstra, E. W., and Hoare, C. A. R., *Structured Programming*, Academic Press, 1972.

32. Brooks, F. P., Jr., *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, 1975.

33. Conway, L., “IBM License to Make and Distribute Copies of Lynn Conway's ACS Archive”, lynnconway.com, August 23, 2000.
http://ai.eecs.umich.edu/people/conway/ACS/IBM/IBM-License.html

34. Mead, C., and Conway, L., *Introductions to VLSI Systems*, Addison-Wesley, 1980.

35. Conway, L., *The M.I.T.'78 VLSI System Design Course: A Guidebook for the Instructor of VLSI System Design*, Xerox PARC, Aug. 12, 1979.
http://ai.eecs.umich.edu/people/conway/VLSI/InstGuide/InstGuide.pdf

36. Marshall, M., Waller, L., and Wolff, H., “The 1981 Achievement Award: For optimal VLSI design efforts, Mead and Conway have fused device fabrication and system-level architecture,” *Electronics*, Oct. 20, 1981.
http://ai.eecs.umich.edu/people/conway/VLSI/Electronics/ElectrAchievAwd.pdf

37. Conway, L., Bell, A., and Newell, M.E., “MPC79: The Large-Scale Demonstration of a New Way to Create Systems in Silicon”, *Lambda*, Second Quarter 1980.
http://ai.eecs.umich.edu/people/conway/VLSI/MPC79/MPC79Report.pdf

38. MOSIS, “The MOSIS Service – More than 50,000 designs in over 25 years of operation”, *mosis.com*.

39. Conway, L., “The MPC Adventures: Experiences with the Generation of VLSI Design and Implementation Methodologies”, Xerox PARC Tech. Report VLSI-81-2, Jan. 1981. http://ai.eecs.umich.edu/people/conway/VLSI/MPCAdv/MPCAdv.pdf

40. Computer Science and Telecommunications Board, National Research Council, *Funding a Revolution: Government Support for Computing Research*, National Academy Press, 1999; Chapter 4, pp. 113-122. http://www.nap.edu/catalog.php?record_id=6323

41. Carroll, P., *Big Blues: The Unmaking of IBM*, Crown Publishers, Inc., 1993.

42. DiCarlo, L., “How Lou Gerstner Got IBM To Dance”, Forbes Magazine, November 11, 2002.

43. Acosta, R. D., Kjelstrup, J., and Torng, H. C., “An Instuction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors”, *IEEE Transactions of Computers*, Vol. C-35, No. 9, September 1986, pp. 815-828. http://ipa.ece.illinois.edu/mmm/papers/Acosta.1986.TC.pdf

44. Steele, B., “CU professor is honored by Intel Corp. for his computer chip invention”, *Cornell Chronicle*, Dec. 11, 1997. http://www.news.cornell.edu/releases/Dec97/Torng.bs.html

45. Steele, B., “Cornell wins $184 million award from Hewlett-Packard for patent infringement”, *Cornell Chronicle*, June 6, 2008. http://www.news.cornell.edu/stories/June08/HPpatent.ws.html

46. Smith, J. E., and Sohi, G. S., “The Microarchitecture of Superscalar Processors.,” *Proceedings of the IEEE* Vol. 83, No. 12, December 1995, pp.1609 - 1624.

47. Agerwala, T., and Cocke, J., “High Performance Reduced Instruction Set Processors," IBM Watson Research Center, RC 12434, 1987.

48. Computer History Museum, Event: “IBM ACS System: A Pioneering Supercomputer Project of the 1960's”, February 18, 2010. http://www.youtube.com/watch?v=pod53_F6urQ

49. IEEE Computer Society, “Lynn Conway, 2009 Computer Pioneer Award Recipient,” January, 2010. http://www.youtube.com/watch?v=i4Txvjia3p0

50. Hasbrouck, L., Madden, W., Rew, R., Sussenguth, E., and Wierzbicki, J., “Instruction execution unit,” U.S. Patent 3,718,912, February 1973. http://www.freepatentsonline.com/3718912.pdf

51. Aspray, W., Interviewer, *Gene Amdahl Oral History*, CHM Reference number: X5988.2011, Computer History Museum, September 24, 2000, p.26. http://archive.computerhistory.org/resources/access/text/Oral_History/102702492.05.01.acc.pdf

52. DeLamarter, R. T., *Big Blue: IBM's Use and Abuse of Power*, Dodd, Mead and Co., 1986.

53. Schorr, H., “Design Principles for a High-Performance System,” *Proceedings of the Symposium on Computers and Automata*, Polytechnic Institute of Brooklyn, April 13-15, 1971, pp. 165-192. http://ai.eecs.umich.edu/people/conway/ACS/People/Herb_Schorr/Schorr1971_ACS_Reflections.pdf

54. Hennessy, J. L., and Patterson, D. A., *Computer Architecture: A Quantitative Approach*, *4nd Ed.*, Morgan-Kaufman, 2007; Chapter Two: Instruction-Level Parallelism and Its Exploitation, pp. 64-151.

55. Hiltzik, M.A., “Through the Gender Labyrinth: How a bright boy with a penchant for tinkering grew up to be one of the top women in her high-tech field”, *Los Angeles Times Magazine*, Cover story, Nov.19, 2000. http://articles.latimes.com/2000/nov/19/magazine/tm-54188 http://ai.eecs.umich.edu/people/conway/Media/Through%20the%20Gender%20Labyrinth.pdf

56. Conway, L., Ed., “The VLSI Archive: An online archive of documents and artifacts from the Mead-Conway VLSI design revolution”, *lynnconway.com*. http://ai.eecs.umich.edu/people/conway/VLSI/VLSIarchive.html