An old idea may influence new computer designs.

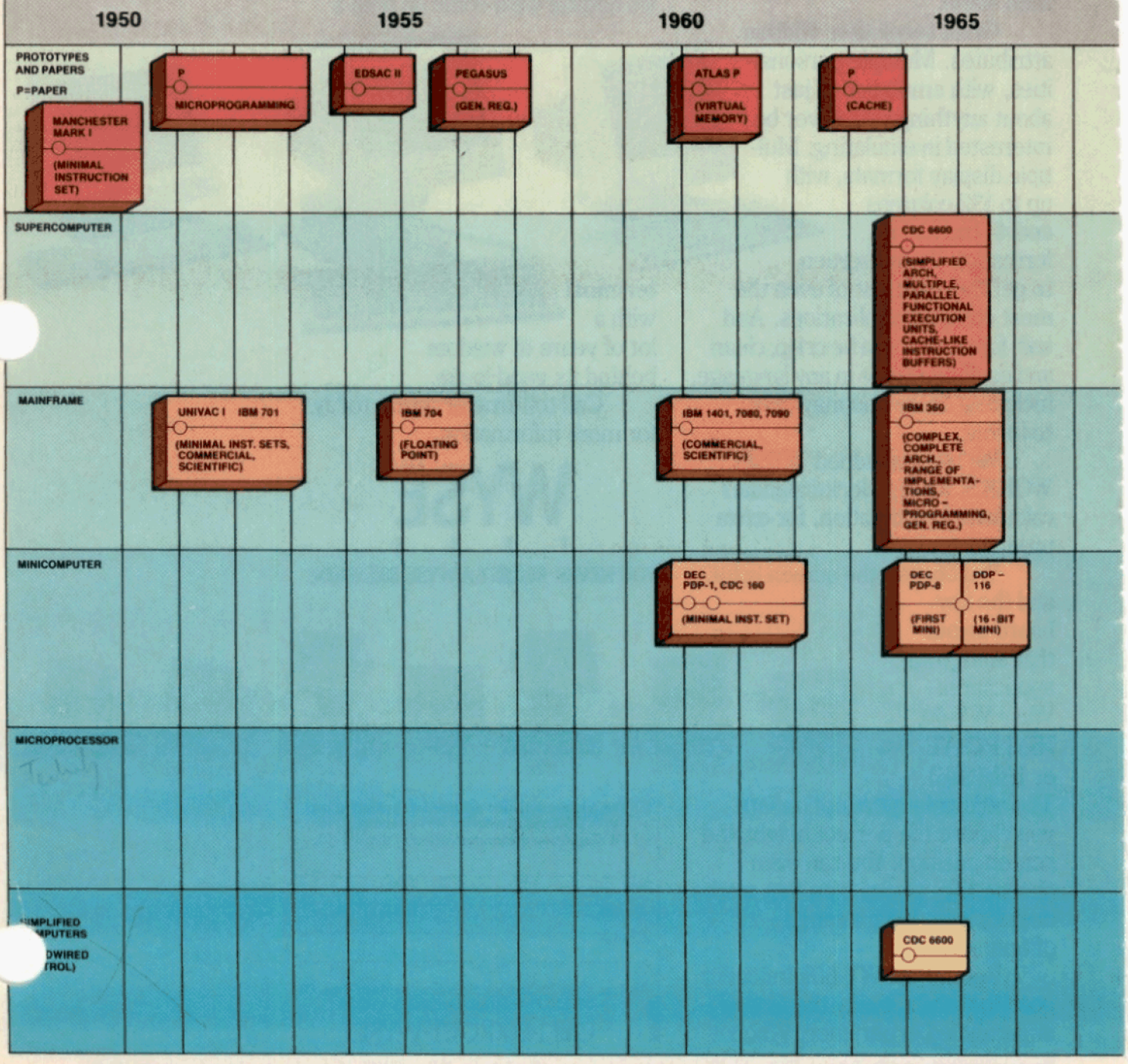
RISC: BACK TO THE FUTURE?

by C. Gordon Bell

Several recent announcements indicate that computers will be changing over the next decade. The new high-speed architectures announced by Hewlett-Packard, IBM, and a startup in Sunnyvale, Calif., called MIPS Computer Systems Inc., challenge today's reliance on microprogrammed processors—a legacy of the original IBM 360.

The primary development involves the application of the so-called reduced instruction set computer (RISC) concept. RISC

FIG. 1
A TIME LINE OF FOUR COMPUTER CLASSES



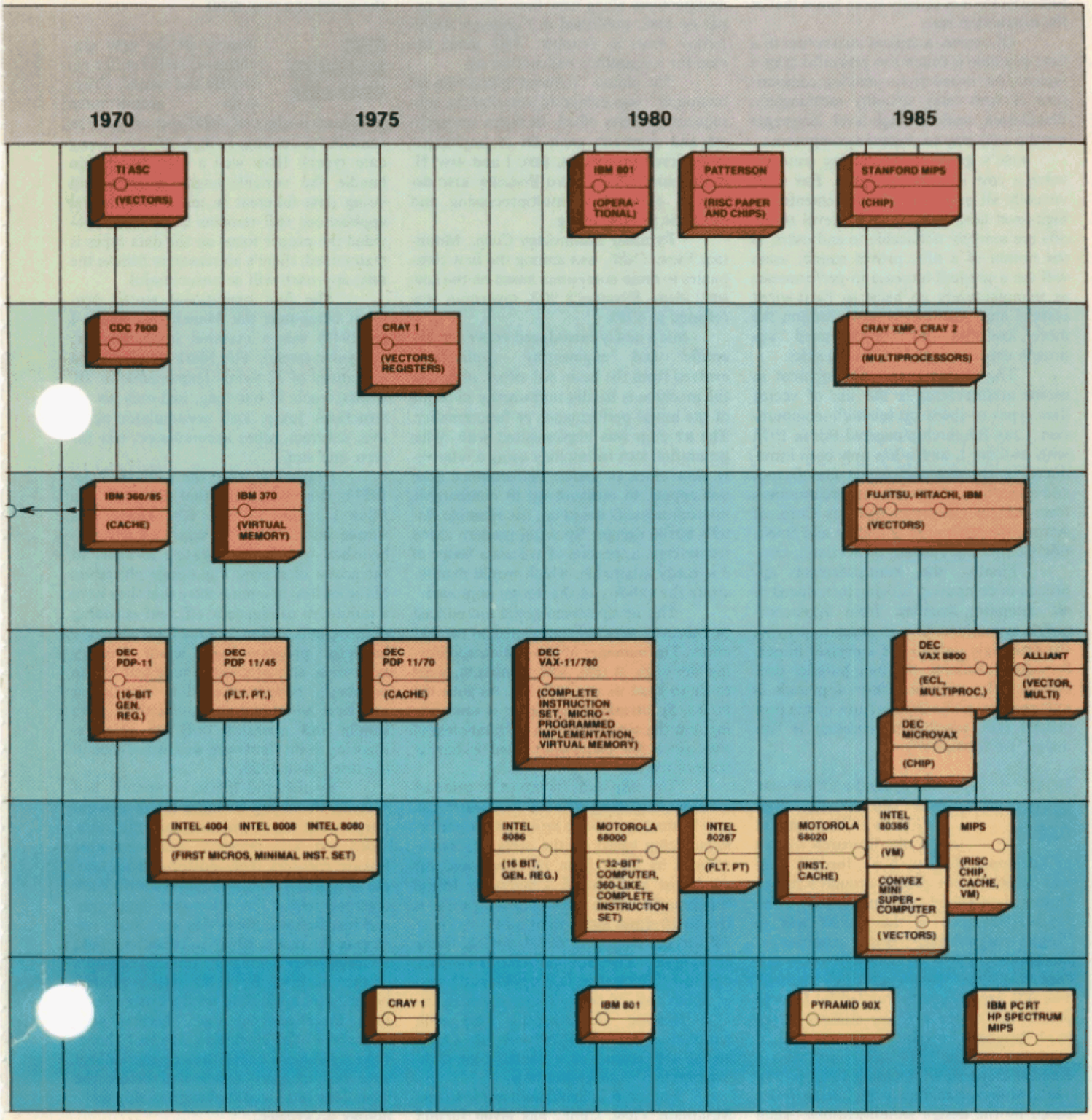
designs contrast with those of complex (or complete) instruction set computers (CISC), which are meant to contain a repertoire of machine instructions to handle all the data types and operations of today's high-level languages. (The CISC approach reduces the size of object programs by attempting to provide each high-level source code statement with a single machine instruction.)

The concept of RISC involves an attempt to reduce execution time by simplifying the central processor's tasks. Conventional microprogrammed architectures are predicated on relatively slow access to primary memory and read-only memories for microprograms that are five to 10 times faster. In CISC machines, which do more processing per instruction than in RISC, the processor requires five to 10 clock ticks to carry out a typical instruction.

But with continuing hardware refinements, especially in chip technology over the last decade, logic and memory speeds are now nearly identical. RISC machines exploit this development by transferring certain logical steps that are required to interpret instructions out of processor microcode and into memory as a run-time library program (see Figs. 2c and 2d). In a RISC design, all high-level language functions are constructed from simple software primitives in a fashion resembling microprogramming, except that they are stored outside the processor as regular programs.

Reducing the instruction set further reduces the work a RISC processor has to do. Since RISC has fewer types of instructions than CISC, a RISC instruction requires less processing logic to interpret than a

CHART BY CYNTHIA STODDARD



Nearly all the RISC machines resemble the simple, Cray-style architectures pioneered in the CDC 6600.

CISC instruction. The effect of such simplification is to speed up the execution rate for RISC instructions. In a RISC implementation it is theoretically possible to execute an instruction each time the computer's logic clock ticks. In practice the clock rate of a RISC processor is usually three times that of the instruction rate.

Of course, a typical instruction in a RISC machine is much less powerful than a typical CISC instruction—making comparisons of MIPS rates virtually meaningless. Benchmark tests in high-level languages are the only way to compare computers.

RISC's gains in processing time involve a cost in memory space. But since virtually all programs are implemented in high-level languages, machine-level tradeoffs are scarcely noticeable to end users. If the notion of a RISC proves sound, users will see a gradual increase in performance as manufacturers go back to hard-wired control implementations and abandon the more complex microprogrammed approach used over the last two decades.

The second major development in recent architectures is the use of vector data types to speed up scientific computation. Cray Research pioneered this in 1975 with its Cray-1, and it has now been introduced in the IBM-compatible mainframes and a plethora of new minisupercomputers from Alliant Computer Systems Corp. of Acton, Mass., Convex of Dallas, and Scientific Computing Systems of Portland, Ore.

Finally, the multiprocessor approach to computing is being introduced in all computer families, from supercomputers to workstations, in order to provide the seemingly unbounded increases in performance available through parallel processing. The multiprocessor approach is independent of the architecture of the processors (i.e., whether a processor is "reduced" or "complex").

HOW RISC BEGAN

The groundwork for RISC was laid in the mid-1970s. In 1975, a team at the IBM Research Center in Yorktown Heights, N.Y., formed what was called the 801 project around a design approach credited to IBM fellow John Cocke. The goal of the project was to "achieve significantly better cost/performance for high-level language programs than that attainable by existing systems." The 801 group, which was led by George Radin, one of the primary authors of the PL/1 language, produced an operational minicomputer in 1979 (see "IBM Mini a Radical Departure," October 1979, p. 53). The IBM group acknowledged being influenced by the design simplifications—such as hard-wired control—pioneered by Seymour Cray in his design of the first supercomputer, the CDC 6600 (c. 1964).

Professor David Patterson of the University of California at Berkeley has been the main proponent of reexamining architectures along RISC lines. His first paper on RISC, published in *Computer Architecture News* in October 1980, made the case for a simplified instruction set.

The phrase "reduced instruction set computer" was coined to describe the subsequent Berkeley effort. Berkeley researchers and engineers went on to implement operational prototypes, RISC I and RISC II, in the early '80s; a third Berkeley RISC design is oriented to multiprocessing and symbolic programming.

Pyramid Technology Corp., Mountain View, Calif., was among the first companies to build computers based on the new RISC ideas. Pyramid's 90X supermini was released in 1983.

IBM's newly introduced PC RT for scientific and engineering applications evolved from the basic 801 effort, although the machine is hardly noteworthy in terms of the initial performance or functionality. The RT chip was implemented with older generation MOS technology using a relatively slow clock of 6MHz. Performance does not appear to measure up to comparable microprocessors based on, for example, Intel's 80386 design. By using modern CMOS technology, a speedup of at least a factor of 3 is easily attainable, which would demonstrate the validity of the design approach.

The HP Spectrum series announced in February may also be related to the 801 effort. The manager of the section sponsoring the work at IBM, Joel Birnbaum, went to HP to head its research lab. As with the PC RT, Spectrum performance is uninspiring, but the series is not an adequate test of RISC since the technology used is hardly state of the art.

The chip built by MIPS Co. presents a more compelling case for RISC. MIPS Co. was formed in 1984 to build a high-performance chip based on the design ideas developed by Prof. John Hennessy and his associates at Stanford University. Initial benchmarks of the MIPS chip indicate it is five to 10 times faster than a DEC VAX-11/780 or the Motorola 68020 for the same clock speed. The MIPS Co. chip is simpler, considerably smaller, and significantly faster than any of today's microprocessors. The Defense Department's R&D arm, which sponsored the MIPS work, has adopted the architecture as a standard for high-performance implementations.

Fairchild Semiconductor Corp., Mountain View, Calif., has given further support to the RISC approach with its recently announced microprocessor called Clipper. The leader of the Clipper development group, Howard Sachs, came from Cray Research. Clipper, implemented as three CMOS chips, is reported to be about the speed of a VAX 8600.

RISC MACHINES CRAY-LIKE

Nearly all the RISC machines unveiled so far resemble the simple, Cray-style architectures pioneered in the CDC 6600 and oriented to scientific processing (which stresses binary data types). How well a RISC design can handle the variable-length decimal and string data inherent in many commercial applications still remains to be seen. Provided the proper focus on the data types is maintained, there's no reason to believe the RISC approach will be unsuccessful.

The first operational stored program computer, the Manchester Mark I (c. 1948) was a minimal instruction set computer (MISC). The Mark I, which had a memory of 32 words (expandable to 8K words), each 32 bits long, had only six instructions: jump, load accumulator negative, subtract, store accumulator, test for zero, and stop.

Beginning with the Univac I (c. 1951), the computers that followed the Mark I in the '50s and '60s likewise had simple instruction sets, appropriately embellished with index registers to assist in the access of arrays. The design objectives of the earliest machines were that they have a minimum of registers, efficient encoding of programs (often oriented to assembly language programming), small primary memories, and processors matched to the memory's performance. The instruction sets were small and the instructions were simple and operated only on integers. Floating point hardware was introduced in the IBM 704 in 1955.

By the mid-1960s, computers had evolved to having a single set of general purpose registers combining accumulators, index and base registers, and subroutine linkage registers. The control unit for simple processors was straightforward. Each machine had a few allowable data types and instructions. Memories were slow, relative to the rate at which information could be transferred among the internal registers of the machine. By 1960, core memories had a 2μsec cycle time, with a 1μsec access time. Since internal logic operated at a 5MHz to 10MHz clock rate, five to 10 hardware operations could thus be carried out after one memory access and before the next. This ratio would change as chip technology progressed.

A simple computer of the early '60s operated at about 250,000 instructions per second since typically two memory references were required per instruction (e.g., load accumulator, add memory to accumulator, store accumulator). Performance was increased by providing overlapped memory so that a processor could access memory at a faster rate. The objective of computer design was to match the instruction processing rate of the processor to the memory. Fig. 2a shows the configuration of a hardwired processor, matched to a memory.

The second computer generation began in 1960 with many important, transistorized, core memory machines brought out by the early leaders in the minicomputer, mainframe, and supercomputer classes: DEC (PDP-1), IBM (1401 for operating on strings, 7070 for operating on decimal numbers, and 7090 for operating on scientific numbers), and Control Data (160, 1604). (Seymour Cray developed the architecture for Control Data's first supercomputer in the 1960s. He later left CDC and, in 1972, founded Cray Research, the leader in supercomputer class.)

By the mid-1960s, a second round of significant computers from these vendors established their three respective classes: DEC PDP-5 (1964), the forerunner of the PDP-8, the first minicomputer; IBM 360 (April 1964), the mainframe; and Cray's CDC 6600 (1964), the first supercomputer. Fig. 1 depicts the evolution of the "mainline" computer classes—micros, as we shall see, came with the 1970s—demonstrating how prototype ideas that first appeared on high-priced machines later spread to wide-scale use.

The IBM 360, introduced in 1964, was one of the earliest computer families to span a range of price and performance. Along with the 360, IBM introduced the word architecture to refer to the various processing characteristics of a machine as seen by the programmer and his programs. In the initial 360 product family, the model 91 exceeded the model 20 in performance by a factor of 300, in memory size by a factor of 512, and in price by a factor of 100.

360 LINE SET NEW STANDARD

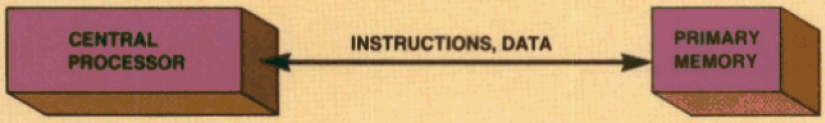
The enormously successful 360 product line set an important new standard for CISC design. By virtually all measures, the instruction set was more complete and complex than any previous design, and included instructions to handle data in both the commercial and scientific environments: integers, floating point, decimal, and character strings.

The primary goal for the product line was to merge IBM's scientific and com-

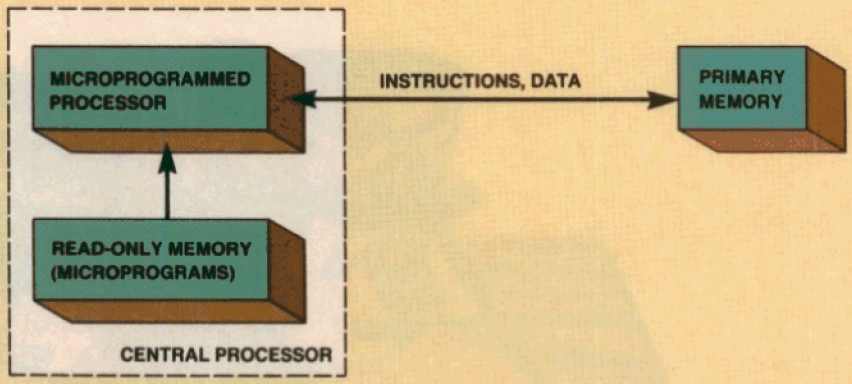
FIG. 2

~~FOUR~~ MAINLINE COMPUTER ARCHITECTURES

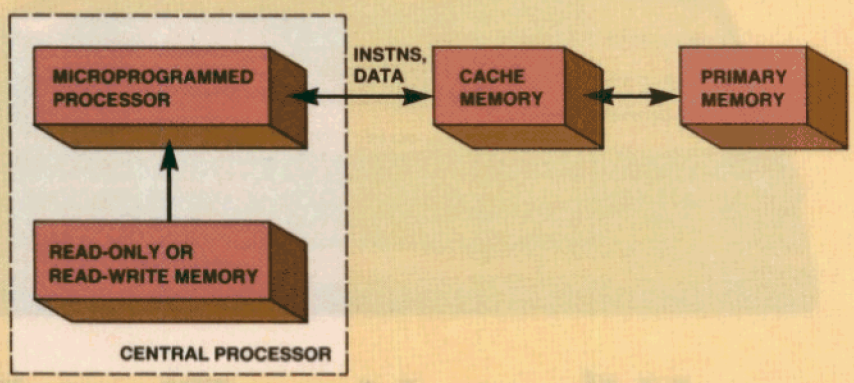
Evolution



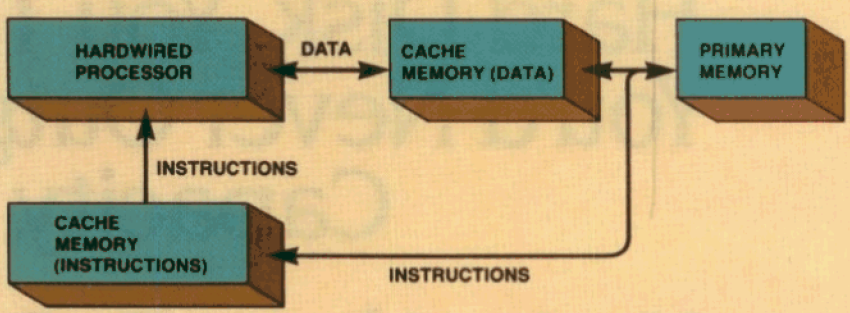
2a. Early, hardwired simple computers (c. 1950 - 60) and ~~conventional~~ early micro-processors (c. 1971 -)



2b. Microprogrammed processor to interpret instruction set (c. 1964)



2c. Microprogrammed processor with cache (c. 1968)



2d. Hardwired, pipelined processor with separate caches for instructions and data (c. 1985)

The goal of these early architectures was to provide processors that operated at the memory rate.

mercial computer families into a single product line in order to have a single architecture and to maximize the compatibility of the peripherals and operating systems. Since the memory sizes were small, a complex architecture was adopted in order to most efficiently encode programs of the various languages. The benefits to users were configuration flexibility and compatibility across the range.

Microprogramming, first described in a 1951 paper by the English computer pioneer Maurice Wilkes, was the technique used to implement most of the 360 line. A microprogrammed processor carries out its functions with a program, stored in a read-only memory, that interprets the larger instruction set. In effect, it is a computer within the central processor (see Fig. 2b). The microprogrammed processor is also useful for implementing the complex I/O processors that were part of the 360 architecture.

Since the 360 included a range of computers, the specific models were separate and distinct. A few of the early 360s were developed using hard-wired (non-microprogram) implementations. These were the highest-priced, high-performance models. The model 44, introduced in the late '60s, used hard-wired control, and provided an exceptionally high price/performance ratio.

CDC 6600 WAS MODEL FOR RISC

The goal behind the CDC 6600 was simply to build the largest, fastest scientific computer of the day. Hard-wired control, radical packaging (heat exchangers, for example), and an emphasis on parallelism were among the innovations of the first supercomputer.

The 6600 architecture was radically simplified compared with contemporary architectures of 1964, especially the 360's. The 6600 is now recognized as something of a RISC prototype because it used a relatively small set of general registers; only register-load and register-store instructions were used to access memory, and a minimum number of instructions were defined to operate on integer, boolean vector, and floating point data. More complex operations such as character string operations were coded from the basic operations.

The processor was completely hard-wired, with a great amount of control capabilities, so that several different instructions could be executed in parallel, through pipelining. In the 6600, a large instruction buffer, which is almost identical to today's instruction caches, was used to hold instructions without having to access the slower core memory.

By 1975, Cray had extended the 6600 architectural concept in the Cray 1 to include a small register array as a set of "vector" registers in order to maximize the performance of the computer for inherently pipelined operations.

With vector operations, a single instruction may specify a list of numbers, such as a column of a matrix, to be operated on at a given time. With the extension of vectors, it is very hard to characterize such an architecture as either simple or RISC-based. A vector processor, however, maintains the goal of delivering a result every clock tick, using a complex data type, namely a vector.

Roughly 10 years later, IBM and the Japanese 360/370-compatible vendors used the same concepts for extending the 360 architecture for high-performance scientific computation. By 1985, IBM had also introduced the vector processor in the 3090, model 200/VF.

MINIS APE LARGER COMPUTERS

Minicomputers followed the same evolutionary path taken by computers in the mainframe and supercomputer classes. Developers of the PDP-8, the first minicomputer, sought to build the smallest computer possible that would still be both widely useful and widely affordable ($18,000 in 1966).

The design of the PDP-8 followed that of its predecessor, the PDP-5, which had been implemented as the smallest (MISC) computer the design team could think of. As in the original Manchester Mark I, only two registers were included in the PDP-5 (and the original PDP-8) architecture: the accumulator and the program counter—which meant that subroutines were required for multiply/divide and floating point operations.

Registers were so expensive when the PDP-5 was built that the program counter was stored in memory location 0. The original architecture had only eight basic instructions. Subsequent additions to the architecture provided arithmetic, including floating point. The PDP-8 continued this pattern of adding complexity within the processor design.

By the early '70s, the 360 had evolved to include virtual memory, a concept first realized a decade earlier in Manchester's Atlas. The 360/85 (c. 1968) was the first computer to use the cache memory, described by computer pioneer Wilkes in 1965 as a small storage area used to hold recent fetches from primary memory (see Fig. 2c). The 370 (c. 1972) represented a high-end introduction of integrated circuits, which were used for, among other things, both primary and cache memories. The gap between memory and logic speed had begun to close.

The advent of low-cost logic using integrated circuits led to the explosive growth of the the minicomputer industry in the 1970s. The evolution of minis followed "mainline" development, including a family of compatible computers, floating point arithmetic, virtual memory, cache memory, multiprocessors, and a complex instruction set (commercial and scientific) as pioneered by the 360.

The introduction of DEC's VAX-11/780 in 1978, which evolved from the PDP-11, marked the emergence of a high-performance or superminicomputer class that combined the design concepts of mainframes with relatively lower-cost technology.

By all accounts, VAX has about the most complete architecture, with separate sets of basic operations for each of the scientific and commercial data types and primitives to assist the operating system, including the management of virtual memory. The architecture also includes single instructions for procedure calls, DO loop control, and case statements. The initial microprogram size, in words or bits, was roughly double that of the first 360s.

MICRO ALSO APES LARGE COMPUTERS

The development of the micro has followed the evolution pattern of the mainframe and minicomputer. The first micro, the Intel 4004, was a minimal instruction set computer. Intel's subsequent microprocessors evolved from an 8-bit data orientation and small (16-bit) address to include floating point arithmetic, provide for memory management, and eventually support a large virtual address space (the 80386). The technology for implementing micros favors the microprogrammed approach in order to simplify the design whereby read-only memory arrays occupy a large area of the chip.

Motorola followed a similarly evolutionary path based on a near-360 architecture. After about 10 years of evolution, the 68020 chip set provides for floating point, a large virtual memory, and an on-chip instruction cache. National Semiconductor adopted a near look-alike to the VAX architecture.

Unlike either mainframes or minis, microprocessors are not fixed architectures. Rather, microprocessors approach a complete architecture as each new model extends the architecture of its predecessor. The chip technology determines the architecture of each implementation, which embellishes the architecture of its predecessor

by adding capabilities. All the "standard" chips by Intel, Motorola, and National have followed the traditional path of evolutionary complexity: all of the chips are substantially larger and more complex (by a factor of 2 to 4) than their RISC counterparts. They require a relatively long design time and operate at a comparatively slower processing rate.

The academic and commercial interest in RISC, implemented with a simple hard-wired control unit, is motivated by trends in memory and logic technology. When memory rates were considerably slower than logic rates, microprogramming improved performance by reducing the time spent outside the processor in memory fetches. Although the microprogrammed idea originated in 1951, IBM pioneered this style of design with the 360 family.

By the early '70s, the cache memory was introduced (in the 360/85), providing in effect a substantially faster memory. New semiconductor memories also offered reduced access and cycle times. Further, the speed of small read-write memories used for registers and caches began to approach that of logic. The result was that cache-based computers underscored CISC performance limitations because no faster memory could be used for the microprogram. It was after the introduction of cache memory that the IBM research project leading to the 801 was established.

Today's large IBM mainframes require about three clock ticks per instruction, which makes the question of whether CISC is a performance limitation almost academic.

FIG. 3

TWO APPROACHES TO COMPUTER DESIGN

	COMPLEX	REDUCED
Registers	8-16 gen. reg. floating pt.	16-32 gen. reg., + opt. floating pt.
Data types	bytes...double precision fl. pt. decimal, byte strings, page tables, queues, etc.	bytes...integers, fl. pt. (opt.), ? decimal, ? byte strings (software processing of O/S data)
Instructions	correspond to data types, instructions assist O/S and run-time utilities	load/store general registers, operations on data types in registers
Inst. formats	variable length, many types: load/store, R := R op R, R := Mem. op R, M: = M, M := Mem op Mem	fixed length, two main types: load/store, R := R op R
Encoding	1 instruction = 1 statement	1 inst. = 1 operand or 1 operation
Design objective	min. program length, max. work per instruction	trade-off program length, minimize time to execute instruction
Implementation	microprogrammed processor; slow, primary memory and fast clock; instructions take var. time; pipeline is complex; larger implementation may result in longer design time	hard-wired processor and software; fast processor and fast cache for instructions; instructions take one clocktime; simple pipeline
Caching	useful	essential for instructions
Compiler Design	should stress finding right instructions	should stress best ordering
Philosophy	move any useful software function into hardware, incl. diagnostics, hardware changes	move all functions to software

RISC GOAL IS MORE FUNCTIONS

The goal of a reduced instruction set is to make a simple hard-wired processor and to carry out as many functions as possible with software. (Fig. 2d shows the essential RISC scheme derived from the 801. Fig. 3 summarizes the two approaches.)

RISC focuses on reducing the number of instructions that operate on a conventional register array, and separating them into two classes: simple load/store of the registers, including use of the registers as base and/or index registers; and operations among the registers utilizing a three-register address format. In effect, this separation into instruction types means that a statement is compiled into the parts that access the data, and the part that performs the arithmetic of the statement.

In contrast, CISC schemes have a range of instruction formats that effectively combine memory access with the operations. The essential goal of the VAX, for example, is to provide a separate microcoded machine instruction for every statement that could be written in a high-level language, e.g., a single VAX instruction is C[i] = A[j] + B[k] and would correspond to at least four RISC instructions:

1. load accumulator 1 with A[j];
2. load accumulator 2 with B[k];
3. add accumulator 1 to accumulator 2;
4. store contents of accumulator 2 in C[i].

Because they are simple, one or more RISC instructions are usually held in a word (typically 32 bits), as in the early, word-oriented RISC machines.

The second way that a RISC scheme simplifies processing (besides having only load/stores and arithmetic operations) is by eliminating the complex data types such as floating point, decimal, and byte strings. In RISC, the necessary "primitive" operations (e.g., decimal add) are best considered part of the architecture. In the case of floating point, separate execution units are used for a completely hard-wired and parallel implementation.

The processor is controlled by a hard-wired logic unit, not a microprogrammed processor. The goal of an implementation is to be able to carry out one operation every clock tick or every memory cycle, using a pipeline of, say, four stages, just as in early processors that matched the memory bandwidth. The single best test for a RISC architecture is to observe whether the instruction rate is within a factor of 2 of the clock rate of the processor.

Here is the basic equation governing the execution rate of a simple, scalar computer:

#P × (clock rate) × (1/ticks/instruction) × (operations + operands)/instruction × (statements/ (operations + operands)) × (compiler efficiency)

(The number of central processors is represented by P.) This general structure using multiprocessors is likely to be the basis of mainline computing in the next decade because of the negligible cost of incremental microprocessors based on CMOS technology. For scientific processors, a vector pro-

The efficiency of the compiler is a major factor that may favor the RISC approach.

cessing unit operating in parallel with the conventional processor is the surest way to increase performance by a factor of 3 to 20, depending on the problem.

PROCESSOR NUMBER IS CRUCIAL

The number of processors potentially has the greatest effect, because it can quite possibly be increased indefinitely, with relatively little extra cost, depending on the amount of parallelism inherent in the problem. Companies such as Alliant, Encore, and Sequent are using multiple microprocessors (i.e., a multi) to increase performance in a radical fashion and provide substantially greater performance in the superminicomputer market. Cray is using the multiprocessor to increase performance in exactly the same fashion.

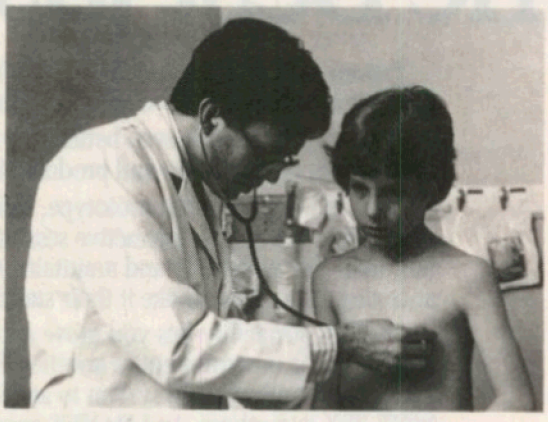
The efficiency of the compiler is a major factor that may favor the RISC approach, since there is usually only one way to carry out a given function. In the case of a complex architecture, the difficulty is finding the best way to carry out a statement, including the use of temporary general registers. There is little understanding or data that indicate that the compiler is substantially different in either case. We simply must wait for some competitive studies.

This review of the evolution of computer architecture has demonstrated the ultimate appeal of simple designs. For example, the advantages of complex microprogramming diminish directly as access time to memory is reduced. Given a relatively constant or smaller amount of a given technology, it now appears that a pipelined RISC computer could outperform a microprogrammed machine by a factor of 2 or 3. Based on the rate at which such slowly evolving technologies as TTL or ECL have been registering performance gains (i.e., 15% yearly), RISC would thus represent an immediate five- to seven-year advance over CISC. For rapidly advancing technology like CMOS, the performance of which has been improving at up to 40% per year, the switch to RISC is equivalent to a two- to three-year advance in the state of the art.

Many factors determine a computer's performance. The number of processors will have the greatest long-term effect, regardless of the number or type of instructions they execute. For the scientific market, the introduction of vectors is essential, which hardly makes the architecture very simple. For a simple instruction set, including vector operations and now multiprocessors, the best advice for today's architect is simply, "Follow Cray." If the computer is to process decimal and string data for the commercial environment, care must be taken to provide the primitive operations for the languages in the architecture. ◉

C. Gordon Bell is the assistant director of the newly formed computer and information science and engineering directorate of the National Science Foundation. He is also the chief scientist for the DANA Group, Sunnyvale, Calif., which is designing a personal supercomputer using a RISC chip. He was a vice president for engineering at DEC from 1960 to 1983, and the chief architect for the PDP-8, the System/20, the PDP-11, and the VAX-11/780. He has taught at Carnegie-Mellon University and is an IEEE fellow.

Most people think heart disease happens only in the elderly.



It happens in children as well. Things like rheumatic heart disease and congenital heart defects. Each year, nearly one million Americans of all ages die of heart disease and stroke. And 20,000 of them die from childhood heart diseases.

The American Heart Association is fighting to reduce early death and disability from heart disease and stroke with research, professional and public education, and community service programs.

But more needs to be done.

You can help us save young lives by sending your dollars today to your local Heart Association, listed in your telephone directory.

Put your money where your Heart is.

American Heart Association

WE'RE FIGHTING FOR YOUR LIFE