YSTEM/360 Model 91: An efficient algorithm
ietic units," *IBM Journal of Research and*
(January 1967). Or see T. C. Chen, "The
TEM/360 Model 92 central processing unit,"
S *Fall Joint Computer Conference* 26, Part 2

ecting and error correcting codes," *The Be*
X, No. 2, 147–160 (April 1950).
al structure of OS/360, Part I, Introductor
l 5, No. 1, 3–11 (1966).
structure of OS/360, Part II, Job and tas
Journal 5, No. 1, 12–29 (1966).
al structure of OS/360, Part III, Data man
nal 5, No. 1, 30–51 (1966).
to Dr. H. Hellerman.

The cache, a high-speed buffer establishing a storage hierarchy in the Model 85, is discussed in depth in this part, since it represents the basic organizational departure from other SYSTEM/360 *computers.*

Discussed are organization and operation of the cache, including the mechanisms used to locate and retrieve data needed by the processor.

The internal performance studies that led to use of the cache are described, and simulated performance of the chosen configuration is compared with that of a theoretical system having an entire 80-nanosecond main storage. Finally, the effects of varying cache parameters are discussed and tabulated.

Structural aspects of the System/360 Model 85

II The cache

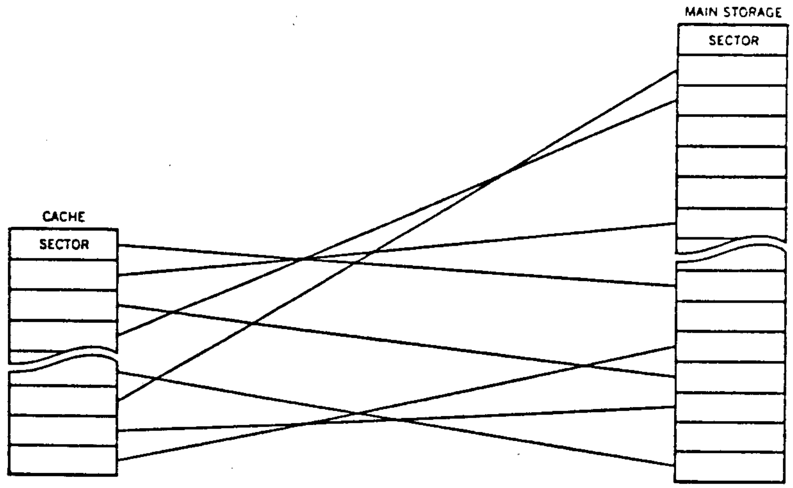
by J. S. Liptay

Among the objectives of the Model 85 is that of providing a SYSTEM/360 compatible processor with both high performance and high throughput. One of the important ingredients of high throughput is a large main storage capacity (see the accompanying article in Part I). However, it is not feasible to provide a large main storage with an access time commensurate with the 80-nanosecond processor cycle of the Model 85. A longer access time can be partially compensated for by an increase in overlap, greater buffering, deeper storage interleaving, more sophistication in the handling of branches, and other improvements in the processor. All of these factors only partially compensate for the slower storage, and, therefore, we decided to use a storage hierarchy instead.

The storage hierarchy consists of a 1.04-microsecond main storage and a small, fast store called a cache,[1] which is integrated into the CPU. The cache is not addressable by a program, but rather is used to hold the contents of those portions of main storage that are currently being used. Most processor fetches can then be handled by referring to the cache, so that most of the time the processor has a short access time. When the program starts operating on data in a different portion of main storage, the data in that portion must be loaded into the cache and the data from some other portion removed. This activity must take place without program assistance, since the Model 85 must be compatible with the rest of the SYSTEM/360 line.

This paper discusses organization of the cache and the studies that led to its use in the Model 85 and to selecting of values for its parameters.

Figure 1 Assignment of cache sectors to main storage sectors



Cache organization

The main storage units that can be used on the Model 85 are the IBM 2365-5 and the 2385. They have a 1.04-microsecond cycle time and make available capacities from 512K bytes to 4096K bytes (K = 1024). The cache is a 16K-byte integrated storage, which is capable of operating every processor cycle. Optionally, it can be expanded to 24K bytes or 32K bytes.

Both the cache and main storage are logically divided into sectors, each consisting of 1K contiguous bytes starting on 1K-byte boundaries. During operation, a correspondence is set up between cache sectors and main storage sectors in which each cache sector is assigned to a single different main storage sector. However, because of the limited number of cache sectors, most main storage sectors do not have any cache sectors assigned to them (see Figure 1). Each of the cache sectors has a 14-bit sector address register, which holds the address of the main storage sector to which it is assigned.

assigning cache sectors

The assignment of cache sectors is dynamically adjusted during operation, so that they are assigned to the main storage sectors that are currently being used by the program. If the program causes a fetch from a main storage sector that does not have a cache sector assigned to it, one of the cache sectors is then reassigned to that main storage sector. To make a good selection of a cache sector to reassign, enough information is maintained to order the cache sectors into an activity list. The sector at the top of the list is the one that was most recently referred to, the second one is the next most recently referred to, and so forth. When a cache sector is referred to, it is moved to the top of the list, and the intervening ones are moved down one position. This is not meant to imply an actual movement of sectors within the cache, but rather refers to a logical ordering of the sectors. When it is necessary to reassign a sector, the one selected is the one at the bottom of the activity list. This cache sector is the one that has gone the longest without being referred to.

When a cache sector is assigned to a different main storage sector, the contents of all of the 1K bytes located in that main storage sector are not loaded into the cache at once. Rather, each sector is divided into 16 blocks of 64 bytes, and the blocks are loaded on a demand basis. When a cache sector is reassigned, the only block that is loaded is the one that was referred to. If they are required, the remaining blocks are loaded later, one at a time. Each block in the cache has a bit associated with it to record whether it has been loaded. This "validity bit" is turned on when the block is loaded and off when the sector is reassigned.

store operations

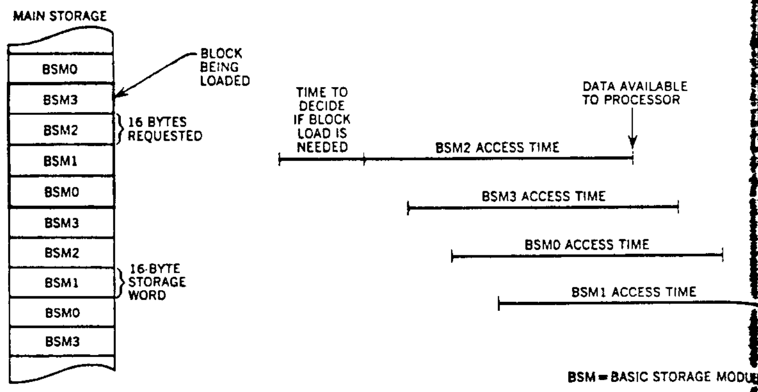
Store operations always cause main storage to be updated. If the main storage sector being changed has a cache sector assigned to it, the cache is also updated; otherwise, no activity related to the cache takes place. Therefore, store operations cannot cause a cache sector to be reassigned, a block to be loaded, or the activity list to be revised. Since all of the data in the cache is also in main storage, it is not necessary on a cache sector reassignment to move any data from the cache to main storage. All that is required is to change the sector address register, reset the validity bits, and initiate loading of a block. The processor is capable of buffering one instruction requesting the storing of information in main storage, so that it can proceed with subsequent instructions even if execution of the store instruction cannot be initiated immediately.

Two processor cycles are required to fetch data that is in the cache. The first cycle is used to examine the sector address registers and the validity bits to determine if the data is in the cache. The second cycle is then used to read the data out of the cache. However, requests can normally be overlapped, so that one request can be processed every cycle. If the data is not present in the cache, additional cycles are required while the block is loaded into the cache from main storage.

The storage word size on which the Model 85 operates internally is 16 bytes. This is the width of the data paths to and from the storage units, and is the amount the processor can store or fetch with a single request. Because a single 2365-5 storage unit operates on an 8-byte-wide interface, two units are paired together and operated simultaneously. Except for the 512K configuration, main storage is interleaved four ways. Since a block is 64 bytes, four fetches to main storage are required to load one block into the cache. With four-way interleaving, this means one request to each basic storage module. To improve performance, the first basic storage module referred to during each block load is the one containing the 16 bytes wanted by the processor. In addition to being loaded into the cache, the data is sent directly to the processor, so that execution can proceed as soon as possible (see Figure 2).

On the Model 85, channels store and fetch data by way of the

Figure 2 Timing for a block load



processor. Channel fetches are processed by getting the require data from main storage without referring to the cache. Channe stores are handled the same way as processor stores. In this way, a channel changes data that is in the cache, the cache is updated b the channels do not have any part of the cache devoted to them

Performance studies

Among the questions that had to be answered to determine wheth the cache approach should be taken were: (1) how effective is and (2) does its effectiveness vary substantially from one progra to another? The principal tools used to answer these questions a the tracing and timing techniques referred to in Part I. The traci technique produces an instruction-by-instruction trace of a pr gram operating under the SYSTEM/360 Operating System. The outp is a sequence of "trace tapes," which contain every instructi executed, whether in the problem program or the operating syste and the necessary information to determine how long it takes to executed. These trace tapes contain about 250,000 instructions ea and are used as input to a timing program, which determines, cyc by-cycle, how the Model 85 would execute that sequence of instru tions. These techniques are intended to determine internal pe formance and do not provide any information concerning throug put. An intensive investigation preceded selection of the progra used in this study.

cache effectiveness

In order to measure the effectiveness of the cache, we postulat a system identical to the Model 85 except that the storage hierarc is replaced by a single-level storage operating at cache speed. T performance of such a system is that which would be achieved the Model 85 if it always found the data it wanted in the cache a if it never encountered interference in main storage due to stor Therefore, it represents an upper limit on the performance of t Model 85; how close the Model 85 approaches this ideal can ser as a measure of how effective the cache is. Nineteen trace tap

Figure 3 Model 85 performance relative to single-level storage operating at cache speed

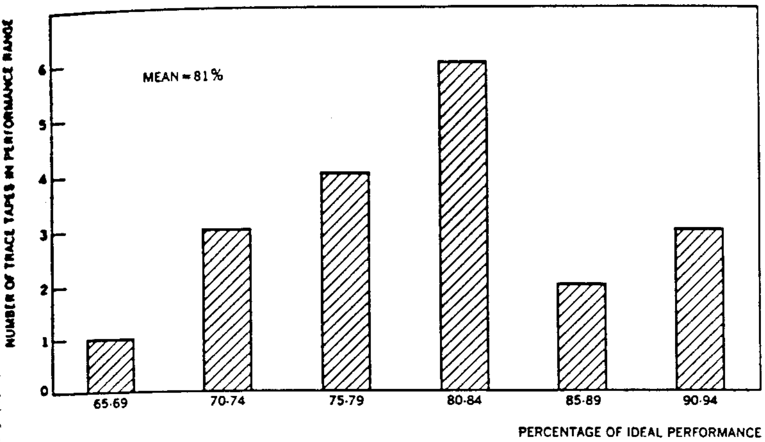
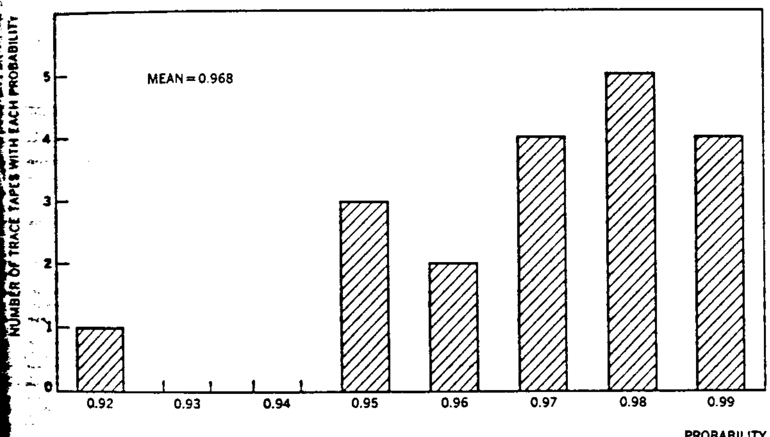


Figure 4 Probability of finding fetched data in cache



were timed for both the Model 85 and the postulated system, and the performance of the Model 85 was expressed as a percentage of the performance of the ideal system. Figure 3 shows the distribution of performance data obtained. The average was 81 percent of the performance of the ideal system, with a range between 66 and 94 percent.

An important statistic related to cache operation is the probability of finding the data wanted for a fetch in the cache. Figure 4 shows the distribution of this probability for the same 19 trace tapes used for Figure 3. The average probability was 0.968. It is worth noting that, if the addresses generated by a program were random, the probability of finding the data wanted in the cache would be much less than 0.01. Therefore, it can be said that what makes the cache work is the fact that *real programs are not random in their addressing patterns.*

Table 1 Average performance relative to an ideal system with cache size and number of sectors varied – Block size = 64 bytes

Number of cache bytes	*Number of sectors* 8	16	32
8K	0.693	0.744	0.793
16K	0.765	0.825	0.861
32K	0.857	0.891	0.902

Selection of cache parameters

Before the final cache design was established, a great deal of effort was expended on the choice of cache parameters.[2] The tools used to make the choice were the trace and timing programs. From among the trace tapes available, we picked five representative ones and ran them for many cache configurations, varying cache size, sector size, and block size. Tables 1 and 2 show the results obtained. In Table 1, block size is always 64 bytes; in Table 2, the number of sectors is always sixteen. In both cases, performance is compared with that of a single-level storage operating at cache speed. The selection of a 16K byte cache with 16 sectors and 64 bytes per block was made as the best balance between cost and performance.

replacement algorithms

The choice of an algorithm for the selection of a sector to reassign was also the object of careful study. From among the algorithms proposed, two were selected as likely candidates and incorporated into the timing program for study.

For one algorithm, the cache sectors are partitioned with an equal number of sectors in each partition. An activity list is maintained for each partition reflecting the use of the sectors within it. Each partition has a binary address, and when a main storage sector needs to be assigned a position in the cache, the low-order bits of its sector address are used to select one of the partitions. The sector at the bottom of that partition's activity list is the one chosen for reassignment.

This algorithm was studied for 1, 2, 4, 8, and 16 partitions. When there is only one partition, the algorithm becomes the Model 85 replacement algorithm. At the opposite extreme, when there are sixteen partitions, there is only one sector in each, and the idea of an activity list for each partition is meaningless. In this case, the choice of a cache sector to reassign depends only on the low-order address bits of the main storage sector for which a place is being found in the cache, and consequently each main storage sector has only one possible place where it can be put in the cache.

The second algorithm involves a single usage bit for each cache sector. When a sector is referred to, its usage bit is turned on if it is not already on. When the last sector bit is turned on, all of the other bits are turned off and the process continues. If a sector has to be assigned, it is selected randomly from among those with their usage bits off.

Table 2 Average performance relative to an ideal system with cache size and number of bytes per block varied – Number of sectors = 16

Number of cache bytes	Number of bytes per block 64	128	256
8K	0.744		
16K	0.825	0.810	0.781
32K	0.891	0.885	0.870

Table 3 Comparative performance using different cache sector replacement algorithms

algorithm	*performance*
1 partition*	1.000
2 partitions	0.990
4 partitions	0.987
8 partitions	0.979
16 partitions	0.933
usage bits	0.931

* Replacement algorithm chosen for the Model 85

Table 3 summarizes the results obtained. The choice of the activity list was made because it provided the best balance between cost and performance.

Summary comment

The inclusion of a storage hierarchy represents one of the major advances in system organization present in the Model 85. Although the concept of a storage hierarchy is not new, the successful implementation of a nanosecond/microsecond level of hierarchy was inhibited until now by the lack of a suitable technology. As implemented in the Model 85, the fast monolithic storage physically integrated with the CPU logic yields the desired machine speed, while the large core storage yields the desired storage capacity, the combination being transparent to the user. It is likely that with future progress in technology this nanosecond/microsecond hierarchy is not merely an innovation that worked out well for the Model 85, but rather it is a fundamental step forward that will be incorporated into most large systems of the future.

CITED REFREENCE AND FOOTNOTE

1. The term cache is synonymous with high-speed buffer, as used in other Model 85 documentation.
2. D. H. Gibson, "Considerations in block-oriented systems design," *AFIPS Conference Proceedings, Spring Joint Computer Conference* 30, Academic Press, New York, New York, 75–80 (1967).