System dependence on channel and direct access device architecture was addressed with the introduction of System/370. Discussed are the alternatives to this problem and their evaluation by a channel architecture program simulator.

Also presented is the solution, a block multiplexer channel and sector addressing in devices, which resulted in more efficient channel utilization and reduced programming overhead.

Channel and direct access device architecture

by D. T. Brown, R. L. Eibsen, and C. A. Thorn

During the development, testing, and use of System/360, difficulties and problems were logged, but few received the attention given to the Direct Access Storage Device (DASD). This was not necessarily because DASD attachment and use had greater inefficiencies or design problems than other devices. With System/360, DASD attained a prominence it never had before, since no previous system was as dependent on a single Input/Output (I/O) device type for satisfactory system performance.

With the introduction of System/370 and the expected continuation of system dependence on DASD, it was necessary that DASD operations be optimized.

Discussed in this paper are the specific problems experienced by medium and large systems and the three major solutions that were developed to address these DASD problems: (1) an I/O processor, (2) a buffered DASD controller, and (3) a modified multiplexer channel with a modified DASD controller. Evaluation of these alternatives was aided by the use of a channel architecture simulator. The third solution, referred to in this paper as the block multiplexer channel proposal, was adopted. The multiplexer channel with selector channel speeds that was defined and a new DASD control unit that was specified to provide rotational position sensing are then presented.

The environment

An assumption in the development of System/360 was that the attachment of direct access storage to the system would be highly desirable and almost universally accepted. This assumption was based on the fact that disk and drum technologies had sufficiently matured to make their use practical on all systems. Thus, IBM's major operating systems, OS/360 and DOS/360, required at least one and preferably more direct access devices for external storage.

The use of DASD by the operating systems was high, for it was necessary for these devices to be accessed many times during the setup and execution of every job processed by the system, in the retrieval of programs and control information, and in the updating of externally stored information. As users adopted DASD for data and program storage, these devices became crucial compared with other I/O devices in affecting system performance. Both simulations and actual practice showed that, in many cases, system performance was limited by DASD use. Because using a Central Processing Unit (CPU) with higher speed capabilities did little to enhance throughput for these systems, other approaches were investigated to improve system performance.

The time taken by a DASD to seek, that is to move the head from one cylinder to another, was the first area for which a solution was sought. A means of reducing the effects of this delay was to prohibit frequently accessed data sets from coresiding on the same device. Such a distribution of data and accesses allowed these data sets to be accessed without head movement in most cases. However, while distributing the frequently used data sets provided the greatest improvement, somewhat less improvement accrued by doing the same with the other data sets. Thus, it was found that the benefits achieved from data set distribution were limited by two factors. First, the number of frequently accessed data sets was often larger than the number of available devices. Secondly, as device activities increased, channel utilization increased until the limit of channel capability was reached.

The second approach for improving system performance was the use of additional channels across which the DASD could be spread to avoid channel capability bottlenecks. As was expected, the total DASD throughput was almost directly proportional (in a very active and well-distributed system) to the number of channels used by the direct access devices. This resulted in the extended capability, which is now provided on some systems, to attach more than seven channels. However, the high cost of this extended capability was justified on only the very large systems.

identified problems

The greatest problems, typical of medium and large OS systems, were found in architectural difficulties: high utilization of the channel and the I/O interface due to searching for records on DASD, and high operating system overhead.

The reading or writing of a record on a direct access device involves at least three specific activities at the device: seeking (selecting the desired track on the device), searching (finding the desired record on the track), and reading or writing. In the case of writing, verification typically follows. While seeking requires very little channel, interface, or control unit activity, searching monopolizes the channel, interface, and control unit facilities for a long period of time since it involves the repeated transfer of the search argument from the channel to the control unit, to be compared against each record encountered on the device. With an IBM 2314, for example, rotation time is 25 ms and a search takes 12.5 ms on the average. After the search, a read or write takes place. During the search, a process which does not make efficient use of the I/O facilities, and the subsequent read or write, no other device is able to use the channel, I/O interface, or control unit.

Of the several programming areas identified as potential problems in system throughput, the Operating System Input/Output Supervisor (IOS), which managed the initiation and termination of all I/O operations, was a concern. A significant percentage of supervisory programming overhead was contributed by IOS, due not only to its size and complexity, but also to the frequency with which it was called upon. While executing, IOS used the CPU facility which could otherwise be used for other program processing. IOS execution also impeded I/O activity because IOS must run, in part, with I/O interruptions disabled, thereby causing any I/O interruption condition arising during the disabled time to be held off until IOS allowed interruptions. Also, when a selector channel caused an I/O interruption, that channel could perform no operations until IOS handled the interruption and reinstructed the channel.

proposed solutions

Three proposals were made for solving the identified problems:

- the I/O processor
- the buffered control unit
- the block multiplexer channel

The most comprehensive proposal of the three was the I/O processor. The processor was assigned many of the functions normally performed by the operating system. To perform these functions, the proposed processor included most of the characteristics of a CPU. A suggested instruction set included arithmetic and logical operations similar to System/360 as well as I/O

operations. As with normal channels on the system, the I/O processor was directly coupled to the CPU main storage where the programs, data, and control information for channel operations resided. Some of the operating system functions subsumed by the proposed I/O processor were simple error recovery, record blocking and deblocking, retrieval of indexed records, and data address conversion. To simplify the performance of these functions, changes to the control unit design, data format on the device, and data indices were also proposed. Furthermore, changes to the operating system were proposed to provide a more functionally oriented interface for communication with the I/O processor than with normal channels.

The buffered control unit proposal was based on the use of a selector channel with data and search argument buffering residing in the control unit. Further, the proposal specified a means for searching for records which could, in a large number of cases, be handled by the control unit without prolonged channel participation. The buffering aspect involved a sufficiently large record buffer in the control unit. Once the record had been loaded from the device on an input operation, for example, the transfer to the channel could take place at a speed limited only by the channel and buffer.

The block multiplexer channel proposal required a new channel and a new control unit. The proposed channel was similar to the selector channel in that both had high date rates, but similar to the byte multiplexer channel in that a degree of multiplexing was specified. Inasmuch as neither the I/O interface nor the device-dependent programming interfaces were necessarily changed by the new channel, current devices of all types could use the channel. In addition, any buffered device would be particularly suited to the channel. The proposed control unit was significantly different in that a new order of addressing was introduced. While current DASD addressing specified cylinder and track, the proposal specified sector addressing, in addition, based on the division of all tracks into fixed sectors. The control unit would then monitor the DASD for a selected sector before beginning the search for the desired record, thus making the searching time relatively small. The multiplexing capability of the channel was intended to make efficient use of the time made available by sector orientation.

Simulation and evaluation of proposals

As solutions were proposed, it became important to evaluate the comparative performance potential of each. Early simulation experiments led to the development of an I/O subsystem simula-

tor as a vehicle for modeling the proposed designs. The simulator that evolved emphasized the following characteristics:

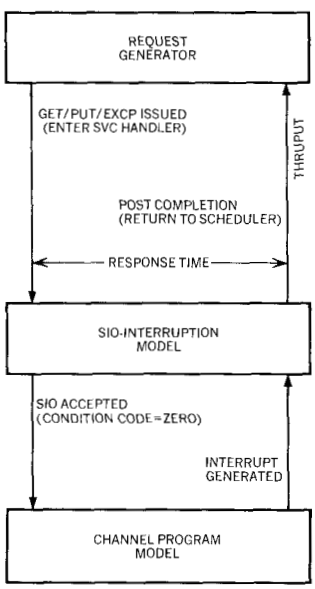
- Detailed modeling of channel, control unit, and device functions integrated with macromodeling of the IOS functions of the CPU
- Use of a flexible, probabilistic request generator to synthesize an EXCP stream representative of the unmodeled part of the system
- Modular organization to handle a multiplicity of models at the channel and control unit level, while minimizing the need to remodel the common parts of the system
- Highly parameterized base model to facilitate representation of a range of synthetic applications and a variety of DASD and related cyclic devices

The simulator's characteristics are now presented in terms of its modular components, as shown in Figure 1.

request generator

The request generator created simulated I/O requests such as GET, PUT, or EXCP supervisor calls which were representative of a large data base system. The rate at which requests were generated was not directly specified. Instead, a *constant system queue* philosophy was employed under which the number of requests coexistent in the modeled system was specified as a constant for each subrun. Once the request generator had injected the specified number of requests into the model, additional requests were injected only to replace those that completed and exited.

Figure 1 Simulator macroflow



The choice of a constant queue drive was somewhat arbitrary, but was consistent with the fact that CPU utilization for execution of application programs was not included in the model. Therefore, the number of I/O requests in the system was logically limited only by the number of active programs or by the capacity of the operating system. The case implied was one where the computer system was fully I/O limited, creating maximum sensitivity to performance of the I/O subsystem. Since system queue size was an independent variable, the corresponding dependent variables became response time and throughput as applied to the I/O request stream clocked at the GET/PUT interface. *Response time* was the elapsed time from the issue of a GET/PUT call to completion of the resulting POST macro, while *throughput* was the mean request completion rate.

The attributes that characterized a simulated I/O request were grouped as primary attributes, which were randomly selected from their respective probability distributions, and secondary attributes, which were specified as dependent functions of the primary attributes, the configuration, and the dynamic state of the request.

The primary attributes included:

- Data descriptors
  - Device address
  - Cylinder address
  - Rotational position of record
  - Record length
- Request descriptors
  - Operation (read/update in place)
  - Access method (direct/indexed)

The update operation was represented by a simulated channel program containing the command chained sequence—read, write and write-check—all to the same record address. The indexed access method modeled a sequence of channel programs consisting of three index references before reading the data record. (Although not ISAM, it incorporated some ISAM facilities.) After random selection of the data location, the simulator mapped backward to the fixed relative locations of the associated indices. The mapping pattern assigned the first two indices to a common cylinder and the third index to the same cylinder as the data.

Secondary attributes included:

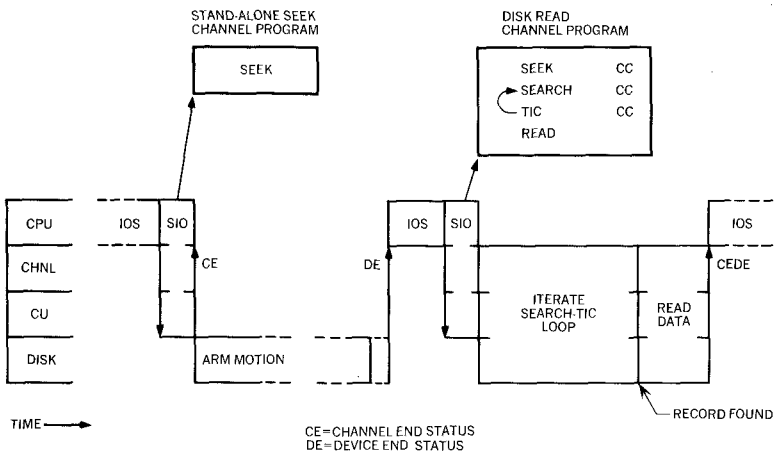
- Index record descriptors (one set per index level)
  - Device address
  - Cylinder address
  - Rotational position of record
  - Record length
- Data path descriptors (one set per device address)
  - Channel address
  - Control unit address

Although the primary data and request descriptors were selected probabilistically, the update operation and indexing injected secondary references that were rigidly interdependent. Also, the dynamic loading of channels and devices could vary significantly in spite of the discipline of a constant system queue.

SIO-interruption model

The SIO-interruption section of the model represented the software and hardware functions that translated a data management macro, such as GET or PUT, into an appropriate channel program, supervised the initiation of the channel program, and reported its completion to the calling program via the POST macro. Software functions included data management and the I/O supervisor, while the hardware functions were limited to the interaction of channel, control unit, and device in responding to the START I/O (SIO) instruction and in propagating interruptions to the CPU. These functions were modeled in only enough detail to

Figure 2 Simple direct read using a selector channel program model and IBM 2314 disk



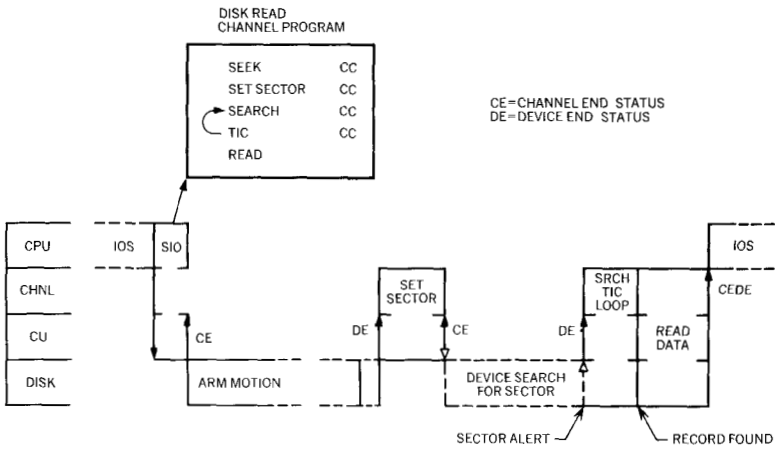
simulate their contribution to response time. For example, data management was modeled only to the extent of holding the simulated CPU for a length of time representing average execution time of a GET/PUT macro, before passing the request to the EXCP interface. IOS involved some logical detail since the management of logical channel queues was modeled in addition to representing CPU utilization for execution of EXCP, SIO, I/O interruption handler, POST and channel-restart routines. However, CPU activity other than data management and IOS was ignored. The most detailed logic of this section modeled the propagation and queuing of interruptions, including generation of the channel available interruption condition by the block multiplexer channels defined later in this paper.

channel program model

The channel program model represented the operation of channels, control units and direct access devices during simulated execution of selected channel programs. This model was unique for each proposal.

A simple example was that of a nonindexed read operation that was initiated by a simulated call specifying Basic Direct Access Method (BDAM). A comparison could be made between the channel programs for the selector and block multiplexer channels. The selector channel attached an IBM 2314 and the block multiplexer channel attached an IBM 3330. In the selector channel model, a nonindexed read invoked a sequence of two separate channel programs illustrated in Figure 2—the "stand-alone seek" program inserted by IOS to free the channel during arm-motion, and the basic read program provided by the access method. The block multiplexer channel model, depicted in Figure 3, mapped the nonindexed read into a single channel program since the stand-alone seek program is not needed. The

Figure 3 Simple direct read using block multiplexer channel program model and IBM 3330 disk



channel disconnected from the device during arm-motion time by storing the address of the next operation to be performed by the device in the subchannel associated with the device. The channel program could then be resumed when the device presented Device-end (DE). The channel program command then primed the device with a rotational position cue and allowed the channel to disconnect again. When the disk rotated to the cued position, the device presented DE and the Transfer-in-channel (TIC) loop of the read program was entered.

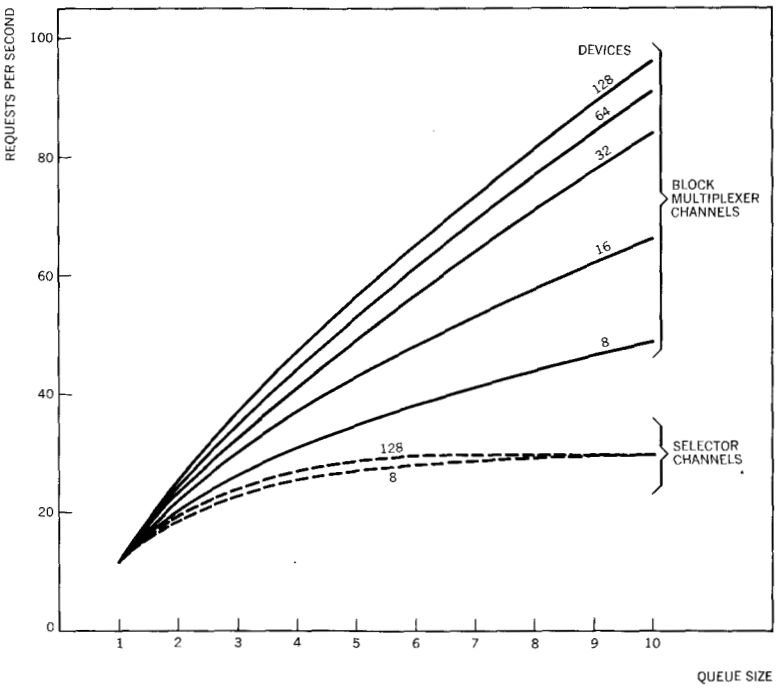
While Figure 2 and Figure 3 show the level of detail at which the channel programs were modeled, contention for the channels and control units that occurred before most operations and resulted in wait times is not explicitly shown.

simulation results

Simulation was used to investigate approximately three-thousand combinations of system configurations, queue sizes and application descriptions including a wide range of CPU's, several device types (with an emphasis on the IBM 2314, 3330, 2301 and 2305 devices), and the various proposed channels and control units. The request mix used for the results obtained consisted of a record length distribution ranging from ten bytes to ten thousand bytes with a mean value of three hundred fifty bytes, uniform device and cylinder distributions except for index records, and a request mix of 75 percent indexed requests and 17 percent updates. Indexed requests and updates were chosen independently, resulting in four request categories:

- Indexed read – 62 percent
- Indexed write – 13 percent
- Direct read – 21 percent
- Direct update – 4 percent

Figure 4 Simulation results using a variable number of IBM 3330-like devices on a single channel



Most requests involved the access of more than one record.

Some results of the block multiplexer channel and the selector channel comparisons are shown in Figures 4, 5, and 6. The term device is used to indicate a single disk pack and arm rather than an entire storage facility. Throughput, in requests per second as depicted in Figures 4 and 5, is given as a function of queue size. All resulting figures apply to a particular work load defined by the above request mix, statistically uniform device loading, and a constant request queue. It is also interesting to note that the model was designed to predict results from queue lengths up to 100, although such lengths are not experienced in current systems.

A device on a selector channel operated independently only for the seek operation and required the channel for searching as well as for reading. For this reason, the channel quickly became a bottleneck as I/O activity (the request queue) increased. Increasing the number of IBM 3330-like devices on a selector channel of the modeled System/360 Model 85 did not proportionately increase the utilizations as shown by the curves of Figure 4. The utilizations of the simulated selector channel with eight devices were 35 percent at a queue of one, 85 percent at a queue of five, and 98 percent at a queue of ten. Increasing the number of devices to 16 produced corresponding utilizations of 35, 93, and 100 percent.

Figure 5 Simulation results using IBM 3330-like devices evenly distributed among a variable number of channels

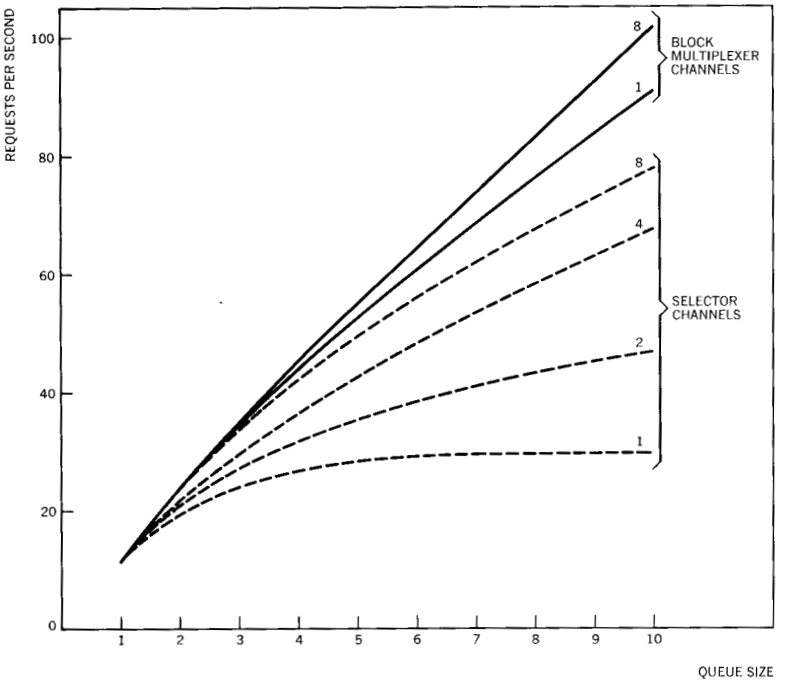
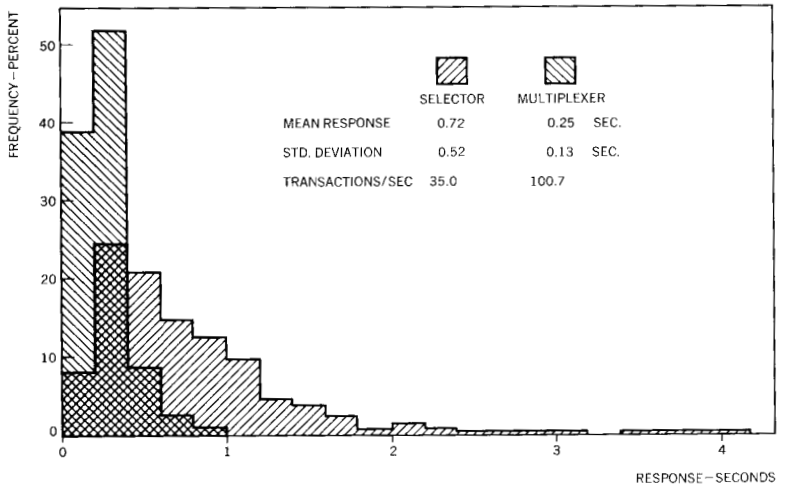


Figure 6 I/O response time distributions for a constant queue of 25



Devices with rotational position sensing on the model of a block multiplexer channel did not require the channel for searching and, thus, performance was limited less by the channel and more by the number of devices, assuming a uniform distribution of activity among the devices. The curves in Figure 4 show the results as the number of devices on a block multiplexer channel

were varied on the modeled System/360 Model 85. Channel utilization was less than 1 percent at a queue of one, 7 percent with eight devices at a queue of 10, and 19 percent with 128 devices at a queue of 10.

The number of channels on the modeled System/360 Model 85 containing 64 IBM 3330-like devices allocated evenly among the channels was also varied and is shown in Figure 5. The performance with eight block multiplexer channels was only slightly better than with four, indicating that simulated performance in this case was almost entirely limited by the devices. Simulated performance continued to improve as selector channels were added, but eight selector channels did not match the performance of one block multiplexer in this application.

Furthermore, the mean response time in a system was lower with a block multiplexer channel than with a comparably driven selector channel because there was less waiting for the channel. Figure 6 shows the response times on a modeled IBM System/370 Model 155 for the two types of channels, both driven by a constant queue of 25 transactions. The block multiplexer channel achieved almost three times the throughput with about one-third the response time of the selector channel. Also, the standard deviation of response time was much lower for the block multiplexer channel. This contradicted a belief that the response time distribution for a block multiplexer had a long tail due to transactions that repeatedly found the channel busy when the desired record became available at each revolution. In fact, this effect was much less significant than the effect of variable wait time to capture a highly used selector channel.

In addition to the block multiplexer and selector channels, an I/O processor and several varieties of buffered control units were modeled. The performance of the I/O processor was comparable to that of the block multiplexer channel except when attached to a slow CPU with high I/O activity. In this case which represented an extreme situation, the I/O processor was better because the CPU became totally utilized in performing IOS functions when connected to a block multiplexer channel. However, as modeling of the I/O processor progressed, it was determined that at that point in time the I/O processor created problems of hardware compatibility and increased cost. It, therefore, was not adopted.

Configurations of more than one buffered control unit per channel, when driven by a large request queue, performed somewhat better than the block multiplexer proposal because data transfers between control units and devices could occur simultaneously. Also, channel time for data transfer was generally reduced. When the request queue was small, however, the additional time required to unload the buffer after the read degraded

performance. After extensive modeling, a buffered control unit was not adopted because its slight performance advantage in a configuration of multiple buffered control units per channel was outweighed by the performance disadvantage with small request queues. Furthermore, it benefited only direct access devices compared to the block multiplexer which could be used advantageously by other I/O devices.

After the block multiplexer proposal had been chosen over the other alternatives, the simulator was still used to evaluate design details such as the necessary resolution of angular address (sector size) and the value of multiple requesting capability for the IBM 2305. One simulation of a 2305 on a block multiplexer channel showed a throughput increase, due to the request queuing facility, of 50 percent at a queue of two and 160 percent at a queue of eight.

Other details that affect performance, but are generally invisible to the system user, were considered. One was an expanding record window that was proposed to operate as follows. After an unsuccessful attempt to capture the channel, the device advances the point at which it requests the channel so that on the next revolution, the channel request begins earlier. The request point is further advanced each revolution until the channel is captured. The effect was to increase the probability of channel capture for requests that had had to wait, thus decreasing the variability of response time. The disadvantage, among others was that channel utilization was increased because early channel capture meant a longer channel hold time. Simulations showed a small decrease in response time variance and some decrease in throughput, as well as some increase in mean response time. Consequently, this feature was not adopted.

As a result of the decision to implement the block multiplexer channel proposal, the new channel type and a new DASD function, rotational position sensing, were defined in complete detail.

Block multiplexer channel

The block multiplexer channel, deriving two major characteristics from the selector and byte multiplexer channels – speed and multiplexing capability, is designed to handle devices with the high data rates normally handled by selector channels. The IBM 2880 Block Multiplexer Channel, for example, can operate at speeds up to 3.0 million bytes per second. Multiplexing implies multiple subchannels, and, like the byte multiplexer channel, the block multiplexer channel has both shared and nonshared subchannels. The multiplexing capability of the block multiplexer

channel, however, is limited to interleaving complete blocks of data because of the expected data rates on the channel. Logical disconnection of a device from the channel is permitted between blocks and occurs only if a significant delay is anticipated before another operation can be executed by the device.[1-3]

Two allied functions were found necessary because of block multiplexing. While the I/O interface definition already accommodated multiplexing of blocks on the interface, no facility existed to efficiently interleave the execution of most I/O instructions with bursts of channel activity on the interface. The *channel available interruption* condition was thus defined to alert the operating system that the channel, which previously rejected an instruction because it was busy, is now available.[2,3]

A new instruction, HALT DEVICE, was also defined largely because of the block multiplexing operation. Its function is similar to HALT I/O except that, when a channel is busy, only the addressed device is affected.[2,3]

Rotational position sensing

The tracks on DASD employing rotational position sensing are divided into sectors. The number of sectors per track is fixed for each device but varies among models. In general, a large number of sectors, not exceeding 256, is implemented. The IBM 3330 uses 128 sectors; the IBM 2305 Model 1 uses 90, and the IBM 2305 Model 2 uses 180.

The one-byte sector number is sent to the DASD by a new command called *set sector*. When it receives the sector number, the control unit logically disconnects from the channel until the desired sector is reached or is about to be reached. Reconnection is then attempted. Since device rotation causes the sector to be available only temporarily, a busy channel can result in a missed sector and a delay of one rotation time before the sector is again ready.[4]

The sector number to be used in the set sector command can be obtained in two ways. When the records on the track are of a fixed length and format, the sector can be derived from the track capacity and the sectors per track. Alternatively, when a record is read, written or searched, its sector can be retrieved by a *read sector* command.

The request queuing facility of the IBM 2305 further uses the block multiplexer channel and the rotational position sensing concept by allowing up to eight operations to be simultaneously in process for a single device. The device is assigned eight de-

vice addresses to which operations can be arbitrarily directed. The control unit for the 2305 effectively sorts these operations so that they are handled in the order in which their respective sectors become ready at the device.[5]

Concluding remarks

The effective use of direct access devices in System/360 was limited by channel and device architecture. Because System/370 was expected to continue with the same dependence on DASD, the problems were defined and solutions were developed. With the assistance of a simulator used during evaluation, the block multiplexer channel proposal was selected. This effort thus provided the basis of the block multiplexer channel and block multiplexing devices that are available today.

ACKNOWLEDGMENT

The authors wish to acknowledge J. E. MacDonald, under whose direction the simulator models were developed, and also those individuals who evaluated the proposals at the Systems Development Division Laboratories in Poughkeepsie, Kingston, Endicott, and San Jose.

CITED REFERENCES

1. *IBM System/360 and System/370 I/O Interface Channel to Control Unit Original Equipment Manufacturers' Information*. Form GA22-6974, International Business Machines Corporation, Data Processing Division, White Plains, New York 10604.
2. *IBM System/360 Principles of Operation*, Form GA22-6821, International Business Machines Corporation, Data Processing Division, White Plains, New York 10604.
3. *IBM System/370 Principles of Operation*, Form GA22-7000, International Business Machines Corporation, Data Processing Division, White Plains, New York 10604.
4. *IBM System/360 Component Summary: 3830 Storage Control and 3330 Disk Storage*, Form GA26-1592, International Business Machines Corporation, Data Processing Division, White Plains, New York 10604.
5. *IBM System/360 Component Description: 2835 Storage Control and 2305 Fixed Head Storage Module*, Form GA26-3599, International Business Machines Corporation, Data Processing Division, White Plains, New York 10604.