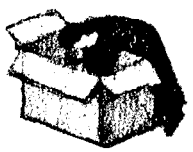
Case Study: IBM'S SYSTEM/360–370 ARCHITECTURE

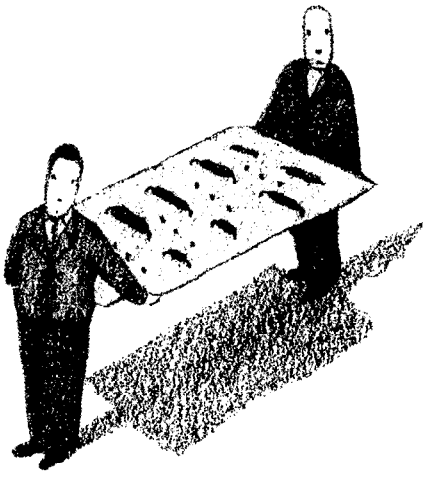


Why has the IBM System/360–370 architecture been so successful for so long? Case-study editors David Gifford and Alfred Spector sought to answer this question by talking to Andris Padegs and Richard Case, two of the key people responsible for the 360/370 architecture, in Cases's IBM office in Thornwood, New York, on August 8, 1986.

As the study shows, instruction-set design constituted only a small fraction of the effort behind the development of the 360/370 architecture. Padegs and Case describe the architecture and its specification, and recount its evolution. They emphasize that the architecture's success can be attributed to a sound initial design, but also to the precision of its definition, the care that has been taken to ensure that implementations meet architectural specifications, and the procedures by which improvements have been managed. And, although the architecture has not survived unchanged since its introduction in 1964, the initial concept was sufficiently good to allow IBM to engineer smooth customer transitions to enhanced versions.

Near the end of the interview, Padegs and Case comment on multiprocessing, reduced-instruction-set computers, and address space limitations. IBM's largest current computers are all 360/370 family multiprocessors; in order to support multiprocessing, the architecture had to be revised to precisely specify the properties of memory accesses. With respect to reduced-instruction-set computers, Padegs and Case report that the common 360/370 instructions execute in a single machine cycle. They also describe their empirical observation that the architecture has needed addressing at a rate of about one bit every 30 months, and comment on the implications of this observation for the future of the architecture.

The editors observe that the careful standardization process used by the architects is applicable outside the realm of hardware. With ever greater investments in user and system interfaces, standardization and controlled evolution may be even more important for software. The example of the efforts of the 360/370 architects may benefit software implementors who want to build and maintain enduring systems.



CASE STUDY: IBM'S SYSTEM/360–370 ARCHITECTURE

The architecture of IBM's System/360–370 series of compatible processors is one of the most durable artifacts of the computer age. Through two major revisions of the product line and 23 years of technological change, it has remained a viable and versatile interface between machine and user.

DAVID GIFFORD and ALFRED SPECTOR

INTRODUCTION TO THE SYSTEM/360–370 ARCHITECTURE

DG **As a starting point, can you tell us what the scope of the 360 architecture is?**
Padegs The architecture of a computer is the interface between the machine and the software, though not everything in this interface is significant to the program and needs to be specified. When we defined the System/360, we identified in the Principles of Operation the parts of the architecture that could differ among models. It might appear to the casual observer that imprecision is a weakness of the architecture. Our experience with new models and features confirms that it is very important not to define everything precisely. Unnecessary specificity can add significant cost to present and future implementations of an architecture. Thus it is important to determine which pieces should be specified and which should be left unpredictable.

AS **Why "unpredictable" rather than "undefined"?**
Padegs Both terms are acceptable, but we wanted to emphasize that the architecture is, in fact, defined; the definition, however, specifies the operation to be unpredictable. That is, the operation may vary among models, and on any one model may differ among instances. Otherwise someone could ask when we are going to complete the definition.

AS **Can you give us an example of something you chose to leave unpredictable?**
Padegs The result of the diagnose instruction is one example. The instruction is used for various maintenance functions, and each model has a separate specification for its operation.

Case There are also instructions where it is unpredictable which of two or more cases, each with its specific result, will apply when a particular instruction is called. Take the character string comparison operation. The operation starts at the beginning of both strings and compares them character by character, and stops when it finds a difference. If the character strings happen to cross a protection boundary, then the operation may or may not cause a protection exception, depending on the data in the strings. However, the architecture will allow a model to generate a protection exception even if the protected data are not necessary for completing the string comparison. We argued to ourselves that this case represents a program bug, and so there's really no harm in raising the protection exception.

Padegs However, when it matters, we will go to a fair amount of work to make the results of an instruction well defined. For example, in the presence of virtual memory we actually may perform a trial execution of the edit instruction in order to ensure that all of the pages that the instruction needs are

available in memory. Once the dry run succeeds without generating a page fault exception, then we perform the actual execution. We introduced this rather cumbersome requirement in System/370 for the sake of compatibility. Normally we check for the required pages on the basis of the designated operand lengths. But the edit instruction can be successfully executed without use of the entire designated source operand; in this case the required size of one of the operands depends on the value of the other. We wanted to make sure that a 360 program that placed the required part of this operand against the end of storage would not cause an unnecessary page fault on a 370 machine.

DG **Where did the idea for a unified processor architecture come from?**

Case There was an internal development project at Poughkeepsie in the years just before 360 that was known as the 8000 project—to the best of my knowledge, by the way, that was when the word "architecture" was first used in computing. People working on that project believed that they could define a single software interface that was common across three different machines with a total performance span of approximately 7 to 1. The advantage they foresaw was in having a single system software development effort and a single application program development effort that would be useful for the three machines. Even though the 8000 was never built, some of the same people were involved in the 1961 SPREAD Committee, which was trying to come up with a new product line for IBM. The SPREAD Committee applied the idea of a unified processor

System/360 in Context

IBM's System/360 was one of the most ambitious projects in the history of the computer industry. First announced in 1964, System/360 was the first line of processors with both upward and downward compatibility, which meant that any program that ran on the most powerful machine would also run on the least powerful, as well as vice versa. It was also the first major product line designed for both business and scientific applications. Twenty-three years later, the architecture pioneered in System/360, and extended to System/370 in 1970 and to System/370-XA in 1983, continues to be widely used. The Soviet Union provided one of the more unique testimonials to the success of System/360 by functionally duplicating several of its models to handle the bulk of its large mainframe computing needs [12].

System/360 required an enormous commitment of resources, both financial and human, and was therefore a gamble for IBM. For the gamble to succeed, it was necessary that the architecture remain salable for upwards of a decade. The technological challenge was to anticipate the needs of users several years down the line without pricing the machines out of the market, or making them too complicated or inconvenient, in the meantime. For instance, IBM would have liked to use integrated circuits in System/360, but found that the technology was not quite ready. The best alternative was a hybrid microminiaturization effort to standardize and streamline circuit technology by using as much solid logic technology as was viable at that time. Microprogramming, which had been pioneered by Maurice Wilkes at Cambridge University in the late 1950s, was used to implement upwards and downwards compatibility across different models of System/360.

A major challenge for the architects of System/360 was a suitable addressing strategy. The report of the SPREAD Committee, which worked out the long-range plan for System/360 in 1961, called for an increase in main memory by two to three orders of magnitude. To do this with the direct addressing approach current at that time would have been inefficient, since the 24-bit addresses that it would have taken to address 16 million memory positions would have affected both storage capacity and performance. The architects developed "base-register offset addressing" to allow 12-bit addresses within a contiguous memory area; they found that instruction sequences would remain within the necessary memory area in a sufficiently large number of cases to justify this approach.

In designing a system with upward and downward compatibility for both scientific and business customers, IBM was attempting to use a single architecture to meet the needs of an unprecedentedly large segment of its customers. To support this effort, large-scale efforts to develop compatible peripheral interfaces and software were necessary. System/360 was designed to be able to handle both decimal and binary formatted information, with both variable-field length and floating-point arithmetic capabilities. Since scientific users tended to use Fortran and business users tended to use Cobol, IBM designed and developed the PL/1 programming language in an attempt to provide a programming bridge between the two communities.

It would be difficult to render judgment as to the extent to which IBM was able to meet each of its individual goals for System/360. Some features were not as successful as others. PL/1, for instance, did not supplant Fortran and Cobol. But the bottom line is impressive: "In the six years from 1965 to 1971, IBM's annual gross income increased 2.3 times from $3.6 billion to $8.3 billion, and net earnings after taxes increased 2.3 times, from $477 million to $1.1 billion. In 1982, the descendants of System/360 accounted for more than half of IBM's gross income."[1]

Robert Papsdorf

[1] This quote is from "System/360: A Retrospective View," by B. O. Evans [5]. IBM asked that the words "In the five years from January 1, 1966, until January 1, 1971, ..." in the original be changed to "In the six years from 1965 to 1971, ..."

architecture to IBM's business problem of serving a diverse set of customers and postulated the 360 line of computers [3, 6, 7]. The original plans called for a range in performance of 20 to 1 over six different designs, including the model 90/91, but it was hoped that the performance spectrum would eventually be broader than that. I don't think any of the people that participated in the SPREAD Committee were able to anticipate what the range would be 22 years later. Today the 3090-400 is over 2000 times faster than the System/360 Model 30.

At the time of the SPREAD Committee, there was a big debate about whether 360 compatibility should be uni- or bidirectional. Everybody accepted the objective that any valid program that ran on a smaller machine should also run on a larger one, but it wasn't obviously true that the reverse would also hold. The initial commitment to do it both ways was received by a majority of the IBM technical community with a great deal of skepticism. In fact, only the appearance on the scene at about the same time of microprogram technology provided a way of implementing bidirectional compatibility.

RICHARD CASE

Richard Case has been involved with many aspects of the 360–370 family during his career with IBM. He started as a large systems logic designer in 1956, and worked on the 1410, the 7040, and the 7044 computers before becoming an engineering manager on the early version of the 360 Model 65. He went on to assume overall responsibility for language compilers for the 360 operating system in 1962, and became assistant manager for the OS/360 project in 1964. From 1966 to 1971, Case had overall division-level responsibility for 370 architecture and system performance evaluation. He was manager of the FS project from 1971 to 1974 and then spent two years in the research division. He went on to become involved in developing the 4300 line and later spent a year on packaging technology for the 3081. Case then became a development executive in the semiconductor division until he went back to Endicott in 1982 to oversee the announcement of the 4381 and VM version 4. In 1984 he was promoted to his present job as IBM director of Technical Personnel Development.

Padegs We were in effect replacing several successful lines: the 701-704-709-7090-7094, which included also the 7040-7044; the 702-705-7080; the 650-7070-7074; and the 1401-1410-7010. We recognized that a large part of the computer usage was common to both scientific and commercial applications—for example, assembling, compiling, and operating-system functions. We had to convince the advocates of the specialized lines that the new architecture was going to meet their specific needs. Compromises had to be made. The edit instruction was introduced as part of the decimal package to placate those who felt that an earlier single-architecture proposal was biased in favor of the scientific users. In the earlier System/360 models, the decimal and floating-point instructions were made optional features, with the expectation that they would only be installed for the specialized applications. In the 370-XA these facilities are part of the basic architecture, and all current models offer them. During the development of the 360, the key was making the case that we had in fact succeeded in developing an architecture that was good for both scientific and commercial applications.

AS **Did you ever add instructions to help justify the machine that in retrospect were not really necessary?**

Case Yes. The edit instructions are one clear instance. There are some other examples that are debatable.

DG **How many distinct interfaces were published when the 360 was announced?**

Padegs There were two model-independent interfaces specified for the 360 line. The first interface was the instruction set; the second was the channel-to-control-unit electrical specification and signaling protocols. The latter was standardized so that a single set of peripheral devices could be designed to work across the entire range of models. The channel-to-control-unit interface has some interesting and important features: It allows I/O equipment to be interchanged between systems; it also has some effect on the software, since the commands and the status indications are defined to be consistent. This makes a compatible or common programming system for I/O control possible. The channel-to-control-unit interface was made public at the same time as the 360 Principles of Operation.

AS **What distinguishes a machine from an architecture?**
Case A model is a realization and an implementation of an architecture, whereas a model is an electrical and physical design, with a production cost, a production schedule, and a price.

Padegs An architecture is a reflection of a conceptual model. A machine is the physical embodiment of the architecture.

AS **Did the fact that you originally announced a spectrum of models help your architecture?**
Padegs I would say it helped. Yes.

Case I would say it's essential.

Padegs [*laughs*] Then maybe it's essential. At least it's essential initially, since it enforces the required discipline. If you build a single machine, there is always a temptation to optimize the architecture for the machine you are building. If you are building a collection of machines from low to high performance, you are forced to take a broader view.

DG **Did your 20-to-1 performance range help your architecture?**
Padegs It did, since it forced us to look at the specific implementation issues for the different types of machines. One example is the byte addressability of most operands. This was motivated by the Model 30, but also simplified programming on the larger models.

DG **If you had not had a low-end machine to start, would the architecture of the 360 have been different in some way?**
Case I know one thing exactly that would have been different if we had not had such a low-end machine: We would never have had the packed decimal formats in the machine. The only reason for packed decimals was to allow the Model 30's one-byte-wide data path to process two digits at a crack rather than one. Packed decimal formats made a substantial enough difference in the decimal performance of the Model 30 that we put them into the architecture. Without a one-byte-wide machine, I believe we would never have had the packed decimal data format.

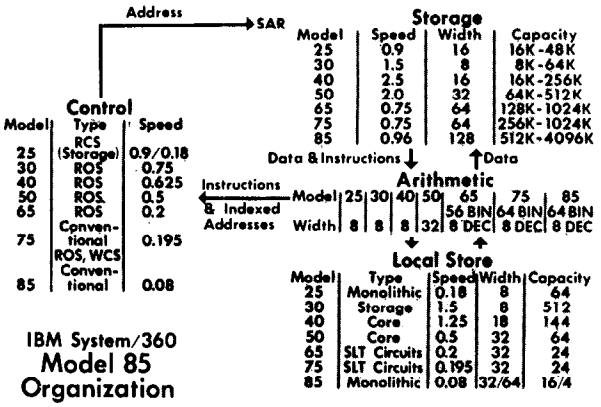
AS **Has your goal of having a collection of compatible machines actually worked out in practice?**
Case I think the record shows that we have successfully developed an architecture that provides compatibility across models. If you go out looking for examples of an application program that runs on one model and does not run on another model, you just don't find them. Professional architects might be able to write programs that will run on a 145 and fail on a 3033, but in practice people don't run into that problem.

ANDRIS PADEGS
Andris Padegs is manager of the department responsible for the System/370 architecture at IBM, and has been involved with the 360–370 architecture throughout its entire 23-year lifetime. He joined IBM in 1958 to work on the Stretch Project. In the early 1960s, he was responsible for defining the System/360 I/O architecture, and he wrote the channel part of the System/360 Principles of Operation. Later he switched into the CPU architecture group, becoming its manager in 1968. Since then he has worked on extensions to System/370 architecture and on the development of other computer structures. In 1975 he assumed his current position as the manager of the central systems architecture department. He has received seven formal awards from IBM for his contributions, including awards for outstanding invention and invention achievement. Padegs has published over 10 papers on computer architecture and organization and holds four patents, including one on the System/360–370 I/O interface.

DG **How many 360/370 compatible machines and different models have actually been manufactured in the 22 years since the architecture was announced?**
Padegs IBM has built around 50 different models. As an order of magnitude, I would estimate that 100,000 machines have been manufactured.

AS **Is the vector processing option included in the architecture?**
Padegs Yes. The vector feature was introduced early in 1986, and the function is offered as an option on the 3090. This architecture extension

is intended for intensive engineering and scientific applications that process arrays of data. It includes 171 new instructions and 16 special vector registers, which on the 3090 contain 128 four-byte words each. By comparison, it is interesting to note that the original System/360 Principles of Operation contained 143 instructions; for the 370-XA Principles of Operation published in 1983, this number had grown to 208. But the structure of the vector unit is simpler than the count of 171 new instructions suggests, because each arithmetic and logical operation is available for several different operand types and instruction formats.



The System/360 line in 1968 included seven models with a performance range of 100 to 1.

DESCRIPTION OF THE ARCHITECTURE

DG **What are the primary documents that describe the architecture of the 360 and 370 family?**
Padegs The Principles of Operation is the key document; so far there have been four major versions. The 360 Principles of Operation was first published in 1964, and the 370 Principles of Operation was first published in 1970 [8]. The third version was an offshoot for the smaller models called 370 virtual storage extended [9]. The fourth was the extended architecture (XA) Principles of Operation [10], first published in 1983. All large models we build today have the XA architecture. Yet another separate document [11] describes the hardware protocol between channels and control units.

AS **How frequently do updates have to be made to the Principles of Operation?**
Case We issued 10 editions in the first eight years, which was slightly more than 1 edition a year. We do it less frequently now—we're getting better.

AS **Was a document ever written to describe why certain things were done, as opposed to how they were done?**
Case There have been two attempts at that. The first was in a paper by Amdahl, Blaauw, and Brooks in the *IBM Journal of Research and Development* in 1964 [1]. The second was an article that Andris and I did in *Communications of the ACM* in 1978 to motivate the 370 architecture [4].

Padegs There are a few more papers on such topics as channels, extended-precision floating point, virtual machines, and the extended architecture. We have probably published a total of a dozen papers on the motivation of the architecture [see references].

DG **We get the impression that the sum of IBM's experience with this architecture is embodied in the Principles of Operation. Is this true?**
Padegs No, some parts such as the specifications of interpretive execution and vector operations are actually published in separate manuals.

Case From a conceptual point of view, the architecture is described in a single document that's published in a number of volumes. The volumes are consistent in style, presentation, and terminology.

AS **The Principles of Operation is a fairly short manual—570 pages. How can you describe the architecture in so few pages?**
Case It is because of the level of professionalism that is dedicated to architecture. The architecture is documented by the same people who are responsible for its content. They are dedicated to making the description precise and succinct. The style was established by Gerry Blaauw and Andris Padegs with the publication of the first System/360 Principles of Operation, and it has been continued ever since.

Padegs I would like to add that the page count is low because it is a reference document with very little tutorial material. It contains only what is essential, and we try to avoid repetition. For example, we have a certain pattern and structure that describes common attributes of instruction execution once, so they do not need to be repeated in the description of every instruction.

AS **If you were going to design another computer system, how would you document its architecture?**
Padegs Well, we have discussed this a number of times, and the question keeps coming up. In the early 1960s, we worked with Falkoff and Iverson on an APL description of the architecture [2]. We decided against using APL in the official description of

the architecture, and I think that we will stick with English in the future. There are a number of reasons for doing so. First, a formal language such as APL may be more precise, but what we really need is flexibility. We would like to say precisely what we mean when we want to make precise statements, but we also need to say things that can't be said very well in a formal language. Second, we have a principle of having a single definition of the architecture. If this single definition was in a formal language, what would its audience be? There might be some people who would appreciate the beauty of a formal definition, but it is difficult to teach the whole community to use a formal language. Thus we provide an English version, and if we permitted two definitions, it's possible that they would be inconsistent in some way.

AS **Can I write an operating system for a 360/370 computer solely from the Principles of Operation?**
Case The manual does not propose to teach somebody who doesn't know anything about operating systems how to do so. But, for someone who's already written an operating system, or who knows how, the manual would be sufficient.

DG **Are there ever model-dependent features in the architecture, and if so, are they public?**
Case Yes, individual model descriptions are used to describe public model-dependent features. There are also certain model-dependent features for which the definition is not public. For example, there are functions built into almost all of our models that we believe are there only for the purpose of enabling us to do error detection and fault isolation.

AS **Does the model-dependent information pertain only to fault finding?**
Case Well, that's the biggest piece of it. We do have some assists that we have implemented to help certain pieces of our operating systems go faster. In general, the operating systems did not require these assists to run. They just ran a little bit better or a little bit faster. The assists are often experimental attempts to improve performance on some processors. As we learn more about the system, we introduce general-purpose extensions. For example, we replaced the assists for VM by a more general-purpose interpretive execution facility. All XA models that we build now include the interpretive execution facility.

THE ARCHITECTURAL PROCESS

DG **How do you establish a customer requirement for a new architectural feature?**
Case In 1964 there was no identifiable customer requirement for a line of compatible processors that did commercial and scientific processing equally well. At least you couldn't have found that requirement by listening to customers. We had to dig deeper, to establish who our customers were, what their applications were, and how they would be likely to react to an entirely new product. Asking customers what they want is important, but its not always possible to work on such a direct and literal level.

AS **How do you agree on a new architecture?**
Padegs The answer to this question is different today from what it was when we first designed the 360. In the early 1960s, we started from scratch, and we had to reason from first principles. We needed to incorporate a complete set of functions that would meet the needs of application programs and make it possible to design an operating system. Today we have a well-established architecture, programming systems, and people who are using the architecture. We look at what is missing for today's applications, and we consider how to extend the architecture. We have considered extensions for both the problem state and the supervisory state. In fact, in the late 1960s we did a user survey to gather suggestions for possible extensions. We received 164 proposals for new instructions, this at a time when we only had 143 instructions in the actual architecture. We narrowed these 164 proposals down to 20 or 30 viable instructions and finally included half a dozen of them in the 370 architecture.

AS **What types of analysis have you done of the use of the 360/370 architecture, and how has this analysis affected the implementation of specific models?**
Case When we started to design System/360, we looked at extensive instructions traces from past ar-

If you build a single machine, there is always a temptation to optimize the architecture for the machine you are building. If you are building a collection of machines from low to high performance, you are forced to take a broader view.

chitectures. When we were developing System/360, though, so much was changing that we could not rely exclusively on these traces. Therefore we also coded up common instruction sequences that were anywhere from about 4 to about 40 instructions long. We called these kernels. In addition to coding kernels with the basic instruction set, certain kernels were also coded to take advantage of different options. We then analyzed the performance of the kernels. I should point out, though, that kernels also have problems. It's necessary to weight the importance of the different kernels, and kernels can be written in many different ways, which can influence the results. Finally, kernels can make you forget about all the other instruction sequences that are out there.

By now, IBM has extensive information on the utilization of the 360 architecture. In fact, in some cases the most sophisticated machine designers will actually go out and collect their own data for some specific purpose when it can't be distilled from the existing tapes.

Padegs It would seem that today we should be able to measure precisely the value and cost of a new instruction—the value in terms of the performance improvement to some kernel, and the cost in terms of the additional circuits and microwords. But it is not that simple. The amount of effort and cost a designer is willing to commit to the new function depends on its anticipated use. The use, in turn, depends, on performance. Thus we have to break the circle. In some cases a number of "steady-state" design points might be possible, depending on agreements reached. However, for some instructions, such as those for setting and testing bits, the potential for usage is so pervasive that it is not possible to project a meaningful usage frequency. For other instructions, such as those for operations on list structures, the justification cannot be based on where the new instructions would be used in current programs, but rather on what new applications and program structures the new instructions would make feasible.

Case Early in the design of a machine, we use very sophisticated machine simulators, along with workload or trace tapes, to determine the performance of a new model to three or four significant digits. The only reason we compute four digits is so we can divide performance by cost figures in order to rank features. This gives us a pretty good idea of what we should include in the model.

DG **Once a machine is designed, do you have simulators that allow you to verify that it properly implements the architecture, even before the machine is actually built?**

Case Yes, absolutely. For example, on the 3090 someone added up all of the simulated running time that we accumulated on the 3090 before the first chips were cut, and it amounted to several seconds of CPU time. It is now beginning to be possible to use simulation to run noticeable amounts of real time at the application level even before a machine is built.

DG **Once a machine is built, what sorts of programs do you run to validate its implementation of the architecture?**

Case The programs that we really depend on are special programs that check out all the subtle and obscure parts of the architecture specification. This is in addition to the main-line operating systems, together with a wide collection of application programs. We run such programs for hundreds if not thousands of hours on a prototype of a new machine before the machine is committed to replication.

DG **Have you noticed the average instruction mix shifting in any significant directions over the last 20 years?**

Padegs I cannot think of any specific changes. But I can think of two factors that would affect the distribution. First of all, the instruction set and the relative performance of instructions have changed over the years. We have increased the number of instructions by almost 50 percent, and each of the new instructions displaces some current usage. Additionally, there is a gradual evolution in the choice of instructions for a particular function. It is an iterative process: Machines are designed to improve the performance of the most common instructions as determined by the latest measurements, and the new kernels are coded in view of the relative performance of instructions on the latest models. Clearly, changes in technology have had a major impact here.

But, considering the mixes as indicated by kernels, there is also a shift in the type of work the kernels are reflecting. Many of our kernels now are based on MVS usage of the machine, and these kernels reflect the evolution of the operating system to virtual storage, new program linkage conventions, and multiprocessing. Thus you will see now the use of compare and swap instead of some general-purpose instructions for the management of locks.

Case My view is that the variations from workload to workload are a lot larger than the changes from year to year.

DG **Could you describe the process that you use to arrive at architectural decisions?**
Padegs Let me answer your question by reviewing the development of the XA architecture in the late 1970s. The XA architecture was developed by a couple of teams of people, each specializing in a certain area, such as channels, virtual machines, and addressing. Each team consisted of half a dozen people and included one or two architects, along with machine designers, programmers, and planners. The architects were in charge of the final conclusion and would probably be considered the leaders. A design would start with a consideration of what the programmers wanted and what the engineers could provide. A memo would then be written stating one possibility for extending the architecture. The memo would initially be only a few pages long, but over time it would be revised to include more detail. Any conflicts that arose were negotiated, and eventually a group proposal resulted. Finally, any incompatibilities between the proposals generated by different teams were resolved.

Case It's also necessary to put multiple iterations in the scenario because design is not a linear process. It's either done in multiple iterations or at succeeding levels of detail, until you finally converge on something you're going to do. It's a collaborative building process that uses estimates of cost, schedules, performance, and so on.

Padegs Once each team evolved a proposal, it was documented, polished, and negotiated to the point where it was detailed and complete enough to suit everybody. The proposal was then given to the architecture board to go through a process called adoption. The architecture board includes representatives from all affected machine and programming groups. Once there were no objections to the proposal from the architecture board (or when the objections were such that we could override them), we adopted the proposal.

AS **Why have there been relatively few revisions to the architecture over the 22 years?**
Case There's a pretty broad consensus that there's hardly anything more to be gained by tweaking the core problem-state instructions. If you want to do something to the problem-state instruction set, it had better result in a big enough performance change to be worth it—the vector facility is one example where we thought that it would be worth the trouble.

Padegs I certainly agree with Richard that there is no point tweaking general-purpose instructions like loads and stores. Too much software needs to be changed to realize any performance benefits. But new concepts and technologies may justify extensions for specific applications, such as sorting. We did this for XA. In this case a change in one software product—the sort program—to make use of the new machine capabilities led to significant performance improvements in the customers' applications. And the supervisory-instruction set is always evolving.

AS **Does a feature ever remain in the architecture even though it may not be necessary?**
Case That happens. Consider storing ahead into the instruction stream. The way the architecture is defined, that case must work properly. On a 3090 the execution time penalty for storing ahead into the instruction stream is very big if it happens, but the hardware cost to detect a store ahead is relatively small. Nobody cares about the time penalty because it never happens. It's almost to the point where it's cheaper to keep building the machines with a store-ahead detection circuit than it is to wonder whether or not it's going to hurt somebody.

EVOLUTION OF THE ARCHITECTURE

DG **Earlier you stated that when you defined the architecture you purposefully left some things undefined. Did you ever make a mistake and leave something undefined that should have been defined?**
Padegs The best examples of that are in the floating-point instruction set. We initially defined floating point such that overflow resulted in an unpredictable value; models could vary as to what they provided for the result value on overflow. It turned out that many of our customers wanted to be able to recover from overflows. The Model 65, unlike other 360 models, actually produced a useful result on overflow. That seemed to be the right way of handling overflow. In fact, people claimed that the other models were wrong. We looked at the problem, and we felt there was enough customer interest to change all of the machines in the field. So we changed the architecture to produce a useful result on floating-point overflow. The result that we defined could be used to scale the operands and to proceed with the operation. A second problem in the floating-point instruction set was remedied by adding a guard digit during postnormalization shifting. Without the guard digit, numbers that were very close together could result in very large errors.

AS **You changed all of the machines in the field?**
Padegs Yes. This cost IBM a substantial amount of money. The problem was caused by a lack of knowl-

Evolution of System/360–370 Architecture

The System/360 architecture was introduced on April 7, 1964, with the announcement of the first six models. In the next couple of years, new models at both ends of the performance range introduced architectural changes to meet particular cost and performance goals. The Model 20, although nominally part of the System/360 family, was incompatible. It offered only 37 of 143 instructions, had 8 instead of 16 general registers, and differed in other ways. At the high end of the performance range, Models 91, 95, and 195 introduced some deviations to accommodate highly overlapped designs by delaying program interruptions and permitting the result of the divide operation to be off by one bit in the low-order bit position. Additionally, Model 44 had a number of extensions for real-time applications. None of these special functions, however, was continued in later models. The time-sharing functions introduced on Model 67 were not continued in that form on the subsequent models, but they were the precursors of virtual storage on System/370.

In 1968, as part of the extended-precision floating-point facility on the Model 85, the 128-bit floating-point format, including instructions for rounding to the next smaller format, was introduced. Model 85 also removed the original System/360 requirement that storage operands of unprivileged instructions be aligned on boundaries equal to a multiple of the operand length, and introduced on the 2880 Block Multiplexer Channel some of the I/O extensions later made part of System/370.

When the next set of changes was introduced, with the announcements of Models 155 and 165 in June 1970, the architecture was renamed System/370. The main architectural extensions were six general-purpose instructions (move long, shift decimal, etc.), the time-of-day clock (with a period of 143 years and resolution of 1 μs), and control registers (to serve as an extension of the program status word (PSW)). The USASCII-8 facility was deleted.

Virtual storage, the single item that most distinguishes System/370, was introduced with the announcements of Models 158 and 168 in August 1972. It includes dynamic address translation and channel indirect data addressing. A number of other extensions were introduced at this time: the CPU timer, the clock comparator, program-event recording (for software debugging), and the new PSW format and interruption controls associated with extended-control (EC) mode. Multiprocessing (CPU identification, program-settable prefix, and the signal-processor instruction) and the conditional-swapping and PSW-key-handling instructions were introduced in February 1973.

After termination of the future-systems project, in the mid-1970s, extensions of the System/370 architecture branched off into two new directions, as additional extensions to the original System/370 architecture were also being introduced. In 1978, with the shipment of the 3033, System/370 was extended to improve MVS performance and availability (low-address protection, the invalidate-page-table instruction, etc.). In 1979, a facility was made available on 3033 multiprocessing systems that permitted switching channels between CPUs. In 1981, the virtual-storage architecture on the 3033 and 308X machines was enhanced by the installation of the dual-address-space facility, which includes a 16-bit address-space number. A total of 64K 16-Mbyte address spaces can be designated, although at any one time addressability exists only to two spaces—the primary and the secondary. Instructions are provided for establishing addressability, calling and returning from programs either in the same or another address space, and performing other functions. The main-storage address at this time was extended to the equivalent of 26 bits (64 Mbytes) by the use of two unassigned bit positions in the page-table entry. In the channel, queueing of I/O instructions was provided.

The architecture for the smaller models (the 4300 processors) was developed to meet the needs of the DOS/VSE operating system. The key distinction was the introduction of an address-translation mechanism that uses a single level of page tables, located in internal machine storage, to replace the segment-table/page-table structure of System/370 and to make I/O work with virtual addresses. Since in the VSE mode DOS/VSE offers a single virtual address space of 16 Mbytes, the architecture is simplified by elimination of the multiple-address-space capability of System/370. Facilities for multiprocessing were also deleted. The Principles of Operation for the VSE mode was published in January 1979 [9].

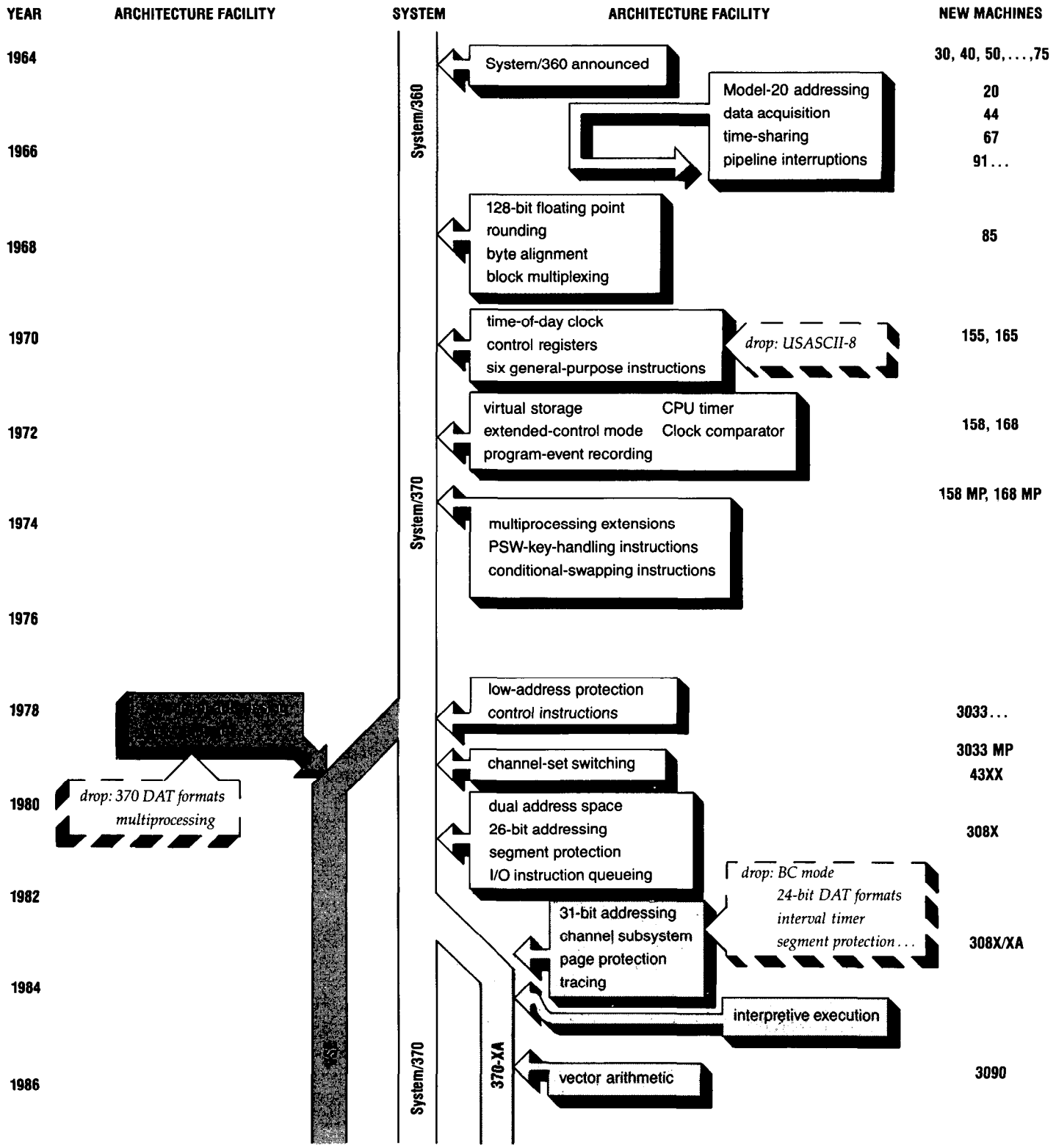
The architectural requirements for system facilities in the high-end machines were set primarily by the MVS operating system. The architecture was published in the System/370 Extended Architecture (370-XA) Principles of Operation in March 1983 [10], and was first made available on the 370-XA mode on the 308X machines. In addition to the System/370 extensions previously made available on the 3033 and the 308X machines, the main new architectural features introduced at this time were 31-bit addressing to permit addressing of up to 2 Gbytes of real and virtual storage and the channel subsystem, which, among other things, removed the channel-CPU affinity and provided for dynamic selection of one among a multiplicity of paths to an I/O device. It also introduced tracing and protection of virtual storage by pages instead of segments (segment protection was introduced in System/370 on the 308X), and deleted some functions that had been superseded: the original BC-mode of the PSW (note that both modes had been available for two generations), the interval timer (it had been maintained concurrently with the new timing facilities for two generations), 2-Kbyte pages and protection blocks (370-XA introduced 4-Kbyte protection blocks), and 64-Kbyte segments (only 4-Kbyte pages and 1-Mbyte segments are offered in 370-XA).

In January 1984 the 370-XA architecture was extended by the introduction of the interpretive execution architecture on the 308X. This extension serves to establish and control virtual machines operating under a hypervisor. Early in 1986 the vector facility, which includes 171 instructions and 16 special vector registers, was made available on the 3090. This extension is intended for engineering and scientific applications involving intensive computation.

Only the more visible and significant extensions are described here; numerous other extensions, many of them for improving recovery and serviceability, are not listed. The "machine" column lists only the first models on which the facility was offered.

Andris Padegs

Evolution of System/360–370 Architecture



edge in defining the architecture in the first place. It wasn't the fault of the hardware designers; it was inherent in the initial architectural specifications.

DG **How much did it cost?**
Padegs I don't know that anybody ever added up the figures.

Case I can give you an idea how much it cost: In order to defend making the change and to get authorization to do it, we had to make at least three different trips to the corporate office to meet with the corporate management board.

AS **Was the floating-point fix the only architectural change ever retrofitted to all the machines in the field?**
Case Well, there was one more, but it was not a change that was visible to the programmer. In the original design of the channel-to-control-unit interface, transmission lines were daisy chained through controllers, and the lines were terminated to a non-ground voltage level at the last controller. If you happened to turn the power off at the last control unit in the chain, the whole channel wouldn't work. In order to fix this problem, we went back and over an 18-month period retrofitted every control unit and channel that we had shipped. It was a big deal.

DG **How else has your architecture changed over time?**
Case Let me distinguish between architectural changes and extensions. An architectural change will change the way current programs operate. The floating-point revisions were architectural changes. Extensions are new functions that are compatible with existing programs. Generally speaking, existing programs will not be able to take advantage of an extension, unless it is something like a cache memory. But cache memories are really an implementation technique, and so they are not visible in the architecture; in other words, they have no effect on software.

Most of the architectural extensions that we have made have been to supervisory-state facilities that are only used by the operating system. For example, we introduced a new format for the program status word in 370, and for XA we have a new set of I/O instructions. In this way we can isolate problem-state programs from the evolution of the architecture. In addition, when we add a new facility to replace a current one, we keep both in for a long time. This allows user sites to convert to the new architecture over a period of years without changing their hardware. With the exception of the floating-point changes, we have never changed the architecture without at least fixing it so that all the old programs could still be run.

Padegs I think it is important to clarify precisely what we mean when we say that extensions are compatible with existing programs. Extensions are only compatible with valid programs. A valid program is one that doesn't intentionally—or unintentionally—cause any error indications. Error indications can result from issuing instructions with an invalid op-code, or changing a set of bytes to an invalid state. The idea is that valid programs do not use the codes and formats that we may need for extending the architecture, such as unassigned op-codes.

We built the whole original 360 line with USASCII-8 capability, and to the best of my knowledge, nobody *ever used it.*

DG **You mentioned that the only programmer-visible change that has been made in the 360 architecture over time was the floating-point change. What about the change that eliminated USASCII-8 mode?**
Case The original 360s had a bit in the program status word that changed the representation of digits and signs. This mode was in anticipation of the ASCII standard that had just been published. USASCII-8 capability was removed in the 370 because we were unable at the time to find a program that used the facility. In fact, we built the whole original 360 line with that USASCII-8 capability, and to the best of my knowledge, *nobody* ever used it. Thus, there were no old programs to worry about in this case.

AS **Amazing.**
Padegs Although the USASCII-8 mode was intended for use by unprivileged programs, the architecture specified that the machine had to be in supervisory state to turn on USASCII-8 mode. However, none of our operating systems were programmed to turn on the bit. That was one assurance of how unlikely it was that anybody could be using the USASCII-8 mode.

Now, to be completely precise and honest technically, a few other minor changes were introduced into 370 that should be considered incompatible, such as the choice of the op-code for the new halt-device instruction, the effect of I/O command retry, and the change in the prefetching of channel control words by the channel. All these changes are highlighted in the Principles of Operation.

MULTIPROCESSING

DG **To what extent is multiprocessing designed into the 360/370 architecture?**
Padegs I assume when you say multiprocessing you mean where there are two or more CPUs sharing main storage. The System/370 architecture offers this type of multiprocessing, and we found that it has two types of architectural implications. The first has to do with new functions, where one CPU tells another to start or stop or do something. The second is much more subtle and pervasive and has to do with the sequencing of events in a multiprocessor.

I can tell you a story about instruction retry to give you an idea of how subtle these sequencing implications are. Our hardware designers included a facility in some of our machines that would retry instructions that got recoverable hardware errors. Initially the designers felt that instruction retry was transparent to the architecture so that it wouldn't concern architects. So did we until we realized what could happen. The problem was that a second CPU could observe all the intermediate values placed into storage by the CPU that is doing the retry. It could observe, for example, a field being changed to some new value (the erroneous result), then back to the original result as stored by the retry procedure, and finally to the correct new value.

Case The test we used was, Can you show me a difference between a multiprocessor system with retry and one without? And then, if you can show me a difference, can you prove to me that any program could ever detect that difference? At first we weren't sure that a program could ever detect retry by another CPU, then we weren't sure that a program could ever care, and then finally somebody said, "Hey, that's wait-post!" Wait-post is the standard queue management discipline in MVS, and it could tell the difference. After some analysis you conclude that, if you're going to get coherence between two different CPUs, the first thing you've got to do is establish some rules about what just one CPU can do. If you don't have any rules about what one CPU can do, you can't get the link between them.

Padegs We were of course really forced to think through the whole thing. There are certain rules that will make operations predictable. We added these rules to the architecture. They are not really part of multiprocessing, but they are important for multiprocessing. For example, certain instructions can only update storage once. The rules also define the permissible orders for fetches and stores.

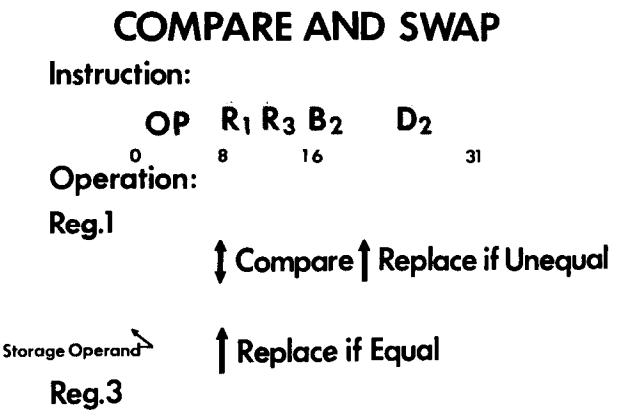
DG **So you define some instructions to be atomic operations and others to be nonatomic?**
Padegs Well, no, there is a difference between atomic instructions and atomic stores. An atomic store of an 8-byte field means that all 8 bytes appear to be placed in storage concurrently; the field cannot be stored one byte at a time, where another CPU could observe only part of the field having been changed. Also, the effects of retry must not be observable for an atomic store. But that doesn't mean that the whole instruction involving an atomic store must be atomic. Instructions are generally not atomic, and another CPU can observe distinct fetches and stores taking place. That of course is the whole reason why we need compare and swap. Compare and swap is atomic, and so is test and set.

AS **How do your rules help the programmer deal with the complexity of multiprocessing?**
Case In both the single processor and the multiprocessor case, we have simplified the architecture at the expense of a potentially complex implementation. For example, in the multiprocessor case everything has got to work as though you had a bunch of slow serial CPUs actually executing out of a memory without any pipelining and caches. We preserve that simple image. There are some exceptions, as when one processor stores into the instruction stream of another processor, but that is the basic idea.

DG **That certainly must complicate the implementation of high-performance pipeline machines like the 3090. Would these machines be less complex if you relaxed your rules?**
Padegs To some degree, oh yes.

AS **Does the architecture now specify that all of the processors in a multiprocessor system must have equal access to I/O devices?**
Padegs Part of the XA structure is that each CPU connects with all channels.

DG **Why did you provide compare and swap in your architecture when you already had test and set?**
Case Test and set is sufficient for implementing a basic semaphore, but for operations like updating a queue, test and set has to be used in a protected region, which is a portion of code that cannot be interrupted. Compare and swap can perform such operations in a single instruction. Because compare and swap is atomic, it does not have to be used in a protected region. Thus for an important set of applications compare and swap is simpler and has higher performance than test and set.



The "Compare and Swap" instruction was added to the 370/XA architecture in 1973 to support multiprocessing.

AS **What are the specific interprocessor communication instructions in the 370 architecture, and when were they introduced?**
Padegs There is one instruction called signal CPU, which was added in 1973. Signal CPU allows one CPU to control and communicate with another. In the 360 architecture, we had a general-purpose instruction for signaling that was used in the Model 65 and 67 multiprocessors.

DG **How do you provide cache coherence between CPUs in a multiprocessor?**
Case Caches use cross interrogation to find out if other caches also hold the same data. If cross interrogation never finds a conflict, execution is at the same speed as it would be without cross interrogation. But there is a sizable hardware cost in the cross-interrogation system.

DG **Just to play devil's advocate, I'm going to assert that the only reason you can make these machines work is that you've been at it for 20 years. The machines are sufficiently complicated now that you would have a very difficult time making them work properly without all your experience.**
Case Possible. Possible.

Padegs Well, I admit there are things that we would do differently if we were starting from scratch. Would we have provided the instructions for serializing, synchronizing, and making the cache nontransparent? That's difficult to answer. Maybe if we were given today's technology without any precedents, we might have allowed private segments that would not provide cache coherence between CPUs. That would certainly have reduced cache cross-interrogation traffic between CPUs.

Case I think I would build hardware that could detect whether a segment was read only or write only just from the actions applied to the segment. This would eliminate the need for people to declare segment types.

MAINTAINING THE SYSTEM/360 ARCHITECTURE

DG **Has the 360 outlasted its original projected lifetime?**
Case There was no official forecast that I know of about the lifetime of the architecture, but there was an official forecast about the lifetime of the initial models. Some of the people who were there—Fred Brooks maybe—will tell you they hoped the architecture would last for 15 years because previous architectures had lasted 5 years.

AS **Why has it lasted for 22 years?**
Padegs I would say that there are three major forces that have contributed to its success. First of all, the basic structure was sound. We've changed the I/O architecture completely, and we've changed functions for the control program and added new instructions, but all of these additions and modifications are based on the same basic structure for instructions, operands, program status words, and so on, that we developed in the early 1960s. These structures are probably not optimum, but they are sound. That's number one.

The second factor is that we were able to build on the basic structure because it was so rigorously and precisely defined.

The third factor is that we have a department in the company with responsibility for defining and maintaining the architecture. Maintenance is just as important as the initial definition. Initially, people worry about how to define an architecture. Once an

The System/360 Model 40 was introduced on April 7, 1964, and had a cycle time of 625 ns and 256 kbytes of main memory.

architecture is defined, in many cases the architectural function fades out. In our case I think that the architecture department has actually grown since the original definition of the 360 in the early 1960s. Architecture needs continuity.

AS **Do computer architects have a relatively large amount of latitude in the design of something that will work?**
Case I think so. All of the reasonable designs that have been developed have been a lot closer to each other than people expected. The thing that most distinguishes different instruction sets in computer architecture is their addressing capabilities. The reason we needed System/360 in the early 1960s was for its 24-bit address space. That was enough for a decade, and then we had to expand to 31 bits. We had laid the groundwork for doing that back in 1964. At present we are using up one bit of address space every 30 months. That means we are going to eventually reach the point where a 31-bit address space is insufficient. For a while we'll be able to get by with a collection of patches and switches, but they will not last forever. Probably by the time we need 34 bits or so there will have to be another major upheaval.

AS **I presume that XA came out just in the nick of time in 1983.**
Case It was late. Some customers could have taken advantage of it earlier.

AS **Okay, so in several years you will run out of 31 bits. You have a couple more bits you said you could handle by ad hoc means. So you've got until 1995, and then your upheaval will occur.**
Case It's plus or minus a few years, at today's rates. Now whether today's rates continue, or whether something else will happen, I don't know. But addressing is the most important driving force in instruction-set architecture.

DG **Have architectural deviations been allowed, to improve performance in a particular model?**
Padegs Performance normally is not the motivation for a deviation from architecture. A deviation normally is allowed after the fact. Furthermore, we do not grant deviations for the basic part of the architecture that would affect the normal operation of a program. For example, in the past we have allowed a deviation when the test light on the panel doesn't come on at the right time. A deviation normally is requested to avoid the cost of correcting the design.

DG **Was the imprecise interrupt in the Model 91 an architectural deviation?**
Padegs I guess you would have to call it a deviation, yes. I look at deviations as minor things that don't affect the basic functioning of a machine. Normally there are very few deviations in a machine, perhaps a couple or none at all. Imprecise interrupts in the Model 91 were a design decision. It was really almost like adding a new function to the machine.

We told the users of that machine that in exchange for increased performance they could not recover from floating-point errors.

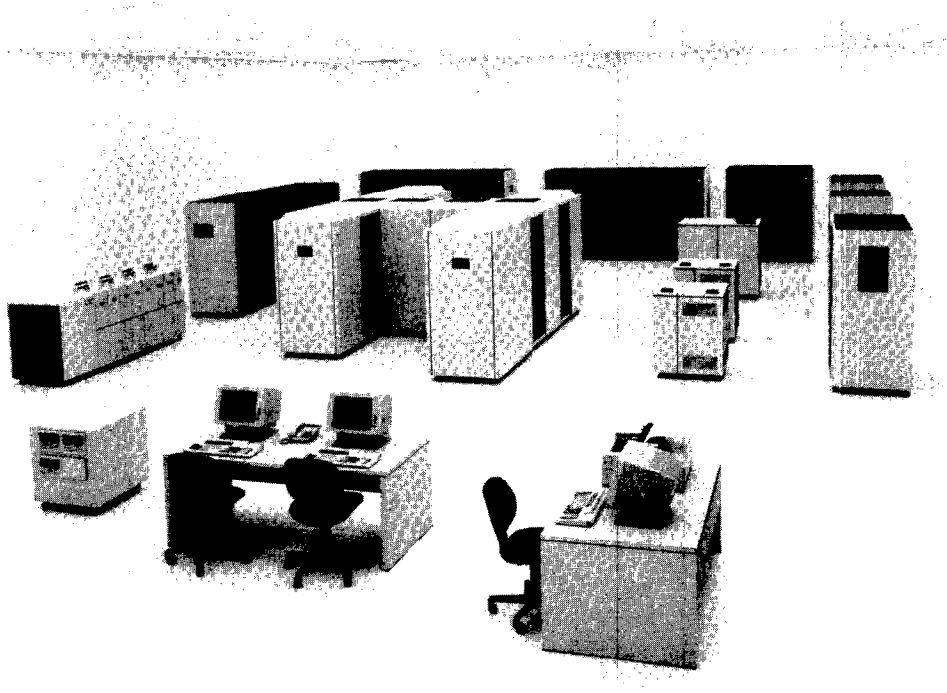
Case Imprecise interrupts did not survive in the successor to the Model 91, by the way.

AS **So-called reduced-instruction-set computers implement a small set of instructions that can be executed in a single cycle. What influence do the arguments for reduced-instruction-set computers have on your architecture today?**
Padegs Let me answer with an example. At one point we considered adding an instruction to our architecture that would, in one operation, load the general registers and change the PSW [program status word]. This is a very common operation in dispatching. The existing method used two instructions, one to load the registers and another to change the PSW. We discovered that the proposed single instruction would be slower than executing the two existing instructions. So we didn't add that instruction. We've always had at least an implicit reduced-instruction-set philosophy.

Case In fact, in the fastest implementations of the architecture there is a whole flock of instructions that execute in one cycle, and these single-cycle instructions are the bulk of the instructions executed.

DG **Doesn't the relative complexity of your instruction set create a need for more hardware to implement your core instructions in a single cycle?**
Case Yes, but the cost is not that great. I believe you could build a high-density single-chip version of a 370-like reduced-instruction-set computer so that the core of the 370 instructions would be implemented in hardware on the chip in about one cycle per instruction. The rest of the instructions would be simulated using the core instructions out of a control store. Hardware complexity could be within a factor of two at the chip level. There are good arguments for reduced-instruction set computers such as the 801 [14]. The 801's compiler ideas were good, along with the idea of having separate data and instruction caches, and the trade-offs in the design of the hardware protection facilities were good. But in the end the total difference between conventional designs like 370 and reduced-instruction-set computers is not huge. In today's technologies there are only a few chips at the CPU level for many implementations, and both RISC and non-RISC designs use the same number of memory chips and the same number of I/O chips.

AS **When John Cocke gave his talk at CMU on reduced-instruction-set technology, one of his arguments was that it's desirable not only to execute an instruction per cycle, but also to minimize the**



The 3090 Model 400, which runs both the 370 and the 370/XA architectures, was introduced on February 12, 1985, and has a cycle time of 18.5 ns, with up to 128 Mbytes of main storage and 512 Mbytes of expanded storage. The newest 370–370/XA machine, the Model 600E, which looks essentially like the 3090 Model 400, was introduced on January 6, 1987, and has a cycle time of 17.2 ns, with up to 258 Mbytes of main memory and 1 Gbyte of expanded storage.

length of a cycle. We have an existence proof that it's possible to build a high-performance 370-XA architecture with an average number of cycles per instruction of close to one. The remaining question is, how much have you added to the length of a cycle to do that?

Case That's a harder one to answer. For example, one experimental FET 370 that we have designed in a reduced-instruction-set style has many more gate delays per cycle than a 3090 does. There are a lot of trade-offs. Cocke's 801 design minimizes the number of gate delays per cycle relative to a 370-XA implementation. The question is how much.

DG **There is at least one feature in your architecture that is fairly complicated, which is the program event recording facility. How was that justified?**

Case The program event recording facility provides the ability to build a general-purpose debugging subsystem that operates on a wide variety of other programs. That feature has been justified more times than any other single feature in the architecture. It has been a candidate for removal every time there was a major change in the architecture because of its complexity. But then we look at the savings in the field expense for basic software maintenance that it provides and change our minds. It allows our field service representatives to identify a failing module with an efficiency that they otherwise could not achieve.

AS **Is the program event recording facility a tough thing to implement in software?**

Case No, not at all. All you've got to do is put a 17-instruction interpretive loop around every instruction. Then it's easy to implement. But of course that is not practical. John Cocke would argue that you only need a program event recording facility for modules written in assembly language, that the program event recording facility is superfluous for modules written in higher level languages.

FUTURE DIRECTIONS

DG **How might you try to design a new architecture today?**

Case When we did the 360, we had to assume that a significant fraction of the programming was going to be in assembly language. Today you might begin by dispensing with a rational human-engineered interface at the assembly-language instruction level.

AS **What sorts of radical changes can you foresee that would cause the 360 architecture to be supplanted by a new architecture?**

Case That's hard to say. If there is ever a breakthrough in harnessing thousands of micros into an effective array for a wide-enough spectrum of big jobs, then it might be time to rethink the major computer product lines. In the meantime, though, I'm underwhelmed by most of the suggestions for such machines that have been made in the literature and by entrepreneurs. Not that a lot of what they're saying is not worthwhile, but the scope of applications is limited in each case. I think that if IBM ever gets convinced that its future objectives can't be met with 360/370 machines and if it believes that there might be another approach that will meet those objectives, then we'll go try.

REFERENCES

1. Amdahl, G.M., Blaauw, G.A., and Brooks, F.P. Architecture of the IBM System/360. *IBM J. Res. Dev. 8*, 2 (Apr. 1964), 87–101.
2. Amdahl, G.M., et al. Special Issue: The structure of SYSTEM/360. *IBM Syst. J. 3*, 2 (1964), 119–263.
3. Aron, J.D., et al. Discussion of the SPREAD report, June 23, 1982. *Ann. Hist. Comput. 5*, 1 (Jan. 1983), 27–44.
4. Case, R.P., and Padegs, A. Architecture of the IBM System/370. *Commun. ACM 21*, 1 (Jan. 1978), 73–96.
5. Evans, B.O. System/360: A retrospective view. *Ann. Hist. Comput. 8*, 2 (Apr. 1986), 155–179.
6. Haanstra, J.W., et al. Processor products—Final report of the SPREAD task group. *Ann. Hist. Comput. 5*, 1 (Jan. 1983), 4–26.
7. Hellerman, H. The SPREAD discussion continued. *Ann. Hist. Comput. 6*, 2 (Apr. 1984), 144–151.
8. IBM. IBM System/370 Principles of Operation. Order no. GA22-7000, available through IBM branch offices.
9. IBM. IBM 4300 Processors Principles of Operation for ECPS: VSE mode. Order no. GA22-7070, available through IBM branch offices.
10. IBM. IBM System/370 Extended Architecture Principles of Operation. Order no. SA22-7085, available through IBM branch offices.
11. IBM. IBM System/360 and System/370 I/O interface: Channel to control unit, original equipment manufacturer's information. Order no. GA22-6974, available through IBM branch offices.
12. McHenry, W.K., and Goodman, S.E. MIS in Soviet industrial enterprises: The limits of reform from above. *Commun. ACM 29*, 11 (Nov. 1986), 1034–1043.
13. Padegs, A. System/360 and beyond. *IBM J. Res. Dev. 25*, 5 (Sept. 1981), 377–390.
14. Radin, G. The 801 Minicomputer. *Symposium on Architectural Support for Programming Languages and Operating Systems, SIGPLAN Not. 10*, 2 (Mar. 1982), 39–47.

CR Categories and Subject Descriptors: B.0 [**Hardware**]: General; C.1.1 [**Processor Architectures**]: Single Data Stream Architectures; C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors); C.5.1 [**Computer System Implementation**]: Large and Medium ("Mainframe") Computers; K.1 [**Computing Milieux**]: The Computer Industry; K.2 [**Computing Milieux**]: History of Computing; K.7.2 [**The Computing Profession**]: Organizations

General Terms: Design, Documentation, Performance

Additional Key Words and Phrases: IBM, SPREAD, System/360, System/370