The IBM 3090 system: An overview

by S. G. Tucker

The first part of this paper places the IBM 3090 system in historical perspective with respect to its predecessors. Treated briefly are the technology and the design process, both of which were critical to the development of the 3090. Presented in detail is the 3090 system itself, with emphasis on its features that differ from those of prior systems.

Historical perspective

The IBM 3090 system has roots in the IBM 3033 and 308X systems while at the same time possessing features unique to it. The IBM 3033 was a highly *pipelined* machine. For example, it had an instruction element to prefetch instructions; it allowed conditional fetching down alternate paths on branching instructions; it did operand prefetching; and it had a four-element queue for instructions ready for execution. The execution element was independent and ran overlapped with the instruction processor. It had a 64-bit data path for binary operations and a one-byte-wide data path for decimal and byte operations. The cache was of a *store-through* design; that is, every store went directly to central storage as well as to the cache. The 3033 system was built in a relatively easily changed card-on-board technology with a fixed set of chips. The cycle time was 57 nanoseconds, and the system was first shipped in 1978.[1]

The 308X system took a major step forward in both packaging and level of integration. The logic chips, with a maximum of 704 transistor-transistor-logic (TTL) circuits, were surface-soldered on ceramic modules, with sites for 100 chips. (There were 118, and later even 133, chip sites for some arrays.) The modules in turn were plugged on either 6-module or 9-module multilayer boards. The effect of the packaging and integration technique was a great improvement in circuit density and associated cycle time. These improvements were balanced by an increase in the time required to make changes. To manage the reduced changeability, the 308X used a simple, straightforward machine organization. There was little pipelining, and microcode was used extensively to provide changeability. At the same time, the 308X introduced a *store-in cache*, in which data stored to the cache are not immediately also stored in the central storage. As part of the development process, there was a higher level of simulation and design verification done prior to building hardware models.[2] The 308X, which was first shipped in 1981, had a cycle time of 26 nanoseconds. The cycle time was reduced to 24 nanoseconds in later models. The 308X introduced the System/370-XA architecture, which added 31-bit addressing and a channel subsystem that included path management.

The 3090 system uses an extension of the 308X technology. It has a faster, more powerful circuit family and a more extensive array menu, and the power and cooling enhancements required to support them. This not only allows a cycle-time re-

duction from the 24 nanoseconds of the 308X to 18.5 nanoseconds, but also allows more levels of logic in a cycle. The store-in cache design of the 308X is used, but with an improved management scheme. The basic design process introduced in the 308X has been improved for use in the 3090 design. The success of the 308X design verification and the expected improvements allow the choice of a more complex, overlapped machine organization. Thus the instruction-processing and execution-element structure is based on that of the 3033, with many additions to improve the instruction-per-cycle rate, especially on engineering and scientific workloads. The system control element, which interconnects the processors and the channel subsystem to the storage system, is a substantially new design, as is the channel subsystem, which implements the System/370-XA I/O architecture.

The major technology change is inside the chip.

The 3090 introduces two new features, the *expanded storage* and the *Vector Facility*. The expanded storage is a synchronous block-transfer storage implemented in semiconductor technology that is managed by the operating system as a pageable extension of central storage. The Vector Facility introduces an extension to the System/370 architecture and is intended primarily for engineering and scientific applications.[3] This facility adds a set of vector registers and extends the System/370 instruction set by adding 171 new instructions for vector processing.

Technology and design process

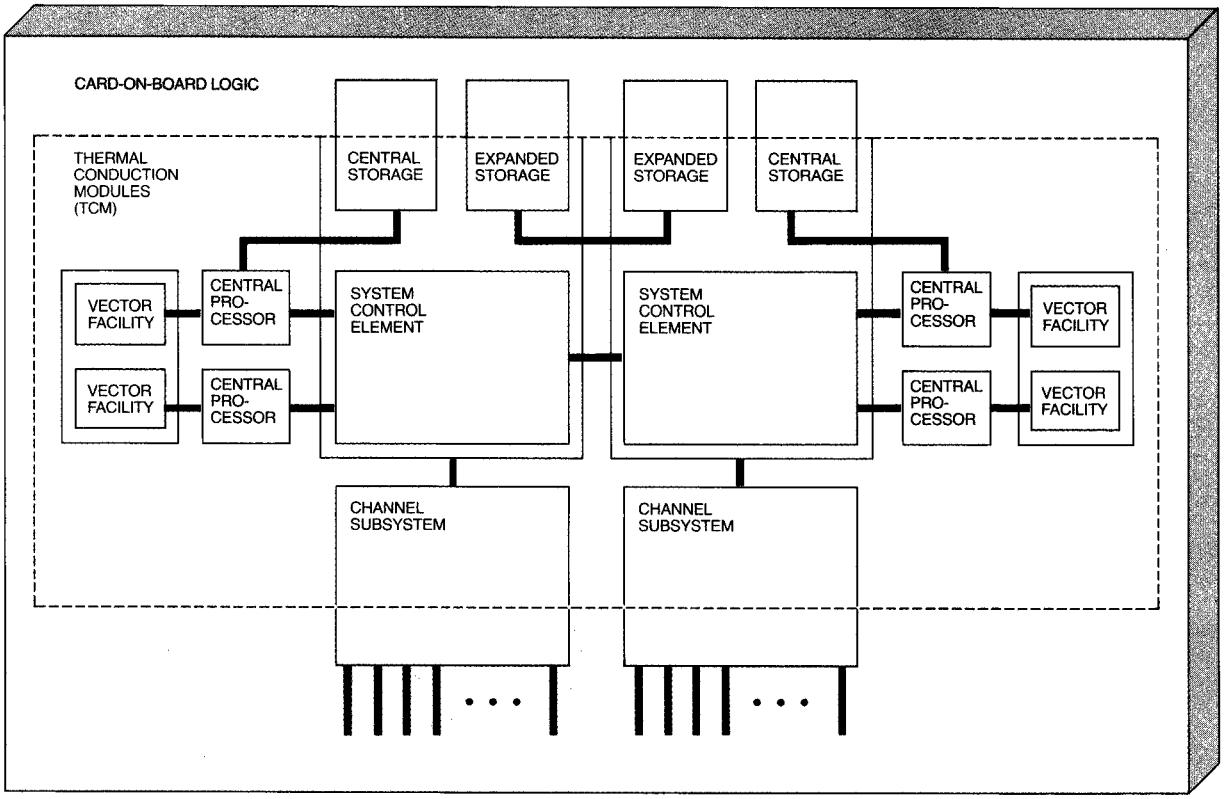
To the casual observer, the 3090 technology and design process appear very much like those of the 308X. The 3090 uses the 100-chip modules mounted on six- or nine-module boards, and the design process is based on cycle simulation, Boolean comparison, and physical design verification. On closer examination, however, it is apparent that very substantial changes have been made.

The major technology change is inside the chip. The 3090 logic chips use current-switch, emitter-coupled logic (ECL). The current-switch circuits are substantially faster than those of the 308X, and they also provide both phases on the output of each circuit. This eliminates the many stages of logic that had been previously required for phase inversion in the 308X TTL circuits. The 3090 logic chips also have the advantage of having all the terminator circuits contained on the logic chips, whereas the 308X required separate terminator chips. The result is not only that the 3090 logic is faster, but it also offers roughly a twenty-five percent improvement in logic density at the module level. The effect on the cycle time is to leave the wire delay time about the same as that of the 308X, while cutting the circuit delay time roughly in half. The 3090 also takes advantage of a more extensive and more tailored menu of arrays for such uses as working stores, directories, and control stores.

These improvements at the chip level required corresponding improvements in the power and cooling system. For example, the current that had to be supplied to a board increased from 600 amperes to over 1000 amperes, and the power that must be removed from a module to keep it cool increased from 300 to 500 watts. To put that into perspective, the module dissipates about the same power as one side of a toaster.

Significant improvements were also required to support the design of a more complex system organization. The design process starts with simulation called *cycle simulation*[2] that was used in the 308X design. Cycle simulation takes a logical description of the design and simulates it on a cycle-by-cycle, bit-by-bit basis. Models of a central processor, cache, and system control element representing over three hundred thousand circuits are run through many millions of System/370 instructions before hardware is built. The physical design is done in parallel with cycle simulation. A Boolean comparison program, also first introduced in the 308X design, is used to prove the logical equivalence of the physical design and the simulation model. Comparisons are now done on full modules of logic, often exceeding thirty thousand circuits.

Figure 1 3090 System Model 400



To minimize cycle time, three separate timing analysis programs are used at various phases of the design. An early program allows an engineer to specify a proposed path that is suspected of being long and calculate the delay and its statistical variance. Later in the design, a program that exhaustively checks all the paths is used. This program does not require a complete design as input, but is written to provide estimates of physical information that is not available, such as actual wire routing or pin assignments. At this step, in the interest of speed, the program does only a rudimentary statistical analysis of the circuit and wire delays to ensure that they meet specified cycle times. Finally, before hardware is built, a full statistical analysis of a complete physical design is done. The design is also supported by a host of checking programs that verify such things as wiring rules, crosstalk minimums, simultaneous switching limits, chip testability, and chip junction temperatures.

As a result of the extensive verification process, the initial hardware typically is able to run the majority of programs and diagnostics. Timing bugs found during hardware test are rare. When problems are found in the hardware, the preferred technique is to reproduce the problem in the simulator and debug it there, because it is easier to see what is wrong in the simulator than in the actual hardware.

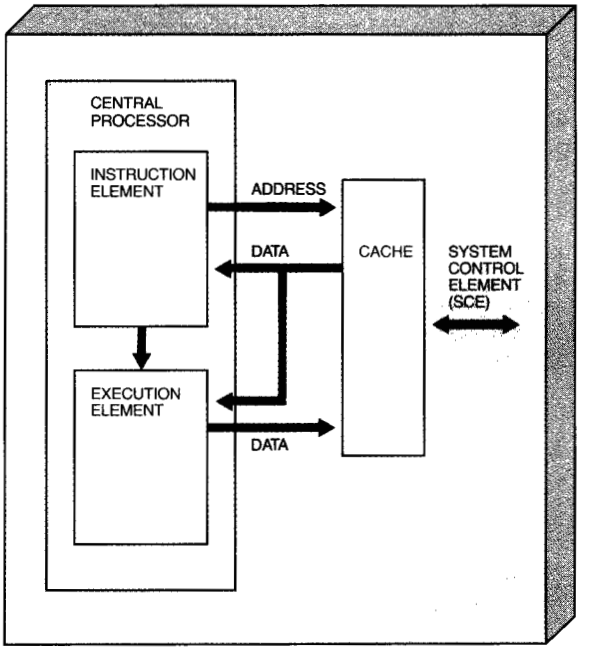
The IBM 3090 system structure

Figure 1 is an overall diagram of the 3090 System Model 400, which is a multiprocessor system. Essentially the Model 400 consists of two Model 200 dyadic systems, interconnected by paths between

their system control elements and expanded storages. A system control element serves to connect two central processors (with Vector Facilities), the channel subsystem, the other side of a Model 400 system, and the central and expanded storages. The dashed line shows which part of the system is implemented using the 100-chip thermal conduction modules (TCMs) and which is implemented in card-on-board logic. Board boundaries are indicated by the shading. The two Vector Facilities share a board. Each central processor, each system control element, and the TCM portion of each channel subsystem occupy a TCM board.

The central processor. The central processor illustrated in Figure 2 consists of an instruction preprocessor element (I) and an execution element (E). The central processor is connected to a cache. The I and E element design incorporates pipelining and overlap features, many of which were introduced on the IBM 3033 or even earlier. The I element prefetches instruction double words into one of three four-double-word instruction buffers. From there instructions are moved into an instruction register where they are decoded. Decoding and address generation are accomplished concurrently, and the address is sent to the cache as an operand fetch all in one cycle. Decoded instructions with a starting control-store address are put into a four-element queue of instructions ready for the E element to execute. The queue acts as a speed-matching buffer between the I and E elements. If there are few delays in the I element, it may advance a few instructions ahead of the E element. Thus, when the I element is delayed—for example, due to waiting for an instruction fetch to return after a branch—the E element may be able to continue executing instructions out of the queue. On a branch instruction, the I element can continue fetching instructions in the main line and also start another instruction stream fetching down the branch path into another instruction buffer. Thus, whether or not the branch is taken, it has the correct instruction already fetched and can switch decoding to that stream. The I element can fetch one double word per cycle. Three instruction streams and the operand fetching compete for priority. The I element contains the necessary logic to ensure that actions taken which are not in the strict architectural one-at-a-time instruction sequence are valid. For example, there is address generation interlock logic to ensure that the I element never uses a general register value for address generation when there is an instruction in the instruction queue that is about to modify that general register. Similarly, there is logic to ensure, for example, that no subsequent store instruction attempts to modify an instruction that has already been prefetched. If there has been a modification, the instruction must be refetched to reflect the updated value after the store.

Figure 2 3090 central processor and cache



The E element contains an operand buffer to hold double words that have been fetched by the I element and returned from the cache to the E element. Whenever there is a valid instruction in the queue (and the operand, if required, is in the operand buffer), the instruction can be executed and removed from the queue. For simple instructions, the E element can do this at a rate of one instruction per cycle. Both the I and E elements have copies of the general registers. The E element has the floating-point registers and the control registers.

The E element is a horizontal microcode-controlled engine. *Horizontal microcode* is written with long microwords—in this case, words over 100 bits in length—containing many fields, each controlling a separate part of the data path, or microcode

branching. Thus, during a single cycle, it is possible to use simultaneously the parallel adder, shifter, and serial adder, to read and write general registers and, at the same time, to update counters. The microcode resides in a mostly read-only control store, but one that also has a writable portion to assist in updating, changing, and debugging the system.

This all results in a pipelined organization that can have many instructions in various stages of processing at a given instant of time. For example, there could be several prefetched instructions in the instruction buffers, one being decoded from the instruction register, several operand fetches outstanding, up to three instructions waiting in the queue, and one in the queue being executed. Thus there might be a total of seven or more partially processed instructions.

Performance. There are many aspects to performance in a computer system. Cycle time has been mentioned, but it is far from the whole story. Relative to a 3081KX, the cycle time improvement contributes about 30 percent of the 3090's performance improvement, whereas the performance overall is improved from about 70 to 90 percent and up to 200 percent for some scientific workloads. Another aspect of performance has to do with the instruction-per-cycle rate obtainable if there were no cache misses. This is often referred to as the *infinite cache rate*, which is determined by the central processor organization.

In designing the 3090, performance simulators were used to evaluate the effect of many possible enhancements. These simulators operated at a higher level than did the circuit-level simulator that was used for logic design verification. Nevertheless, the performance simulators were detailed enough to show cycle-by-cycle performance on selected job streams containing many millions of instructions. As a result of this work, many design enhancements that were found to offer good cost/performance trade-offs were included in the central processor. Some of the more significant ones are discussed here.

A study of workloads indicated that in certain applications decimal instructions were heavily used. Thus the I element was changed to prefetch decimal operands and overlap decimal execution. Decimal multiply and divide were made much faster by handling decimal multipliers and divisors in parallel.

Half-word instructions on the 3033 execute in two cycles, one to propagate the sign into the left-half 16 bits and a second to execute as if the instruction were a full-word operation. These instructions have been improved on the IBM 3090 to allow one-cycle execution.

LOAD ADDRESS is an instruction that adds the contents of two general registers to a displacement field from the instruction and puts the result into a general register. That instruction is frequently followed by an instruction that uses the result as an address. LOAD ADDRESS used to be placed in the queue to await E element execution in turn. In the 3090 it is pre-executed. Thus, a subsequent instruction that needs the result of the LOAD ADDRESS for address generation can proceed without delay.

Of particular importance in the engineering and scientific workloads are loop-closing branches [BXLE, BXH, and BCT(R)]. BXLE and BXH are instructions that increment a general register, compare the result to a limit, and branch if the limit has not been reached. These instructions are heavily used as the last instruction in critical DO-loops. In the 3033, they had been guessed as successful in the I element and then awaited their turn in the queue for execution in the E element. In the 3090, these instructions are pre-executed in the I element at decode time. When this has been done, the outcome of the branch has been determined and need not be guessed. Also, the result of the incrementing operation is saved, and it is very likely that one of the next few instructions to be decoded will use that result for generating an address. Again, as in the LOAD ADDRESS case, the saved result can be used immediately instead of waiting for the BXLE to be executed in the E element. The BRANCH ON COUNT instructions benefit from similar pre-execution, which eliminates guessing the way branches will go.

Branches on the condition code have always been disruptive in pipelined machines, because the results of previous instructions in the pipeline awaiting execution are required before the branch direction can be determined. Three techniques are used to alleviate this situation.

A very simple one is to keep track of all the instructions in the queue. If none of them is a condition-code-setting instruction, the current value of the condition code can be used with assurance that it is correct. With this enhancement and loop-closing branches taken care of, the only branches left that need to be guessed are branch-on-condition-code instructions that actually do have a condition-code-setting operation ahead of them in the pipeline. For them, a decode-history-table scheme is used, in which a table keeps the history of branches. Just as past references are used to predict the data that should be kept in a cache, so past branching results can be used to predict future branch behavior. The location of the branch is used to reference the table, and, if the table indicates that the last branch was successful, the current branch is predicted to be successful. Finally, the instruction LOAD AND TEST REGISTER, COMPARE LOGICAL IMMEDIATE, or TEST UNDER MASK followed by a BRANCH ON CONDITION is recognized and given special treatment. If the branch is incorrectly predicted, the alternate stream instruction can be decoded on the same cycle in which the condition code is set, rather than on the following cycle.

Additional improvements were made to the multiply and floating-point add operations.

Additional improvements were made to the multiply and floating-point add operations. All fixed- and floating-point multiply instructions were built using a half cycle (9.25-nanosecond clock) and a highly parallel carry-save adder design. This makes possible a MULTIPLY LONG product generation in three cycles, not including the final add and normalization cycles. Multiply by zero is also detected and treated as a trivial fast case. Furthermore, floating-point add instructions have been improved by a technique that eliminates most unnecessary recomplement and normalization cycles.

The storage system. The central storage system of the 3090 includes a cache for each central processor, a system control element (SCE) (or one for each side of a Model 400), the central storage arrays, and the expanded storage feature.

The cache. Every large IBM system since the System/360 Model 85 has had a cache.[4,5] A cache is a relatively small storage with a fast access time that is used to hold portions of central storage that have been referenced most recently. In the case of the 3090, the cache is a 64K-byte buffer with pipelined, two-cycle access.

An aspect of performance that must be considered in the design of a large system is the penalty added to the infinite cache performance by the cache design. The relatively simple cache organization, called *store-through*, is one in which every store goes not only to the cache, but also to the central storage array. In this organization, the central storage array always holds the latest copy of the store. However, as new methods and circuits reduce the cycle time of the processor with respect to the central storage cycle time, the penalty of storing through on every store action becomes relatively larger. Thus, on the 3090 as on the IBM 308X, a *store-in-cache* design is used. In a store-in-cache design, a store operand goes into the cache of the processor doing the store operation, but it is not sent to the central storage immediately. A *cache line* (i.e., the unit of transfer between the cache and the central storage) is kept solely in the cache until another processor requests that line or until the cache location is required for a different line. This adds the complication to the system control element that, on any storage request, it must determine where the latest copy of a line of storage resides; it could be in the central storage arrays or in any one of the caches.

The access time requirements prohibit the luxury of using the address to look up the location of a double word in cache and then accessing the double word.[6] Instead, the cache is divided into four sets of 2K double words each. A given double word, as determined by the low-order bits of its address, can reside in one particular double-word location of any one of the four sets. The directory containing the addresses of the lines in the cache

is similarly divided into the same four sets, one of which (or none on a miss) indicates that the word is in that set. On an access, four directory entries and four cache double words are read out at the same time. The directory entries are used to determine which, if any, of the four cache double words to select. To minimize the critical access time, the directory chip was custom designed with built-in logic in addition to the directory array.

An interesting complexity in cache design that has been given special treatment in the 3090 cache has to do with synonyms. Virtual storage in System/370-XA architecture allows relocation of 4K-byte pages. This means that the low-order 12 address bits that address a byte within a page are the same for both a virtual and a real address. Architecture, however, allows different virtual addresses to map to the same real address. Thus the cache is managed by real addresses, despite the fact that it is accessed by virtual address. Since it takes 16 bits to address a 64K-byte cache and there are only 12 real bits available, we lack four bits. There are thus 16 places in the cache where an operand might reside. Four of these locations are read out of the cache simultaneously on the initial cache read operation. The directory, however, is built to read out all 16 entries simultaneously. Thus, if there is a miss on all of the primary four locations but a hit on one of the other 12, the cache can be read correctly with a minimum delay.

The cache can be accessed in the following four modes:

- Processor fetches
- Processor stores
- Cache line transfers from the system control element (SCE)
- Cache line transfers to the SCE

In all of these modes, the transfers can be done at a rate of one double word (8 bytes) per cycle. The cache, however, is built to read or write one quad-word (16 bytes) per cycle, thus providing it with double the bandwidth. As a result, it becomes possible to interleave the various cache access types. For example, if a cache line is being transferred from cache to central storage, every other cache cycle is free for a processor access, which prevents the processor from being locked out for the duration of a cache line transfer.

A computer system design is influenced by many aspects of earlier systems. The diagram traces the roots of the IBM 3090 system. The first IBM machines were built using relays as active logic and control elements, as exemplified by the IBM Automatic Sequence Controlled Calculator, also known as the Harvard Mark I Machine, built in 1943. In the early 1950s, electronic stored program machines used electron tubes, one of the first of which was the IBM Type 650 Magnetic Drum Calculator. By the late 1950s, the electron tubes had been replaced by transistor technology.

The architecture of a machine is a statement of what the machine does, its data formats, and its instruction set. On the basis of their architecture, the IBM systems of the fifties and early sixties can be grouped into several lines, among which are the following:

- The 701 line (701, 704, 709, 7090, 7094, and 7094-II) primarily for scientific applications
- The 702 (702, 705, and 7080) and the 7070 (7070 and 7074) lines primarily for large commercial use
- The 1401 line (1401, 1410, and 7010) primarily for smaller commercial use
- The 7030 (Stretch) a scientific "supercomputer" in its day

In 1964, IBM announced the System/360 series, which was unique in that it provided a line of machines with widely varying performance and storage sizes, all with a common architecture. The initial System/360 machines shipped were the Models 30, 40, 50, 65, and 75. They ranged in organization from the Model 30, which was a one-byte-wide, microprogrammed machine, to the Model 75, an eight-byte-wide hardware-controlled machine. The 3090 processor organization can be traced back to the Model 65, which was an eight-byte-wide, microprogram-controlled machine.

Solid Logic Technology (SLT) was introduced with System/360. Previous technology had used transistors individually sealed in small cans. With SLT, transistor chips were directly mounted on half-inch-square ceramic modules that contained one or two circuits each.

The heritage of the IBM 3090 system

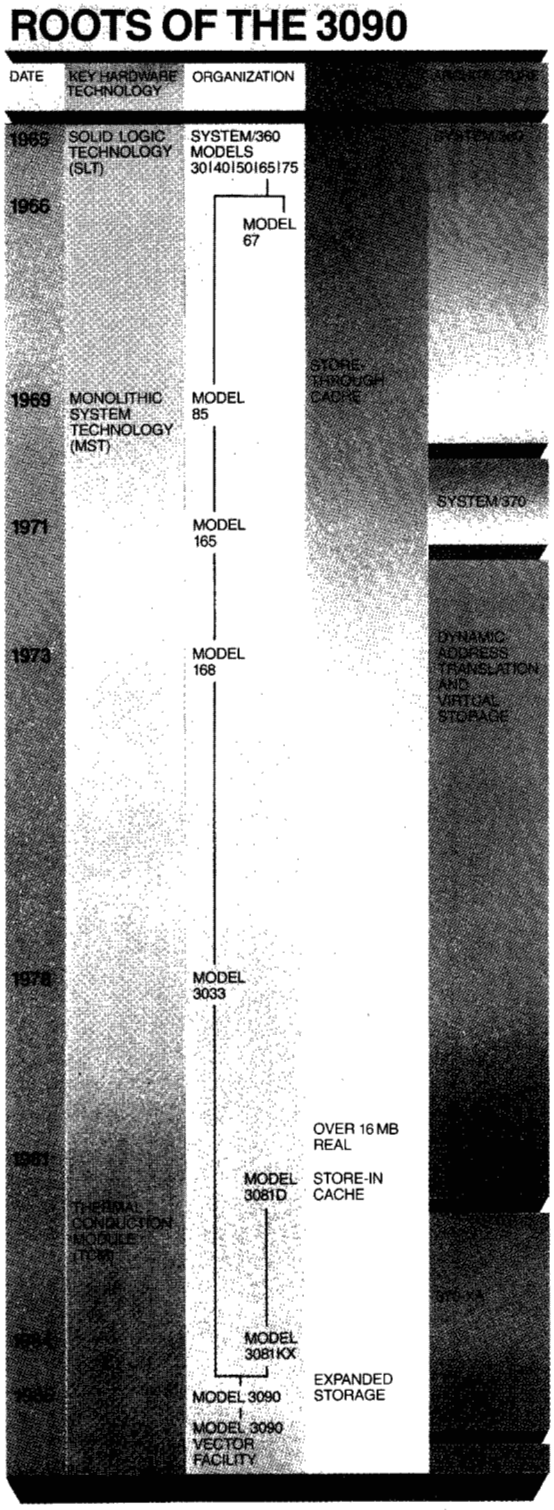
A series of processors followed the Model 65, each using more advanced technology and increasingly pipelined machine organizations. The Model 85 was the first to incorporate the cache concept, in which a small, fast storage was used to hold recently referenced portions of the central store. Integrated circuits were introduced to the line with Monolithic System Technology (MST). Here, several entire circuits were built on a single chip. The number of circuits per chip continued to increase, reaching as high as 40 by the time of the Model 3033.

New architecture was also introduced. The Model 67, a derivative of the Model 65, had dynamic address translation (DAT), a feature key to supporting virtual storage. The Model 67 was used in time-sharing applications and was first shipped in 1966. A modification of the Model 67 DAT architecture was later added to the System/370 architecture and implemented in the Model 168 and later models. Architecture allowing addressing up to 64 megabytes of real storage was introduced on later models of the 3033 and the early 308X models.

The thermal conduction module (TCM), another major step in technology, was introduced with the 308X series. The logic chips have cells for over 700 circuits and mount on ceramic modules that have sites for 100 chips, thus greatly improving the circuit density. The 3090 uses an enhanced version of the TCM technology with faster circuits. Another aspect of the 308X that strongly influenced the 3090 storage structure is the store-in cache design of the 308X. Enhancements to the architecture continued with the introduction of the 370-XA architecture and its implementation on 308X models and the 3090. The 370-XA architecture provides the ability to address two gigabytes of both virtual and real storage, and provides new I/O facilities.

The 3090 storage system has the new expanded storage feature, which allows electronic block transfers to and from the central storage. The Vector Facility, and implementation of the System/370 vector extension architecture, was also introduced on the 3090.

Stuart G.Tucker



The system control element. Physically, the system control element (SCE) serves to connect the central processors, the channel subsystem, and the other side of a Model 400 system to the central storage arrays. Logically, it provides access to central storage for the central processors and the channel subsystems. However, with store-in caches, any lines in a cache that have been stored into become part of the central storage system. Thus the latest copy may be in the central storage arrays, or it may be in either cache attached to the local SCE, or in either cache attached to the SCE on the remote side of a Model 400 system. Thus the SCE must ensure that there is always a known single latest copy of any line and must be able to find it and transfer it to any of the requestors.

Another component of system performance is the degree to which the SCE can do this for increasing aggregate processor rates and I/O rates without introducing delays greater than those that would be encountered in a single-requestor system.

The requestors to a 3090 SCE are the two central processors, the channel subsystem, and the remote SCE. The SCE has two request registers for each requestor, one for fetches and one for store requests. These registers are connected by independent request and address buses, and thus can be set by a requestor on any cycle in which they are free. The requests can be for the transfer of from one double word up to a full cache line. On each cycle, the SCE can examine all eight request registers and select one on a priority basis.

The 308X shared the address and data buses and generally allowed data to be transmitted in only one direction at a time. The 3090 SCE not only has independent address buses, but it also allows double words to be transmitted both to and from the central processors and to and from the central storage arrays on any given cycle. The data portion of the SCE can be viewed as three independent double-word, cross-point switches, each of which can transfer a double word from any input data bus to any output data bus.

To further increase the efficiency of the SCE, the cross-point switches are assigned only for the actual duration of the data transfer, rather than being assigned for the entire time required to process a request. Consider, for example, a processor request to fetch a line. During the early part of the command the other caches must be tested to ensure that they do not have the latest copy, and the address must be sent to the central storage array and the line read out of the array. During this time, no cross-point switch is tied up by the operation. The assignment is delayed until the transfer from the storage array to the requesting central processor is ready to start. This minimizes contention for cross points.

An interesting feature of the storage structure is the *fast path*.

Building the cross points to connect any input bus to any output bus supports the implementation of many transfer modes other than simply requestor to or from the central storage arrays. One of these is direct cache-to-cache transfer. In a store-in-cache system, it is possible to find that the desired line is in another cache. One approach is to force the line to be transferred to the central storage array and then reprocess the request. Another, used in the 3090, is to set up the SCE cross points to provide a cache-to-cache transfer without ever sending the line to the central storage arrays. This can be done even when the cache that has the line is on the far side of a Model 400 system. Both SCEs participate in the transfer. The remote SCE does a cache-to-inter-SCE-bus transfer while the local SCE is transferring the double words from the inter-SCE bus to the requesting processor's cache. The structure is also useful when the line is found to be in a central storage array serviced by the remote-side SCE. The transfer can be set up just as previously mentioned, except that the remote SCE takes its input from the storage array bus instead of the bus from the cache. Similar techniques permit direct transfers from any cache to either channel subsystem.

A feature that provides a surprising speed advantage is the special handling of the pad characters

on the MOVE CHARACTERS LONG (MVCL) instruction. The architecture of the MVCL instruction states that if the result operand is longer than that of the source, the rest of the result field is to be filled with pad bytes taken from a general register. A common application of the instruction is to use a zero-length source and a zero pad to clear large blocks of central storage. The special treatment was designed to provide an SCE command to fill a line with pad bytes. A double word full of the pad bytes is sent to the SCE, which forwards it to the central storage where it is replicated as a full line of pad bytes. Thus a full-line transfer is effected at the cost of a double-word transfer.

Technology has contributed to a cycle-time improvement of about thirty percent.

Another interesting feature of the storage structure is known as the *fast path*. Here, one of the two processors of a Model 200 (or one processor on each side of a Model 400) has a direct, line-fetch, request bus to the central storage arrays and a corresponding double-word, data-return bus. A request is sent in parallel to the SCE and to the storage array. If the SCE determines that the requested line is in fact in the array, it simply lets the independent fetch path proceed. If not, the SCE cancels the independent fetch and handles the request routinely. This saves a few cycles in the normal case.

Thus far we have discussed several contributions to the 3090 performance. Technology has contributed to a cycle-time improvement of about thirty percent. The increased pipelining and overlap in the I and E elements contribute to the infinite-cache rate. The cache and SCE bandwidths and parallelism contribute further by reducing the cache-miss penalties. The net of these contributions adds another thirty to fifty percent, yielding an overall improvement of 1.7 to 1.9 in the internal throughput rate over the 3081KX in commercial environments. The improvements are even greater in the engineering and scientific environment due to the looping and floating-point improvements.

The expanded storage. The expanded storage feature is a new storage designed for electronic block transfers of 4096-byte pages. It is not, however, attached in the asynchronous manner of an I/O device, nor is it simply additional central storage. It is a separate storage that allows block transfers of 4K pages between itself and central storage, under control of the operating system. The expanded storage feature runs synchronously with respect to the central processor that requests the transfer. The decision to design this storage feature for synchronous transfer was based on a study comparing the central processor times required for synchronous and asynchronous operation. In the synchronous mode, the time starting with the processor's requesting a block transfer and ending when the transfer is complete is lost to the processor; the processor can only wait for the transfer to complete. In the asynchronous mode, after the block transfer has been initiated, the operating system must do a task switch, and incur the associated save and restore overhead, if it is to utilize the processor for something else while the transfer is in process. Transfer completion would then cause an interrupt that the operating system would handle, after which it would dispatch another task. The path length through this processing was studied and found to take substantially longer than the time required for the actual transfer. The added complexity of building an asynchronous transfer results in a net loss of useful central processor time; thus the synchronous design was chosen.

The block-data transfer is done directly between the expanded storage and central storage, with little interference to the rest of the storage system. The expanded storage arrays can be accessed at a peak rate of a quad-word every four cycles. Accesses to the central storage arrays are interleaved by line so as not to tie them up for the duration of a block transfer. The SCE need be involved only enough to ensure that the most recent copy of a line is in fact in the central storage array. If this is the case, the SCE need have no further involvement in the transfer. Even if the expanded storage and the central storage arrays are on opposite sides of a Model 400, the transfer is done over a separate bus, so that there is no SCE involvement

in the data transfer or the associated interference. The result is the ability to achieve a complete page transfer operation, including the operating system code, in about 75 microseconds, with relatively little interference to the rest of the system.

There are other advantages to not designing the expanded storage as an additional central storage:

- The expanded storage is designed for high bandwidth without requiring fast access to it on central processor requests. This offers the critical advantage of not having to make the expanded storage accessible through the caches. Unless a line has its latest copy in one of the caches, a transfer to or from the expanded storage has no effect on the caches.
- From an addressing viewpoint, the expanded storage offers the advantage of providing a larger address space for future use. Whereas each central storage address designates one byte, each expanded storage address points to a 4K-byte block.
- Protection functions are handled directly by the operating system, which controls all transfers. This eliminates the need for storage keys in the expanded storage and the costs of implementing and managing the associated protection, reference, and change bits.

Reliability features of the storage system. The 3090 storage system has many enhancements relative to its predecessors that improve its fault-tolerance and serviceability.

Both the central and expanded storages have error-correcting codes. The central storage has a single-error-correcting, double-error-detecting code on each double word of data. The code is designed to detect all four-bit errors on a single card. The correcting code is passed to the caches on a fetch operation so that it can cover transmission errors as well as storage-array errors. The expanded storage is even more fault-tolerant. Each quad-word of the expanded storage has a double-error-correcting, triple-error-detecting code. Again, a four-bit error is always detected if caused by a single-card-level failure.

Both the central storage and the expanded storage have provisions for the correction of some multiple errors that exceed the limit of the error-correcting code. The technique is based on a concept called *complement/recomplement* or *double complement.*[7-9] To see how double complement works, consider the errors in storage to be of two classes, hard errors and soft errors. *Hard errors* are those that would persist even if the correct data were to be restored; these are termed "stuck" bits. *Soft errors* are those for which the storage would work correctly if the correct data were to be written again and reread. A soft error might be caused, for example, if an alpha particle were to disturb the charge in an array cell, thus causing the cell state to be lost.[10] Double complement can be used to correct hard errors, leaving fewer soft errors for correction by the error-correcting code. Consider a stored word that has one hard failure and one soft failure, which when read out has a double error that is not correctable in the central storage. The word is then complemented and written back into the same cell, reread, and complemented again as follows:

a. Correct word	1 0 1 0 0 1	
b. Failure mode	−−HS−−	
c. Read out	1 0 0 1 0 1	Double-bit error
d. Complemented	0 1 1 0 1 0	
e. Write and reread	0 1 0 0 1 0	
f. Complement	1 0 1 1 0 1	Single-bit error remains
g. After error correction	1 0 1 0 0 1	

Note: A comparison of steps c and e for equality gives 001000, which locates the hard-failing bit for service purposes.

The effect of the double-complement action is to give a corrected value for any bits that are stuck in the wrong state. Soft errors (or bits that are stuck in the right state) come out wrong. Of course, the underlying hard errors remain, and the page should be deallocated by software and/or the offending storage scheduled for replacement. The central storage implementation allows for the correction of up to two errors, as long as only one is soft. Expanded storage can correct all double errors and triple errors that are not all hard or all soft.[11]

To further reduce the probability of accumulating multiple soft errors, the expanded storage implements a technique known as *scrubbing*. By this technique, double lines are read from the expanded storage array periodically, run through error cor-

rection, and rewritten if corrected. Thus soft errors are periodically removed. The scrubbing rate is set low enough so that interference with normal operation is negligible.

The channel subsystem. Over the last several generations of large systems, the fastest I/O device rate has remained the same, while processor speeds have continued to increase. To support the correspondingly higher I/O demands, the number of channels has been increased accordingly. This has led to I/O workloads that, at times, have stressed the shared-channel engine designs of the 3033 and the 308X. Therefore, the designers of the 3090 adopted a channel subsystem design that incorporates individual engines for each of up to 96 channels on a Model 400.

The 3090 channel subsystem is shown in Figure 3 and consists of the following components:

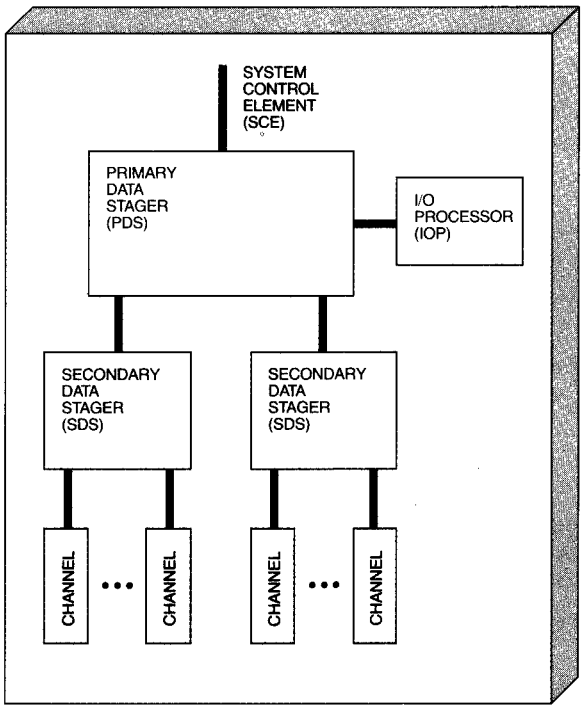
- I/O Processor (IOP)
- Channels (CHANs)
- Primary data stager (PDS)
- One or two secondary data stagers (SDS)

Each side of a Model 400 system has a channel subsystem.

The 3090 system implements the System/370-XA architecture that was first introduced on the 308X system. The System/370-XA I/O architecture can be viewed as being made up of two parts: a non-time-critical part, dealing with path management, and a time-critical part, dealing with channel command processing and interface control. The I/O processor (IOP) performs the non-time-critical portions of an I/O operation while leaving the central processor free to proceed. It is a separate engine that executes pageable vertical microcode and uses a reduced instruction set computer (RISC) architecture, which is well suited to this function. It is built in TCM technology. The IOP communicates with the central processors, receiving I/O instructions from them and posting interruptions to them, and handles the path-management functions of the System/370-XA architecture.

The channels are individual microprocessors built in card-on-board technology and controlled by horizontal microcode from a writable control store. Different microcode loads are used for different combinations of block- or byte-multiplex

Figure 3 3090 channel subsystem



channels and System/370 or System/370-XA modes. The channels perform the time-critical part of the I/O operations. These are essentially the same functions as were done by the System/360 and System/370 channels, i.e., channel control word (CCW) processing and chaining, and I/O interface control and protocol management.

The primary and secondary data stagers (PDS and SDS) act as speed-matching buffers and as funnels to concentrate up to 48 channels down to a single SCE port for access to the central storage.

The increased SCE capacity has reduced data overruns, and the provision of individual rather than shared engines to perform the channel function has the following effects:

- Virtually eliminates command overruns
- Increases byte-multiplexing channel performance (Therefore, byte-channel-configuration performance analysis is no longer done at installation time.)

- Enables the off-loading of all but essential work from the IOP, thereby increasing the channel subsystem I/O-operations-per-second capability

The Vector Facility. The Vector Facility provides for the addition of a pipelined arithmetic unit to each central processor in a 3090 system and thereby provides a substantial performance improvement for many engineering and scientific applications. The performance of the Vector Facility is discussed more fully in this issue by Gibson et al.[12] and by Clark and Wilson.[13]

When the number of elements becomes large, the average time per element approaches one cycle.

The Vector Facility has roots in an IBM study in the early 1970s.[14,15] At that time two types of vector facilities were being considered. One possibility considered was a vector arithmetic unit integrated with the host processor; the other was a stand-alone unit sharing central storage with the host processor. Implementations that accessed all operands directly from central storage were considered but rejected in favor of an architecture based on a set of vector registers. The vector registers provide the bandwidth that is critical to supporting the high vector-processing rates while minimizing the additional load on the central storage. The vector registers can also be tailored to a carefully matched architecture so as to provide multiple, conflict-free accesses on each cycle.

The 3090 Vector Facility is integrated in that the System/370-XA Vector Architecture[3] provides 171 new vector instructions that can appear in the instruction stream of any central processor with a Vector Facility. It is separate in the sense that it is implemented using a set of vector registers and a pipelined arithmetic unit packaged on a separate board. The board can hold two Vector Facilities, one for each central processor in a Model 200. This arrangement allows the Vector Facility to be offered as a field upgrade.

The vector instructions allow a single instruction to specify that the same arithmetic operation be performed on corresponding elements of the vector operands. For example, the addition of vectors $A_1, A_2, A_3, \cdots, A_n$ and $B_1, B_2, B_3, \cdots, B_n$ implies formation of the sums $A_1 + B_1, A_2 + B_2, A_3 + B_3, \cdots, A_n + B_n$. The pipeline is arranged so that individual element pairs flow through it advancing a stage every cycle. Typical stages are vector register (VR) read out, add preshift, add, normalization, and VR write. A new pair of elements can be sent into the pipeline on every cycle. After a pipeline-fill delay, a result can be produced on every cycle.

The vector registers (VRs) consist of 16 registers, each with 128 elements of 32 bits each. The VRs can also be coupled to form eight registers, each with 128 64-bit elements. The number of elements per register is a model-dependent tradeoff for the machine designer. The time needed to do a vector operation consists essentially of the overhead for pipeline starting and filling plus one cycle per vector element. Thus, using a large number of elements allows allocating the startup time over many elements. When the number of elements becomes large, the average time per element approaches one cycle. However, this efficiency must be balanced against the added time required for saving and restoring VRs on a task switch and the practical cost and space limits. This consideration led to the choice of 128 elements per vector register for the 3090 Vector Facility. The interleaving and organization of the VRs have been designed to allow two elements to be read out and one element written in on every cycle. Pipeline lengths are arranged for every instruction to ensure that the two reads and the write do not cause a conflict.

Multiply was considered to be sufficiently important in engineering and scientific applications to set an objective of doing multiplies at a one-per-cycle rate as well as adds. Several rather elaborate and lengthy pipeline schemes were investigated before the following relatively simple approach was adopted. The base 3090 multiplier requires three cycles, if stripped of its final add and normalization, which are easily pipelined. To allow a one-cycle rate, the vector element has three

base 3090 multipliers. Operand pairs are dealt into them, one to *A*, one to *B*, and one to *C*. By the fourth cycle, the first result is out of *A*, and *A* is ready to receive the fourth pair. The three multipliers behave just as though they were a three-cycle multiply pipeline.

Since the multiply pipeline is essentially independent of the add pipeline, it is possible to configure the two pipelines as one long pipeline that can do MULTIPLY and ADD instructions. The compound instructions then allow both a multiply and an add to proceed at a one-cycle rate.

A Vector Facility is attached to the instruction and execution elements of a central processor and has no direct connection to the storage system. The instruction element fetches and decodes all the instructions including any vector instructions that are in the instruction stream. The instruction and execution elements access storage, much as they do in System/370 operations, and then transfer the operands to or from the Vector Facility. The cache is used in the normal manner, except for two special modes that optimize for the case in which storage operands are stored with *stride one* (for elements that are contiguous in storage) or *stride two* (for every other element in storage). For contiguous double-word elements, a line-fetch mode is used. If there is a cache miss, a line must be brought into the cache. In line-fetch mode, the double words are passed on to the Vector Facility at the same time they are being put into the cache. If need be, the next line is automatically started. All this is done in the attempt to come as close as possible to providing a stream of double words to the Vector Facility at a one-per-cycle rate. This method works for stride-two single-word vectors too. However, for contiguous single-word operations, elements show up at the rate of two per cycle and would flood the pipeline, which runs at a one-cycle rate. For this case a cache-fill mode is used. In this mode lines are fetched ahead into the cache, but the forwarding to the Vector Facility is not done. For larger strides and irregularly spaced vectors, only the normal cache mechanism is used.

The System/370 architecture requires the reporting of precise interrupts. Prior to virtual storage, exceptions were considered relatively rare events associated with arithmetic anomalies as overflows. Thus, imprecise interrupts were tolerated in the Model 91 line of processors. With the advent of virtual storage systems, paging interrupts have become normal, and precise interrupts required. In the vector architecture, precise interrupts require locating both the instruction and the offending element. After considerable grappling with the management of precise interrupts in a pipeline environment, the notion of an exception pipeline emerged. A separate pipeline has been built to flow exception indications through exactly the same stages as the actual operands. If an exception is encountered at any stage of access or arithmetic, it follows that element to the final write-in stage and inhibits the write-in. All prior elements are completed normally. The interrupt is then taken, and provisions are made in the architecture to resume from that element after a fix-up.

The Vector Facility provides significant performance improvements at very reasonable cost for vectorizable routines. It offers the convenience of appearing as an addition to the System/370 instruction set.[3]

The processor controller. A description of the 3090 system structure is not complete without mention of the processor controller. There are a host of functions, other than the running of the instructions, that are an important part of a large computer. Many of these functions are done by the processor controller on the 3090 system.

The processor controller is actually two identical engines so arranged that if one fails the other can take over, in most cases, with little or no disruption. This is accomplished by having one engine run as the active controller while the other acts as a standby. The active controller leaves enough information about its state that the standby can pick it up and continue. A switch-over can be done if the active controller fails.

The processor controller has direct hardware connections to the 3090, and can read or write many internal control states while the clocks are running. With the normal clocks stopped, the processor controller can read or write any trigger or bit in the machine.

The services performed by the processor controller are the following:

- Manual controls provide an operator the ability to use a keyboard and console display to do things such as alter, display, reset, initial microcode load, IPL, start/stop clocks, and set various service modes.
- Power/Thermal control activates power on/off; regulates voltages; monitors currents, voltages, and temperatures (and automatically shuts down the system if they exceed limits); and controls marginal testing.
- *SAD*, the systems activity display, uses the processor controller to collect and display data.
- Configuration changes are made possible by an interface with the control program that performs physical configuration changes and stores and provides I/O configurations as requested.
- Recovery provides the capability of recovering from many of the errors that can occur in the 3090 and the capability for the processor controller to re-establish the state that existed prior to the error and invoke a retry.
- Diagnostic control is performed by the processor controller through the use of two types of diagnostic functions. The processor controller can run a variety of diagnostic tests for either functional or gate-level testing. It also has a set of analysis routines that attempt to record and subsequently analyze any error triggers that come on and attempt to identify the unit that needs to be replaced. To the extent that the analysis of the first-error capture data is successful, there is no need to take machine time to attempt to recreate the failure with the diagnostic tests.
- Remote service interface provides the 3090, as it did the 308X, the ability to dial out to a remote service support center. What is new with the 3090 is the ability to auto-dial. In the 308X, the dial-out was made, with customer authorization, only after the customer engineer arrived. In the 3090 system, the processor can dial out, again only with customer authorization, but at the time of error analysis. Thus the parts-required information can be available at the time of the initial error analysis.

Concluding remarks

The IBM 3090 unites an extension of the thermal conduction module (TCM) technology, first introduced with the IBM 308X series, with a state-of-the art pipelined and overlapped machine organization to provide a powerful computing facility. The 3090 Model 400 has an advanced system control element (SCE) design that allows four central processors and two channel subsystems to access the common central storage with minimal loss due to contention. The new expanded storage effectively expands the size of the central storage without affecting the normal access times to the central storage. A third dimension to the 3090 performance is the addition of an optional Vector Facility that offers improved performance at low cost for applications that are suitable for vector processing.

Cited references and notes

1. W. D. Connors, J. H. Florkowski, and S. K. Patton, "The IBM 3033: An inside look," *Datamation* **25**, No. 5, 198–218 (May 1, 1979).
2. *IBM Journal of Research and Development* **26**, No. 1 (January 1982); the entire issue. See especially the paper by R. N. Gustafson and F. J. Sparacio, pp. 12–21.
3. W. Buchholz, "The IBM System/370 vector architecture," *IBM Systems Journal* **25**, No. 1, 51–62 (1986, this issue).
4. D. H. Gibson, "Considerations in block oriented system design," *AFIPS Conference Proceedings* **30** (1967 Spring Joint Computer Conference), 75–80 (1967).
5. J. S. Liptay, "Structural aspects of the System/360 Model 85, II, The cache," *IBM Systems Journal* **7**, No. 1, 15–21 (1968).
6. C. J. Conti, "Concepts for buffer storage," *IEEE Computer Group News*, 9–13 (March 1969).
7. F. J. Aichelmann, Jr., "Fault-tolerant design techniques for semiconductor memory applications," 177–183; and C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," 124–134; *IBM Journal of Research and Development* **28**, No. 2 (March 1984).
8. B. E. Bachman and S. M. Dobrzynski, "Multiple error correction," *IBM Technical Disclosure Bulletin* **13**, No. 8, 2190 (1971).
9. J. Datres, E. Klass, J. Matcham, T. Papanastasiu, and G. Zvaigzne, "Multiple memory error correction," *IBM Technical Disclosure Bulletin* **24**, No. 6, 2690 (1981).
10. T. C. May and M. H. Woods, "Alpha-particle-induced soft errors in dynamic memories," *IEEE Transactions on Electron Devices* **ED-26**, No. 1, 2–9 (January 1979).
11. An amusing sidelight is an anomaly in terminology this has created. For years we have called multiple errors UEs (i.e., uncorrectable errors). Now we have UUEs (uncorrectable UEs) for use when double complement fails, and the oxymoron CUEs (correctable uncorrectable errors) when it works.
12. D. H. Gibson, D. W. Rain, and H. F. Walsh, "Engineering and scientific processing on the IBM 3090," *IBM Systems Journal* **25**, No. 1, 36–50 (1986, this issue).
13. R. S. Clark and T. L. Wilson, "Vector system performance of the IBM 3090," *IBM Systems Journal* **25**, No. 1, 63–82 (1986, this issue).
14. L. M. Moss, J. H. Shelly, and S. G. Tucker, "Processing vectors as a plurality of segments," *IBM Technical Disclosure Bulletin* **13**, No. 12, 3804 (May 1971).

15. S. G. Tucker, "Storage system with conflict free simultaneous access," U.S. Patent 3,812,473 (issued May 21, 1974).

Stuart G. Tucker *IBM Data Systems Division, P.O. Box 390, Poughkeepsie, New York 12602.* Mr. Tucker is currently a senior engineer in the Advanced Processor Design Department at the IBM Poughkeepsie Laboratory. He received his Bachelor of Engineering degree in electrical engineering from Yale University in 1955 and joined IBM Poughkeepsie in that year. He has held a number of technical and management positions related to large-processor design. Past projects have included work on the 7030 (Stretch), the System/360 Model 65, the 7000 series emulators on the Model 65, several study projects including early work on vector processors, and, most recently, the 3090. He is a member of the Association for Computing Machinery and SIGmicro.

Reprint Order No. G321-5258.