IBM

Z20-1788-0

SALES and SYSTEMS GUIDE

IBM Time—Sharing System/360 Concepts and Facilities

This document is being furnished for System/360 instructional use only. It contains preliminary technical information, and the contents are subject to change. The contents represent "best available information" as of the publication date. This material is to be considered automatically replaced by the normal SRL publication upon its release.

This publication describes the basic concepts of Time-Sharing System/360 and guides the programmer in the use of its various facilities.

Time-Sharing System/360 is an advanced set of language translators and service programs, operating under the supervisory control and coordination of a complex control program. TSS/360 permits the concurrent sharing of a single or multiple processor system by many tasks and by programmer-users, at terminals conversing with the system. The virtual memory concept of TSS/360 permits each task, or programmers, to assume a memory of up to 4 billion bytes of memory. It is designed for use with IBM System/360, Model 67.

Address comments concerning the contents of this publication to
IBM, Technical Publications Department, 112 East Post Road, White Plains, N.Y. 10601

CONTENTS

INTRODUCTION

The Time-Sharing System/360 (TSS/360) provides to many simultaneous users a wide variety of program services to assist them in obtaining a solution to their individual computer problems. TSS/360 users call for these services through commands which they introduce to the system through a remote terminal consisting of a keyboard, a printer and, optionally, a card reader, a paper tape reader, or a display device. They can enter and construct programs and data sets, debug programs in a conversation mode with the system, and request execution of programs using specified data sets. In addition, users can specify that their programs and data sets are to be added to, deleted from, modified, copied, or moved to and from input/output units according to their data processing needs. While the user(s) is performing these functions, he is in constant communication with the system, and is aware of the functions being performed by it. As requests for various services are made, the system informs the user(s) of action it has taken, of additional information required to complete services requested, and of any mistakes made by the user in requesting services.

To provide efficient service to all users, the System/360 time-sharing system allows them easy and open access to the central data processing facility. Multiple remote consoles provide to many simultaneous users a direct means of monitoring and controlling the computers that are servicing their needs. These users, at remote terminals, can, if they desire, limit themselves to the services to which they are accustomed in a multiprogrammed, batch-processing environment. Experience has indicated, however, that users take advantage of the broader range of services that are part of a time-sharing system. These services provide an easy man-machine communication that facilitates such problems as program writing and checkout, and complex design problems where rapid computer arithmetic and human judgment are required.

The Time-Sharing System/360 includes:

- A time-sharing supervisor with a time-slicing capability
- A command language for communication between user and system in both conversational and batch modes, including a program checkout subsystem, for source level debugging
- Data management and cataloging facilities
- A mnemonic assembly language compiler with macro capability — batch and conversational modes
- A FORTRAN IV compiler — batch and conversational modes
- A Programming Language I (PL/I) incremental compiler
- A COBOL compiler — batch mode
- A sort/merge program, oriented to direct access storage
- An open-ended library of mathematical and utility programs

The Computing System/360, Model 67, was designed specifically for time sharing. The special time-sharing features include:

- Multi-tailed storage units
- Multiple central processing units (CPU's)
- Dynamic storage relocation
- High-speed multiplexor channels
- Dual input/output paths
- Store and fetch protection
- System partitioning capabilities

In addition, the Time-Sharing System/360 takes advantage of the standard System/360 multiprogramming features, such as the privileged instructions, the interval timer, and the supervisory mode.

DEFINITIONS

In relation to the TSS/360, the term "users" represents the totality of all potential terminal customers. That is, the system has a list of all legitimate users. This is a predefined list, usually compiled jointly by the manager of the computer center and all departments serviced by the center. Active users, then are the subset users who are currently working on terminals or whose jobs are being run in the nonconversational mode (batch or background). A session is defined as the total work done by an active user at a terminal from the moment he requests service from the system (LOGON) to the time he notifies the system that he is relinquishing his terminal (LOGOFF). A task is a basic unit of work, such as a FORTRAN compilation or a program module execution. The task is associated with the discrete virtual memory that it is a part of and is the basic multiprogramming unit under the TSS/360 supervisor. A job can be composed of several tasks, but becomes meaningful only in the background mode. That is, active users at terminals create tasks rather than jobs.

Operational Characteristics

The normal mode of operation of the TSS/360 is to have users at remote terminals working in a conversational mode with the system. In the conversational mode, the user can direct the system as he proceeds and may change his tactics in solving the problem or modify his approach as he examines the results of his action and finds that modifications must be made. By having the user at a terminal conversing with the system in this manner, the system can be used as a powerful tool to aid in the solution of the user's problem. Figure 1 is a graphic representation of the general conversational operation of TSS/360 and can be used as a guide throughout the manual.

Conversational Mode of Operation

Any user at any terminal has full access to the system and its services without regard to any other user who may simultaneously be using the system. When a user initially logs on at a terminal, the system establishes a task monitor to service him for the duration of his session at the terminal. The task monitor expects inputs from, and direct messages and instructions back to, the user. The input expected from the user at the terminal consists of commands that specify the programs to be executed. The source of these input commands is defined to be a system input (SYSIN). Similarly, messages sent from the task to the user who is directing or commanding it are sent via a system output (SYSOUT). In a conversational mode of operation, the SYSIN device is the keyboard or, possibly, a card reader or a paper tape reader that is associated with the terminal that the user is using. The SYSOUT device is the printer that is associated with the terminal that the user is using.

Nonconversational Mode of Operation

In many situations, the programs that are to be executed can be defined ahead of time, so that no variation occurs as execution of the program proceeds. In such a case, a user can establish a sequence of commands to perform the functions he desires and can store these commands in the system as a data set. The user can request execution of these commands in nonconversational mode by establishing a nonconversational task (background job). The SYSIN associated with this nonconversational task is not a terminal; instead it is a data set that is stored on a direct access device or tape or some other medium internal to the system. When a nonconversational task has been established, commands are taken, one at a time, from this internal data set to direct the program execution required.

Similarly, SYSOUT for a nonconversational task is not a terminal but may be a data set that is printed eventually on a systems printer by the system. This printout is returned to the originating user; he can then examine it at his convenience to determine how the execution of his nonconversational task proceeded.

THE TIME-SHARING CONCEPT

Time sharing is a technique that is used in remote computing systems to provide more equitable response to a large number of simultaneous users at different terminals. Each user is given the illusion that he alone has all of the computer resources at his disposal. The system techniques that provide this illusion are not evident to the user.

Time slicing is a method used in time sharing to provide each active task a period of time when it alone is using a central processing unit. A time slice to be optimum should be long enough to do meaningful work for the task and short enough so that other waiting tasks are not delayed unreasonably. The supervisor allocates time slices to the queue of tasks in accordance with preset priorities set by the installation.

VIRTUAL MEMORY CONCEPT AND EFFECTS

The virtual memory concept is a solution to two distinct but related problems:

The 24-bit addressing capability of standard System/360 allows a maximum of 16 million addressable bytes. In practice, however, the programmer is restricted to using addresses that represent the actual physical storage on his machine. Thus, the programmer does not have the ability to write a program addressing 16 million contiguous bytes. If the total program size exceeds the available core storage, the programmer must overlay his program. This places an undue clerical burden on large programs and is a frequent source of error.

In a multitasking environment, the supervisor has the job of paging (swapping) active tasks into whatever core storage is actually available. During execution of tasks, this physical fragmenting of programs must not be apparent to the user, so that he may think of his program as being in contiguous memory locations.

Both problems are solved through the virtual memory concept, a combined programming system-equipment approach, which is described below.

In a multitasking environment, the supervisor has the job of space sharing the programs of many users. The more users that are in core storage,

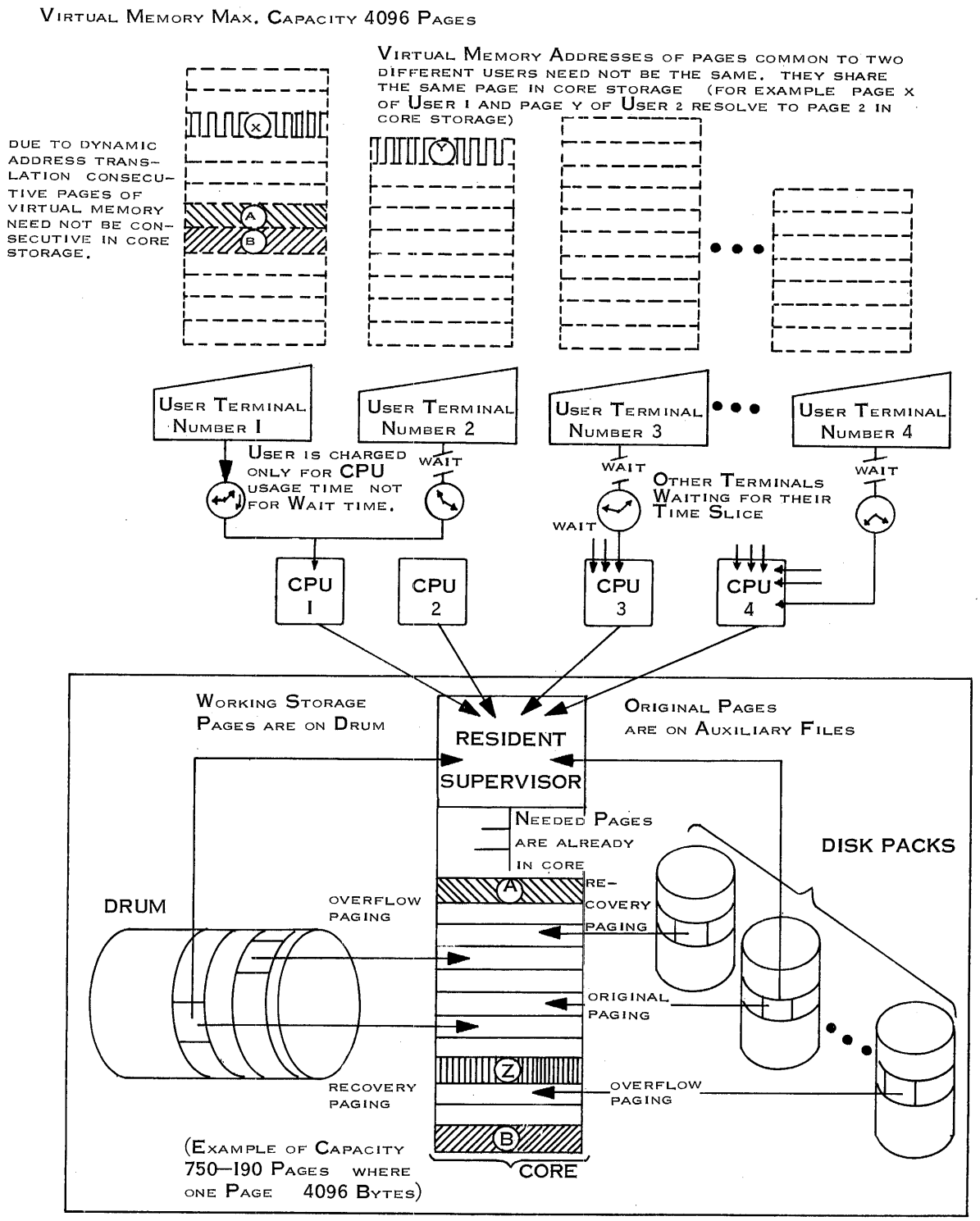


Figure 1. Operation of time-sharing system/360

the faster the control can be transferred between users, and the lower the resulting overhead cost. In the time-sharing system, space sharing and time sharing are facilitated by breaking up programs into pages of 4096 eight-bit bytes. Core storage is allocated in page increments.

Programs on the Computing System/360, Model 67, operate in one of two states, that is, unrelocated or relocated. Programs that run unrelocated run essentially as do programs on other models of System/360. Programs that run relocated are divided automatically into pages of 4096 bytes (see Figure 2), and a map is created consisting of one entry per page. All pages present in the map are logically existent. The total number of pages in the map represents the size of the program's virtual memory, up to a maximum of 1,048,776 pages. The entries in the map also indicate whether the corresponding page is or is not currently resident in core storage. Those pages that are currently resident are physically existent. Only those pages of a task that are actively used are brought into core storage. Core storage never contains the pages of a task that are not referenced. At execution time, core storage holds only a few parts of a user's task, but what is more important, it holds only the active parts.

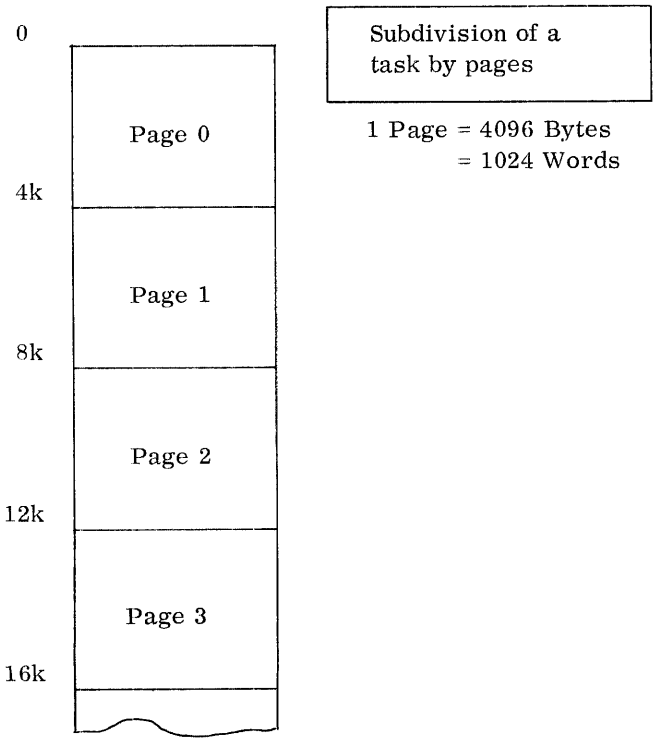


Figure 2.

The heart of this technique is an equipment development called dynamic address translation. This technique could be called address mapping, for it involves the transformation of virtual memory addresses into core storage addresses by means of an equipment table lookup (see Figure 3). Each task in the system (the work introduced at a single terminal) requires for its operation a set of dynamic storage relocation tables (called page tables), with entries for each page of virtual memory used. These tables are developed by the TSS/360 supervisor as the task is created and as new page requirements are made known by the task. As required pages are fetched into core storage, the supervisor enters into the page table the location of each page. During execution, the equipment automatically does a table lookup on each address as it is referred to by the user, and the corresponding core storage address is chosen. If the user refers to a location that is not in core storage, an automatic interrupt occurs that causes control to be transferred to the supervisor, which sets up a page-turning routine to fetch the missing page. All of this is not apparent to the user. To him, core storage is limited only by the addressing capabilities of the equipment.

Other advantages are derived from the dynamic address translation. Since multiple users can be served concurrently, it is inefficient to require each user to provide a copy of subroutine or language processors. For example, if eight users require a square root subroutine, just one "read only" copy of the routine need be in core storage. Each user's tables need contain only a reference to the subroutine. During the execution of a particular user's task, his tables will send him to the square root subroutine and then (again through his private tables) back to his own pages in core storage.

To exploit this feature, it is required that the location of parameters be raised in general registers, that the routine do no internal storing, and that results be stored in locations passed in general registers. (A full description of the conventions for entering "read only" (reenterable) routines may be found under "Coding and Linkage Conventions".)

Still another advantage to the virtual memory concept is the protection provided both to the system and to the various tasks. The TSS/360 supervisor is automatically protected against accidental or mischievous action within any task by virtue of the fact that each task operates only within its own virtual memory (as defined by its relocation tables); the core storage area occupied by the supervisor is not addressable through the relocation table of any task. It therefore is not a part of the same virtual memory as the task and is completely protected.

Tasks are protected against interference by each other in essentially the same way. Each task has

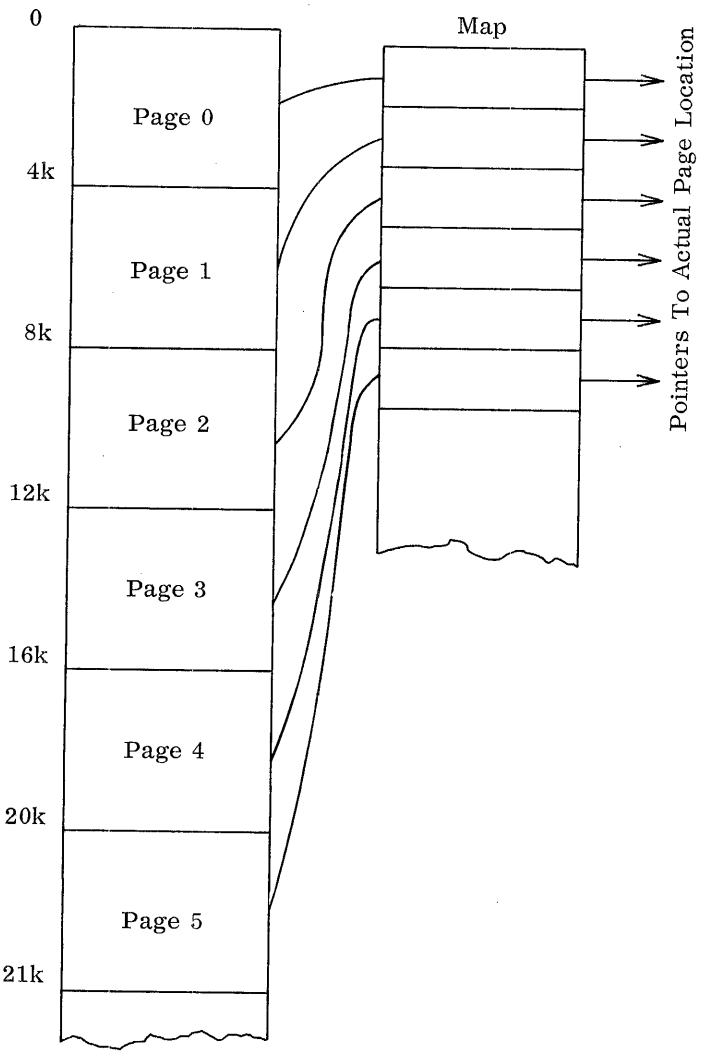


Figure 3.

its own virtual memory and relocation tables. As the supervisor controls the use of core storage, no task has the ability to address any portion of another task's core. Shared programs and data sets are an exception to this.

A page of parallel reenterable code is also sharable. Sharable means that the code may be executed by multiple users concurrently, and only one copy of the shared page is required (see Figure 4). In other words, the maps of the different tasks point to the same page in core storage. The total number of physical pages is less than the total number of logical pages. The shared pages are a physical intersection of the two programs. Since the two programs have distinct maps, they are logically disjointed (see Figure 5).

In summary, the virtual memory concept allows each task in a time-sharing system to operate as though the full virtual memory were available to it, with all necessary storage bookkeeping being performed by a combination of equipment and programming features. Note, however, that the transfer of pages from auxiliary storage into core storage, consumes a certain amount of time. The most efficient total use of the system, therefore, is dependent on minimizing this as much as possible. A program that makes many random widely-spaced references to a large table, or that transfers control frequently over large sections of code, can cause enough paging effort to degrade the total system performance. This should be avoided.

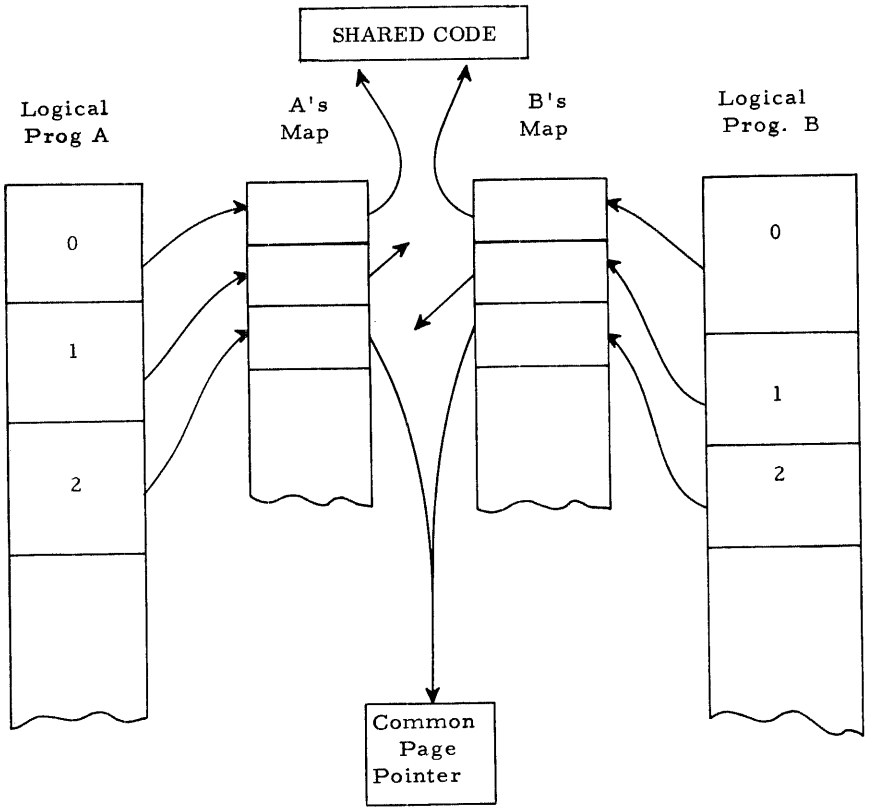


Figure 4. Example of shared code

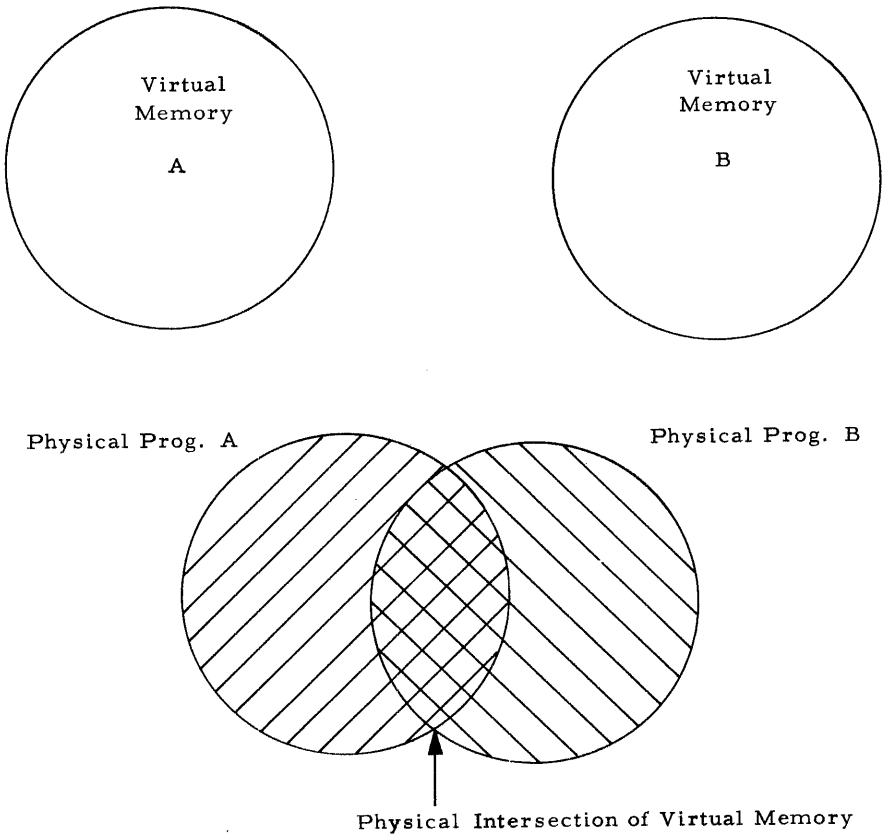


Figure 5. Shared code intersection

THE PAGING CONCEPT

The TSS/360 supervisor has the job of placing active tasks into available core storage. The task runs in core storage, but because of space limitations or relocation of programs, the program contiguous in virtual memory may be scattered throughout core storage. This physical fragmenting of programs is not apparent to users, who should think of their programs as being in contiguous virtual memory locations.

Physical fragmenting, or paging, has the following effects:

- The entire task need not be in core storage at one time. Instead, pages of many tasks are present, and several may be ready for execution. The system, therefore, has many opportunities to do useful work while swapping a page.
- Swap time is reduced as an overhead factor, since only active pages of a task ever require movement between core storage and auxiliary storage.
- Actual core size ceases to have any significance to a programmer. Although written and executed as a classical set of contiguous instructions and working space, a task exists in the machine as scattered pages. Therefore, the programmer's concept of the task in execution is oriented to virtual memory rather than to actual core storage.
- With so much virtual memory space available, it becomes possible to extend the page-turning concept to replace input/output operations upon data. If a task in virtual memory allocates a large area for data storage, the space that is occupied by the data set can be treated as an extension of the virtual memory, and can be operated upon directly without explicit read or write commands. As each page of the data is referred to, it is page-turned into core storage by the system. The effect, to the programmer, is the same as if the entire data set were always present in core storage.

The page-turning concept (see Figure 6), then, provides for efficient use of the equipment, while allowing the system to handle many tasks in rapid succession. In addition, it frees the computer user from the arbitrary bounds of actual core size.

MULTIPROCESSING

The number of tasks that can be serviced efficiently in a time-sharing system is increased by adding central processing units in parallel. The TSS/360 is designed to operate using two or more central processing units (CPU's) in parallel. With two or more CPU's operating in parallel, the system can recover from the failure of one, and continue to provide efficient service to users. It also allows for system growth without upheaval. The system is designed to accept up to four CPU's, each of which is a device to be allocated by the TSS/360 supervisor.

The supervisor is not changed by execution, and thus presents no problems with respect to simultaneous execution by more than one CPU. The supervisor does, however, modify certain system tables. To prevent confusion within the time-sharing environment, the supervisor makes the modifications in a sequential manner, rather than in parallel. In these circumstances, the CPU that is executing the supervisor that first begins to modify the system tables locks out all other CPU's until the modifications have been completed. The locking out of CPU's is accomplished through the

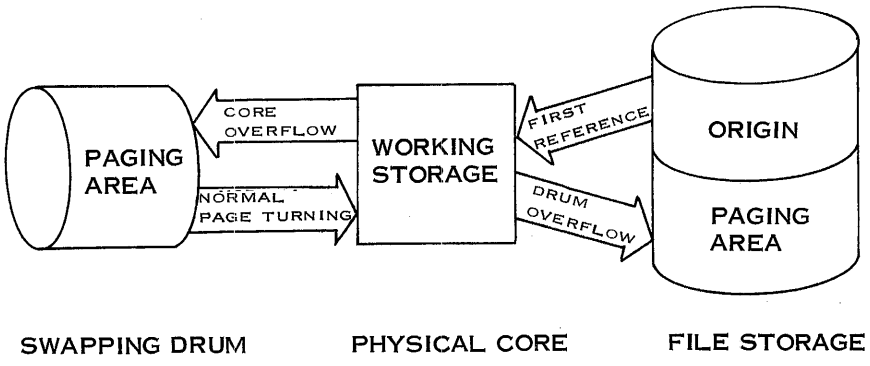


Figure 6. Page turning concept

use of a combination of programming mechanisms and wired-in (equipment) interlocks. CPU's are locked out only when necessity demands, and the lockout time is held to a minimum.

In summary, two or more processors in the Time-Sharing System/360 provide computational power, reliability, and flexibility. The system can allocate tasks among several CPU's, or it can operate with only one. In addition, the system can be "partitioned". This means that a selected CPU plus storage and I/O devices can operate as an autonomous unit.

SYSTEM PARTITIONING

Normal operation must be altered under two conditions (this altering is called partitioning):

1. If a control device becomes inoperative or is required for servicing, it must be made logically unavailable to the resources of the supervisor; at the same time, it should be disconnected physically from the system. Partition switches are located on each tail coming from a unit. To remove a unit from the system physically means that all partition switches connecting the unit are turned off manually. Since a unit may be actively engaged at a time when a physical partition is required, it is first necessary to notify the supervisor, via the operator's console, of the intention; as soon as the supervisor can logically disconnect the unit (remove it from the list of available devices), it so signals the machine room operator. At that point, the device can be disconnected physically. When a partitioned unit is returned to the system, a message is sent to the supervisor, through the operator's console, and the physical partition switches are turned on. The supervisor updates its device availability table and can begin to schedule its use.

2. If a particular subset of the total system resources is required for special runs that are best scheduled in a non-time-sharing environment, the supervisor again must be required to produce this logical partition. As soon as the supervisor can remove the required units from its pool of devices available to time sharing, it logically does so. It may have to wait for some I/O to complete or for a particular CPU to be released from an active program. Since all system activity is under its scrutiny, the supervisor eventually knows when all of the required units are available. It then sets up the partitioned subsystem with the job that requires it, and starts it up. During the partition time, an observer can see two systems working in parallel. After the job completes its run, the CPU signals the supervisor, which logically returns the units to its table of available devices. As soon as it can, the supervisor begins to reschedule time-sharing activity on the units.

LEVELS OF TSS/360

The Time-Sharing System/360 programming environment is divided into three privilege levels:

- Level 1, the highest privilege level, is the supervisor.
- Level 2 contains the task monitor and many service routines.
- Level 3, the lowest level, is the TASK or problem program.

Level 1, the supervisor, is parallel reenterable and resides in core storage in the unrelocated mode. It runs in the supervisor state, that is, it has access to all instructions and occupies about 25 pages. The supervisor is responsible for handling all interrupts, scheduling and allocation of processor execution time, core space, I/O data paths, and auxiliary storage. The supervisor initiates all channel programs, but is responsible for access and error recovery of only those I/O devices that are being used for paging.

The supervisor is not accessible to users and therefore is completely protected from interference. The only way that a user's task can communicate with the supervisor is through the use of a service (SVC) call, which causes an interrupt. Thus the supervisor can perform work only in response to an interrupt either through a user call or through an equipment action, such as machine-check of I/O action.

Level 2 modules are reenterable; they run using time slices as do users tasks, and are relocatable, residing in a task's virtual memory. Because the modules are reenterable, only a single copy need be in core storage, and the copy can be shared among several tasks. Because level 2 modules operate in a semiprivileged state, they cannot communicate directly with the problem's program; instead, control is passed to or from these modules by use of SVC's, which create interrupts that are processed by the supervisor in level 1, which in turn passes the requests to level 2 modules. Level 2 modules allocate and access all I/O devices and channel programs for nonpaging I/O, allocate and reserve virtual memory, and consist of service routines, such as catalog services and the task monitor. The task monitor module initiates and services tasks.

Level 3 consists of problem programs that run in the nonprivileged mode, that need not be reenterable, that are relocatable, and that reside in virtual memory. Level 3 modules include the language processors, assembly program, and user-written processing modules.

Nonprivileged SVC's are available to level 3 modules to request service of level 2 programs. No interrupt scheme exists for level 3 programs, since they do not process interrupts. All instructions are executed sequentially, and each is

finished before the next is started. This creates a very simplified operating environment and eases the programming burden for the user. The user may control problem execution by specifying certain pages of his program module as read only pages. This bounds the operating module and simplifies debugging.

Privilege relates to the type of machine language instruction a module can execute; priority relates to the time that the task will be executed; and command privilege relates to the types of commands the user can execute.

COMMAND SYSTEM

The command system consists of the command language interpreter (CLI), the various command programs, and the program checkout subsystem (PCS). CLI and PCS are the interface between user and system; they enable users to enter, manipulate, and control the running of programs. CLI enables operators to control the operation of the system. CLI and PCS occupy part of each user's virtual memory. Command system is a level 3 program.

DATA MANAGEMENT

Data management facilities control I/O devices and provide device-independent operation for system modules and problem tasks. The command system interfaces with the data management system through macro instructions. The data management system is relocatable, and required routines are loaded into the user's virtual memory. Data management is a level 2 function.

LANGUAGE PROCESSORS

The TSS/360 language processors translate source language program modules into object program modules. Conversational facility and language diagnostics are provided for terminal users during the analysis passes of the FORTRAN and macro assembly processors. The processors reside in the user's virtual memory.

SORT/MERGE

The sort/merge program is a level 3 task, residing in the user's virtual memory, that uses virtual memory as a scratch area to sort or to merge data sets.

INTERFACES

The interface between the various TSS/360 components are designed so that modifications and additions can easily be made to the system (see Figure 7). IBM-supplied programs — such as the compilers, utility programs, etc. — are treated no differently by the supervisor than are user-supplied programs. This permits easy additions to the program libraries by both IBM and the user.

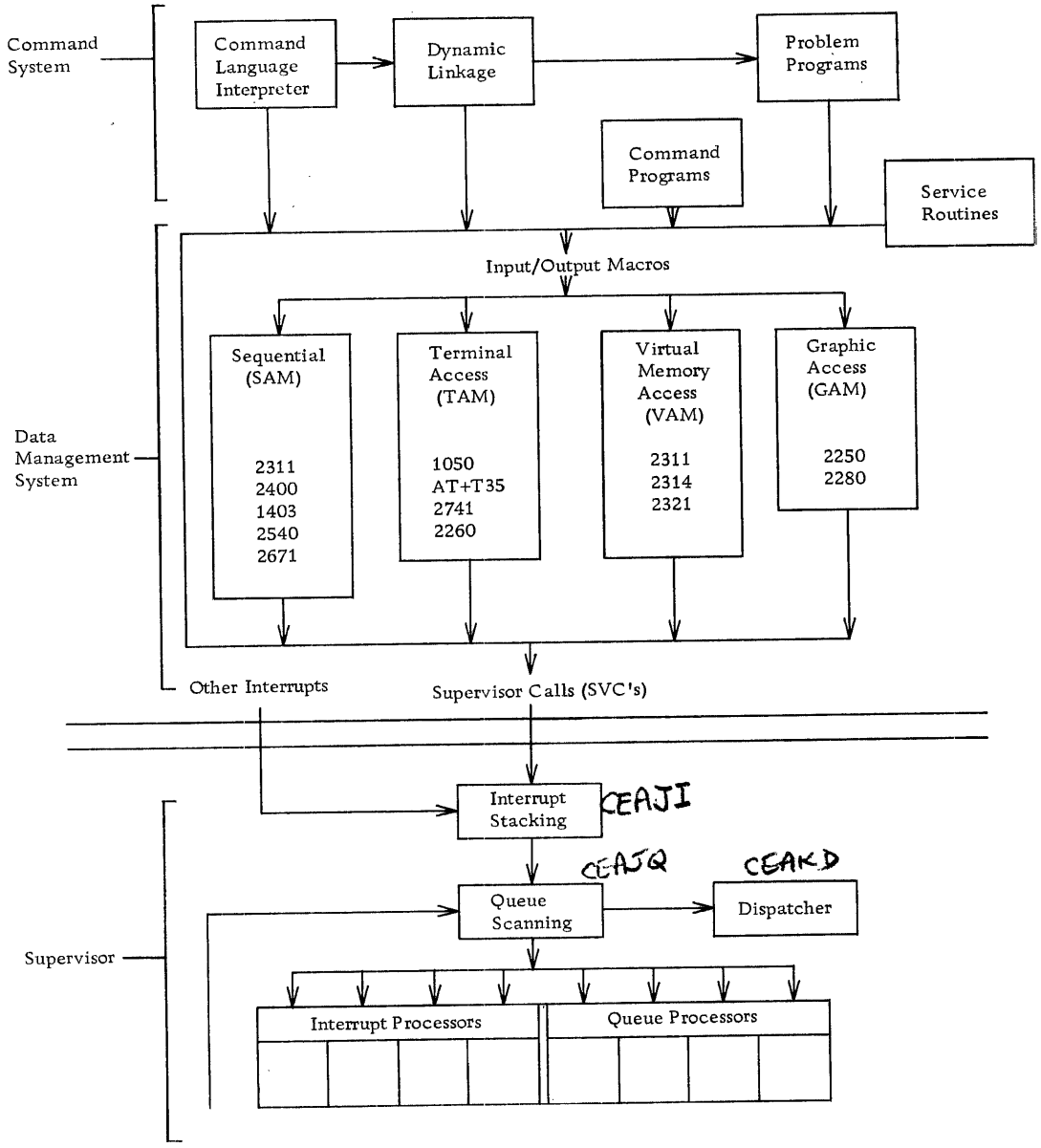


Figure 7. TSS/360 system interfaces

Time sharing, by its very nature, requires extremely large storage capacity. However, not all information must be immediately available in core storage, since all tasks and data sets are not simultaneously active. When tasks or data sets, or portions, become active, they must be moved quickly into core storage so that delays are minimized.

DYNAMIC ADDRESS TRANSLATION

The dynamic address translation feature of System/360, Model 67, minimizes the housekeeping required in the scattering of the program and its data in core storage. It permits the use of the virtual memory, which is far larger than actual core storage. The virtual addresses are translated to actual addresses by relocation tables that are set up and changed on a continuing basis by the supervisor as information is moved back and forth between core storage and drums or other storage devices. This moving of information is controlled by the supervisor in response to user demands.

Part of the equipment provided is a set of eight associative registers that maximize processor performance by doing an automatic table lookup for address translation. These registers are called associative registers, since, in the translation process, they are referred to by their contents rather than by physical location.

Another feature provided to maximize address translation speed, is storage of the instruction address (that is, the instruction counter) in the translated or relocated form. This feature obviates the need for instruction address translation until a branch occurs or a page boundary is crossed.

The IBM 2067 processor uses either a 24-bit or a 32-bit addressing for the time-sharing system, so that a program may have up to 16 million 8-bit bytes or 4096 segments of 256 pages. The TSS/360 programming system works with equal facility in either addressing mode.

The processor recognizes the relocation mode only when it has previously been switched into the extended control mode, as described below. In the extended control mode, bit 4 of the program status word (PSW) specifies the extent of logical addressing, that is, when bit 4 is a zero, 24-bit logical addressing takes place; when bit 4 is a one, 32-bit logical addressing takes place. Bit 5 of the PSW (in extended control mode) specifies relocation vs nonrelocation mode. When bit 5 is a one, relocation takes place; when bit 5 is a zero, no relocation takes place. When bit 4 is a one, bit 5 must be a one, otherwise a specification exception is recognized.

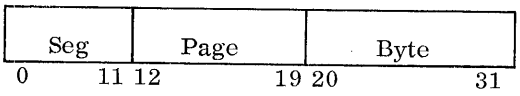
The following summarizes the modes of operation specified by bits 4-5 of the PSW in the extended control mode.

Bit 4	Bit 5	Mode of Operation
0	0	No relocation, 24-bit address arithmetic
0	1	Relocation, 24-bit address arithmetic
1	0	Specification exception
1	1	Relocation, 32-bit address arithmetic (specification exception if 32-bit addressing option is not provided)

ADDRESS TRANSLATION

The relocation tables used to translate a logical address into an actual address consist of segment tables and page tables. These tables are placed in main storage at the segment table origin and page table origin, respectively. Each table occupies the number of storage locations specified by the respective table length amount.

For the purpose of address translation, the maximum addressable storage capacity of 2^{32} bytes is divided into 4096 segments, each segment is divided into 256 pages, and each page contains 4096 bytes. Consequently, a logical address consists of a segment field, a page field, and a byte field. The segment field consists of twelve bits (bits 0-11); the page field of a logical address consists of eight bits (bits 12-19); and the byte field consists of twelve bits (bits 20-31).



Segment table origin is specified by the contents of bit positions 8-31 of the table register (control register 0). The length is generally 64 bytes, or one group of entries (16 entries per group, 4 bytes per entry). The address of the table origin must be a multiple of 64; hence, bits 26-31 of the table register must be zeros, or a data exception is recognized. Each 4-byte entry in the segment table defines a page table. The first byte (bits 0-7) defines the length of the page table; the remaining three bytes (bits 8-31) define the page table origin. The unit of length for a page table is a 2-byte entry.

Thus, the table is variable in multiples of two bytes. Each page table's origin is located at a byte address that is a multiple of 2. Thus, bit 31 of each segment table entry that defines a page table is zero. If bit 31 is one, no translation takes place, and a segment relocation exception is recognized (program interruption with interruption code 16).

Figure 8 illustrates the translation action. Bits 0-19 of the logical (or virtual) address are first compared with the corresponding bits of each associative register having bit 36 set to one. If a match is found, bits 20-31 of that register are used as bits 0-19 of the core storage address, and bit 37 at the associative register is set to one. Bits 20-31 of the logical address are used directly as bits 20-31 of the core storage address.

If no match is found, or if no register in the associative array has bit 36 set to one, the logical address must be translated by means of the segment and page tables. This translation proceeds as indicated by the dotted lines in Figure 8. The segment field of the logical address (bits 0-11) is first added to the origin address portion of the table register (bits 8-31). (For this addition, the segment field of the logical address is aligned with bits 18-29 of the table register, since the entry to be fetched is four bytes long and has a byte address that is a multiple of 4.) The quantity obtained by this addition is the address of the segment table entry.

The segment table entry is used with the page field of the logical address in much the same manner as the table register contents were used with the segment field.

Bits 12-19 of the logical address are aligned with bits 23-30 of the origin portion of the segment table entry (bit 8-31), and the two quantities are added. The resultant 24-bit quantity is used as the address of of a 2-byte page table entry, which is later fetched from storage. As described earlier, if bit 31 of the segment table entry is one, a segment relocation exception is recognized. In addition, bits 12-19 of the logical address are compared to bits 0-7 of the segment table entry, and, if the former is greater than the latter, a page relocation exception is recognized (program interruption, with interruption code 17).

The 2-byte table entry consists of a page address portion (bits 0-11) and control bits (bits 12-15). Bits 13-15 must be zeros, or a specification exception is recognized, and the instruction is suppressed. Bit 12 defines the status of the page address portion of the entry. If bit 12 is zero, the page address is used as bits 0-19 of the core storage address. If bit 12 is one, a page relocation exception is recognized. Thus, bit 12 serves to indicate whether the page referenced is actually available in core. Bits 13-15 are reserved for future use.

The actual core address obtained by the translation method described above is not only used to address core storage but is also loaded into an associative register along with the segment and page fields of the virtual address. Thus, it is made available for future use without the need for repeating the translation process. When an associative register is so loaded, bit positions 36 and 37 in the register are set to one. (Selection of the registers to be loaded is under control of a usage algorithm, which uses, in sequence, the registers with bit 37 set to zero. When bit 37 is one in all registers, this bit is reset to zero in each register.) Bit 36 is used to indicate the presence of a valid entry in the associative register. It is reset to zero each time the content of the table register is changed.

RELOCATION MODE

Relocation of addresses provided by the processor is specified by bit 5 of the PSW. When the bit is a one, relocation takes place; when the bit is a zero, the logical address is used as the actual address.

All core storage locations where information is stored in the course of an operation are subject to relocation.

Addresses provided by the channels, either for fetching channel control words from core storage or for fetching data from or storing data into core storage, are never relocated, regardless of the setting of bit 5 in the PSW.

Locations whose addresses are generated by the processor or channels for updating or interruption purposes (equipment-generated addresses), such as the timer, channel status words, or PSW addresses, are not relocated via the relocation tables. However, when the program module specifies these locations, they are subject to relocation, as defined above.

Core storage addresses in the range 0-4095 (including the aforementioned equipment-generated addresses) are relocated by means of the primary or alternate prefix, as defined in System/360 Principles of Operation, unless the prefix is disabled by means of the prefix deactivation switch. Consequently, the prefix is applied when the actual core address, that is, either the virtual address when no relocation takes place or the translated address obtained via the relocation tables, falls within the range 0-4095.

When relocation is specified, the storage protection, by means of the protection keys, is still active.

Whenever access to core storage is made by the equipment for the purpose of fetching an entry from a relocation table in the course of an address translation process, storage protection is ignored, that is, the equipment acts as though the block of storage

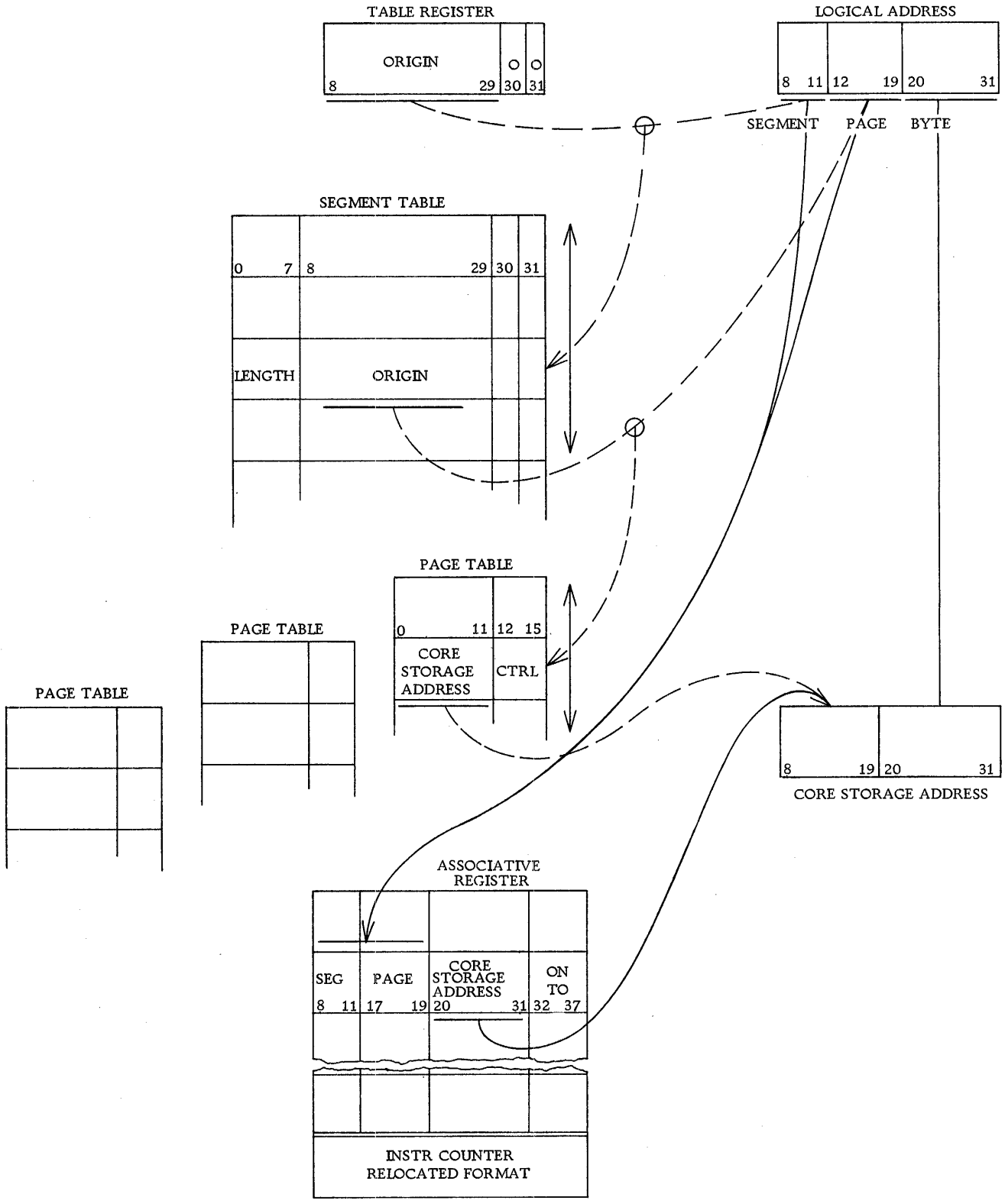


Figure 8. Simplified data flow for dynamic relocation

containing the relocation tables were not fetch-protected during the storage cycle in which the relocation table entry is fetched. However, if the addresses at which the relocation tables are located are generated by the task, they are subject to storage and fetch protection in the normal manner.

If the storage address, generated in the address translation process for fetching a relocation table entry, exceeds the storage capacity of the installation, an addressing exception is recognized, resulting in an interruption (interruption code 5).

STORAGE PROTECTION EXTENSIONS

The advanced storage protection features of the IBM System/360 have been made still more useful and flexible in the IBM 2067 processor and core storage units. Although the storage protection keys are functionally a part of core storage, they are discussed here because they are logically related to the new and expanded features of the processor.

Storage protection is achieved in System/360 by providing main storage into blocks of 2048 bytes. A 4-bit key is associated with each block. When storing of data is specified by an instruction from the processor or a channel, this storage protection key is matched with another key supplied by the current program status word (PSW) or channel address word (CAW). When a mismatch is detected, storing is suspended, and a processor interrupt occurs. The protected storage location remains unchanged.

The 4-bit storage protection keys have been extended to seven bits. Bits 0-3 are the standard 4-bit storage protection keys. Bit 4 is the fetch protection bit, bit 5 is the reference bit, and bit 6 is the change bit. The protection keys in the PSW and the CAW remain unchanged at four bits in length.

FETCH PROTECTION

When the fetch protection bit is zero, no protection against fetching by either a processor or a channel is indicated, regardless of the value of the 4-bit storage protection key in the PSW or the CAW, or the value of bits 0-3 of the storage key.

When the fetch protection bit is one, the data or instruction in the corresponding storage block are protected against fetching whenever they are protected for storing.

When an instruction causes a fetch protection violation, execution of the instruction is terminated, a program interruption occurs, and a protection exception is indicated in the old PSW. The protected information is never loaded into a register or moved to another location.

Fetch protection violations caused by a channel result in a termination of data transmission; thus, the protected information is never transferred to any output medium. The violation is indicated in the channel status work (CSW), which is stored as a result of the channel operation.

Locations whose addresses are generated by the processor for updating or interruption purposes — such as the timer, CAW, and PSW — are not fetch-protected. When a program specifies these locations, however, they are subject to protections.

REFERENCE AND CHANGE

Bit 5 of the storage key, the reference bit, is set to one each time the corresponding storage block is accessed for storing or fetching by a processor or a channel. Bit 6, the change bit, is set to one each time data is stored in the corresponding storage block by a processor or a channel.

The storage key is not part of addressable storage. The key is changed by the set-storage-key instruction, and is inspected by the insert-storage-key instruction. These instructions, bits 0-6 of the storage key, correspond to bits 24-30 of the register designated by the R1 field.

The reference and change recording is always active. It is independent of the problem supervisor or masked state of the processor, of the type instruction of I/O command being executed, and of the manner in which the address is generated. Hence, reference for updating or interruption purposes — such as the timer, CSW, or PSW locations — are included in the reference and change recording.

OPERATIONAL DIFFERENCES IN SYSTEM/360 MODEL 67, MACHINE INSTRUCTIONS

Five new instructions have been added to the System/360, Model 67, and the operation of certain previously established instructions is modified when the 32-bit addressing option is operative.

Two of the new instructions have been added to the basic machine operation set. Load multiple control (LMC) and store multiple control (STMC) are privileged operations that do not concern the problem programmer, and load real address (LRA) is used only by the system. Accordingly, their description is omitted here.

The remaining two instructions, branch and store (RR and RX formats), are shown below.

Instruction	Mnemonic	Type	Code
BRANCH and STORE	BASR	RR	0D
BRANCH and STORE	BAS	RX	4D

The branch and store instructions are very similar to the branch and link instructions and other System/360 models. In coding for TSS/360, the programmer must use the BAS instruction instead of the BAL instruction for linking to a subroutine that uses the stored PSW value to return. This is required to allow compatibility with the 32-bit addressing option wherein the BAL instruction stores only 24 bits of the 32-bit location counter value from the PSW. The BALR instruction should still be used to obtain the condition code, instruction length, and program mask in a general purpose register, since the BAS or BASR instructions do not provide this information.

The updated logical instruction address of the PSW is stored as link information in the general register, specified by R_1. The logical instruction address is later replaced by the logical branch address.

Condition code:	The code remains unchanged.
Program interruption:	None.

Programming note: The link information is stored without branching when in the RR format and when the R_2 field contains zeros.

When branch and store is the subject instruction of execute, the instruction-length code is 2.

OPERATIONAL DIFFERENCES IN SYSTEM/360, MODEL 67, INPUT/OUTPUT

The model 67 extends the normal System/360 input/output capabilities to include:

1. Up to 28 input/output channels
2. Multiple path I/O — any CPU (of the one to four available) may access any and all I/O devices on all attached channels
3. The fetch storage protection feature, which allows system control over the ability to read from certain portions of core storage

The dynamic address translation feature is not available for translating addresses contained in I/O commands (CCW's); hence, the TSS/360 supervisor intercepts all I/O requests and translates the CCW addresses from virtual memory addresses (which the user or system routines supply) to actual core storage addresses.

The command language is the medium of communication between the TSS/360 and its users. The command language is designed primarily for users who communicate with the TSS/360 during execution of their tasks; this operating mode is described as conversational in that the user remains online to the system, engaging in a dialogue with it, during the execution of his tasks.

The various commands in the language enable the user to construct, checkout, and execute program modules and/or manipulate his data sets; for example, he may issue commands specifying that they are to be modified in a specific manner, shared with other users, copied, and moved to or from I/O devices located at the computer center. Other commands are reserved for the main system operator to control and monitor the system configuration and to facilitate servicing of system requests for I/O device handling. Still other commands are reserved for the system administrators to establish user accounting records and to control user access to the system.

A subset of the command language system is the program checkout subsystem (PCS), which provides the user with conditional dump capabilities during program testing. The portion of the command system that is active for a user occupies part of that user's virtual memory.

The device management facilities of the command system allocate I/O devices, directing operator actions with them and with data volumes in response to user requirements. The user need not be directly involved with devices. Instead, his commands are contained in data sets that are interpreted by the command language as device requests and that are presented to the system operator for execution.

The command language also serves as the control language for users who prefer the nonconversational mode of operation; that is, offline preparation of programs and data for batch submission and execution by TSS/360 without user monitoring. The user is free to switch from conversational to nonconversational mode if he satisfies the requirements for that mode before he initiates it. For example, he may modify a prestored program or set of test data in the conversational mode, then initiate nonconversational execution of the modified data sets. With this approach, the user need not attend the system during execution of his program; he simply postpones receipt of the results of that execution.

CONVERSATIONAL MODE

For conversation operation with TSS/360, the user employs a terminal that can be remote from the computer center. Each terminal consists of a printer-keyboard, such as an IBM 1052 or an IBM 2741 or a Teletype Model 35 KSR; optionally, an IBM 1056 Card Reader can be supplied with the IBM 1052. TSS/360 users are authorized to use the system by a system administrator. Once authorized, the user has access, through his terminal, to the total facilities of TSS/360, including the system assembler, compilers, and service routines. As he requests these facilities (either by commands on his keyboard or by feeding them in punched-card form through the card reader), he is engaged in a dialogue with the system. He is told of the actions taken by the system in response to each command, asked for additional information needed to complete an action, informed of errors in his command entry, and told of the options he may exercise. The commands, command abbreviations, and the system responses are in English prose, designed for clarity and ease of comprehension.

NONCONVERSATIONAL MODE

For nonconversational operations with TSS/360, the user need not ever see, much less use, a terminal. Instead, he can prepare his programs and data on punched cards or magnetic tape, together with a sequence of job control instructions for the program and data, and submit the job in that form for batch execution. The job control instructions are written in TSS/360 command language.

A nonconversational user with access to a terminal can use the conversational mode to enter changes, etc., to programs and/or data previously entered in nonconversational mode. Since the user is not available to correct errors during nonconversational operation, his job is terminated when any errors are encountered. The nonconversational mode is thus most useful for jobs that do not justify the user's attendance during execution or that do not require his resolution of problems. The conversational mode, conversely, is most suited to interactive program development, which involves the user in a self-corrective dialogue, and to use of the system in a tutorial role.

TSS/360 commands are executed interpretively; that is, each command is analyzed and then executed immediately after entry from a keyboard, a terminal card reader, or after retrieval from a prestored data set. The command and its parameters are validated before execution. In conversational mode, any errors or omissions produce requests to the user for correction or completion of the parameters; in nonconversational mode, any errors or omissions force termination of the job. Once the command is accepted as correct, all actions it requires are performed before the next command is sought. The system does not record the commands it executes; the user has a record of these commands — on his printer output, his input card deck to the terminal card reader, or in his prestored command sequence. The results produced by command execution are retained by the system.

COMMAND LANGUAGE SUBSYSTEM

The command language subsystem consists of the programs, routines, and subroutines that support the command language. The subsystem includes three major functional parts:

- The command language interpreter (CLI), which recognizes each command, links to the appropriate command routine, and handles messages to and from the user
- The command routines, each of which performs the action(s) required to carry out its corresponding command
- The batch monitor, which initiates and supervises nonconversational processing

When a user starts conversational operation by turning on his terminal, dialing up the system, and pressing the ATTENTION button, a task is created for him. Each task is active only during its time slices, making use of CPU time only during successive time slices. A task exists from log on, whether explicit or implicit, through log off. Upon creation of the task, a copy of the command language interpreter is loaded into the task's virtual memory, and, since the first operation must be a log on, the CLI adds a copy of the LOGON command routine. For a conversational user, an implicit log on is assumed, causing the LOGON command routine to issue prompting messages through the CLI to the user. The CLI reads the user's responses, passing them on to the LOGON command routine to check the user's credentials, to establish accounting records for the charge number that he supplies, and to save his options concerning confirmation and system messages for other command routines.

When the implicit log on procedure is completed under control of the LOGON command routine, the command routine informs the CLI, which then asks the user to enter his next command. This command is recognized by the CLI, which moves the corresponding command routine into the user's virtual memory, replacing the now unnecessary LOGON command routine. As before, the command routine passes prompting or diagnostic messages through the CLI to the user if parameters are missing or in error, obtaining user responses through the CLI. When the user enters or calls in program data sets from external storage, these data sets are also added to his task. Similarly, copies of the system assembler and/or compiler(s) are moved to the user's virtual memory as he requests them; these language processors are also executed conversationally, informing the user of system response to each of his source language statements, so enabling him to alter his tactics in the light of each response.

The user can interrupt execution of his task by actuating the ATTENTION button at his terminal; the responses to this interrupt depend upon the operation in progress. If the user's task is executing a compiled or assembled program, the ATTENTION interrupt stops that processing without disrupting it; the user can, with the appropriate command, direct the system to resume processing from the point of interruption. If the user is entering a command and has not completed that operation, he can use the ATTENTION button to cancel that partial entry.

A task ends when a LOGOFF command is issued. The CLI examines the command, causes the LOGOFF command routine to be added to the task, and passes to the conversational user queries on the disposition of his uncataloged data sets. The data sets he chooses to save are entered in his catalog; all others are lost (like a console patch) as soon as his task ends. Upon disposition of his last data set, the system is so notified, the user's task is eliminated from the system, and all of his influences are removed from the system.

A nonconversational job differs from the above description in several ways. The nonconversational job can be submitted to the system operator(s) on punched cards and/or magnetic tape. A task is created for the job by the batch monitor when the facilities (for example, I/O devices) required for the job are available for assignment to it. The job must contain explicit LOGON and LOGOFF commands, since the user is not available to respond to prompting. For the same reason, the task for the nonconversational job is terminated upon encountering a situation that requires user resolution, that is, a situation resolved in conversational mode by user responses to prompting or diagnostic messages. In this nonconversational mode, the system input (SYSIN) to the task is not a terminal but a prestored set of commands.

Similarly, the system output (SYSOUT) from the task is a data set, not a terminal. This data set is printed out eventually and returned to the user for examination.

A user may change from conversational to nonconversational operation if he has developed a prestored set of commands to direct the latter. As before, the CLI interprets his request to change mode, links to the appropriate command routine, and handles any communications between that routine and the user. The command routine next establishes the new task in nonconversational mode and gets a batch sequence number from the batch monitor. The task established initially for the user is eliminated, since no further communication with the user is expected or needed by the nonconversational task. The user may, if he so desires, initiate further conversational operation from his terminal by logging on again; however, this operation should be unrelated to the nonconversational job. If no task space were available for the user's nonconversational task, his command would be rejected. The user could reissue the command later, after continuation in conversational mode.

PROGRAM CHECKOUT SUBSYSTEM

The IBM Time-Sharing System/360 Program Checkout Subsystem (PCS) is designed to assist users in the checkout of new or revised modules. Among the new approaches made possible by the overall design of the IBM Time-Sharing System/360 is direct user intervention in the loading and execution of a task. This permits the user to check on its progress; modify it, where the source of difficulty is immediately evident, and, if the output is not acceptable, systematically pinpoint where the trouble originated. This flexibility is achieved without interfering with the system's power to serve many users simultaneously and efficiently, and without including any statements that are needed only for checkout or for program logic error or malfunction detection in the module while it is being written. Instead, PCS, under user supervision, effectively controls the execution of a particular module or set of modules, and is responsive to requests for a wide variety of actions that bring the user into direct communication with the process of module loading and execution — which is the task that is to be completed successfully. The following is a general introduction to the objectives and concepts underlying PCS.

PCS Objectives

The functional specifications underlying PCS design are to:

- Make data items and instructions within a module "visible" to a user, who can request their display by means of symbolic references
- Enable the user to modify the contents of data items within a module by assignment statements, expressed in symbolic terms and with standard representation of integer numbers, floating-point numbers, and character strings
- Permit the user to specify — either symbolically or by virtual memory addresses — points within a module at which execution will start and stop; on stopping, items may be displayed or modified as indicated above, followed by either resumption of execution or a pause for user intervention
- Provide a means of defining logical conditions or specific event occurrences that will determine whether action is to be taken at stated points in the module
- Give the user a programming language and syntax that are easy to remember, and that lend themselves to operation from remote-access terminals

Symbolic Referencing Capability

PCS is designed to work in conjunction with the language processors, implemented for the IBM Time-Sharing System/360, and with the command language interpreter (CLI), the dynamic loader, and the linkage editor.

A prerequisite for the most effective use of PCS is that the user retain the internal symbol dictionary, which the language processors produce; it is the retention of the dictionary that permits use of the program's own symbols in writing checkout statements. Because CLI recognizes checkout statements and passes them to PCS for interpretation and response, the services of PCS are always available to the user. Note that the full power of PCS cannot be used in a shared module; the user is limited to investigatory action, such as displaying the contents of any location in virtual memory. The ability to change variable, or to use dynamic checkout statements that take effect upon the occurrence of specified object module events, or to intervene in a task's execution is not available until a "nonshared" module is loaded.

References to a source module are made in terms of the programmer's own internal symbols; this avoids need for concern about core storage locations. If the user had not earlier suppressed the output of the internal symbol dictionary (ISD) by the language processor, he has access to most of the symbols contained in his program, along with their attributes, at checkout time. If he had not anticipated the need for program checkout, and does not have an ISD, the user can still refer to his program in terms of its external symbols.

Automatic Display Format

A drawback of conventional snapshot and core dumps is that it is necessary to specify how internal machine code is to be represented, so that dump output may be easily readable; for instance, a portion of core storage containing integral numbers should be shown differently from one containing floating-point numbers. To present data in the form in which it is most likely to be understood PCS utilizes various types of information from the ISD's produced by the language processors. This is automatic, and the user need not specify either conversion or format requirements.

Statement Types

The user communicates with PCS conversationally through input of PCS statements at the terminal. PCS statements are of several types.

Statements whose requests are acted upon immediately by PCS are termed immediate statements. Dynamic statements are those which are not acted upon until the occurrence of a specified event during object program execution. Immediate and dynamic statements may be either conditional or unconditional. Consequently, there are basically four statement types:

- Unconditional immediate
- Conditional immediate
- Unconditional dynamic
- Conditional dynamic

In addition to these four statement types, PCS control directives exist, which control the effect of prior statements and modify the processing of later statements.

Event Specification

The execution of a program is one event that is made up of an indefinite number of constituent events. PCS allows definition of specific events — such as executing certain object program module statements.

PCS permits symbolic reference to events by the user's symbolic labels. The occurrence of a defined event while the task is being executed can trigger a variety of PCS actions, such as the display, modification, or testing of data and/or the cessation of task execution.

Conditionally

This function provides for comprehensive logical statements that can determine whether an action is to be performed by PCS while monitoring the program's execution.

Statement Examples

This section presents a few examples of the type of operations possible in PCS. All internal symbols are written with preceding periods.

```
AT . ERREXT DISPLAY 0:15R 0:6D DUMP
. TOP:. BOT
```

When program control arrives at ERREXT, all 16 general purpose registers are displayed at the terminal, and all four floating-point registers and memory locations, TOP through BOT, are dumped offline.

```
AT . CALL SET . POINT = A'. DATA'
```

When the instruction at CALL is reached, the variable POINT is set to the address of DATA.

```
AT . S3
```

The user is informed each time control reaches the instruction at S3.

```
AT . DIAG IF . DLEVEL LT 3 RUN . CONT
```

When control arrives at DIAG, program control is transferred to CONT if the value of the variable DLEVEL is less than 3.

```
AT . LOOP DISPLAY %, . I
```

When control arrives at LOOP, a count of the number of times LOOP has been reached and the value of the variable I are displayed. (% is a counter that is incremented each time the PCS statement is executed.)

```
AT . ALPHA IF . X LT 0 SET . X = 0.
```

When control arrives at location ALPHA, if X is less than zero, it is set to zero.

```
AT .GOBACK IF .SORTAB(.I) GT
.SORTAB(.I+1) STOP
```

At location GOBACK, if the Ith entry in SORTAB is greater than I+1st, the program stops.

```
SET .Z(.I,.J) = .X(.I)*. Y(.J)
```

The I,J entry in the array Z is set to the product of the Ith entry in X and the Jth entry in Y immediately after the statement is input.

```
AT .99 IF .WRONG STOP
```

At statement number 99, if the logical variable WRONG is true, the program stops.

LANGUAGE PROCESSORS

The TSS/360 FORTRAN and COBOL compilers produce executable object programs from source programs written in the FORTRAN IV and COBOL languages, respectively. The compilers and their object program modules are intended for execution on the TSS/360 computer configuration, in conjunction with the supervisor, loader, data management, command language interpreter (CLI), and other components of the TSS/360.

In addition to the production of executable program modules, the compilers also detect and give notification of source program errors, and produce various documentation describing the object program module(s).

The TSS/360 assembler produces, from source language programs written in the System/360 assembler language, machine language programs in a format suitable for operation within the Time-Sharing System/360.

The assembler is integrated with other components of the TSS/360 in a way that enables users at remote terminals to control the portion of the assembly that is concerned with source statement syntax. The assembler design is such that a full range of diagnostic messages can be supplied for the benefit of the conversational user. Other components of the Time-Sharing System/360 provide facilities for the correction of source language statements during the course of the assembly.

The assembler can also function as a batch-mode language processor when conversational operation is not desired.

The Time-Sharing System/360 is designed to help the user enter source language statements as required by the user-specified language processor. The user initiates source language processing by issuing a LOAD command that names the language processor he wants. The language processor is then loaded. After the user commands RUN, he is prompted for the information required by the language processor. At this time, the user supplies the name of his source data set, tells whether it is to be created or is prestored, and gives the starting number and increment value for line numbers. He also chooses any options available for the language processor that he has selected. After its initialization, the language processor requests source language statements as they are needed. These statements are either read from a prestored source data set or requested from the terminal user. Diagnostics are issued on a statement-by-statement basis. All diagnostics pertaining to a statement are put out before the next statement is requested.

In conversational mode, a line number or, in some cases, a pound sign — that is, # — is printed at the terminal when it is ready to accept the next source statement. The user also uses these line numbers when he wishes to modify his program online if an error is spotted. To do this, the user types in %, signaling the processor that the user is now entering his own line number and a modification line. If the new line number is less than the last line number issued, a correction is assumed, and the last line number is reissued to resume the normal sequence. If the new line number is greater than the last line number issued, the new line is rejected, and the last line number is reissued. If the last line number issued is itself used to enter a line, the next sequential line number is issued. The user can delete lines by prefixing a D to the line numbers. For example:

```
500 % D, 300
```

means that 500 is the last line number issued, and the user wants to delete line 300. The processor handles the deletion, then reissues line 500. Note that modifications of any but the last entered statement cause the language processor to restart from the first statement.

When the entire source program is entered, the user is given the option of terminating the language processing (due to uncorrected fatal errors, usually), making modification to his source program and restarting, or running the processor to produce an object module. After the processor is finished, it indicates its results by a severity code. This code is placed in the object module, and a message is printed to inform the user. The severity codes are:

0 — no errors

1 — warning diagnostics issued

2 — fatal diagnostics issued
3 — fatal diagnostics issued and no object code produced

If language processing is done in the nonconversational mode, commands and the source program must be supplied as prestored data sets. There is no prompting, thus no chance for source statement modifications and corrections. The language processor accepts one statement at a time until the entire source program is handled; it is then run to get an object module.

DATA MANAGEMENT

The data management commands free the programmer from much of the clerical work involved in manipulating his data sets. Supported by the time-sharing catalog system, these commands enable the programmer to access his data sets by symbolic name alone. The commands provide for the reading in of data sets from tape, card reader, or the terminal. They enable the programmer to modify his data sets easily and to erase unwanted data sets. They permit the cataloging and uncataloging of data sets, and they also furnish automatic control of direct access storage space allocation, relieving the programmer of the details of allocating such space.

Aside from its control features, data management also provides security for every data set. Unauthorized access to a data set is prohibited, ensuring each programmer that no other users can access his data sets without his permission. The data management commands enable the data set owner to grant various levels of access to other programmers so that they can share his data set on his terms.

Data management also provides data access facilities to schedule and control the transfer of data between input/output devices and main storage. This includes reading and writing records, blocking and deblocking records, overlapping read-write and processing operations, reading, validating, and writing volume and data set labels, and repositioning volumes automatically. By the use of data management commands, the programmer can copy data sets or have them printed, punched, or written on tape without having to learn to access characteristics of the various input/output devices attached to the system.

TIME-SHARING SYSTEM COMMANDS

Command Format

Each command consists of a verb and, in most cases, a body. The verb identifies the command and the action to be performed (for example, ERASE). Every command verb, except LOGON, has a standard abbreviation that can be used in place of the verb, whenever the command is entered. The body of the command contains parameters, which are separated from the verb by a blank or a tab.

The TSS/360 commands may be divided into four functional groups — user commands, system operator commands, system administrator commands, and program checkout commands.

- User commands are intended for the programmer. Almost all of these commands may be entered from his terminal or may be placed in a stored data set for nonconversational execution. Table 1 lists the user commands and gives the function of and privilege class of each.
- System operator commands are reserved for the main system operator only. They permit him to monitor system operation, to adjust the system configuration, to control system time and data information, and to communicate with tasks and with users. Table 2 lists the system operator commands, their functions, and their privilege class.
- System administrator commands are also reserved, and enable the system administrator to identify users to the system, obtain accounting information, alter priority, and delete former users from the system. Table 3 lists the system administrator commands, their functions, and their privilege class.
- Program checkout commands provide for conditional dumps and halts during object module execution, as well as display of information addressed symbolically. Table 4 shows the program checkout subsystem commands.

Command Privilege Classes

Each user is assigned one or more privilege classes, which determine the commands he may

use. There are four privilege classes: A, B, C, and D. Class A is assigned to the system control operator. Class B is assigned to the system administrator. Class C is for subordinate system operators. Class D is assigned to TSS/360 users at terminals and nonconversational task operations.

Whenever a user attempts to enter a command for which he does not have the correct privilege class, a diagnostic message is issued, and the command is ignored. Some commands are available to several privilege classes, but the exact function of the command may differ according to the class of the user entering the command. Such is the case, for example, with the PRESENT LINE command.

Note that a single user is not limited to a single privilege class. The system operator, for example, normally holds privilege classes A, C, and D. In a small installation, one user might have all four privilege classes.

Some duplicate commands appear in the following tables, but there are no duplicate functions. For example, the command PRESENT is included in Tables 1 and 2, because the user can present only his own priority while a system administrator can present the priority of any user that he has joined. Both the main system operator and the system administrator(s) can use commands other than those specifically reserved for them.

Table 1. User commands

Command	Command Abbreviation	Function	Privilege Class
BULKIO PRINT	BU PR	To print a data set	D
BULKIO PUNCH	BU PU	To punch a data set onto cards	D
BULKIO TAPE	BU T	To write a data set on tape for offline printing	D
CANCEL	CAN	To cancel a nonconversational task	A, D
CATALOG	CA	To enter or change a data set name in the catalog	D
CHANGE CONFIRMATION	C CONF	To change the confirmation status of a task	D
CHANGE PASSWORD	C PA	To change the user's password	D
CHECKPT	CHP	To checkpoint a task by saving its current status and thus allow an eventual restart	D
COPY	DSC	To make a copy of an existing data set	D
DATA	DT	To build a data set from input records on SYSIN	D
DATADEF	DD	To identify and describe a data set to the system	D
DDCALL	DDC	To retrieve DATADEF commands from a cataloged line data set	D
ERASE	E	To erase the direct access storage belonging to a data set or data set member; if the data set is cataloged, its name is also removed from catalog	D
ERASEID	ER	To remove the internal symbol dictionary (ISD) from an object module	D
LOAD	L	To load an object module into virtual memory for execution	D
LOGOFF	LO	To indicate the end of a task	D
LOGON	none	To identify the user to the system at task initiation	D
MODIFY	M	To insert, delete, and/or replace line(s) in a data set	D
PERMIT	PE	To permit or restrict sharing of a data set by other users	D

Table 1. User commands (continued)

Command	Command Abbreviation	Function	Privilege Class
PRESENT ACCOUNTING	P A	To print accounting information for user's charge number(s)	A, B, C, D
PRESENT CHARGE NUMBER	P CH	To present a user's charge number(s)	B
PRESENT COMMAND	P CO	To print a description of any TSS/360 command	D
PRESENT DATA SET STATUS	P D	To print class, last usage, page count, and type of user's data set	A, B, C, D
PRESENT DATE	P DA	To print the current date and clock time	D
PRESENT LINE	P L	To present one or more lines of the user's line data set	A, B, C, D
PRESENT RESOURCES	P R	To present a list of allocated equipment elements	A, B, C, D
PRESENT TASK STATUS	P T	To tell a user whether a task is currently in the system and how long it has been logged on	A, B, C, D
PRESENT VIRTUAL MEMORY MAP	PV	To get a list of what is contained in the user's virtual memory	D
RELEASE	REL	To release a data set or to remove a job library from the program library list	D
REMOTE	REMO	To change a conversational task to the nonconversational mode	D
RESTART	RES	To restart a previously checkpointed task	D
RUN	R	To begin execution of a loaded program or to restart an interrupted program	D
SHARE	SH	To allow a user to share cataloged data sets belonging to another user	D
UNCATLG	UNC	To delete a catalog entry for a user's data set or sets	D
UNLOAD	UNL	To remove an object module from virtual memory	D
BATCH	B	To enter a nonconversational task into the system	C

Table 2. System operator commands

Command	Command Abbreviation	Function	Privilege Class
ATTACH	A	To logically include and, if a physical switch exists on the configuration console, to physically connect an equipment element to the system	A
~~BROADCAST~~	BR	To send a message from the main system operator to all terminal users currently active in the system	A
BULKIO RDCARD	BU RDC	To read input from a high-speed card reader, create and catalog a new data set, and, if the data set is a command procedure, request initiation of a nonconversational task	A
BULKIO RDTAPE	BU RDT	To read a data set from tape and create a VAM data set within the system	A
CANCEL	CAN	To cancel a nonconversational task	A, D
CHANGE CONSOLE	C CON	To change the primary or backup consoles for the system operator	A
CHANGE DATE	C DA	To change the system date	A
CHANGE TIME	C TI	To change the system time of day	A
DETACH	DE	To logically phase out and, if a physical switch exists on the configuration console, to physically remove an equipment element from the system	A
MESSAGE	ME	To allow the main system operator to direct message to a specified user or add a message to the operator log	A
PARTITION	PAR	To phase out and physically remove from TSS/360 a predefined list of equipment	A
PRESENT CONSOLE	P CON	To present the subchannel numbers and backup consoles for the system operator	A, B
PRESENT OWNER	P O	To present the current user of a device	A
PRESENT RESOURCES	P R	To get a list of all equipment elements in the time-sharing partition or a list of all equipment elements assigned to any task or user	A, B, C, D
PRESENT TASK STATUS	P T	To get the amount of processing time used thus far by any task in the system	A, B, C, D
REPLY	REP	To allow the operator to answer a message from the system	C

Table 2. System operator commands (continued)

Command	Command Abbreviation	Function	Privilege Class
RETURN	RET	To physically connect and make available to TSS/360 the equipment removed by a previous PARTITION command	A
SHUTDOWN	SD	To allow the main system operator to terminate the time-sharing operation	A

Table 3. System administrator commands

Command	Command Abbreviation	Function	Privilege Class
CHANGE CHARGE NUMBER	C CH	To change a user's charge number	B
CHANGE PRIORITY	C P	To change the priority of any user	B
CHANGE PRIVILEGE	C PR	To change a user's privilege	B
JOIN	J	To announce a new user and his initial status to the system	A, B
PRESENT ACCOUNTING	P A	To get accounting information about any user in the system	A, B, C, D
PRESENT CHARGE NUMBER	P CH	To print the charge number(s) of any user in the system	B
PRESENT DATA SET STATUS	P D	To get class, last usage, page count, and type of any data set in the system	A, B, C, D
PRESENT LINE	P L	To present one or more lines from a data set in the system	A, B, C, D
PRESENT PASSWORD	P PA	To get the password of any user in the system	A, B
PRESENT PRIORITY	P P	To get any user's priority	A, B
PRESENT PRIVILEGE	P PR	To have a user's privilege class(es) presented	A, B
QUIT	QU	To deprive a currently joined user of access to the system	B

Table 4. Program checkout subsystem commands

Command	Command Abbreviation	Function	Privilege Class
DISPLAY	D	To present the contents of data fields and/or machine registers	D
DUMP	DU	To dump offline the contents of data fields and/or machine registers	D
QUALIFY	QUAL	To qualify internal symbols of a specified program	D
REMOVE	REM	To remove specific PCS statements that resulted from use of AT phrases	D
SET	S	To change the contents of data locations or machine registers within a program	D
STOP	ST	To stop execution of an object program, return control to the user terminal, and present current location of the object program, as well as program status information	D
AT	AT	To inform the user when specific locations are reached during execution of his object program (when AT is used without a following action verb)	D
IF	IF	Used with a dynamic and/or action verb to form a valid PCS statement	D

TERMINAL AND MAIN OPERATOR FUNCTIONS

The main system operator at the computer center is responsible for control of the computer and its peripheral equipment. Prime responsibility is exercised by the main system operator, who may be assisted in tending input/output devices by subsidiary system operators.

The main system operator is joined (that is, granted access to the system) at system initialization and logged on (that is, has a task created for him) whenever TSS/360 is started up. He is authorized automatically to use all commands in the command language (except those which identify a user to the system and those used to initiate user program execution). The commands reserved exclusively for his use allow him to assign equipment, maintain the operator log, request configuration displays, send messages to other terminals, enter general system information, and respond to system messages. In addition, the main system operator can join and can quit subsidiary system operators who aid in, or perform all, peripheral equipment handling.

System Messages

The time-sharing system issues three types of messages to the user: (1) prompting, (2) response, and (3) confirmation messages. Included within the prompting category are a subclass of diagnostic messages, which identify errors and request corrective action by the user.

All system messages issued to the user are printed on the system output unit (the terminal in conversational mode) unless the user has requested, with one parameter in the LOGON command, that only message numbers be printed. If this option is chosen, only the variable field (for example, parameter in error) and the message code number are printed. A PRESENT LINE command, designating the message code number, can be used to display the text of the message.

Prompting

Under certain conditions, the system prompts the user if he does not supply all parameters for a command. To prompt, it prints out a message asking for the missing parameter. This occurs only during conversational operations. The general conditions for prompting are as follows:

- The system always prompts for a missing mandatory parameter.
- The system prompts for an optional parameter only if the user has previously asked for confirmation.
- The system never prompts for an optional parameter if the user indicates, by commas, that the omission was intentional.
- The system always prompts for the user's identification, password, charge number, and confirmation requirement when he starts a conversational task.
- The system always prompts for the disposition of the user's uncataloged data sets when he ends a conversational task.

An extension of prompting tells the user of any error that he made in entering a parameter. In this case, the system prints a diagnostic message (for example, USERID INVALID. REENTER.), which asks the user to enter the correct parameter.

To signal the user that it is ready to accept the next command or statement, the system prints an underscore beneath the first character position of a new line. This asks the user to enter his next command or statement. However, when the user is entering source language statements to a language processor, a line number is printed at the start of a new line to show that the next line can be entered. One other exception occurs when the user is entering lines for a DATA or MODIFY command. The system then requests new input by printing either a pound sign (#) or a line number.

Confirmation

When the user starts a task, he is asked whether he wishes confirmation. If he requests it, the system issues a confirming message for many commands. (In some cases, such as the PRESENT commands, the action itself confirms the correctness of the command, and no message is required.) The user who requests confirmation is also prompted for all optional parameters in commands, provided he does not show, by commas, that he has intentionally omitted the parameters. Some sample confirmation messages are:

```
(program name) LOADED.
DATADEF  COMMAND  ACCEPTED.
```

The user can change the option that he selected at log on by issuing the CHANGE CONFIRMATION command.

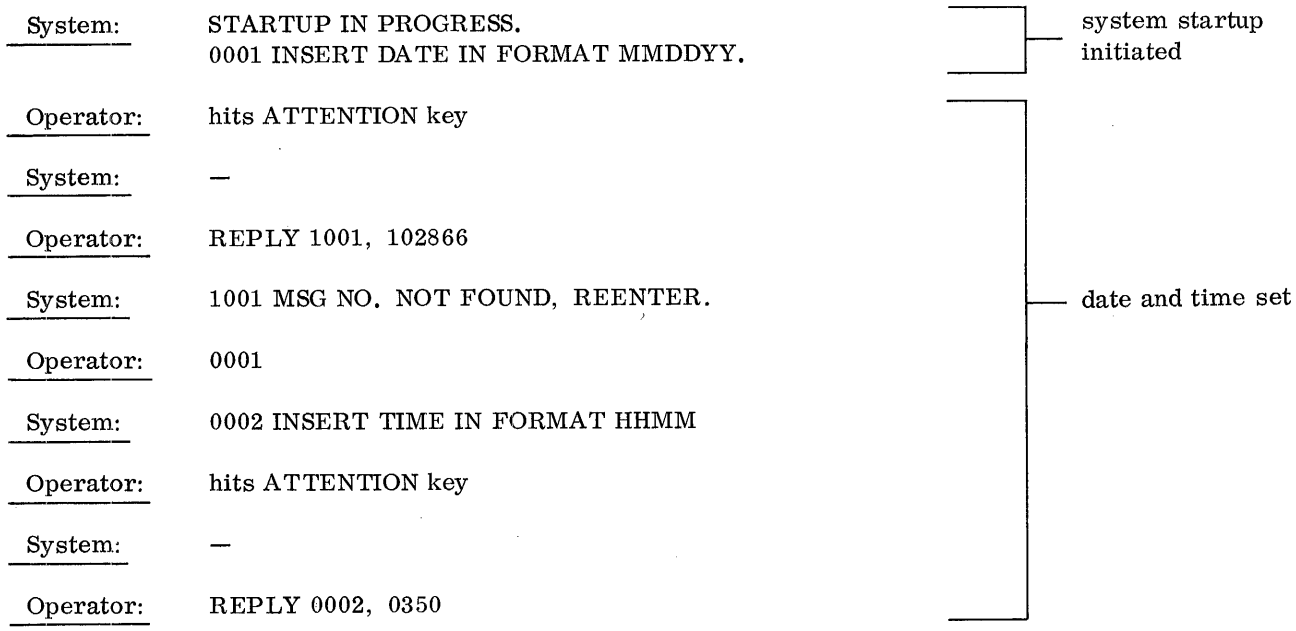
Responses

The system prints out a response message for certain commands to request information from the user or to indicate the result of an action. Responses are similar to confirmation messages, but are always issued. Some typical response messages are:

(data set name) MODIFIED.
LOGON ACCEPTED (date) AT (time).
ENTER MODIFICATIONS.

TSS/360 Main Operator Terminal Session (Example)

A sample main operator terminal session follows. It illustrates the dialogue between the operator and the system from startup to shutdown, showing typical uses of operator commands. Note that the operator must always strike the ATTENTION key before entering any input, except when responding to a diagnostic message requesting reentry of an erroneous parameter. An underscore issued by the system in response to an attention interrupt indicates that the system is ready to receive the operator's input. In the following example the underscores are shown as system output on a separate line from the operator's input. In actual operation, however, the operator types his input in directly over the underscore.



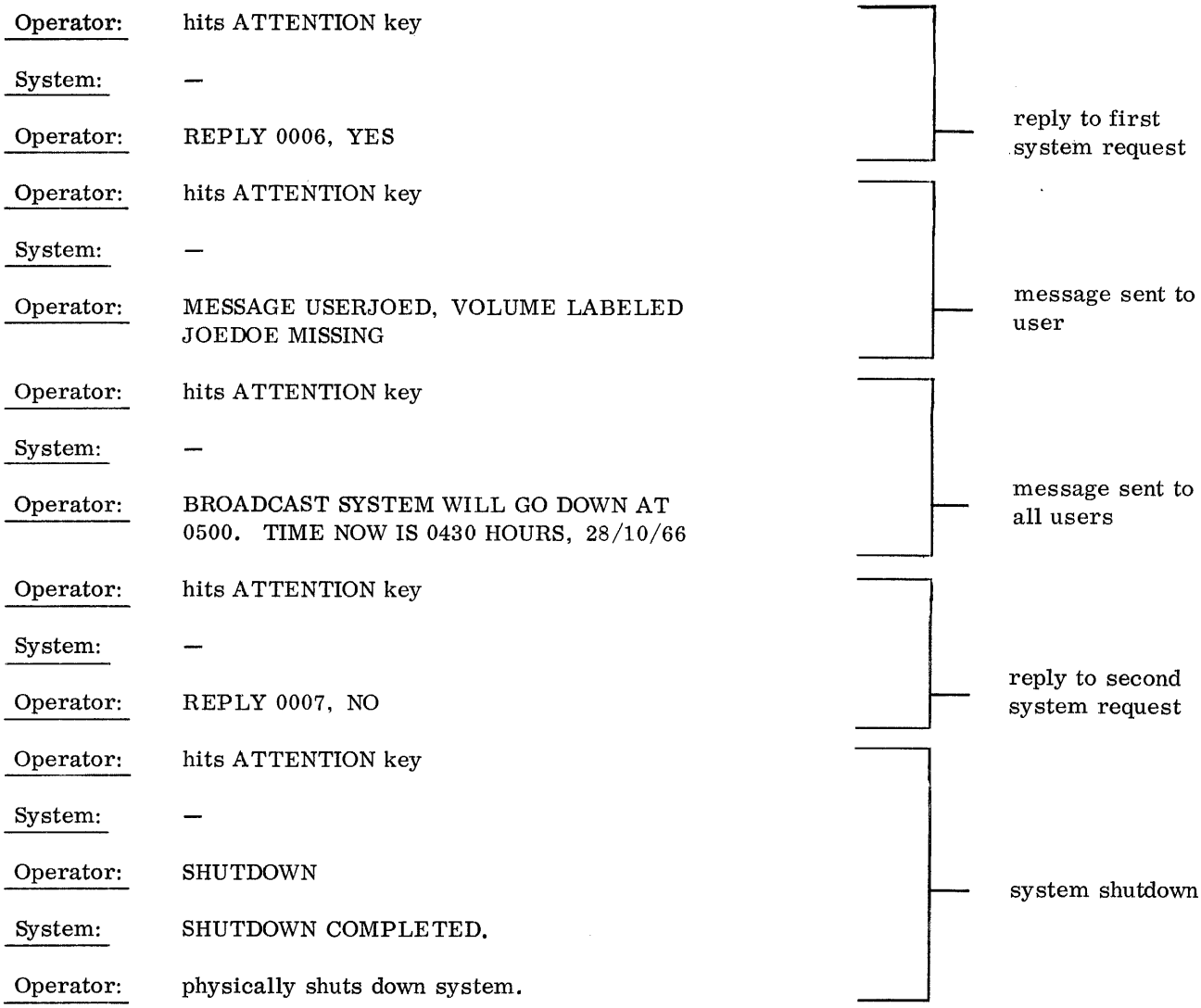
System:	RESOURCES	NAME
	2067.1	CENTRAL PROCESSING UNIT
	2067.2-1, 2, 3, 4	CENTRAL PROCESSING UNIT
	2365.12-1, 2, 4, 5, 8	STORAGE ELEMENT
	2167	CONFIGURATION UNIT
	2846-1, 2, 3, 4	CHANNEL CONTROLLER
	2860.1	SELECTOR CHANNEL
	2860.2	SELECTOR CHANNEL
	2870	MULTIPLEXOR CHANNEL
	2825.5	CONTROL UNIT
	2822	PAPER TAPE READER CONTROL
	2150.1	CONSOLE
	2403.2-1, 3	MAGNETIC TAPE UNIT AND CONTROL
	2702.1	TRANSMISSION CONTROL
	2803.1	TAPE CONTROL
	2816	TAPE SWITCHING UNIT
	2820.1	STORAGE CONTROL
	2841.1-1, 2	DISK STORAGE CONTROL
	1051.1	PRINTER KEYBOARD CONTROL UNIT
	2301	DRUM STORAGE
	2311-1, 2	DISK STORAGE
	2401.1-1, 2, 3	MAGNETIC TAPE UNIT
	2402.3-1, 2, 3, 6	MAGNETIC TAPE UNIT
	1052-1, 2	PRINTER KEYBOARD
	1403.2	PRINTER
	2540.1	CARD READ PUNCH
	2671.1	PAPER TAPE READER
	2741	COMMUNICATIONS TERMINAL
	1316-1, 2	DISK PACK
	2973	DISK SWITCH
	0003 REPLY OK WHEN CONFIGURATION IS CORRECT	

system configuration presented

Operator:	hits ATTENTION key
System:	—
Operator:	DETACH 2067.2-4
System:	0004 SET SWITCH 36 TO OFF. REPLY OK WHEN DONE.
Operator:	sets switch as directed, then hits ATTENTION key
System:	—
Operator:	REPLY 0004, OK
System:	2067.2-4 DETACHED

unit detached

Operator:	hits ATTENTION key	units attached
System:	—	
Operator:	ATTACH 2065.12-6	
System:	0005 SET SWITCH 24 TO ON. REPLY OK WHEN DONE.	
Operator:	sets switch as directed, then hits ATTENTION key	
System:	—	
Operator:	REPLY 0005, OK	
System:	2065.12-6 ATTACHED.	
Operator:	hits ATTENTION key	
System:	—	
Operator:	ATTACH 2402.3-4	
System:	2402.3-4 ATTACHED	
Operator:	hits ATTENTION key	configuration affirmed
System:	—	
Operator:	REPLY 0003, OK	
System:	STARTUP COMPLETED. OTHER TASKS START RUNNING. 0006 MOUNT VOLUME LABELED JOEDOE ON 1316-1. REPLY YES WHEN DONE OR NO IF ACTION CANNOT BE DONE. 0007 ATTENTION REQUIRED BY 1403.2. REPLY YES WHEN DONE OR NO IF ACTION CANNOT BE DONE.	system requests operator action
Operator:	hits ATTENTION key	nonconversational task initiated
System:	—	
Operator:	BULKIO RDCARD	
System:	BULKIO RDCARD COMMAND ACCEPTED AND ASSIGNED BSN 0081.	



TSS/360 User Terminal Session (Example)

This sample terminal session illustrates an assembly of a prestored data set. The assembly is conducted conversationally up to its last segment. The user then issues an attention interrupt, enabling him to create and catalog a data set of prestored commands. He then remotes this command data set, effecting nonconversational resumption and completion of assembly, thus freeing the terminal for a new task.

A description of terms that appear in the session follows.

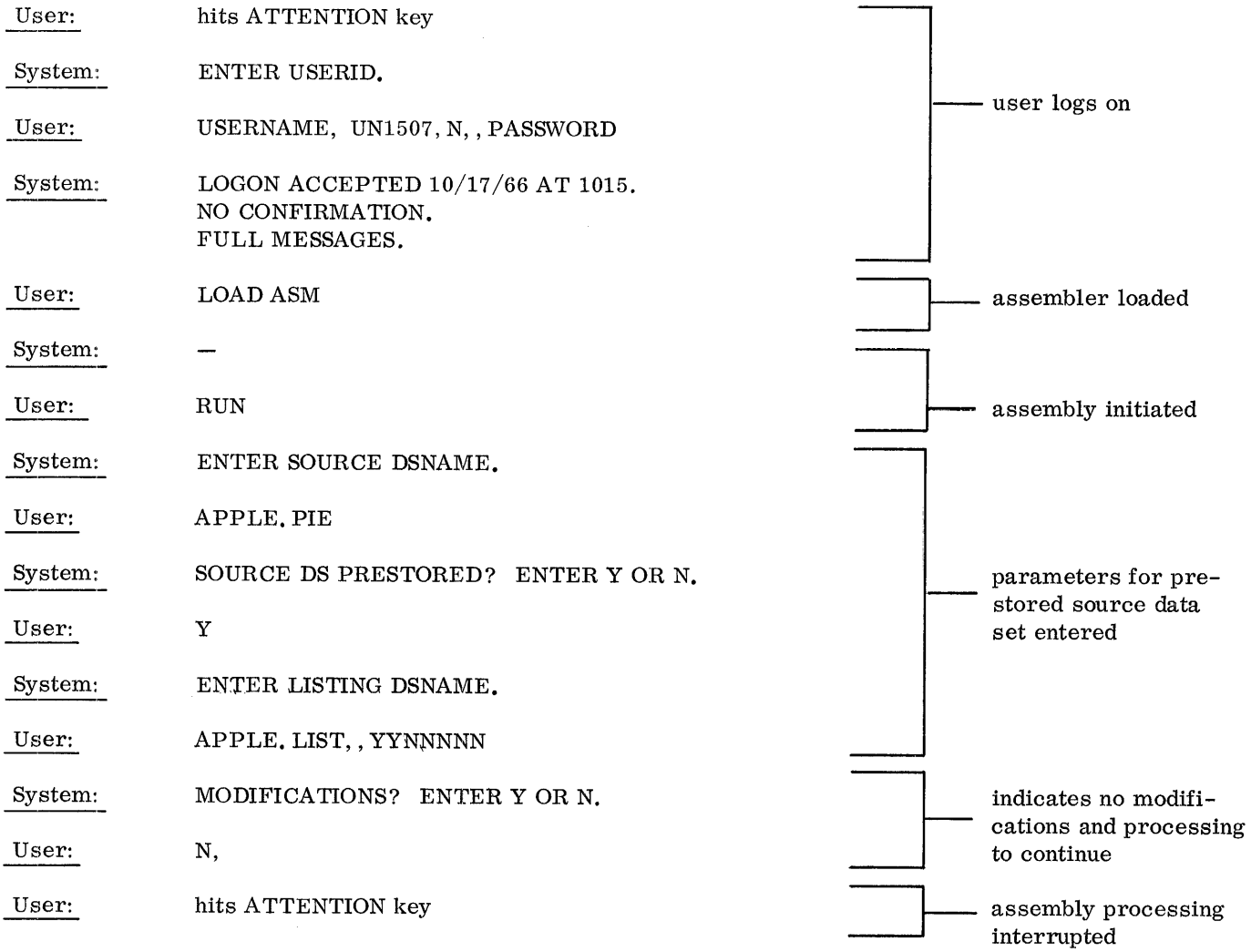
userid — the user's identification consisting of from three to eight alphameric characters, the first of which is alphabetic (alphameric refers to the letters A through Z and the numerals 0 through 9; blanks and special characters are not included). Each userid must be unique within the system.

password — a code word containing from one to eight nonblank characters. Alphabetic, numeric, and special characters except for (), and tab are valid. The password validates the user to the system.

charge number — the user's account number, consisting of from one to eight alphameric characters.

priority — a one-digit numeric code from zero to nine. Zero is the lowest priority, nine the highest.

prompting	messages issued by the system, to which the user must respond. Prompting is issued for any mandatory parameter that is omitted. Prompting messages for optional parameters are issued only if the user has requested confirmation.
confirmation	messages informing the user of the action taken by the system in response to a command or command parameter. Confirmation messages are supplied only if the user has requested confirmation.
response	messages that inform the user of the system action taken for a command or command parameter. Response messages are issued without regard for the user's confirmation option.



System:	—	command data set created
User:	DATA APPLE. CONTINUE	
System:	#	
User:	RUN	
System:	#	
User:	N, N (To answer prompts about cataloging source and listing data sets)	
System:	#	
User:	LOGOFF	
System:	#	
User:	% END	
System:	—	command data set cataloged
User:	CATALOG APPLE. CONTINUE	
System:	—	
User:	REMOTE APPLE. CONTINUE	execution resumed nonconversationally
System:	TASK RUNNING NONCONVERSATIONALLY. ASSIGNED BSN 0359.	

SYSTEM ADMINISTRATION AND ACCOUNTING

Access to TSS/360 is controlled by system administrators, who can authorize or deny such access to the users for whom they are responsible. Each system administrator is joined (that is, granted access to the system) by the main system administrator. The administrators, in turn, can join users by identifying them to the system, assigning each a password, a user ID, privilege classes (up to but not including administrator level), a priority. and charge numbers. To assure accountability, the system prefixes a user's ID and charge numbers with a unique code derived from his administrator's ID. Once he is joined by an administrator, a user can access TSS/360 freely until his authorization is revoked by his administrator.

The system maintains accounting statistics from which charges can be derived for each TSS/360 user. To some extent, the user can study his statistics to learn more effective ways of employing the system. The PRESENT ACCOUNTING command enables the user to get all accounting statistics for each charge number assigned to him.

Accounting statistics fall into two main categories — static and dynamic. Static statistics apply to the storage of cataloged data sets; dynamic statistics deal with the time the user actually works with the TSS/360.

Static statistics are kept for every cataloged data set residing on public storage in the system. These statistics are a function of the type of storage used for the data set, the data set page length, and the number of days the data set is cataloged. Such statistics are accumulated under the charge number assigned to the user when he created the data set.

Dynamic statistics are accumulated while the user is logged on the system. When the user logs on, a table is set up for his dynamic statistics, and is continually updated as long as the user remains on the system. At log off, the table is again updated to show total times, and the dynamic statistics are transferred to a master set of statistics accumulated for the user under his current charge number. Dynamic statistics include:

- Total CPU time used.
- Accumulated time for each reserved I/O unit. This information reflects both the length of time that an I/O unit was reserved for the user and the type of unit reserved. Type refers to the terminal model, tape drives, disk pack, printer, punch, etc.
- Accumulated channel usage. This indicates the number of channel accesses for I/O made during the time that the user is active on the system.
- Accumulated time for active pages in storage during the user's time slice. This reflects the actual number of pages of storage required during each time slice and is thus the user's total "page - time slice" usage.
- Accumulated time for pages in virtual memory. This accounts for the time that a page is part of the task's virtual memory.

All accounting statistics are accumulated for a user identification and for the charge number associated with that identification. Thus, a main account (that is, a charge number) can have subaccounts, each identified by the charge number that is linked with a user identification.

It is often convenient to write a large program in sections. The sections may be assembled or compiled separately, then combined later into one executable object program module. The TSS/360 language processors provide facilities for creating multisectioned programs and symbolically referencing separately compiled programs or program sections.

The concept of program sectioning is a consideration at coding time, compile time, and load time. To the programmer, a program is a logical unit. He may want to structure it into sections commonly called control sections; if so, he writes it in such a way that control passes properly from one control section to another, regardless of the relative position of the control sections in storage.

In writing a TSS/360 program, the basic unit of structure is a control section comprised of coding, the virtual memory location assignments of which can be adjusted, independently of other coding, at linkage or load time without altering or impairing the operating logic of the program. Each control section is assigned at least one page (4096 bytes) of virtual memory and an origin at a page boundary by the dynamic loader routine. The assigning of a page(s) to control sections facilitates the linking to, and loading of, control sections, since a page is the smallest piece of information that can be relocated independently within the time-sharing environment.

A TSS/360 program can be sectioned using three types of control sections, one of which is called simply "control section" (CSECT), and the other two, "prototype control section" (PSECT) and "common control section" (COM). Regardless of the degree to which a program is sectioned, the programmer knows the elements that constitute his virtual memory, because he has described them symbolically. He cannot, however, make any assumptions about the position or ordering of control sections, since their virtual memory location assignments may have been adjusted by the linkage editor and/or the dynamic loader routines, and their core storage addresses may be changing constantly with the time-sharing environment.

In the following discussions, the term "control section" is used inclusively for all three types of control sections (CSECT, PSECT, and COM), unless specified otherwise.

LANGUAGE PROCESSING

Processing by the language processors — assembler and FORTRAN compiler — primarily involves the translation of source statements into executable machine language. The output from the language processors is an object program module, a machine language equivalent of the source program. The processors also produce a printed listing of the source statements and object statements, and additional information useful to the programmer in analyzing his program. The object program module consists of one or more control sections and a control dictionary. The control dictionary contains information that the linkage editor and the dynamic loader routines require to complete cross-referencing between control sections.

The name of an object module produced by the assembler consists of the first eight characters of the list data set name supplied by the user, while the name of the program module produced by the FORTRAN compiler consists of the first six characters of the list data set name supplied by the user. This program module is divided into three identifiable subsets: the program module dictionary (PMD), the text module (TXT), and the internal symbol dictionary (ISD). A brief summary is provided below:

1. The PMD comprises a heading that contains the standard entry point to the module and other information common to the whole module, together with one control section dictionary (CSD) for each control section in the module. Each CSD contains information regarding the external symbol definitions and references, relocation pointers, and paging information — in short, all information required to produce from the text a fully linked, executable, object module.

2. The text module contains the pure binary text of instructions and constants generated by the assembler or compiler. This is acted on and allocated to virtual memory by the information in the CSD. The text for each control section in the module begins on the page boundary in the text module.

3. The internal symbol dictionary is provided optionally for use with the program checkout subsystem.

CONTROL SECTION GENERATION

Control sections are generated in response to four specific directives (CSECT, PSECT, COM, START). They may be assigned names. In default of a specific directive, one unnamed control section per module is generated.

Control sections are generated by the FORTRAN compiler according to the following rules:

1. One fixed-length, reenterable, read-only section is generated to contain the instructions and constants.
2. One fixed-length prototype section is generated to contain variables.
3. One common section is generated for each common block declared in the source code.

The main entry point and the control sections generated for instructions and variable are named by affixing standard suffixes either to the FORTRAN names of the routines (in the case of subroutines) or to the program module name. The common sections are named from the declared common block names. The suffixes are:

1. Main entry point: #E
2. PSECT: #P
3. CSECT: #C

CONTROL SECTION ATTRIBUTES

To facilitate dynamic linkage and loading within the time-sharing system, it is often necessary to indicate that certain attributes are characteristic of the data or instructions within a control section. One or more of the following operands may be used in CSECT, PSECT, COM, and START directives to indicate which attributes are to be assigned to the section. Attributes may be specified singly or in combination, where meaningful.

- Variable-length. This is a section of indeterminate length. At load time, each such section is allocated 20 pages.
- Read only. This attribute specifies that the control section may not be stored into; it causes memory protection.
- Public. This attribute specifies a control section that may be shared by several users.
- Prototype control. This is a special control section used by reenterable routines for communication and temporary storage. A separate copy of the prototype section is provided to each user of a public reenterable routine.
- Common. This is a control section common to all modules in which it is declared.
- Privileged. This is a control section eligible to be classed as a privileged system program when placed into the TSS/360 System Library (SYSLIB).

CSECT — CONTROL SECTION

The CSECT identifies the beginning of a control section. If a symbol names the CSECT instruction, the symbol is established as the name of the control section; otherwise, the section is considered to be unnamed. All statements following the CSECT are assembled as part of that control section until a statement identifying a different control section is encountered. The text of each control section starts on a page boundary, and a virtual memory page table is constructed as the text is produced.

PSECT — PROTOTYPE CONTROL SECTION

Within the time-sharing system, a single copy of a commonly used reenterable routine appears to have different virtual memory location assignments to different users, although its physical disposition in storage remains unchanged. When control is transferred to a reenterable routine, the calling task must supply an address constant that reflects the virtual memory assignments of the calling program so that the reenterable routine may obtain data storage unique to the user. Ordinarily, this would imply that a program which calls a reenterable routine should be knowledgeable about all address constants that might be required within the hierarchy of reenterable programs. To minimize this clerical burden, a prototype control section is defined for use by reenterable programs to simplify the handling of address constants and working storage.

On loading of the reenterable routine, a copy of the contents of all nonpublic sections (prototype sections) is made and assigned virtual memory locations within the domain of the calling program.

In a reenterable program, all working storage and address constants are assembled within a prototype section or within some other nonpublic, nonread-only control section; the user need not know any of the internal requirements of the routine he calls.

Communication of prototype section information is accomplished through the use of an R-type address constant. This supplies the location of the control section, which is required by the reenterable program for working storage and address constants unique to the calling program. Use of the R-type address constant causes the loader to supply the location of the section in which the entry point name was declared.

COM — COMMON CONTROL SECTION

The COM directive identifies and reserves common areas of storage that may be referred to by independent assemblies or compilations that have been

linked and/or loaded for execution as one overall program.

One blank common section and any number of named common control sections can be designated in an assembly or compilation.

No instructions or constants are assembled in the blank common control section. Data can be placed there only through execution of the program. Instructions and constants, however, can be assembled in named common control sections.

R-TYPE ADDRESS CONSTANT

The R-type address constant is used to link to reenterable modules. It supplies the location of the control section in which the entry point was declared (usually a PSECT). It is not necessary for the programmer to know the name given to the control section by the reenterable module, since he needs only to know the desired entry point within the reenterable module. Use of the R-type address constant causes the dynamic loader to supply the location of the control section as a function of the entry point name.

V-TYPE ADDRESS CONSTANT

A V-type address constant is used to reserve storage for the address of an external symbol that is used for effecting branches to other modules. The constant may not be used for external data references; it is specified as one relocatable symbol that does not require identification in an EXTRN statement. Whatever symbol is used is assumed to be an external symbol, since it is supplied in a V-type address constant.

REENTERABLE ROUTINES

As previously stated, each task in the time-sharing system is protected by virtual memory addressing. There still remains the problem of shared routines — that is, routines addressable by two or more tasks simultaneously. For this sharing to be feasible, such routines must be incapable of alteration by either task — either through referencing or through execution.

In addition, to minimize paging, it is desirable that as many pages of a task as possible remain unchanged by execution. This arises because an unchanged page in core storage does not require copying out to auxiliary storage at the end of the current task's time slice, since a fresh copy of the original page suffices for future needs.

Reentrancy permits both of these needs to be satisfied. A reenterable routine is one that meets several criteria:

1. It must not be alterable by any user task. This is achieved by making it read only through the use of the equipment storage protection keys.
2. It must not be altered by loading or relocation. This is achieved through forbidding the inclusion of any relocatable address constants within the reenterable routine.
3. It must not modify itself when executed. This is achieved by the programmer who codes the routine and is monitored by the equipment storage protection keys.

A routine that meets all these criteria would itself be capable only of operating on the general registers of the System/360 and in locations whose addresses were passed in a general register as a parameter. This restriction is avoided without compromising the above requirements by separating all the data that is modified by the relocation or operation of the reenterable routine into a prototype control section. This includes parameters, temporary storage, and the relocatable address constants (Adcons) required to refer to and cover the reenterable routine. The location of the prototype control section is furnished to the reenterable routine in the SAVE area of the calling routine at the time the routine is called, and is referenced by the routine loading a base register. In general, addressability (base register coverage) of the PSECT must be maintained throughout the execution of a reenterable routine.

Protection of the sharing tasks against interference, each from the other, is provided by giving each sharing task a separate copy of the PSECT. Neither task's copy is addressable by the other, and each task's copy is addressable by the reenterable routine only while it is under control of that task.

Minimization of paging is achieved by the concentration of changeable items into PSECT's, so that only the pages of the PSECT require recopying when paged out.

CODING CONVENTIONS — REENTERABLE ROUTINES

FORTRAN-compiled programs are generated as reenterable modules (with PSECT's) by the compiler. However, the programmer who wishes to write a reenterable routine in assembler language has several responsibilities:

- Entry and exit must be in accordance with the system linkage conventions. A save area must be provided if any other subroutines is to be called.
- Both a reenterable code control section (CSECT) and a prototype section (PSECT) must be written and assembled together unless the routine contains no relocatable Adcons and requires no local storage of its own.

- All Adcons, temporary storage, program switches, or other data that may be relocated or modified by execution must be written in the PSECT or in some other nonpublic, nonread-only control section.
- The ENTRY statement that defines the entry point(s) to the reenterable routine must also be written in the PSECT or in some other nonpublic, nonread-only control section. The label that is made an entry point is defined in the control section. This convention permits the resolution of the R-type address constants in referring to the routine.
- Care must be taken that no byte within the CSECT is in any way subject to modification by loading or execution of the code.
- The reenterable and read-only attributes must be assigned to the CSECT. If the routine is to be shared, the PUBLIC attribute must also be assigned.

LINKAGE CONVENTIONS

Linkage conventions are those conventions which govern communication among programs. Basically, there are two types of communication: the type that involves the supervisor (supervisor-assisted linkages are characterized by an SVC) and the type that is independent of the supervisor.

There are two classes of modules that operate in virtual memory — privileged and nonprivileged. All language processors and most user modules are nonprivileged. Generally, privileged modules require some special capability, such as access to tables internal to the supervisor or the ability to give privileged SVC's. Using the definition of privileged and nonprivileged modules, and understanding that any module must necessarily belong to one of these classes, the following three types of linkage conventions have been developed:

Type I — Between two nonprivileged or between two privileged modules

Type II — From a nonprivileged to a privileged module

Type III — From a privileged to a nonprivileged module

Type I linkages are independent of the supervisor, whereas type II and type III linkages are supervisor-assisted. The design is such that the called module does not know the type of linkage involved (that is, whether the calling module is privileged). Macro instructions are provided in the system macro library, which provides the appropriate linkage.

In the TSS/360, all linkage among modules residing in virtual memory conforms to the conventions outlined below.

Conventions for Type I, Type II, and Type III Linkages

General Register	Usage
15, 0	Supervisor parameter registers
1	Parameter list register, supervisor parameter register
13	Save area register
14	Return register
15	Entry point register, return code register

It is the responsibility of the called module to maintain the integrity of general registers 2-12, so that their contents are the same at exit as they were at entry to the called module. It is the responsibility of the calling module to maintain the floating-point registers around a call. General registers 0, 1 and 13-15 must conform to the previously stated conventions.

Save Area

In any type I linkage, whenever one module calls another, the calling module provides a save area for use by the called module; the calling module is known as the owner of the save area. This save area is addressed by the save area register on entry to a called module and has the following format:

1. Word 1 — contains the length of the save area and any appendages to it in bytes. This field is set by the calling module in its own save area.
2. Word 2 — contains a pointer to the save area of the calling module. This field is set by the called module in its own save area. This allows all save areas of active modules to be linked in a reverse chain.
3. Word 3 — contains a pointer to the save area of a called module after its invocation. This field is set by the called module in the calling module's save area. This allows all save areas of active programs to be linked in a forward chain. When the called module is complete, if trace forward has been specified, it sets the low-order bit of this field to 1, to stop the forward chain.
4. Word 4 — contains the return linkage for use by the called module when it is complete. This field is set by the called module in the calling program's save area.

5. Word 5 — contains the entry point address to the called module. This field is set by the called module in the calling module's save area.

6. Word 6-18, register save area — these fields are set by the called module in the calling module's save area, as necessary, to preserve registers 0-12.

7. Word 19 — contains the address of the PSECT belonging to the called module. This field is set by the calling module in its own save area.

A module may use its own PSECT for a save area provided that the head of the PSECT is formatted as indicated above. However, a called module should never use its own PSECT to save registers under any circumstances. The calling module always provides save area locations.

Various linkage techniques are provided in macro form in TSS/360, including:

CALL — both implicit and explicit linkage (see below)
LOAD — explicit load of a module by an external name
SAVE — save registers (used normally at entry points)
RETURN — restore registers
DELETE — explicit unlinking of a module

Implicit Linkage

The use of a V-type or an R-type address constant to refer to an external symbol constitutes an implicit linkage call. When a previously undefined external symbol is referred to, the loader is invoked by a relocation interrupt, and performs the necessary action to link together the referenced and referencing module. This action is automatic and requires nothing of the user beyond specification of external symbols and Adcon types, as required by the assembler.

Explicit Linkage

In certain types of programming, situations arise when a given run of a module may cause only one of a number of independent subroutines to be required. Since dependence on normal implicit linkage requires the presence of Adcons in the calling module for all such subroutines, some unnecessary overhead is experienced in preparing the unused ones for linking.

To allow for this situation, two explicit linkage functions are provided. These cause the desired subroutine module to be retrieved and linked upon the execution of an explicit dynamic LOAD or CALL macro.

Use in FORTRAN

If a call is to be made to an explicitly-linked subroutine, an implicit call is made to a library load subroutine, which takes the first parameter in the calling sequence as the name of the subroutine to be linked and executes a LOAD macro, after which it executes a normal FORTRAN call to the newly-linked routine, using the remainder of the original parameter list.

If a parameter to be passed is the address of another subroutine, a pointer to two Adcons (V-type and R-type) is passed instead of passing both Adcons.

Explicit Unlinking

Occasions may arise when a user wishes to dispense with a linked module because it is no longer needed, and he wishes to free virtual memory space. This facility is provided by the DELETE macro. The unlinking process consists of the following steps:

1. Deletion of the dictionary for the unlinked module, removal of its external references from their use chains, maintenance of memory map, and explicit call chain tabulations
2. Release of pages and page tables relevant to the unlinked module
3. Deletion of any modules (as above) that, as a result of unlinking this module, have become superfluous to the allocation. Specifically, deletion of a module occurs if it no longer has either:
   a. Any explicit callers
   b. Any users of its external definitions

PROGRAM TYPES, SYSTEM SYMBOLS, AND EXTERNAL NAME CONVENTIONS

The TSS/360 system provides user facilities to link to many of the system service routines, such as the I/O access method entry points. Most references to these system entry points are provided in the TSS/360 macro library, and the user need not actually use the external symbols that the system has defined. However, as system facilities are invoked, the system-defined external symbols contained therein are added to the task dictionary for that user.

The following discussion describes the types of external symbols that the system defines and that are hence restricted from user definition, and the types of external symbols that the system has defined for user referability.

Logically, the TSS/360 operating system contains and handles three types of modules:

1. Resident supervisor module operating with core storage addresses
2. System modules operating in virtual memory with privileged status; that is, modules possessing a greater freedom than user modules in the use of other system modules and resident supervisor services. The I/O access method routines are examples of such system routines.
3. User modules — there are two types:
   a. IBM-provided
   b. User-written

The linking of system modules together with IBM-provided and user-written modules introduces the external symbols defined by the TSS/360 operating system modules into user referability. The IBM-defined external symbols are:

1. User-referable system symbols. These are all of the form SYSxxxxx. A user module may not define such a symbol.
2. User-referable system function symbols. A user module may define such a symbol.

All system routine modules, control sections, prototype sections, and entry point names start with the alphabetic letter C. User-written modules must avoid the use of external symbols starting with the letter C.

SYMBOLIC LINKAGES

Symbols may be defined in one module and referred to in another, thus effecting symbolic linkages between independently assembled modules. The linkages can be effected only if the language processor is able to provide information about the linkage symbols to the dynamic loader, which resolves these linkage references at load time. The assembler or compiler places the necessary information in the control dictionary on the basis of the linkage symbols identified by the ENTRY and EXTRN statements. Note that these symbolic linkages describe linkages between independently compiled control sections.

In the program where the linkage symbol is defined (that is, used as a name), it must also be identified to the language processor by an ENTRY statement. It is identified as a symbol that names an entry point; this means that another module will use that symbol to effect a branch operation or a data reference. This information is placed in the control dictionary.

Similarly, the module that uses a symbol defined in some other module must identify it by an EXTRN statement or a V-type address constant reference. It is identified as an externally defined symbol (that is, defined in another module) that is used to effect linkage to the point of definition. This kind of information is also placed in the control dictionary.

PROTECTION CLASSES

Since the supervisor programs are protected by virtue of nonaddressability, the service routines and the user's own routines are the only ones with which he is concerned regarding protection. Certain service routines are not accessible to the user directly. These routines are assigned a storage class of C, which indicates that the user module may neither write into them nor refer to them with a read or branch. His only communication with these routines is through supervisor calls, which provide a monitored and protected entry to the service routines.

Another category of service routines, for example, FORTRAN library subroutines, are given a storage class of B. This permits the user to read and transfer to them, but not to write into them. The user may provide this level of protection to his own routines through specification of the read-only attribute on the control section. The code CSECT generated by the FORTRAN compiler is protected automatically in this way. The lowest category of protection, used for user routines that do not have the read-only attribute, is storage class A. Areas within this class may be freely read and written by user routines.

Good coding practice dictates that read-only protection be used whenever there is not a specific reason for storing into a section of a program. In addition, it must be used for reenterable or public routines.

LINKAGE EDITOR AND DYNAMIC LOADER

The linkage editor and the dynamic loader can take control sections from various assemblies and compilations and combine them properly with the help of the corresponding control section dictionaries. Successful combination of separately produced control sections depends on the techniques used to provide symbolic linkages between the control sections.

Linkage Editor

The linkage editor is a service program, used optionally in association with the language processors.

The output of a single compilation or assembly is called an object program module or simply a module. Normally, this single module is input directly to the dynamic loader, and all other modules necessary to form a task are dynamically linked together. However, the user may find it desirable to process one or more modules through the linkage editor before employing the loader.

The linkage editor can perform the following functions, as specified by the user:

- Two or more modules, created by the same or by different language processors, can be statically linked together to form one program module. A statically linked module requires somewhat less loader processing time and makes more efficient use of virtual memory (at the cost of linkage editor processing time).
- Control sections within modules can be replaced, deleted, or renamed.
- Entry names and external references can be renamed.
- Entry names can be deleted.
- A new module entry point (transfer point) can be defined.
- The attributes of control sections can be changed.
- Two or more control sections can be combined into a single control section.

Processing by the linkage editor is governed by statements arriving from a remote terminal device (conversational mode) or from the primary input device (nonconversational mode). Users at the terminals can correct control statement errors detected by the linkage editor.

Subroutines from a user library can be included in the output module at specified points. However, if it is desirable that this type of routine be included dynamically at execution time, a library statement to the linkage editor creates tables for the dynamic loader, facilitating the loading of these routines at execution time. Thus, user libraries need not contain standard system subroutines; if the routine is sharable, it is possible that the copy linked to is already in core storage.

Output from the linkage editor is the same as the output from any language processor and is in a form processable by the dynamic loader. The output module consists of three parts: the dictionary, the text, and, optionally, the internal symbol dictionary. The dictionary contains all the information necessary to process and load the text (including information about all control sections within the module), relocation information, and the initial entry point to the program. Also supplied is information pertaining to user library requirements and the total blank COMMON requirements of the output module. The output module from the linkage editor can be executed or processed again as input to the linkage editor.

Dynamic Loader

The dynamic loader is one of a subset of service routines that is initialized into the user's virtual memory at LOGON time. It is available for use by other system routines and by the user, in either conversational or nonconversational mode. With this facility the user need not use the linkage editor's services between creation of program modules by a language processor and execution of the program (task). This is useful especially when a given run of a module may cause only one of a number of independent routines to be required. That particular routine may then be explicitly called at execution time, eliminating the requirement for linking the unused routines.

The dynamic loader allocates virtual memory and links external symbols among various separately assembled or compiled program modules on a dynamic basis. To the dynamic loader, there are no programs — only control sections that must be fashioned into an object task. The dynamic loader allocates virtual memory as follows:

- CSECT's of specified length are allocated a fixed number of pages.
- PSECT's are handled similarly to CSECT's. However, when the PSECT is associated with a shared public routine, its external page table is saved, and a fresh copy is made for each user of the routine.
- COM control sections are allocated a variable number of pages, depending on the longest length declared in any module already loaded.

The dynamic loader consists of three independent routines. The first, the explicit linkage routine, is called by the user through a supervisor call (SVC). The user provides a name as input to the routine. The loader finds the module, and if the module has not already been processed, information about the module is placed in the loader tables, virtual memory addresses are assigned to its various control sections, and page tables are constructed for the module. If any relocation is necessary, the pages are marked "unavailable" and "unprocessed by loader".

The implicit linkage routine is called when a page-unavailable interrupt occurs due to a user program reference to a page for which the loader has constructed a page table entry but has not performed the relocation of address constants. This routine relocates the address constants in the

referenced page. External references to new modules from the page cause the loader to "link" them in a manner similar to that described above for an explicit linkage call.

Another facility available is the unlinking of modules. The explicit unlinkage routine is entered through an SVC, which causes the loader to delete all table entries for the module specified by name.

Later linkages may be caused either by implicit references from the linked module or by explicit calls. Implicit and explicit linkage are further described below.

When a symbol within a module is referred to initially, either explicitly or implicitly, the following action occurs in the dynamic loader:

1. The PMD for the module containing the symbol definition is obtained from the library and inserted in a task dictionary.

2. The external page table for the module is retrieved from the library; from this, and from the virtual memory tables within the CSD, a new page table entry is constructed for each control section. These pages are assigned virtual memory addresses, and a memory map is constructed with an entry for each control section.

3. All pages for the new sections are marked "unavailable". Those which require relocation of Adcons within the test are marked "unprocessed by loader".

4. When a page-unavailable interrupt is detected by the paging supervisor, the page is retrieved from auxiliary storage and placed in the user's virtual memory. If the unprocessed bit is set, the loader is provided with the virtual memory address of the retrieved page.

5. The loader then locates (through the memory map) the dictionary for the control section that includes the referred-to page.

6. All test modifiers for this page are executed, thus relocating all Adcons in that page. Any new symbols referred to by the page cause this cycle to be started over at step 1 above.

The management of data in a time-sharing system poses many problems that either do not exist or can be ignored with less severe consequences in a more conventional batch processing system. All data in the entire installation must be available to each terminal user within a "reasonable" time span; this capability should not be restricted by the data management system. Any restrictions on the unlimited accessibility of data should be imposed only by the users themselves, for security reasons, for example. The size of the data sets may require large amounts of storage online and accessibility within a matter of milliseconds. IBM supplies this magnitude of storage in a wide range of devices having large capacity and various degrees of accessibility.

The data management function of Time-Sharing System/360 assists programmers in achieving maximum efficiency in managing the mass of data and the many programs that are processed in an installation. To attain this objective, data management facilities have been designed that provide systematic and effective means of classifying, identifying, storing, cataloging, and retrieving all data (including loadable programs) processed by the TSS/360.

All data in the Time-Sharing System/360 is maintained in the form of data sets. A data set is a named, organized collection of one or more related records. Information in data sets is not restricted to a specific type, purpose, or storage medium.

The facilities provided can be grouped into three major categories: data set control, data access, and storage allocation.

DATA SET CONTROL

This section describes the data set control facilities of TSS/360. Data set identification, cataloging, security, storage space allocation, and data definition procedures are described. Information is also provided on label and record formats.

TSS/360 provides a comprehensive group of facilities that feature automatic and efficient control of many data processing operations performed previously by programming personnel as clerical tasks. A number of these control facilities are described in the following paragraphs.

Data set location control, supported by an extensive cataloging system, enables programmers to retrieve data and program modules by symbolic name alone, without specifying device or volume serial number. In freeing computing personnel from the necessity of maintaining involved volume serial number inventory lists of data and programs stored with the system, the catalog reduces manual intervention and its concomitant human error.

Another major facility of the cataloging system enables programmers to classify successive generations (updates) of related data. These generations can be referred to relative to the current generation. The system automatically maintains a list of the most recent generations that have been produced.

Control of confidential data is a recurring managerial problem that is solved by the data set protection facility of TSS/360. Using this facility, a programmer can prevent unauthorized access to data sets requiring protection. Unauthorized attempts to gain access to such data sets result in an abnormal termination.

In addition to data set location control and protection, control of direct access storage space allocation is provided by the system. This control facility frees programmers of the details involved in allocating direct access storage space to a data set. The programmer need specify only the amount of space required and, optionally, the storage device required, and space is allocated accordingly.

The system, as well as providing a wide range of control facilities, permits the programmer to organize data sets and individual records in a variety of standard formats that can be selected to meet specific data processing needs.

DATA ACCESS

The data access facilities provided by the operating system are a major expansion of the input/output control systems (IOCS) of previous operating systems.

Input/output routines are provided to schedule and control the transfer of data between core storage and input/output devices. Routines are available to perform the following functions:

- Read data
- Write data
- Block and deblock records
- Overlap reading/writing and processing operations
- Read, verify, and write volume and data set labels
- Reposition volumes automatically
- Detect error conditions, and correct them when possible
- Provide exits to user-written error and label routines

Flexibility has been a major design principle in the system's data access facilities. The program-

mer and the installation can select from several methods of data access to obtain a group of facilities tailored to their processing requirements. Each access method supplies a comprehensive group of macro instructions that permit the programmer to specify input/output requests with a minimum of effort; the programmer need not be concerned with learning the individual access characteristics of the many input/output devices supported by the system.

The following access methods are available to the user of TSS/360:

1. SAM — Sequential access method
2. VAM — Virtual access method
3. TAM — Terminal access method
4. GAM — Graphic access method

Within VAM, various data organizational methods are provided to assist the user in organizing data.

STORAGE ALLOCATION

Data storage within the proposed time-sharing system is organized into a hierarchy that might be described as constituting a triangle of variable dimensions (see Figure 9).

Storage media closer to the base of the triangle store data at a lower unit cost but require longer access time. Storage media closer to the apex permit fast access at a higher cost per unit. The total area of the triangle corresponds to the total data capacity of the system. Response time can be improved by increasing the number and capacity of the devices with faster accessibility. The design of the data management system permits this flexibility without requiring recompilation of programs or reformatting of data.

The vertical data set flow is triggered automatically whenever idle time is present on either CPU, and core space and channel time are available. If any device has a data set density above an installation-designated normal level, a task is created to reduce its content. The activity of each data set can be determined as a function of two sets of data:

1. • time in the system
   • total number of calls, c
2. • time of last call, t_n
   • time of next-to-last call, $t_n - 1$
   • time of second-to-last call, $t_n - 2$

Each of these sets can be used to calculate an "activity" rate (r_1 and r_2), as follows:

$$r_1 = \frac{c}{\text{time in system}}$$

$$r_2 = \left[\alpha \frac{1}{t_n - t_n - 1} + (1 - \alpha) \frac{1}{t_n - 1 - t_n - 2}\right] \Big/ 2$$

These can be combined to give a usage rate, r, in the following manner:

$$r = \left[\beta r_2 + (1 - \beta)\, r_1\right] \Big/ 2$$

α and β are parameters that can be set by the installation to adjust the responsiveness of the vertical data set movement to variations in activity. When an imbalance is indicated, based on usage rate, a task can be invoked by the system that schedules an interchange from device to device, as space permits, moving higher-rate data sets from devices lower in the hierarchy and exchanging them with lower-rate data sets.

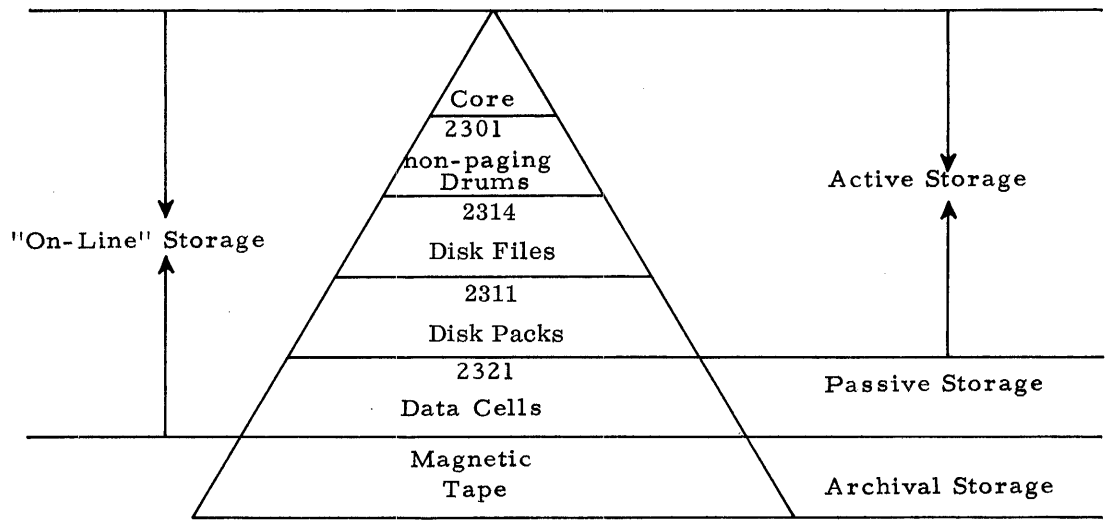


Figure 9. Storage allocation

DATA SET NAMES

The name of a data set identifies a group of records as a data set. All data sets are recognized by name alone (that is, referable without volume identification), and all data sets residing on a given volume must be distinguished from one another by unique names. To assist in this, the system provides a means of qualifying data set names.

A data set name is a series of one or more simple names joined together so that each represents a level of qualification. For example, the data set name DEPT999.SMITH.DATA3 is composed of three simple names that are delimited to indicate a hierarchy of categories. Starting from the left, each simple name is a category within which the next simple name is a unique subcategory.

Every simple name consists of from one to eight alphameric characters, the first of which must be alphabetic. The special character period (.) separates simple names from each other. Including all simple names and periods, the length of a data set name must not exceed 44 characters. Thus, a maximum of 22 qualification levels is possible for a data set name. As the TSS system adds an eight-character user ID qualifier to all user names, the user is restricted to 35 characters.

To specify the use of a particular data set by a problem program, the programmer denotes the data set name and other pertinent information (for example, volume number) in a command language statement called the data definition (DATADEF) statement.

To permit different data sets to be processed without module reassembly, the programmer does not refer to the data set by name, but refers to a data control block associated with the DATADEF (DD) statement at object time. The programmer reserves space for a data control block at assembly time by issuing a DCB macro instruction.

DATA SET CATALOGING

The cataloging facility of the Time-Sharing System/360 enables the programmer to refer to data sets without specifying their physical locations. When a data set is cataloged, the volume serial number is associated in the catalog with the name of the data set. All TSS/360 data set organizations are capable of being cataloged.

THE CATALOG

The catalog of data sets is a data set residing on a direct access volume. It is organized into indices that connect data set names to corresponding volume numbers (and to data set sequence numbers for magnetic tape volumes).

As described previously, a data set name consists of one or more simple names. For every distinct level of qualification in the name of a cataloged data set, the catalog includes a group of one or more index blocks.

The hierarchy of index levels is determined by the order of the data set names, with the highest level being the leftmost name. The records in each index consist of the simple names and physical locations of all subordinate indices or data sets. Thus, only the lowest-level index indicates the volume number of the data set.

CATALOGING DATA SETS

To insert, delete, and modify data set entries in the catalog, statements are provided within the TSS/360 command language. To catalog a data set, the CATALOG statement is provided. To allow the sharing of data sets among two or more users, the PERMIT and SHARE statements are available for the modification of the catalog. The ERASE statement deletes data set entries from the catalog, and thus deletes the data sets.

DATA SET SECURITY PROTECTION

Through the facilities provided by the catalog, the protection of data sets from unauthorized use is enforced. Unless the owner, or creator, of a data set authorizes (via the PERMIT command language statement) the sharing of a data set, no other user is allowed access to that data set. The PERMIT statement allows the user to make data sets available to any subset of users, or to all users of the system. The access of the sharing users may be restricted to reading only, may be accorded the privilege of both reading and writing a shared data set, or may be given unlimited access, which includes the addition or deletion of data set names in the catalog. Once a user has been authorized to share a data set, he must issue a SHARE command to indicate that he does indeed wish to share the data set.

VAM data sets may be shared among users.

GENERATION DATA GROUPS

Certain data sets that are periodically updated may be chronologically related to each other. For example, similar payroll data sets may be created every week. Cataloging such data sets with unique data set names would be as inconvenient as giving them all the same name and accounting for volume identification. For this reason, the system provides an option within the cataloging facility that assigns numbers to individual data sets in a chronological collection, thereby enabling the programmer to catalog the entire collection under a single name. The programmer can distinguish among successive data sets in the collection without assigning a new name to each data set. Since each data set is normally created by updating the data set created on the previous run, the update is called a generation number.

A generation data group is a collection of related cataloged data sets that can be referred to by a common name in a DATADEF statement. The programmer can refer to a particular generation by specifying, with the common name of the group, either the generation name or the relative generation number of the data set.

DATA SET STORAGE AND VOLUMES

System/360 provides a variety of devices for collecting, storing, and distributing data. Despite this variety, the storage units have many common characteristics. For convenience, therefore, the generic term "volume" is used to refer to such diverse storage media as tape reels, disk packs, data cells, and drums.

Direct access volumes play a major role in TSS/360. These volumes are used to store not only the operating system itself, but also all program modules. In addition, direct access volumes are used by many installations as the chief storage media for the vast number of data sets processed each day.

Before a direct access volume can be used for data storage, it is initialized by a volume initialization utility program. This program creates a volume label and reserves the area on the volume that the system uses for space management. This area is called the volume table of contents (VTOC). Volume initialization effectively clears the volume of any existing data by indicating to the VTOC that all the space on that volume (except that space occupied by the volume label and VTOC) is available for allocation.

1. Volume label — identifies the volume and contains a pointer to the VTOC. The volume labels of direct access volumes are always on cylinder 0 and track 0.
2. Volume table of contents — describes the contents of a direct access volume. Basically, there are two types of entries in a VTOC: an entry for each data set stored on the volume, and an entry for each available set of contiguous tracks.

Each data set entry contains the name, description, and location on the volume of its associated data set. This entry is called the data set control block (DSCB). Each entry of available storage indicates a group of contiguous tracks that are available for allocation.

DATA STORAGE ON MAGNETIC TAPE VOLUMES

Because of the serial nature of magnetic tape devices, the system does not provide space allocation facilities comparable to those for direct access volumes. New data sets may be added after those that already exist on a volume.

Although the system does not reserve space for a data set on a magnetic tape volume, it can position the volume so that the data set is written on unused space. When a new data set is to be placed on a magnetic tape volume, the programmer should specify the relative position of this data set in the data set sequence number parameter of its DATADEF statement. When this parameter is specified for a data set with standard labels or with no labels, the system positions the volume so that the data set can be written. Nonstandard labels are not provided for in TSS/360.

VOLUME LABELING

Various groups of labels are used in secondary storage of TSS/360 to identify magnetic tape and direct access volumes, as well as the data sets they contain.

Magnetic tapes can have standard labels, or they can be unlabeled. Only standard label formats are used on direct access volumes. Volume, data set control block, and user labels are used; however, user labels are allowed only within SAM.

DATA SET RECORD FORMATS

Data processing operations are concerned with individual data records within a data set. All records of a data set must have the same format. The formats of these records is the subject of the following discussion.

Time-Sharing System/360 data sets can be organized in three ways:

Sequential

This is the familiar tape-like structure, in which physical records are placed in sequence. Thus, given one record, the "next" record is uniquely determined. The sequential organization is used for all magnetic tapes, and may be selected for direct access devices. Punched tape, punched cards, and printed output are considered to be sequentially organized.

Index Sequential

Records are arranged in logical sequence (according to a key that is part of every record) on the tracks of a direct access device. In addition, a separate index or set of indices maintained by the system gives the location of certain principal records. This permits direct, as well as sequential, access to any record. Records may be added to and deleted from the data set as required, with appropriate updating of the index.

Partitioned

This structure has characteristics of both the sequential and indexed sequential organizations. Independent groups of sequentially organized data, each called a member, are in direct access storage. Each member has a simple name stored in a directory that is part of the data set and contains the location of that member's starting point. An example of partitioned data set use is the storage of program modules in a form structurally similar to program libraries. As a result, partitioned data sets are often referred to as "libraries".

LOGICAL RECORDS

A data set is made up of a collection of logical records that usually have some relation to one another. The logical record is usually the basic unit of information for a data processing program. A logical record might be, for example, either a single character, all information resulting from a given business transaction, or parameters from a given point in an experiment. The maximum logical record lengths are shown below.

Maximum Logical Record Lengths

Access Method	Data Set Organization	Maximum Number of Bytes in a Record
Sequential access method (SAM)	Sequential	32,767
Virtual access method	Sequential	1,048,767
Virtual access method	Index sequential	4,000

RECORD BLOCKING

Blocking of records is the process of grouping a number of logical records before writing them on a volume. Such a grouping is called a block. Blocking improves effective data rate and conserves storage space on the volume by reducing the number of inter-record gaps in the data set. In many cases, blocking also increases processing efficiency by reducing the number of input/output operations required to process a data set.

RECORD FORMATS

Logical records may be in one of three formats: fixed-length (format F), variable-length (format V), or undefined (format U).

The prime consideration in the selection of a record format is the nature of the data set itself. The programmer knows the type of input his task will receive and the type of output it will produce. His selection of a record format is based on this knowledge, as well as on an understanding of the type of input/output devices that are to handle the data set and of the access method used to read or write the data set.

GENERAL SERVICES

Certain macro instructions provide services that prepare data sets for processing, and core storage for use as buffers and buffer pools. Of these macro instructions, DCB, DCBD, OPEN, and CLOSE are used with all of the access methods.

Macro Instruction	Function
DCB	Constructs a data control block for interfacing with the supervisor
DCBD	Provides symbolic names for data control block parameters
OPEN	Connects the data set to the user's task
CLOSE	Disconnects the data set from the user's task
FEOV	Forces an end of volume
GETPOOL	Gets a buffer pool
FREEPOOL	Frees a buffer pool
GETBUF	Gets a buffer from a pool
FREEBUF	Returns a buffer to a pool

The DCB macro, OPEN macro, and CLOSE macro are required for each of the access methods discussed in the following sections.

SEQUENTIAL ACCESS METHOD (SAM)

SAM supports the greatest variety of devices of the TSS/360 access methods. Its organizational methods may be used for the following devices:

- Magnetic tapes
- Direct access
- Paper tape readers
- Card readers and punches
- Printers

Two organizational methods are supplied within SAM. The queued sequential access method (QSAM) performs blocking and deblocking of logical records, whereas the basic sequential access method (BSAM) operates only upon physical blocks.

QUEUED SEQUENTIAL ACCESS METHOD (QSAM)

The queued sequential access method (QSAM) is, for the most part, device-independent, permitting program modules that can be written to use one of a number of different input/output devices. Records of a sequential data set can be stored and retrieved without the writing of blocking/deblocking, and buffering routines by the user. Simple buffering is used.

The QSAM macro instructions (GET, RELSE, PUT, PUTX, TRUNC), the device-dependent macro instructions (CNTRL, PRTOV), and the general service macro instructions (GETPOOL, FREEPOOL, OPEN, CLOSE, FEOV) are used with the queued sequential access method.

Macro instructions available under QSAM are:

Macro Instruction	Function
GET	Gets a logical record from a sequential data set
PUT	Includes a logical record in an output data set
PUTX	Returns an updated record to a sequential data set or includes a record of an input data set in an output data set
RELSE	Causes the remaining logical records in a buffer to be ignored
TRUNC	Causes the next logical record to be written on the first record of the next block — that is, the current buffer area is regarded as filled
CNTRL	Controls a printer or a card reader
PRTOV	Tests for printer carriage overflow

BASIC SEQUENTIAL ACCESS METHOD (BSAM)

The macro instructions included in this group permit the user to gain access to blocks of a sequentially organized data set. The user can remain device-independent by restricting himself to a subset of macro instructions. For users to whom device independence is not a limiting factor, a macro extensive set can be used.

Macro Instruction	Function
Device Independence:	
READ	Reads a block
WRITE	Writes a block
CHECK	Tests for completion, and waits if not complete
Tape/Direct Access Device Independence:	
NOTE	Notes where a block was written
POINT	Points to a block to read
Device-Dependent:	
PRTOV	Tests for printer carriage overflow
CNTRL	Controls a card reader, a printer, or a magnetic tape drive

VIRTUAL ACCESS METHOD (VAM)

VAM is designed specially for TSS/360, and it performs the input/output of data by using the paging supervisor. Because of its use of the paging supervisor, VAM data sets are limited to direct access devices, and all actual input/output through VAM is in blocks of pages, which are 4096 bytes in length.

VAM organizes data sets by relative page number. That is, each block of a data set is assigned, as it is created, a page number that is relative to the beginning of the data set. These relative page numbers are then related to an input/output device address through a relative page/external page correspondence table (RESTBL), which is created from data in data set control blocks (DSCB) and maintained within virtual memory by VAM routines. Appropriate external device addresses from RESTBL are passed to the paging supervisor, as required, for the building of external page table entries for virtual memory areas associated with a VAM data set.

VAM is designed to minimize the number of virtual memory pages associated with an open data set, since only the data set pages on which the user is currently operating are located in virtual memory. In addition, one or more pages are required for the RESTBL. For a partition organization data set, the partition organization directory is required. For index sequential, an overflow page and a directory space may be required, depending upon the data set. VAM also provides, within the sequential organizational method, the facility for the user to input or to output data set pages into or from virtual memory areas of any size up to a segment (256 pages).

The user must open a data set before it can be processed by VAM, and he must close the data set when he has finished processing it. Again, as for the other access methods, the primary means of communication between the user and the VAM routine is a data control block (DCB).

VAM also provides facilities for the sharing of data sets between many users with no inconvenience to the users. The sharing is controlled on a page basis. Any number of users up to 255 may be simultaneously reading the same page of a data set. However, once a user has obtained a data set page for updating, or is creating a data set page, no other users are allowed to access this page until it has been written or rewritten.

Three data set organization methods are available under VAM: sequential, index sequential, and partitioned. Each of these methods is described briefly in the following paragraphs.

SEQUENTIAL ORGANIZATION (SVAM)

A sequential data set is one in which the order of records is determined by their physical position within the data set. During the creation of a sequential data set, logical records are linked together one behind the other. Normally, these records are then read back in the order in which they were written, although a capability is provided whereby logical records may be read and updated nonsequentially.

Sequential processing has value in cases where the user makes an orderly sweep through his data set, or where the user defines his own random accessing criteria.

The logical records of which a VAM sequential data set is comprised may be variable-length, fixed-length, or undefined, wherein logical record length is assumed to be a multiple of 4096 bytes.

User references to a virtual access method sequential data set are made through the following:

Macro Instruction	Function
SETL	Specifies the point at which sequential retrieval is to begin
GET	Gets the next sequential logical record
PUT	Adds a new logical record to to the data set
PUTX	Returns an updated logical record to a data set

INDEX SEQUENTIAL ORGANIZATION (VISAM)

In an indexed sequential data organization, the records are ordered on a data key. The data key may be a control field that is in an intrinsic part of the information in the record (for example, a part number), or it may be some arbitrary identifier associated with the record, such as a line number. When the record is stored, the data key must be associated directly with the record, and is considered part of the record.

The data set also contains directories and locators relating to the keys and actual addresses of the records in the data set. For the data set as a whole, a page directory gives the value of the key for the first record associated with each data page. On each page an ordered set of locators specifies either the location of the record on the page or the position of a corresponding locator on an overflow page.

A reference to an indexed sequential data set must be made through the following indexed sequential access method macro instructions:

Macro Instruction	Function
SETL	Specifies the point in an indexed sequential data set at which sequential retrieval is to begin
ESTEL	Ends sequential retrieval
GET	Gets the next sequential logical record
PUT	Places a logical record in an output buffer, from which it is written
READ	Gets a logical record with corresponding keys
WRITE	Places a logical record in the data set, positioned in the data set according to the value of the key on the record being written
DELREC	Deletes a logical record from a data set
RELEX	Releases an exclusive control of a record

PARTITIONED ORGANIZATION (VPAM)

The partitioned data organization combines individually organized groups of data into a single data set. Each group of data is referred to as a member, and each member is identified by a unique eight-character alphmeric name. The data set contains a partitioned organization directory (POD), which points to the first block of each member and gives the length of each member. The POD originally occupies the first page allocated to a partitioned data set, and expands into additional pages as required.

Members are located within the partitioned data set by using the FIND macro instruction, and new members are combined into the data set by use of the STOW macro instruction. Facilities are provided to enable the user to relate additional names to a member. These additional names are referred to as aliases, and a member may be located by use of either the member name or an alias name.

A DCB for a partitioned data set must be opened (by use of the OPEN macro) before any processing of the data set can be done. Similarly, a DCB must be closed (by use of the CLOSE macro) when processing with that DCB is completed.

A FIND of a member is a limited "open", the attributes of the member are filled into the DCB and the DCB is logically positioned to the member. Similarly, certain types of STOW's are limited "close" as final buffers are written and the attributes of the member are updated within the POD.

A member of a partitioned data set may be index sequential organization or sequential organization. Whenever an existing partitioned data set is opened for output, the associated DCB is positioned to the logical end of the data set.

A new member can then be created using any of the organization methods and later stowed, at which time a directory entry is created for the new member.

Once a member is stowed, it may later be referenced only after a FIND for the member has been executed. A member may be read, extended, or updated in place.

A reference to a partitioned data set must be made through the following macros:

Macro Instruction	Function
FIND	Prepares a data set member for processing
STOW	Updates the partitioned organization directory and, in certain cases, disconnects a data set member from the user's task

In addition to these macros, the sequential and index sequential macros are available to the user for performing operations upon partitioned data set members.

PRINTER FORMS CONTROL

The BULKIO command provides to the user the facility to specify a form number on which the data set is to be printed. TSS/360 instructs a system operator to mount the specified form before printing begins.

Print Control Characters

For output data sets to be printed, the user may insert his own print control characters as the first byte of each logical record, or he may elect to omit print control characters and have the printing edited with a fixed spacing between lines.

If print control characters are used, they may be either FORTRAN or machine code, but may not be intermixed within the same data set.

Terminal Access Method (TAM)

TAM provides the necessary communication with a terminal. Routines are included to cause the proper action to be taken upon the termination of any channel program operating a low-speed terminal device.

Graphic Access Method (GAM)

The graphic access method (GAM) provides programming systems support for the following graphic devices:

- 2250 Display Unit
- 2280 Film Recorder
- 2282 Film Recorder/Scanner

GAM generates graphic orders for the control of these graphic devices; it facilitates handling, both in core storage and in the graphic device buffer; it accomplishes input/output control functions, and it controls the dispatching of asynchronous light pen, alphameric keyboard, program function keyboard, attention, and error interrupts. Embodied in GAM is a graphic interrupt supervisor, which queues and distributes graphic interrupt conditions. GAM is also responsible for the management of graphic device buffer storage.

DATA FLOW

Data flow through the system is from any auxiliary storage device into core and, if necessary, back into auxiliary storage as processing is completed. The data management routines within the time-sharing supervisor maintain statistics on activity of each data set and assign the sets to storage-level hierarchies on the basis of activity. As a data set is used frequently, it moves upward within the storage hierarchy to devices of faster accessibility, and thus "closer" to core storage. As activity diminishes, it tends to drift lower in the hierarchy to devices of less cost per unit of storage capacity.

The method of data management and flow has several advantages:

- The supervisor achieves an automatic balance between the activity of data and the storage level to which the data is allocated.
- The bulk of the data and programs is kept in devices of lowest cost per unit of storage capacity without compromising response time.
- The system capacity and response can be matched to the user's requirements without program module modification.
- The ultimate in "fail-soft" capability can be achieved at the lowest cost. Duplexing of secondary storage is kept to the minimum commensurate with total capacity and response time.

The entire system philosophy of data flow is based on the page, the basic unit of data in the system. A page may contain either a program or data. Pages are ordinarily the units that are extracted from cataloged locations within secondary storage and loaded into core. These pages are then copied into a paging area on drum or, occasionally, on disk. The page flow, then, is from drum to core storage and back, as determined by the

task-scheduling and core storage allocation routines within the supervisor.

Each data set resident in the system is cataloged in an index that is maintained on disk file. Information concerning the size and location of the data set is contained within the index, thus enabling the system to gain access to the data set, or to any page within it.

When a page is brought into core storage, its availability is recorded in the appropriate page table entry.

Programs and data sets may be introduced into the system in several ways. Certain tasks may create them, as is the case with compilers. Users may create them by requesting the system to save, under a new name, a data set with which they have been working, or data may enter from a wide variety of sources:

- Keyboard input
- Low-speed and high-speed card readers
- Paper tape readers
- Tape units
- Disk packs
- Remote data links

Data, by whatever means it arrives, must be cataloged, and storage of the required size must be assigned. Control information may govern the device type into which it is loaded, but not the specific I/O device. It need not remain in the device into which it was loaded originally. Depending on its activity, it may rise within the hierarchy of storage to a device with faster access speeds or, if comparatively inactive, sink to a device of slower accessibility.

Data may be loaded with control information specifying that it is to be kept in the load location, or it may be left to the system's regulatory ability to locate it in the device most suited to its usage and system storage capacity.

Two levels of data flow are present in the time-sharing system. The first is the flow into core storage, in response to calls, of pages from a cataloged storage location within the storage hierarchy. Modified pages flow back to the cataloged secondary storage.

The second is a vertical movement between devices within different levels of the storage hierarchy to balance the system dynamically. The system maintains the density of each storage device and brings programs and data sets into a level of storage accessibility commensurate with their activity.

As additional data sets are introduced into the system, provision must be made for a periodic purging of the lowest or the most passive level of online storage to some type of archival storage. The archival storage may be tape, removable disk packs, or removable data cells.

The object of this data flow activity is twofold. First, to use the various I/O devices in the most efficient manner; second, to have files located as close in the hierarchy to core storage as their usage requires and storage capacity permits.

An additional advantage of this dynamic method of storage allocation is the minimizing of the cost of fail-soft capability. Ideally, the failure of any I/O device in the system should not be of concern to the terminal user. Duplexing of all input/output devices provides the ultimate in fail-soft capability. However, full duplexing of large numbers of fast-accessing storage devices may be prohibitively expensive. This automatic method of keeping only the high-activity files in the fast-access devices tends to reduce the total cost of secondary storage, thus making full duplexing more attractive.

MAJOR DIFFERENCES BETWEEN OS/360 AND TSS/360 AS THEY AFFECT THE PROGRAMMER

DATA MANAGEMENT FUNCTIONS

The facilities for addressing extremely large areas of virtual memory permit TSS/360 to offer data management services that differ from those of OS/360.

The following compares organization and access methods for the two systems:

Operating System			Time-Sharing System		
			Sequential		Direct Access Only
Organization	Queued	Basic	Queued	Basic	
Sequential	QSAM	BSAM	QSAM	BSAM	VSAM
Indexed sequential	QISAM	BISAM			VISAM
Direct		BDAM			
Partitioned		BPAM			VPAM
Telecommunication	QTAM	BTAM		TAM	
Graphic				GAM	

Data set identification is comparable in the two systems for QSAM or BSAM.

Cataloging and library management on OS/360 is primarily via the data definition statement, whereas on TSS/360, separate commands exist for this function. The protection of data sets, which is a special feature of OS/360, is standard on TSS/360.

The operation of data management macros is similar in both systems; in particular, the data required to OPEN data sets is identical (derived from DCB's DD statements and data set labels). However, under VSAM, physical block size is set to conform with the paging mechanism.

Allocation of input/output devices is automatic on TSS/360. The reservation of physical units takes place in the DD statements, which reserve a device for a logical data set directly, instead of indirectly, as in OS/360.

TSS/360 provides a different level of device-independence than does OS/360. Also, it is not possible to interchange all data sets (for example, TSS/360 VAM data sets) from OS/360 to TSS/360, and vice versa, because of differences in labels and organization conventions.

Note: TSS/360 offers special facilities for terminal communication via the GATE routine of CLI.

JOB MANAGEMENT FUNCTIONS

A correspondence exists between the effects of command language commands for the time-sharing system and the JOB CONTROL language of OS/360. However, note the following differences:

Statement Format

OS/360	//name operation operands
TSS/360	operation operands

Whereas OS/360 statements may have a name and are identified by the presence of slashes in the first two positions, TSS/360 statements cannot have a name and are identified only by the initial presence of an operand.

Number of Operations

From the user's viewpoint there are only three OS/360 operations: JOB, EXEC, and DD. From the same viewpoint there are 49 TSS/360 operations (and an equal number of abbreviations). Comparable keywords are:

OS/360	JOB, EXEC, DD
TSS/360	LOGON, LOAD, RUN, DD

Parameter Syntax

OS/360	TSS/360
Parentheses are used to delimit lists.	Parameters separated by commas are treated as lists (same for TSS on card input).
A specified card position holds a possible continuation character, and the continuation card starts at a specified column 16.	A special keyboard combination -- tab and carriage return — implies that a continuation statement follows. The latter starts in the usual position (same as OS/360 for card input).

Definition of Named Job Steps

This is possible on OS/360, but not on TSS/360.

Definition of Procedures

OS/360 permits any set of control cards to be cataloged as a procedure by entering a keyword on the JOB card. To do this on TSS/360 a separate run of DATA is required. On TSS/360, predefined procedures cannot be called as part of other command strings.

Setting of Time and Page Constraints

No facilities exist on TSS/360.

USER PROGRAM CONVENTIONS

The principal difference between writing programs for TSS/360 and for OS/360 is that account must be taken of a different effective address size (a possible 32 bits instead of 24 bits). This means that the use of high-order bits of Adcons and the use of short Adcons is no longer possible with safety. Any program written for OS/360 may require careful examination before it can be reassembled to run on TSS/360.

GLOSSARY

Access method. Any of the data management techniques available to the user for transferring data between main storage and an input/output device.

Adcon. (Same as "address constant".)

Address constant. A value, or an expression representing a value, used in the calculation of virtual memory addresses.

Alias. An alternate name that may be used to refer to a member of a partitioned data set; an alternate entry point at which execution of a program can begin.

Allocate. To grant a resource to, or reserve it for, a task.

Asynchronous. Without regular time relationship; hence, as applied to task execution, unexpected or unpredictable with respect to instruction sequence.

Attribute. A characteristic; for example, attributes of data include record length, record format, data set name, associated device type and volume identification, use, creation date, etc.

Auxiliary storage. Data storage other than core storage used to hold active programs.

Basic access method. Any access method in which each input/output statement causes a corresponding machine input/output operation to occur. (The primary macro instructions used are READ and WRITE.)

Batch processing. The processing of a task under the direction and control of a prestored command sequence.

Block (records). (1) To group records for the purpose of conserving storage space or increasing the efficiency of access or processing. (2) A physical record so constituted, or a portion of a telecommunications message defined to be a unit of data transmission.

Buffer (program input/output). A portion of core storage into which data is read, or from which it is written.

Catalog. (1) The collection of all data set names and location indices. (2) To include the volume identification of a data set in the catalog.

Cataloged data set. A data set that is represented in an index or hierarchy of indices that provide the means for locating it.

Checkpoint. (1) A point at which information about the status of a task can be recorded so that the task can be restarted. (2) To record such information.

Control block. A storage area through which a particular type of information required for control of the operating system is communicated among its parts.

Control dictionary. The external symbol dictionary and relocation dictionary collectively, of an object or a load module.

Control section. The smallest separately relocatable unit of a module; that portion of text specified by the programmer to be an entity, all elements of which are to be loaded into contiguous main storage locations, starting at a page boundary.

Conversational. Able to control, interrogate, modify, and observe a task, from a terminal remote from the control data processing facility.

Core storage. The total amount of core storage available for TSS/360 use in the IBM 2365 core storage units.

Data control block. A control block through which the information required by access routines to store and retrieve data is communicated to them.

Data definition name (dd name). A name appearing in the data control block of a program that corresponds to a name field of a data definition statement.

Data definition (DATADEF) statement. A command language statement that describes a data set associated with a particular task.

Data management. A general term that collectively describes the functions of the supervisor that provide access to data sets, enforce data storage conventions, and regulate the use of input/output devices.

Data organization. A term that refers to any one of the data management conventions for the arrangement of a data set.

Data set. The major unit of data storage and retrieval (consisting of a collection of data in one of several prescribed arrangements and described by control information) to which the system has access.

Data set control block (DSCB). A data set label for a data set in direct access storage.

Data set label (DSL). A collection of information that describes the attributes of a data set, and that is normally stored with the data set; a general term for data set control blocks and tape data set labels.

Device independence. The ability to request input/output operations without regard to the characteristics of the input/output devices.

Direct access. (1) Retrieval or storage of data by a reference to its location on a volume, rather than relative to the previously retrieved or stored data. (2) A type of storage device, such as IBM's 2301, 2311, 2314, and 2321.

Dump. (1) To copy the contents of all or part of storage onto an output device so that it can be examined. (2) The data resulting from (1). (3) A routine that will accomplish (1).

Dynamic address translation. The translation of virtual memory addresses to core storage addresses.

Entry point. Any location in a program to which control can be passed by another program.

Extent. The locations on input/output devices occupied by or reserved for a particular data set.

External reference. A reference to a symbol defined in another module.

External symbol. A control section name, an entry point name, or an external reference.

External symbol definitions. Control information associated with a program module that identifies the external symbols in the module.

Fetch storage protect. The equipment feature that disallows any reference to a fetch-protected storage block; an attempted reference causes an equipment interrupt.

Generation data group. A collection of successive, historically related data sets.

Index (data management). (1) A table in the catalog structure used to locate data sets. (2) A table used to locate the records of an indexed sequential data set.

Installation. A general term for a particular computing system, in the context of the overall function it serves and the individuals who manage it, operate it, apply it to problems, service it, and use the results it produces.

Job library. A series of interconnected user-identified program modules.

Language translator. A general term for any assembler, compiler, or other routine that accepts statements in one language and produces equivalent statements in another language.

Library. (1) In general, a collection of objects (for example, data sets, volumes, card decks) associated with a particular use, and the location of which is identified in a directory of some type. (2) Any partitioned data set.

Linkage. The means by which communication is effected between two routines or modules.

Linkage editor. A program that produces an object program module by transforming other program modules into a format that is acceptable to the dynamic loader, optionally combining separate control sections into a single control section, resolving symbolic cross-references among them,

replacing, deleting, and adding control sections automatically on request.

Load. To dynamically link to, that is, to bring a program module into virtual memory when it is referenced.

Locate mode. A transmittal mode in which data is pointed to rather than moved.

Logical record. A record from the standpoint of its content, function, and use, rather than its physical attributes; that is, one that is defined in terms of the information it contains.

Macro instruction. A general term used to collectively describe a macro instruction statement, the corresponding macro instruction definition, the resulting assembler language statements, and the machine language instructions and other data produced from the assembler language statements; loosely, any one of these representations of a machine language instruction sequence.

Module (programming). The input to, or output from, a single execution of an assembler, compiler, or linkage editor; a source, or program module; hence, a program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading.

Move mode. A transmittal mode in which data is moved between the buffer and the user's work area.

Multiprogramming. A general term that expresses use of the computing system to fulfill two or more different requirements concurrently.

Multitask operation. Multiprogramming; called multitask operation to express parallel processing not only of many programs, but also of a single reenterable module used by many tasks.

Name. A set of one or more characters that identifies a statement, data set, module, etc., and that is usually associated with the location of that which it identifies.

Object module. (see "program module".)

Operator command. A statement to the command language program, issued via a console device, which causes it to provide requested information, alter normal operations, initiate new operations, or terminate existing operations.

Page. A total of 4096 bytes, the first byte of which whose address ends in twelve binary zeros.

Paging. Transcription of a page to or from core storage and a direct access device.

Parallel processing. Concurrent execution of one or more tasks.

Physical record. A record from the standpoint of the manner or form in which it is stored, retrieved, and moved; that is, one that is defined in terms of physical qualities.

Polling. A technique by which each of the terminals sharing a communications line is periodically interrogated to determine whether it requires servicing.

Private library. Any partitioned data set that is not the system library (SYSLIB).

Privileged. (1) Privileged user: one capable of executing certain system control commands from a terminal. (2) Privileged module: a system module that is allowed to communicate with the TSS/360 supervisor. (3) Privilege of access: an attribute of the level of allowable sharability of a shared data set; could be read-only access, read/write access, or no access.

Problem program module. Any of the class of routines that perform processing of the type for which a computing system is intended, and including routines that solve problems, monitor and control industrial processes, sort and merge records, perform computations, process transactions against stored records, etc.

Processing program. A general term for any module that is not a supervisor program.

Program. A collection of related program modules.

Program module. The output of a single execution of an assembler or a compiler, which constitutes input to the dynamic loader or the linkage editor. A program module consists of one or more control sections in relocatable (though not executable) form and an associated program module dictionary.

Protection key. An indicator associated with a task that appears in the program status word whenever the task is in control, and that must match the storage keys of all storage blocks that it is to use.

Qualified name. A data set name that is composed of multiple names separated by periods (for example, TREE.FRUIT.APPLE).

Qualifier. All component names in a qualified name other than the rightmost (which is called the simple name).

Queued access method. Any access method that automatically synchronizes the transfer of data between the program using the access method and input/output devices, thereby eliminating delays for input/output operations. (The primary macro instructions used are GET and PUT.)

Ready condition. The condition of a task that is in contention for the central processing unit, all other requirements for its activation having been satisfied.

Record. A general term for any unit of data that is distinct from all others when considered in a particular context.

Reenterable. The attribute of a control section that allows the same copy of the section to be executed concurrently by two or more tasks.

Relocation. The modification of address constants required to compensate for a change of origin of a module or control section.

Resource. Any facility of the computing system or operating system required by a job or task, including main storage, input/output devices, the central processing unit, data sets, and control and processing programs.

Restart. To reestablish the status of a task using the information recorded at a checkpoint.

Return code. A value that is by system convention placed in a designated register (the return code register) at the completion of a module. The value of the code, which is established by user convention, may be used to influence the execution of succeeding modules, or, in the case of an abnormal end-of-task, it may simply be printed for programmer analysis.

Return code register. A register identified by system convention in which a user-specified condition code is placed, at the completion of a program.

Reusable. The attribute of a routine that the same copy of the routine can be used by two or more tasks. (See "reenterable", "serially reusable".)

Secondary storage. Auxiliary storage.

Seek. To position the access mechanism of a direct access device at a specified location.

Serially reusable. The attribute of a routine that when in core storage the same copy of the routine can be used by another task after the current use has been concluded.

Service program. Any of the class of standard routines that assist in the use of a computing system and in the successful execution of problem modules, without contributing directly to control of the system or production of results, and including utilities, simulators, test and debugging routines, etc.

Shared routines. Routines containing one or more control section addressable simultaneously by two or more tasks.

Simple buffering. A technique for controlling buffers in such a way that the buffers are assigned to a single data control block and remain so assigned until the data control block is closed.

Simple name. The rightmost component of a qualified name (for example, APPLE is the simple name in TREE.FRUIT.APPLE).

Source module. A series of statements in the symbolic language of an assembler or a compiler, which constitutes the entire input to a single execution of the assembler or compiler.

Stacked job processing. A technique that permits many tasks to be grouped (stacked) for presentation to the system, which automatically recognizes each task, one after the other.

Storage block. A contiguous area of core storage consisting of 2048 bytes to which a storage key can be assigned.

Storage key. An indicator associated with a storage block or blocks, which requires that tasks have a matching protection key to use the blocks.

Store storage protect. The equipment feature that disallows a store into a protected storage block; an attempted reference causes an equipment interrupt.

Supervisor. The program modules supplied by IBM that control and monitor the usage of the model 67.

Synchronous. Occurring concurrently and with a regular or predictable time relationship.

SYSIN. A name used conventionally as the data definition name of a data set in the input task stream.

SYSOUT. An indicator used in data definition statements to signify that a data set is to be written into the system output data set.

System macro instruction. A predefined macro instruction that provides access to TSS/360 system facilities.

Task. A unit of work for the central processing unit, from the standpoint of the control program; therefore, the basic multiprogramming unit under the supervisor.

Task monitor. The supervisor function that selects from the task queue the task that is to have control of a central processing unit, and gives control to the task.

Task management. A general term that collectively describes those functions of the supervisor that regulate the use by tasks of the central processing unit and other resources (except for input/output devices).

Teleprocessing. A term associated with telecommunications equipment and systems.

Text. The control sections of a program module.

Throughput. A measure of system efficiency; the rate at which work can be handled by a computing system.

Time slice. The maximum period of time each task is allowed to execute before that task is placed in a queue of tasks that are vying for the system resources.

Transmittal mode. The method by which the contents of an input buffer are made available to the program, and the method by which a program makes records available for output.

Turn-around time. The elapsed time between submission of a task to a computing center and the return of results.

U format. A data set format in which blocks are of unspecified or otherwise unknown length.

User. Anyone who utilizes the services of a computing system.

V format. A data set format in which logical records are of varying length and include a length indicator; and in which V format logical records may be blocked, with each block containing a block length indicator.

Printed in U. S. A. Z20-1788-0

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601