Operation of several systems as one multisystem to obtain increased availability, improved cost/performance, or both, is considered.

System requirements for various applications are formulated, and the multisystem capabilities of SYSTEM/360 *are discussed in context.*

The structure of SYSTEM/360

Part V - Multisystem organization

by G. A. Blaauw

A system consisting of two or more central processing units that can communicate without manual intervention is called a *multisystem*. Thus defined, this term encompasses a large variety of system configurations and should be distinguished from such terms as *multicomputer system* or *multiprocessing system*, which usually are given a more restricted definition.[1]

The motivation for multisystems stems principally from two considerations: availability, and the ratio of cost to performance.

The term *availability* is used to describe the degree to which a system can function in the presence of malfunctioning equipment. When extra-high availability is not required, a multisystem must be justified on the basis of improved *cost/performance* only. The cost/performance ratio may be influenced by such considerations as the limitation, specialization, separation, or pooling of equipment.

For the purpose of this discussion, the different cost/performance considerations will be discussed separately of each other and of the availability requirements. An actual multisystem configuration will most often be justified by a combination of these reasons.

Cost/performance

To justify a multisystem solely by cost/performance, the multi-

system must show an economic advantage (for given workload, function, and time constraints) over alternative solutions, such as multiple independent systems or one single system of increased capability. Competition from a single system is severe, since it has been shown many times that the performance of one central processing unit (CPU), built for a cost equal to that of two smaller CPU's, easily exceeds the combined performance of the two. If however, one CPU operating at the limit of available technology has insufficient power, a multisystem suggests itself.

technology limitation

Constructing a multisystem out of several independent CPU's increases the combined cost/performance ratio of the CPU's. This increase is due to the cost of communication expressed in added equipment, reduced speed of operation, and additional program residence and execution time. Therefore, in the absence of a time constraint, the multiple independent systems are more attractive. But if the period in which a major job within the workload must be completed is so short that a single system is inadequate, a multisystem may provide a solution.

A multisystem in a real-time environment is still in competition with a single system having multiple arithmetic and logical units. In such a system, logical circuits dynamically select strings of instructions that can be performed simultaneously. Simultaneity is sought within one program segment, rather than between several program segments, as in the multisystem approach. Because of the greater integration of design and the absence of programmed interlocks, multiple arithmetic units are often more efficient than the comparable multisystem.

processing unit specialization

In contrast to the use of multisystems to overcome technology limitations, which has not been prominent, multisystems based on processing unit specialization are finding increasing acceptance, as in the case of attached peripheral computers.

A single CPU must be able to perform a variety of tasks, but is not equally adept at each. In particular, high CPU performance is of no avail for a task that is input/output (I/O) limited. Furthermore, technological limitations become sooner apparent in character-oriented operations than in word-oriented floating-point operations. In compiling or editing, processing time of a high-performance CPU may be only marginally lower than that of a medium-size CPU more adept at this class of operations, and a functional division of work between the CPU's often results in improved cost/performance.

The logical design of SYSTEM/360 permits, within its strictly compatible definition, a wide degree of specialization that makes it attractive for this type of multisystem. The advantage of compatibility becomes apparent in the reduction of programming and educational costs.[2]

Because the specialized computers join in a common job, data and programs must be communicated between them. This communication may be manual, hence the application does not in itself require a multisystem; e.g., an operator may carry tapes

from the tape drives of one system to those of another. The high overhead of manual intervention due to the short duration of an average processing run can be reduced by *batching*, i.e., by treating several programs jointly. However, any manual intervention is, by nature, unreliable. The ability to switch tapes under program control, which is one method of interconnecting systems into a multisystem, eliminates the time and hazard of manual intervention. Communication media with shorter access time can further reduce *turn-around time*, the interval between the moment a program is presented to a computation center and the moment the results of the run are available. This time may greatly affect the productivity of the center and, therefore, justify a multisystem of specialized CPU's.

equipment separation

When data are collected and results are made available at widely separated locations, cost/performance is often improved by local systems that communicate with each other or with a central system. Local processing can be justified where the savings in reduced data transmission outweigh the cost of the additional system.

component pooling

Finally, cost/performance may be improved by pooling storage and peripheral equipment among CPU's. In this case, the tasks of the individual systems may be basically independent, but the sharing of facilities permits each system to perform a wide variety of tasks with a reduced storage and I/O complement for each. Pooling is even more attractive when not only storage but also stored information can be shared, as exemplified by systems programs on a shared disk file.

Equipment pooling is only attractive if the savings in equipment are not offset by the cost of equipment interconnection. The introduction of switching in what otherwise would be a single connection always involves some cost and performance penalty. The gating, selection, priority-determination, and powering involves not only additional equipment but also some delay of transmission. This delay is crucial for high-speed storage, where connection of CPU and storage often leads to critical timing problems.

The concurrent use of one storage array by several CPU's results in delays, called *interference*, during which one CPU waits because the storage array is performing a cycle for another CPU. Since the CPU is not time-dependent, the loss of time causes no further complications. However, when an I/O device competes with other I/O devices for the use of shared storage, the permissible rate of transmission is affected and an overrun may result. As a result, the pooling of parts of a high-speed storage array is less attractive than the pooling of individual arrays of a multiple-array storage. Either case is advantageous only when additional gains, such as increased availability, can be achieved.

Availability

The character of a multisystem designed for high availability is primarily determined by the time allowed for reconfiguration, the

ability to fail safely or softly, and the multiplicity and modularity of the system. High availability may be required around the clock or, perhaps, just during banking hours; in each case, special measures must be taken to reduce the chance of failure during the critical period.

reconfiguration time

The system components that communicate in the performance of a given task are said to form a *configuration*. When components are eliminated or introduced, a new configuration is formed, the process of changing being called *reconfiguration*. The time required to reconfigure upon the occurrence of a malfunction may be a critical systems parameter. The time to reconfigure includes the time required for fault detection, fault location, switching, manual intervention, program restart, as well as for supervisory program execution.

The time required to reconfigure is critical only in a real-time environment. When much time is allowed, reconfiguration may be entirely manual and, therefore, no multisystem is required. In payroll applications, for example, reconfiguration is often achieved by carrying the job from one installation to another.

When time constraints permit, the use of multiple independent systems may be superior to the multisystem approach because multiple independent systems have greater overall reliability. This is explained by the same factors that apply in the case of cost/performance. The added communication circuits, increased chance of interference, and additional program steps required to prepare for reconfiguration, detract from overall systems reliability, even if only marginally.

In an increasing number of cases, however, reconfiguration time must be limited to a few minutes, seconds, or fractions of seconds, and the ability of a multisystem to communicate without manual intervention becomes essential.

fail safe

A system that can perform its entire workload in the presence of any single malfunction is said to be *fail safe*. Since a malfunction may have catastrophic consequences, each system component should have a potential replacement that is not required by the workload and hence is *redundant*. Also, *isolation* should be provided so that failure of one component does not cause its replacement to fail.

A conceptually simple, and historically early, fail-safe system is the stand-by system. Here the system required for the full workload is duplicated, and thus has 100 percent redundancy. The two systems communicate at several points, and reconfiguration time can be of the order of a few instruction execution times. Processing proceeds in duplicate and in parallel, each system being internally checked. The supervisory task can be held to a minimum and can be implemented by hardware.

When enough time is allowed for reconfiguration, i.e., files can be accessed and several subroutines can be executed, less than 100 percent redundancy can be achieved by a system having fail-softness or a greater degree of multiplicity.

A *fail-soft* system performs only the essentials of its workload in the presence of malfunction. An analysis of the tasks to be performed must determine which part of the workload can be delayed in case the system has to revert to the reduced performance level. **fail soft**

The fail-soft approach provides two advantages. First, only the processing power for the essential, time-dependent workload needs duplication. Second, in the absence of malfunction, the duplicated processing power can be used to perform the nontime-dependent workload. This workload can include diagnostic programs that serve to detect, at an early and convenient time, actual or potential equipment malfunction. As a result, the efficiency and availability are improved.

A typical fail-soft configuration consists of two equal or unequal CPU's with a complement of storage and I/O equipment. The smaller CPU should be able to perform the time-dependent tasks, whereas the processing power of both CPU's satisfies the overall workload, including any work deferred because of malfunction.

The strict compatibility within the wide range of models of SYSTEM/360 is particularly useful in matching a multisystem to fail-soft requirements. Processing units of equal or unequal power may be chosen as needed.

The term *multiplicity* is used to indicate the number of CPU's within the multisystem. **multiplicity**

If one computer of power X is required for a full workload, then two computers with total power $2X$ and 100 percent redundancy permit a fail-safe system. But if we replace the single computer by n computers with power X/n, then $(n + 1)$ computers with total power $X(n + 1)/n$ and only $100/n$ percent redundancy are required for fail-safe operation. Thus, increasing the multiplicity reduces the redundancy.

This over-simplification ignores several factors. First, the cost of $n + 1$ computers of power X/n may turn out to be of the order of $(n + 1)/\sqrt{n}$ times the cost of one computer if we assume that performance increases as the square of equipment increase. In that case, a two-computer system ($n = 1$) is the least costly. Second, the increased amount of equipment, as reflected in the cost, reduces the reliability and availability of the system. Third, it may be impossible to divide an actual workload efficiently among a multiplicity of computers. Fourth, actual equipment, which always appears in discrete units, may not provide the capacity ideally desired.

In practice, the number of CPU's within a multisystem may be expected to be small; two and three being the most frequent. As a typical example, a three-computer system could provide, first, fail-safe operation and, upon the occurrence of malfunction, fail-soft operation.

The term *modularity* is used here to indicate the number of system components that can function independently. System components are considered logically independent if they are **modularity**

interconnected by well-defined interfaces so that they can, upon reconfiguration, operate without communicating with each other. Examples of logically independent system components are storage units, CPU's, I/O control units, and I/O devices.

Although logically independent, these system components may still be physically dependent because they share common equipment. Thus, storage and CPU may be built within a common frame and share power supplies; an I/O channel may share the logical circuits of the CPU; or several I/O channels may use common logical circuits as in the case of the multiplexor channel. These are examples of *integrated* design. The decision to use an integrated design is dictated by cost and performance considerations, and not by logical dependency. Thus, the SYSTEM/360 models differ in the degree of integration across the performance range, yet they have identical logical structure.

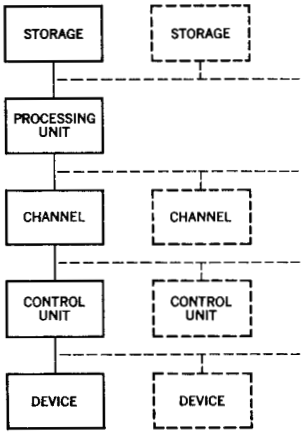
Storage units may be considered logically independent when they are served by separate access mechanisms. A storage unit may therefore be divided into as many physically independent units as is economically justifiable. A major consideration in this case is system performance. Increased modularity (i.e., many small storage units) reduces interference for a single CPU as well as for several CPU's within a multisystem. For this reason, the large SYSTEM/360 models use multiple storage units. This also points to the desirability of multiple storage units once main storage must be shared.

Since failure of a circuit element shared by several logically independent system components causes malfunctioning of all these components, physical as well as logical independence is required if modularity is to help in obtaining improved systems availability. Pooling of equipment, on the other hand, requires only logical independence.

For SYSTEM/360, an integrated design was used only in those instances where major savings in equipment could be obtained for the particular performance goal. Such a reduction in equipment increases the reliability and, in turn, the availability of the overall system. This increase in system availability through component reduction more than offsets the increase in availability which physical independence would provide.

Even though the system components that are integrated may fail at once, the integrated design is functionally acceptable for several reasons. First, normally the programming system which provides for the eventuality of an individual component failure can handle the case of several failures at once. Second, the failure of one component typically reduces the demand upon other components. Third, the components eliminated from the operating part of the system are often needed for diagnosing the failing part of the system. Consider, for example, the multisystem components storage and CPU. When one CPU fails, the multisystem system must still anticipate the potential failure of a storage unit, while the need for this second storage unit is reduced because less processing

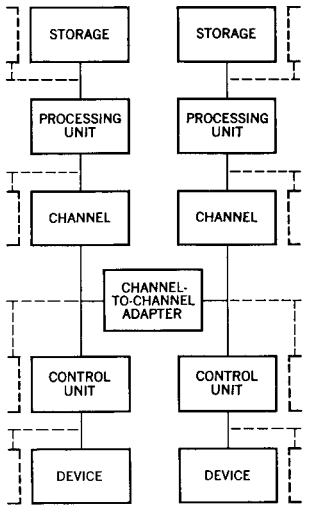
Figure 1 Functional structure of basic system



can be performed. Moreover, the storage unit associated with the CPU is likely to be needed for performing diagnostics upon the CPU. These functional reasons combined with reduced cost and increased availability make the integrated design attractive when performance permits.

In summary, the modularity of a multisystem is determined primarily by cost/performance rather than availability considerations.

Figure 2 Channel-to-channel adapter



Means of communication

Communication between the CPU's of a multisystem may be achieved by transmitting information from one CPU to another through a connecting link or by giving them access to a shared storage medium.

Before considering these two types of communication, it is well to review the major components of a single, or *simplex*, system. Figure 1 shows these components, and their functional dependencies, in simplified fashion. The interconnection between channel and control unit is independent of the particular control units connected or the particular implementation of the channel logic. This interconnection in SYSTEM/360 is called the I/O interface (see Part IV). In contrast, the interconnection between control unit and device is specialized for the device at hand and thus differs for tape drives, communication equipment, files, etc. The interconnection between CPU and storage, the storage bus, also serves for transmission of data between storage and channel.

To obtain communication between CPU's, the storage media and interface connections already available in a simplex system are used. The amount of extra equipment or equipment change that must be introduced to achieve multisystem operation is thus minimized.

transmission

Transmission of information is made possible by the channel-to-channel adapter and by the transmission control units.

A channel-to-channel adapter permits connection of I/O interfaces of two channels, as shown in Figure 2. The adapter's main purpose is to make each channel appear as a control unit to the other channel. This is necessary, because the I/O interface is not symmetric, but assumes the presence of a channel on one end, and one or more control units on the other.

The channel-to-channel adapter normally connects channels associated with different CPU's, thus establishing a multisystem. Transmission proceeds by byte at a rate established by the two channels. Because of the standardization of the I/O interface, the channel-to-channel adapter may connect any model of the SYSTEM/360 to any other model, and may use any type of channel on a given model. Any number of channel-to-channel adapters may be used in a multisystem, but in most cases, one or two suffice. Their main application is in a multisystem emphasizing medium reconfiguration time or equipment specialization.

A transmission control unit permits CPU-to-CPU communication by private line or common carrier. As indicated in Figure 3, communication is established by means of the specialized device interface rather than via the channel interface. Rate of transmission is determined primarily by the line capacity. Any two models of SYSTEM/360, as well as those of any other system, can be connected. The major application is for geographically separated components. The maximum number of connections differs among the SYSTEM/360 models. Multisystems with hundreds of line connections are possible.

shared storage media

When two or more CPU's have access to a common storage medium, information placed in the medium by one CPU can be read by another CPU. In contrast to transmission, sending and receiving are not simultaneous, and a one-to-one relation between recording and retrieving is not necessary. The storage media must be read and written without human intervention, thus barring, for the present, punched cards and the printed page. The choice of shared media is determined by access time, transmission rate, capacity, and cost per bit.

Communication differing in application and implementation is achieved by the sharing of disk files, drums, data cells, and tape drives.

Shared devices are useful for restart information, which permits recovery upon reconfiguration. Disk or drum may be pooled for storage of programming systems. Disk, drum, and tape drives are also useful as a means of communication between specialized CPU's to achieve improved turn-around time.

Because one control unit normally suffices for several files, a switch between channel and control unit allows efficient sharing of a control unit between two CPU's. This possibility is depicted in Figure 4.

As suggested by Figure 5, tape drives are shared between control units, rather than control units being shared between channels. This choice is made because pooling of tape drives between control units permits the simultaneous operation of any combination of tape drives. This logical ability increases the power of both a simplex system and a multisystem. As an example, the sharing of any pair of tape drives by two control units can improve sorting time significantly.

Either high-speed or large-capacity storage can be shared, as is shown in Figure 6 for two CPU's. Configurations can have all storage shared, or can combine shared with private storage.

Where one program is executed in turn by different CPU's, it is desirable that the locations of instructions and associated data be identified for every CPU by the same address. This addressing convention was adopted uniformly in SYSTEM/360 for multisystem operation.

The main application of shared storage is in multisystems requiring very short reconfiguration time. Typically, two or three connections to each storage unit are expected.

Figure 3 Transmission control units

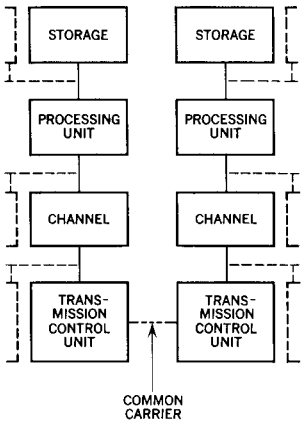
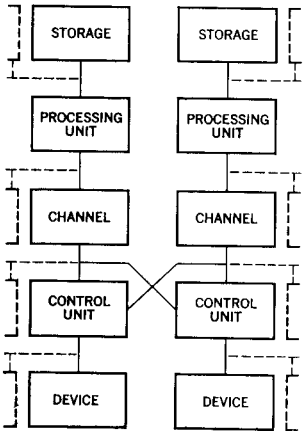


Figure 4 Shared control units



Methods of interconnection

The method by which components of a multisystem are interconnected should be general, to permit freedom in choice of systems components; expandable, to permit economic systems growth; and reliable, to enhance systems availability.

SYSTEM/360 channel-to-channel adapters adhere to one I/O interface definition, and the transmission control units adhere to the industry standards for communication equipment. Since any pair of SYSTEM/360 CPU's can be connected, systems growth can be accomplished by adding more CPU's to the system or, preferably, by replacing a given CPU with one of a greater power. Improved availability can be attained by using several transmission control units and the alternate paths normally present in a communication network.

Interconnections for the sharing of system components may be established between (1) main storage and CPU, (2) channel and control unit, as for disk and drum, and (3) control unit and device, as for tapes. Because the logical approach is the same for all three cases, the interconnection of main storage and CPU serves to illustrate the discussion.

Figure 7 shows an interconnection technique, known as the *crossbar switch*, that allows connection of each of M CPU's to any one of N storage units. The M connections can be made simultaneously, provided that M is smaller than or equal to N. This well-known switch can be implemented in two ways. In the *centralized* implementation, each CPU and each storage unit has a single interconnection with a central switching network. Each connection being simple, all complexity is concentrated in the switching network, which must be viewed as an additional major system component. Where availability is required, the switch should be duplicated. Yet, the additional equipment adds no additional processing power in either fail-safe or fail-soft mode.

An alternative is the *distributed* implementation shown in Figure 8. This figure can be derived from Figure 7 when the components represented by the vertical lines of Figure 7 are made part of the storage units to which the vertical lines belong; CPU's still have single connections, but the storage units have multiple connections, called *multiple tails*. The equipment of the crossbar switch is thus divided among storage units.

Economic considerations favor a distributed crossbar switch because separate frame and powering are not required. Since each added storage unit comes with its own section of the switch, smooth system growth is possible. Additional CPU's are accommodated by additional tails on storage units. Flexible system design is possible because the number of tails need not be the same for all storage units. Moreover, storage units private to one CPU can be incorporated, and high-speed storage can be treated differently from large-capacity storage.

High availability can be attained without duplication since

Figure 5 Shared devices

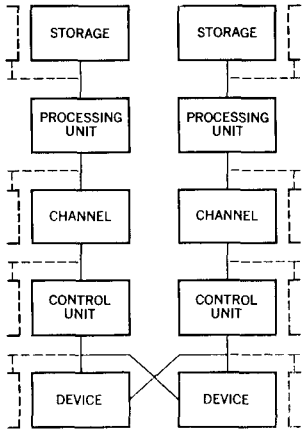
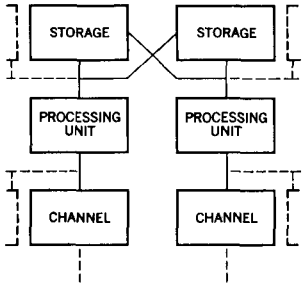


Figure 6 Shared storage



failure of a switch element is counted as part of storage unit failure. The fractional reduction in storage unit reliability is easily offset by the removal of a major systems component (the stand-alone switch).

Figure 7 Centralized crossbar switch

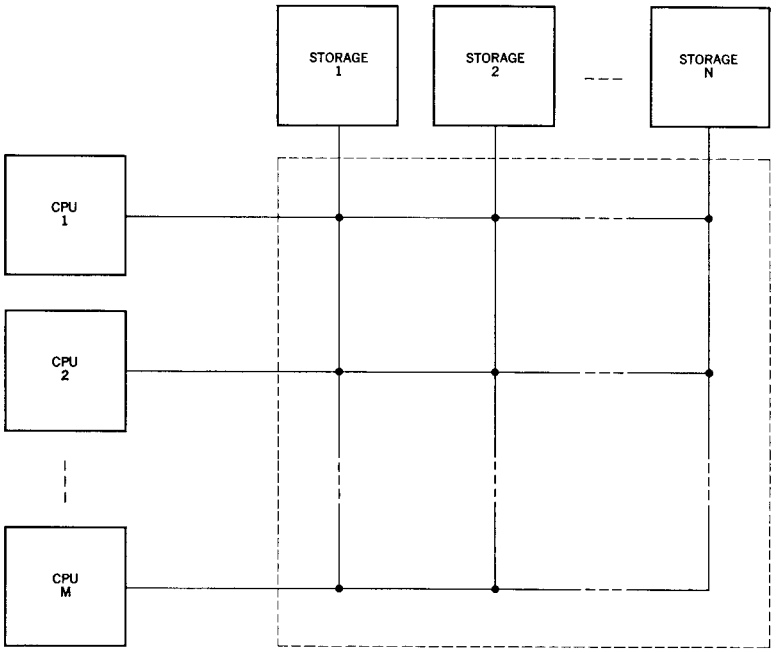
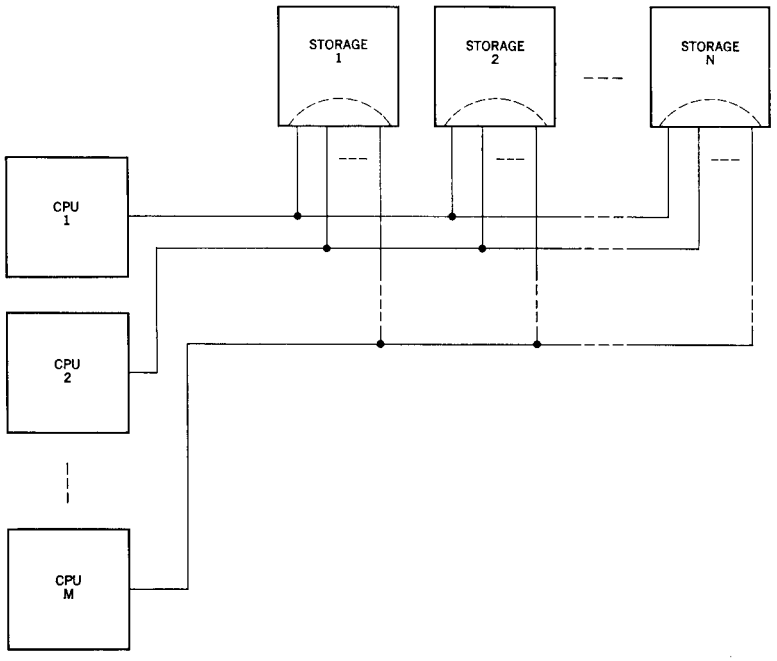


Figure 8 Distributed crossbar switch



For high availability, the bus[3] connecting a CPU with multiple-tail storage units requires proper isolation. Failure of a storage unit should not impede the operation of the buses to which its tails are connected. Neither should the failure of a bus, driven and controlled by a CPU, prevent storage units connected to it from communicating with other buses. Such a design can eliminate the need for physically disconnecting a component when failure occurs.

A distributed switch proves equally desirable for the connection of control units to channels; control units can be connected, through multiple tails, to different channels.

Because the I/O interface is standard for all channels and control units, this interconnection is independent of the models involved. For main storage, the bus must be specialized for the particular CPU and storage type to attain maximum performance. Therefore, normally only CPU's of the same model share high-speed storage.

A centralized switch connects tape control units to tape drives; here, availability requirements can usually tolerate equipment failure upon power malfunction. Tape-switching often occurs in a simplex system. When tape drives are shared between several CPU's, long reconfiguration time is necessary to permit rewinding and manual tape handling. For these reasons, the tape switch normally need not be duplicated. Furthermore, since the number of shared tape-drive configurations is limited, it is possible to design a centralized switch with enough features to satisfy anticipated requirements.

In a multisystem, the allocation of shared storage to a CPU is entirely the responsibility of the programming system. In some applications, a multisystem is separated into two or more independently working parts, this process being called *partitioning*. Partitioning requires no additional functions as long as the supervisory program can be trusted. If it cannot be assumed to operate correctly, manual controls provide a more dependable method of partitioning.

Communication aids

Communication between CPU's should be supplemented in various applications by

- Multiprogramming of supervisory programs and problem programs.
- Signaling that a message has been or is to be transmitted.
- Interlocking the use of storage to prevent conflict.
- Requesting intervention in the case of malfunction.
- Permitting initialization to attempt recovery from malfunction.

multiprogramming

In *multiprogramming*, the attention of the CPU is switched among two or more programs that are simultaneously in storage. Multiprogramming is assumed to be the normal mode of operation

for both simplex and multisystem SYSTEM/360 configurations. This assumption is based on the fact that a large majority of existing systems already use some kind of supervisory program, part of which resides in storage during problem program execution.

In a multisystem, the main occasions for multiprogramming are twofold. First, the supervisory program that directs communication among CPU's makes use of multiprogramming akin to that used by the supervisor of a simplex system. The program requires the storage protection, privileged operations, and efficient program switching included within SYSTEM/360. Second, in systems sharing main storage, one subroutine may be executed simultaneously by several CPU's. This type of program operation is essentially the same as for multiple programs in a simplex CPU sharing the same subroutine, and again, the multisystem can use the simplex features. An essential requirement for such a subroutine is that its instructions remain unaltered during execution. The instruction set and formats of SYSTEM/360 were specified with this in mind. Address modification is amply provided by base and index registers. The modification of operation codes, length, and register specification can be performed with the EXECUTE instruction. For correct execution of a subroutine by multiple CPU's, or by multiple programs within one CPU, it is further necessary that the data areas belonging to the different programs do not overlap and are properly identified by base addresses. A subroutine satisfying these requirements is said to be *re-entrant.*

signaling

Before one CPU sends information to another, it should alert the receiving CPU. Once alerted, the receiving CPU can specify the proper storage area and transfer incoming bytes under control of a read command, while the transmitting CPU transfers bytes under control of a write command. SYSTEM/360 alerts a program to an incoming message in a general fashion by an attention signal which interrupts the CPU. In the case of the channel-to-channel adapter, the initiating write command generates the attention signal in the receiving channel. Once the receiving CPU responds with a read command, communication is established. When transmission is completed, both receiving and transmitting CPU's can be interrupted.

For communication through a common storage facility, a CPU must be alerted to the fact that a message has been prepared for it by another CPU. This alert can be programmed, of course, by periodic inspection of a signal bit. A more rapid response, however, is attained through interruption. The direct-control instructions and external-interruption lines of SYSTEM/360 are intended for this purpose.

Associated with the direct-control instructions is an interface at which eight signals can be made available. A signal from one CPU may be connected to one of the external-interruption lines of another CPU. By the instructions READ DIRECT and WRITE DIRECT, the program in one CPU can cause an external interruption in another CPU. The direct-control feature further provides

static signals that can be sent to, or read from, other CPU's or special-purpose external devices.

interlocking

Shared storage is often a common medium for data, restart information, programs, and results that are updated by different CPU's of a multisystem. To prevent logical conflict, sufficient interlocks must be provided.

A control unit or I/O device responds with a busy indication when a new data transfer is attempted before a preceding transfer has been completed. Because only one message at a time can be transmitted between any CPU and a shared I/O device, a satisfactory interlock exists. Where time is longer, as for the arm movement of a disk mechanism, additional controls are supplied by which a program can reserve and release an access mechanism.

When main storage is the means of communication, interlocks are provided for the period of one storage cycle. When two CPU's simultaneously request access to storage, a tie-breaking priority circuit grants access to one CPU, then gives the next cycle to the other CPU. This simple rule prevents one CPU from locking out, and therefore effectively halting, another CPU.

Access to storage is granted for the duration of one storage cycle, not for the duration of an entire instruction. Since, as a rule, information is processed in internal registers, a typical procedure is to fetch data from storage, process the data in registers, and store the result in storage. The storage interlocking mechanism, unaware of the relation between successive storage accesses, makes it possible for one CPU to store results in a location after a second CPU has fetched its operands from that location, but prior to storage of the results by the second CPU. This possibility necessitates a programmed interlock in the use of core storage as a shared medium.

malfunction alerting

In a multisystem designed for high availability, it is not sufficient to back one CPU by other CPU's, but it is necessary to determine when this backup should take place. The properly operating part of the multisystem should be alerted to the fact that malfunctioning has occurred in a CPU. This malfunction may be caused by a program or the hardware.

Program malfunctioning, as such, does not require multiprocessing, but only multiprogramming; it suffices to identify and discontinue the faulty program under control of the supervisor, and subsequently to proceed with the next available task.

To help detect the malfunctioning of the program, two basic tools are provided. First, the execution of a faulty program often leads to execution of an invalid instruction, use of invalid data, and reference to an invalid address or protected location. Since all programming exceptions are verified (*policed*) in SYSTEM/360, this type of malfunction is soon identified and signaled by a program interruption. As a second tool, the timer may be used to detect programming errors, such as unending loops, which are not detected by policing. The timer can be set to cause an interruption when the time allowed to a program segment is exceeded.

Perhaps the most important use of the extensive checking included in all SYSTEM/360 equipment is fault location. A high degree of checking makes it possible to recognize the occurrence of a malfunction on short notice and, thus, to preserve the state of the CPU for a subsequent diagnosis. The information provided to the engineer servicing the equipment reduces the mean repair time and contributes to availability.

When the malfunction is intermittent, the machine-check interruption and a programmed restart make it possible for the CPU to recover. When the malfunction is solid, the CPU cannot recover and a second CPU should take over. A malfunction signal, of the same nature as a direct-control signal, can give an external interruption in the second CPU. The signal, which requires no programming, is issued as soon as a machine malfunction is detected.

recovering

Each CPU of SYSTEM/360 uses permanently assigned storage (locations 0 through 127) for program status words, channel address and status words, the timer, and diagnostic scan areas. Were these locations common, they would be shared by several CPU's. Therefore, to provide each CPU with separate preferred storage, a quantity called the *prefix* is used to relocate dynamically all addresses referring to the first 4,096 storage locations. Since each CPU can have a different prefix, the sharing of these locations can be avoided.

The prefix relocates all locations that can be directly addressed (using a zero base and zero index specification) by the displacement. Such absolute addressing is useful when the supervisor must store the general purpose registers in program switching. The prefix makes this programming technique possible even if locations 0 through 4095 are not available to a system.

When only storage is malfunctioning, the system can resume operation immediately by eliminating the faulty storage unit. If the faulty storage contains the permanently assigned storage locations for the CPU, new locations can be provided by introducing an alternate prefix. For this reason, a second prefix quantity is provided for a CPU as part of the SYSTEM/360 multisystem feature. Normally, the two prefix quantities relocate the preferred-storage locations to different storage units. Thus, the CPU becomes independent of a specific storage unit for its operation.

The identity of the CPU executing a program may be determined by setting apart one of the addresses in the range of 0 through 4095 as the address of an identifying location. Since the actual location is determined by the prefix, the content of this location may serve to identify CPU and prefix currently used.

When the CPU can resume operation, it is desirable to minimize operator action. Introduction of a new program status word and the corresponding instructions may best be performed by the still-operating part of the multisystem. For this reason, means are provided for one CPU to initiate the initial program loading of another CPU. This signaling again has been defined to be consistent with the signals of the direct-control circuitry. Two signal

inputs are provided, each of which causes initial program loading when a signal is received. The choice between the two signal inputs determines which prefix is used and, hence, the location of the permanently assigned storage addresses. In this case, initial program loading consists of loading an initial program status word from location 0 and performing the necessary system reset. Prior to the initial program loading, the necessary program status words should have been established.

Summary

SYSTEM/360 is designed for multiprogramming as a basic mode of operation, and for multisystem operation as an increasingly important mode of operation.

The design of SYSTEM/360 for use with an operating system led to the inclusion of many multiprogramming features, such as storage protection, privileged operations, program switching, read-only instructions, instruction policing, and the timer. The extension from a multiprogrammed simplex system to a multisystem in which multiple CPU's interact with each other requires:

- The basic ability to communicate among CPU's by means of transmission or shared storage media.
- The proper signaling for message alert, storage interlocking, malfunction alerting, and recovery.

Multisystem operation is already established as a means of dividing tasks between an I/O-oriented peripheral CPU and a high-performance main CPU, resulting in an improved overall cost/performance ratio. The need for multisystems of highly improved availability is clearly foreseen as new computer applications continue to be formulated and developed.

CITED REFERENCE AND FOOTNOTES

1. The name multisystem is also associated with a feature available in some SYSTEM/360 models. This feature, which makes several types of multisystem operation possible, should not be considered essential for all multisystem operations.
2. G. M. Amdahl, G. A. Blaauw, and F. P. Brooks, Jr., "Architecture of the IBM System/360," *IBM Journal of Research and Development* 8, No. 2, 87–101, April 1964.
3. The detailed structure of a bus, which may be either a multiplexed or multiple simplex design, is determined by cost, transmission rate, and interference considerations and differs for the various models of SYSTEM/360.