*A basic design objective for the Model 85 was to add a computer to the* SYSTEM/360 *line that offers high performance over a wide range of job types. Simulation studies indicate that the Model 85 will provide an average three- to five-fold increase in internal performance with main storage capacities of up to four million bytes.*

*This part of the paper discusses the major elements of the Model 85 within the architectural context of* SYSTEM/360, *including the addition of a high-speed buffer, called a cache.*

*Also summarized are the simulation studies that led to use of the cache, selection of its parameters, and verification of internal performance of the system.*

# Structural aspects of the System/360 Model 85

# I General organization

**by C. J. Conti, D. H. Gibson, and S. H. Pitkowsky**

Intended primarily as a growth system for users of Models 65 and 75, the Model 85 was developed as a natural evolutionary phase in SYSTEM/360 architecture. The Model 85 central processing unit is based upon solid-state technology that offers significant advantages over the circuit technology employed in Models 65 and 75. These advantages take the form of improved reliability, speed, and packaging densities. However, going beyond a performance gain attributable to technology, the Model 85 design objectives set out to raise throughput performance by exploiting system organizational potentialities within SYSTEM/360 architecture.

The Model 85 is designed for high performance across the entire spectrum of editing, file maintenance, and computational workloads. As a result of combined improvements in organization and technology, the internal performance of the Model 85 averages in the range of 3 to 5 times the performance of the Model 65—depending upon the nature of the job being measured. A high-speed multiply unit is optional for installations with unusually heavy computational loads.

System performance can be enhanced by attaching as much as four million bytes of main storage. Fast tape units are now available to support high throughput in a tape oriented environment.[1] The Model 85 storage and storage bus configuration will permit the attachment of faster direct-access devices as such units become available. Moreover, the Model 85 design incorporates additional

Table 1 Design parameters

| Function | System/360 Model | | | | |
|---|---|---|---|---|---|
| | 50 | 65 | 75 | 85 | 91 |
| CPU cycle (nanoseconds) | 500 | 200 | 195 | 80 | 60 |
| Memory access (microseconds) | 1.0 | 0.6 | 0.585 | 0.88 | 0.60 |
| Memory cycle (microseconds) | 2.0 | 0.75 | 0.75 | 1.04 | 0.75 |
| Memory interleaving | None | 2-way | 2–4 way | 2–4 way | 8–16 way |
| Data path width | 4 bytes | 8 bytes | 8 bytes | 16 bytes | 8 bytes |

checking, error correction, instruction retry, storage reconfiguration, and diagnostic techniques that further reduce unscheduled maintenance.

Our main purpose here is to introduce the Model 85 at the level of overall structure and performance. The reader is assumed to be familiar with the general architecture of SYSTEM/360.[2] We first discuss the organization of the Model 85, emphasizing features that differ from previous models. Second, we discuss the performance of the system and indicate the nature of the simulation studies that were deemed necessary to the design effort. Third, we briefly summarize the Model 85 inclusions that supplement performance and provide incremental gains in installation throughput.

## System elements

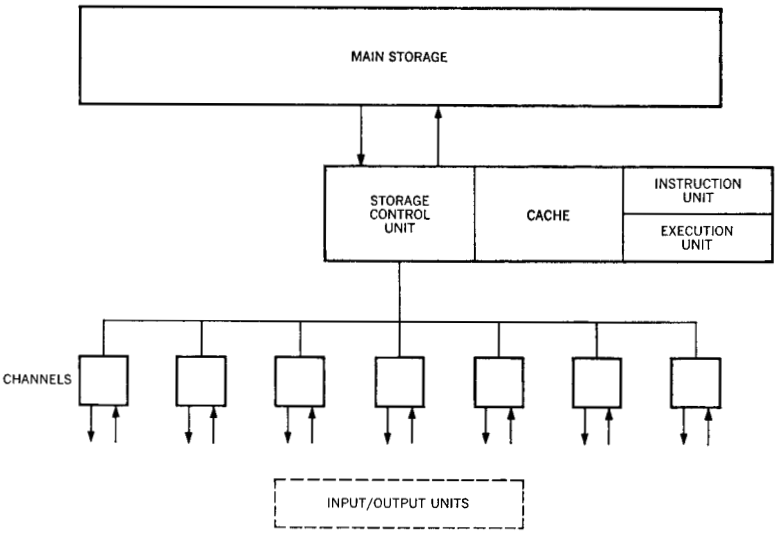
The relative position of the Model 85 in the high-performance end of the SYSTEM/360 family is indicated in Table 1. The basic machine cycle time of the Model 85 is 40 percent as long as that of the Model 65. On the other hand, the Model 85 cycle time is fairly close to that of the Model 91.

An important design feature of the Model 85 is its storage data path width of 16 bytes. The importance of storage organization to the Model 85 (and the 91 as well) is implicit in the fact that the main storages of Models 65, 75, 85, 91 are all designed from the same basic storage units; for the most part, the variations on this unit have related to packaging considerations and environmental factors. We will return to this subject later.

The main functional units in the Model 85, as shown in Figure 1, appear similar to those of any other SYSTEM/360 model. Within the central processing unit (CPU), however, one unit is distinctive to the Model 85. This is the *cache*,[3] a high-speed storage (cycle time of 80 nanoseconds) that serves as a buffer between main storage and the CPU. The purpose of the cache is to effect a work-

the cache

Figure 1 Functional schematic of the Model 85




able match between main storage and the CPU—despite a tenfold disparity in their access and cycle times. Hence, the cache—although completely invisible to the programmer—is the key organizational element in the Model 85.

Before discussing the cache, let us define 1024 contiguous bytes of storage as a *sector* and 64 contiguous bytes of storage as a *block*. In the Model 85 context, main storage is assumed to be divided into sectors and each sector subdivided into blocks. The address of a sector or block is simply the address of the first word in the sector or block.

In a basic Model 85, the cache holds sixteen sectors. Accompanying each sector is an address *tag* (the high-order 14 bits of a 24-bit address) that identifies a sector in main storage with which a cache sector is associated. At any given time, the tags point to the sixteen sectors of main storage from which words have been used most recently. Moreover, the cache contains the information fetched from these sixteen sectors and thereby avoids the need for fetching the same information more than once from main storage.

In transfers from main storage to cache, a block is a more suitable unit of information than a sector. To hold down the overhead in main storage accesses, therefore, the Model 85 moves a block as a unit in transfers from main storage to cache. A "validity bit" accompanying each cache block is set to 1 whenever the block is filled from main storage. Depending upon the pattern of word fetches, from 1 to 16 of the validity bits in a sector may be set to 1. A block consists of 64 bytes and the width of the data path is 16 bytes. The block is therefore moved in four parts of 16 bytes each—the parts following one another on the data path. (Because of the way in which main-storage addresses are assigned, the four

parts are in separate main storage units that can operate in parallel in storage configurations of one million bytes or greater.) For the sake of CPU performance, the part that contains the desired word is routed first; moreover, this part is routed not only to the cache, but also directly to the processor.

Instructions that store information in main storage in no way affect the assignment of cache sectors—such instructions are executed in a conventional manner. However, if the instruction refers to a word that is currently in the cache, the word is updated in both cache and main storage. The necessity for dual updating is evident when we realize that the input/output channels communicate with main storage in parallel with the cache.

In the course of fetching an instruction or operand, an effective (main-storage) address is generated in the usual way and compared with the cache tags. If the relevant sector is "active," i.e., pointed to by the cache, its validity bits are examined to determine if the relevant block is "valid"; if the block is valid, a main-storage fetch is unnecessary; if not, the block is fetched and the validity bit is set to 1. If the sector needed is not pointed to by the cache, it must be activated by replacing the address of an active sector in the cache.

The sectors of the cache are associated with main storage sectors by entries in an activity list. The list is ordered such that the sector most recently referred to is at the top of the list. Thus, entries for less active sectors drift to the bottom of the list. If a sector not in the cache is referred to, the entry for the sector that has gone longest without being referred to is displaced. One of the advantages of this scheme is that the number of sectors in the cache can be readily altered. Although an instruction retry mechanism in the Model 85 (discussed later) usually allows successful re-execution of instructions, the cache permits continued operation despite a persistent fault in a sector. If the checking circuits indicate such a persistent fault (which would degrade performance), the sector can be logically removed from the cache.

The net effect of the cache is to sharply reduce the number of required main storage fetches (Liptay discusses this in Part II). As a result, CPU performance is much less dependent on storage access time, and the designer becomes free to consider new alternatives. For example, he can allow for more main storage, although this implies longer cables and thus longer access times. The access time penalty of additional error detecting and correcting circuitry also becomes more acceptable. In addition, the effects of channel interference become less significant, because the cache reduces CPU referencing of main storage; in other words, channels with higher priorities than the CPU are less likely to "steal" memory cycles from the CPU.

the processor

The CPU contains an instruction unit (I Unit) and an execution unit (E Unit) that are spoken of together as the *processor*. Since the I Unit is capable of a one cycle rate of decoding and issuance of instructions to the E Unit, an instruction can be processed by

Table 2 Sample E-unit execution cycles (I-unit cycles, storage access cycles, etc., are excluded)

| *Operation* | *Number of E-unit cycles*** |
|---|---|
| Fixed-point instructions | |
| load full-word | 1 |
| store full-word | 1 |
| add | 1 |
| multiply | 9* |
| high-speed multiply | 5* |
| Floating-point instructions | |
| load (long) | 1 |
| store (long) | 1 |
| add normalized (long) | 4* |
| multiply (long) | 24* |
| high-speed multiply (long) | 7* |
| add normalized (extended) | 16* |
| Decimal instruction | |
| add decimal | 14* |

* Number of cycles varies with data.

** Execution unit cycles are defined for this table as the number of machine cycles taken by the execution unit to process the indicated operation. The additional cycles taken by the instruction unit to prepare the instruction and to obtain the operand are not included.

the I Unit each 80 nanoseconds. An instruction buffer in the I Unit can queue up to three instructions ahead of current execution. For branch instructions, as many as 16 bytes of each leg of the branch may be prefetched. This helps to minimize the lost time due to branches that depend, during some interim, upon incompleted executions. Although several instructions may be in process at a given instant of time, strict instruction sequence is preserved: instruction $N$ is never completed before instruction $N - 1$. Thus, the capability for precise interruptions is preserved as in the Models 65 and 75. Once the I Unit has decoded an instruction and fetched the required operand from main storage (if such is necessary), the instruction is passed on to the execution unit as a pseudo register-to-register (RR) instruction, much as in the Model 91.[4] Two operands from storage may be buffered in this manner.

The E Unit contains buffers that receive pseudo RR instructions from the I Unit. The E Unit examines the pseudo RR instruction to determine which (if either) of the two operand buffers will be the source of data for the operation. An interlock prevents the E Unit from executing instructions with the wrong data. Because the controls of the E Unit are implemented with read-only storage, a considerable degree of flexibility is built into the Model 85, which facilitates, for example, emulation. The E Unit also contains the general purpose and floating-point registers, a 64-bit parallel adder, a 32-bit logical unit, an 8-bit binary or decimal adder, a 64-bit shifter, and an optional high-speed multiply unit. Table 2 shows sample instruction execution times in the E Unit.

The bus control unit controls the movement of data from main storage to cache, between main storage and processor, between cache and processor, and between main storage and channels. The unit is interlocked to maintain data integrity under all circumstances.

main storage

The SYSTEM/360 addressing format permits the attachment of approximately 16 million bytes of main storage. Nonetheless, the largest amount of main storage offered to date has been for the Model 91, which permits 2048K, or about 2 million bytes of main storage (we assume K = 1024). Adding more bytes of main storage would result in a slower access time for the physical distances involved in a Model 91. One of the advantages of the cache in the Model 85 is that more storage can be attached without significantly degrading system performance as a result of longer access times.

Models 65 through 91 utilize the same basic main storage, which takes the form of an IBM 2365 or 2395 (and, in the case of the Model 85, the 2385) storage unit; the choice depends largely upon packaging considerations. The 2385 contains twice as many bits of storage as the 2365 in the same physical frame and volume. (The 2385 as well as the 2395 features unit interchangeability, which helps to facilitate rapid changeover when a storage error is encountered and must be repaired.) The Models 65, 75, and 91 utilize storage interleaving and internal CPU buffering to provide the storage bandwidth (theoretical upper limit on the bit rate) required in each of the cases. The Model 91 buffers and interleaves to an unusual extent in order to support its high-performance CPU. To achieve a suitable bandwidth and average access time, the Model 85 employs less interleaving but combines the buffering advantages of a cache with the advantages of a wider data path. Data is transferred from main storage sixteen bytes at a time, as compared with eight bytes on the Model 91. (In the case of the 2365, two units are tied together to feed the wide path.) The 16-byte groupings are interleaved four ways for the two and the four million byte 2385 configurations and for the one million byte 2365 configurations. For the half million byte 2365, interleaving is two-way.

An error checking and correcting code (ECC), carried with each eight-byte grouping in main storage, corrects any single-bit failure and detects double-bit failures on main storage fetches.[5] In store operations involving fewer than eight bytes, the ECC is used to detect and possibly correct errors in data that will be regenerated. The redundant bits in this code are not transferred to other system elements but converted to odd parity on a byte basis, which is subsequently checked by the processor. The objective of the ECC is to preserve the validity of information while in main storage.

The storage units are designed with a "floating address" capability such that any storage unit can be manually assigned any appropriate address. This permits the Model 85 to function in a "degraded mode" even though one storage unit has a solid error. For the 2365 storage configuration of one million bytes, the Model 85 can function in this mode with a one-half million byte con-

figuration. For the 2385 memory configuration of two million bytes, a one million byte configuration is possible. For the 2385 storage configuration of four million bytes, a three, two, or one million byte degraded configuration is achievable. In addition, a tester built into the 2385 allows servicing of a one-million byte unit while the remaining storage is still being used.

channels

Channels are attached to main storage through the bus control unit. The IBM 2860 selector channels and the 2870 multiplexor channels are both available for the Model 85. The design of the Model 85 is sufficiently open-ended to permit the attachment of faster channels and input/output devices if and when they become available in the future.

In the Model 85, all data transfers and arithmetic operations are accompanied by parity bits, which are used to check data at the receiving point. When the checking circuitry detects an error in a CPU execution, the Model 85 will typically retry the instruction from the beginning, as though the instruction had never entered the CPU. Up to eight retries are attempted; if any is successful, there is no loss of information and the system continues to run. Although most problem program instructions are retriable from the beginning, a successful retry may not be possible in certain cases, such as a failure in storing data. Because of the relative speeds of the processor and main storage, the error may not have been detected until the processor has proceeded beyond the point where retry is possible.

ac coupling

The Model 85 technology employs "ac coupling," as a result of which the CPU, channel groups, and storage units are isolated from one another in the direct-current sense. This permits power to be shut down on any given box without affecting the operation of the remaining boxes. The field engineer can therefore operate on any of the coupled elements without affecting other portions similarly attached provided appropriate precautions are taken.

fault isolation

The Model 85 design also extends the SYSTEM/360 diagnostic capability to facilitate a more direct diagnosis of error location. Some of the diagnostic routines are written in a microinstruction language and executed out of a read-and-write form of control storage. Since a microinstruction utilizes fewer circuits than a typical machine instruction, the location of the failing element can be ascertained more precisely than is possible with conventional diagnostic programs.

operating system

Because the Model 85 is compatible with the SYSTEM/360 line, existing versions of OS/360 will run on the Model 85.[6–8] At delivery of the Model 85, OS/360 will include such features as MVT (Multitasking with a Variable number of Tasks) with storage protection; data management with data set protection and password security; and sequential, indexed sequential, direct, and teleprocessing access methods. Planned extensions to MVT include rollin/rollout, on-line diagnostics, automatic recovery procedures, and QTAM error recovery procedures. Remote job entry will be provided, as well as basic support for graphics applications.

To take advantage of the distinctive features of the Model 85, extensions to OS/360 are planned. Specifically provided will be the programming required for an optional integrated operator display console and assembler language statements for extended-precision floating-point arithmetic, a development discussed by Padegs in Part III of this paper. A recovery program will also be provided, which is designed to exploit the built-in availability features of the system. The latter includes a recording function, as well as error analysis and recovery using both hardware and programming means.

## System performance

There are at least two main areas to attack in designing an efficient system: performance in terms of more jobs per hour while the system is running (throughput) and keeping the system running more of the time (availability). Throughput, in turn, is affected by CPU-storage performance. Improved availability decreases scheduled or unscheduled maintenance time and thereby raises the proportion of useful system time. The Model 85 was designed with both of these areas in mind, as is clear from the description of system elements.

need for simulation

The significance of accurate performance evaluations to the design of a high-speed computing system was evident to Model 85 designers from the start. The anticipated degree of overlapping, buffering, and queuing in the Model 85 appeared to largely invalidate conventional performance measures based on instruction mixes and program kernels. Although it seemed beyond the state of the art to obtain direct quantitative comparisons on average throughput per day, it seemed feasible and necessary to rely very heavily upon digital simulation for quantitative comparisons of performances in the CPU-memory complex. Experience gained from a special-purpose timing simulator developed for the Model 91 gave encouragement; actual verification runs on the Model 91 have shown the timing program to yield an accuracy within two percent.

general methodology

The performance studies were conducted in essentially three phases, the first to test the idea of the cache, the second to optimize it, and the last to assess the performance ranges of the final design. All of this effort required gathering data to be used with the timing simulator that reflected the anticipated workload of the Model 85.

Although the methodology used in these studies is a subject suitable for a paper in itself, the basic procedure is to run typical jobs on a small SYSTEM/360 computer. The small computer is placed under control of a special instruction-level tracer, which yields an output tape that lists every instruction dynamically executed in completing the job. The tracer using the output tape also calculates the instruction execution times for the Models 30 through 75. The output tape identifies the instruction itself, the location of the instruction, the operand address or addresses, and the operand or operands themselves.

The trace tape provides the data to the timing program, which simulates every cycle of the CPU being analyzed. Although this procedure provides extremely accurate results, it requires a great deal of machine time, and it is not economical to handle all of every job in this fashion.

To reduce the volume of data to manageable proportions without losing its statistical significance, an address-level trace was obtained using a monitoring device. In this case, the output tape includes only the address generated by the CPU. By feeding this data to a statistical analyzer program, which assumes a Model 85 cache, the percentage of main storage activity for the program on a particular machine is obtained. This procedure permits selection of jobs or portions of jobs sufficiently representative of the range of conditions anticipated. These job segments are then timed using the instruction-level tracer and the timing program.

Another program used to reduce the volume of data provides a dynamic count of executions of object code produced by each statement in a FORTRAN source deck. This program determines which segments of the FORTRAN program are most critical of job-required CPU running time. With critical segments, it is also possible to make more economical use of the time-consuming tracer and timing programs.

The timing program used in these studies is highly parameterized. The designer can vary cache size, block size, the number of sectors, the type of main storage, multiply time, store algorithms, fetch algorithms, and, of course, the cache replacement algorithm. This flexibility permitted optimizing of the cache parameters after the cache approach had been chosen.

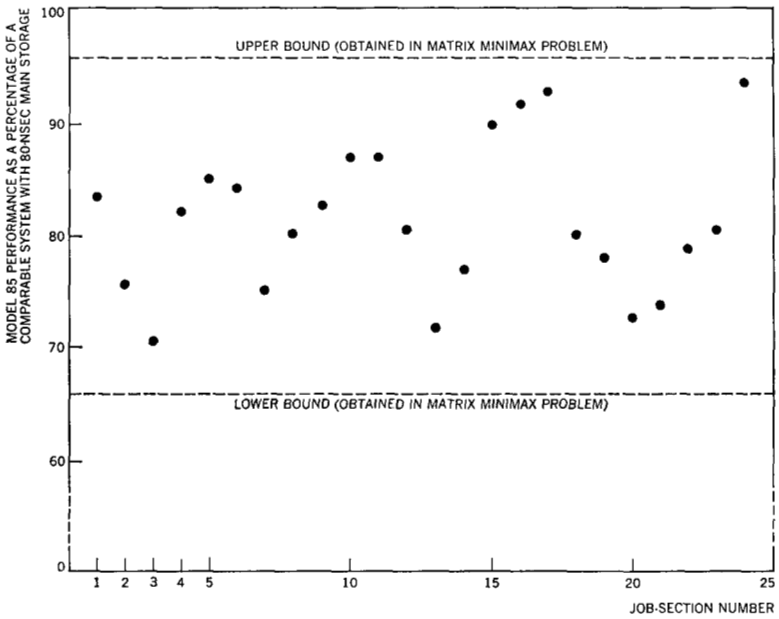
The performance studies made both to decide on the cache approach and to validate the choice included a wide range of applications. Detailed timing studies of 26 different jobs were made, covering the complete range of characteristics anticipated for the Model 85. Traces of 160 million instructions were made, resulting in 714 tape reels of trace information. Since some of the reels were used more than once, over a thousand reels of tape were processed in arriving at the Model 85 cache design.

cache justification

The major design lesson learned from these performance studies was that with a cache of 80 nanosecond cycle and a main storage in the order of one microsecond cycle, one could achieve a performance approximately equivalent to 80 percent of that obtainable with a conventional main storage of 80 nanosecond cycle. It was also observed that a 16K byte cache corresponded to an optimum cost-performance point in the design.

In addition to representative programs, a search was made for programs written specifically to defeat sector-replacement algorithms of the sort employed for the Model 85 cache. In one such problem, the "matrix minimax" problem, the task is to find the minimum of the row or column maximums in a 64 $\times$ 64 word matrix.[9] The program is instructive in that when the matrix is accessed in the direction in which data are stored, the Model 85

Figure 2 Scatter chart of the performance estimates obtained from the timing simulation of 24 job sectors




with its cache will run at or very near its best performance relative to a conventional machine. But when data are accessed in the direction perpendicular to optimum, the Model 85 will perform at or near its worst relative level. Thus, for all practical purposes, the timing results provide upper and lower bounds relative to a conventional main storage and bus configuration.

normalized performance

Figure 2 illustrates Model 85 performance normalized to the performance of a comparable machine except for an 80-nanosecond main storage attached in a conventional manner. The base is analytically useful but unrealistic—a large quantity of 80-nanosecond memory cannot presently be attached without a significant degradation in access time due to bus-length problems, as well as for reasons of economy. In Figure 2, the two minimax bounds are shown as dotted lines.

To test the hypothesis that the bounds are indeed best and worst, a large number of other program cross sections have been plotted. All of these cases do indeed fall between the given bounds. Note that the lower bound represents 66 percent of the performance expected with an idealized 80-nanosecond storage. This fact shows the cache to be remarkably insensitive to addressing anomalies.

The results of a number of representative analyses are given in Table 3. For the sake of interpretation, we can divide jobs into two classes: E-unit limited and access dependent.

relative performance

As examples of the class of programs that are E-Unit limited, consider the sample sequences numbers 9 and 10 of Table 3. For these two sequences, the internal performance relative to a Model 65 improves about 20 percent when the high-speed multiply option is added to the Model 85, indicating that the E-Unit time required to execute the multiply instruction is a critical speed determinant. This conclusion is strengthened by noting that no noticeable performance improvement occurs when the cache is expanded, even though the sequences require over 63K and 105K bytes of main storage, respectively, for execution. Sample sequence number 11 is an additional example of an E-Unit limited program.

The class of program that is access-dependent is illustrated by sample sequence number 6 of Table 3. When the cache is expanded for the Model 85, thereby improving access time to additional data for this sequence, the internal performance is seen to improve about 15 percent. This would indicate that access time is the critical speed determinant, a conclusion that is borne out by the relatively smaller improvement with the addition of the high-speed multiply

**Table 3** Relative internal performance for execution of sample instruction sequences. Numbers shown are ratios, compared to Model 65 CPU.

| *Instruction sequences taken from the following job steps* | *Storage allocated to segment* | *System/360 Model* | | | | |
|---|---|---|---|---|---|---|
| | | *75* | *85 with 16K cache* | *85 with 32K cache* | *85 with 16K cache and high-speed multiply* | *91* |
| 1. Compile job step using OS/360 FORTRAN IV (H) | 218 | 1.3 | 3.2 | 3.6 | 3.2 | 3.6 |
| 2. COBOL compile | 82 | 1.3 | 3.7 | 4.3 | 3.7 | N.A. |
| 3. Assembly job step using OS/360 Assembler (F) | 44 | 1.3 | 3.6 | 4.5 | 3.6 | 3.5 |
| 4. Link edit job step | 96 | 1.3 | 3.8 | 4.5 | 3.8 | 3.8 |
| 5. Sorting operation using OS/360 SORT | 16 | 1.2 | 3.2 | 4.1 | 3.2 | 3.4 |
| 6. Heat transfer problem | 17 | 1.5 | 3.9 | 4.5 | 4.1 | 4.1 |
| 7. Data reduction problem | 199 | 1.5 | 4.2 | 4.4 | 4.4 | 4.1 |
| 8. Curve fitting by least squares method | 173 | 1.6 | 4.6 | 4.8 | 5.3 | 5.2 |
| 9. Integral evaluation within a 3-level nested DO-loop | 63 | 1.7 | 4.5 | 4.5 | 5.4 | 7.6 |
| 10. Matrix eigenvalue calculations within nested DO-loops | 105 | 1.7 | 3.6 | 3.6 | 4.6 | 9.1 |
| 11. Partial differential equation solution using ADI method | 28 | 1.8 | 4.6 | 4.6 | 6.4 | 13.9 |

In arriving at these figures, only CPU-main storage activity was simulated. Thus, these figures represent internal performance only; they do not reflect either the total times required to run the programs or throughput. The above ratios are subject to a 10% tolerance except for the 75 ratios which are subject to a 15% tolerance. These ratios are only for the instruction sequences tested and are not necessarily correct for a complete job step.

option. A further interesting characteristic of access-dependent instruction sequences is inherent in this illustration: performance improvement resulting from expanded cache size cannot be correlated with program size. This sequence requires only a little over 17K bytes of problem program storage area for execution, yet the interplay between the problem program and the operating system is such that the expanded cache size improves performance. In contrast, sample sequence number 7 requires over 199K bytes of problem program storage area for execution, yet the expanded cache size improves performance only about 5 percent. This inability to correlate program size to performance improvement resulting from expanded cache size is an interesting and somewhat surprising characteristic of access-dependent instruction sequences.

The internal performance information provided in Table 3 should not be confused with total job time or throughput. Actual throughput depends on factors not included in the values in the table. However, the values in Table 3 represent a number of distinctive applications, the characteristics of which can be well documented. Thus it is feasible as the subject of a future study to form a weighted average that applies reasonably well to a given installation. Moreover, this data can be used as one of the inputs in arriving at relative estimates of throughput that take account of input/output traffic. Of course, these informed evaluations will not do full justice to the throughput potentialities of the large main storage in the Model 85. For example, with more storage, the programmer can reduce the need for overlays, provide larger buffers for terminal devices, hold more data in readiness, and in some cases reduce computational requirements by storing precomputed tables.

## Summary

The Model 85 is a high-performance system that will execute programs that run on smaller SYSTEM/360 computers. Its basic design includes monolithic circuits, very wide data paths, an 80-nanosecond internal buffer of at least 16K bytes, and a high-speed multiply feature. The system is designed for high throughput over a full range of applications. Among the features that bolster throughput, in addition to the high internal performance, are an exceptionally large main storage, instruction retry, error correction in main storage, and improved diagnostic and maintenance facilities.

CITED REFERENCES AND FOOTNOTES

1. The recently announced IBM 2420.
2. G. A. Blaauw and F. P. Brooks, Jr., "The structure of SYSTEM/360, Part I, Outline of the logical structure," *IBM Systems Journal* **3**, No. 2, 119–135 (1964).
3. Cache is synonymous with high-speed buffer, as used in other Model 85 documentation.

4. R. M. Tomasulo, "The IBM SYSTEM/360 Model 91: An efficient algorithm for exploiting multiple arithmetic units," *IBM Journal of Research and Development* **11,** No. 1, 25–33 (January 1967). Or see T. C. Chen, "The overlap design of the IBM SYSTEM/360 Model 92 central processing unit," *Proceedings of the 1964 AFIPS Fall Joint Computer Conference* **26,** Part 2, 73–80 (1964).
5. R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal* XXIX, No. 2, 147–160 (April 1950).
6. G. H. Mealy, "The functional structure of OS/360, Part I, Introductory survey," *IBM Systems Journal* **5,** No. 1, 3–11 (1966).
7. B. I. Witt, "The functional structure of OS/360, Part II, Job and task management," *IBM Systems Journal* **5,** No. 1, 12–29 (1966).
8. W. A. Clark, "The functional structure of OS/360, Part III, Data management," *IBM Systems Journal* **5,** No. 1, 30–51 (1966).
9. The minimax program is due to Dr. H. Hellerman.