File No. S360-36
GY28-2009-2

Program Logic

IBM System/360 Time Sharing System

System Logic Summary

Program Logic Manual

This publication describes the logic of the IBM System/360 Time Sharing System (TSS/360). The emphasis is on describing the interrelationship of system components in performing system functions. Where an individual component is described, only the highest level of logic is discussed.

This program logic manual is directed to the IBM customer engineer who is responsible for program maintenance. It can be used to determine the interrelationship among the various areas of the system, and it enables the reader to relate these areas to the corresponding program logic manuals.

PREFACE

This publication is divided into three parts. The first part contains a systematic description of TSS/360. The second part explains various detailed features of TSS/360 such as allocation algorithms, sharing, and paging and gives examples of system operation. The third part discusses the relationship of language processors and auxiliary programs to the system.

There are three appendixes. One relates major system components to the appropriate Program Logic Manual, the second contains a summary of important system control blocks, and the third is a directory listing all modules for which flowcharts exist and the PLM which contains each flowchart.

PREREQUISITE PUBLICATIONS:

IBM System/360 Time Sharing System: Concepts and Facilities, Form C28-2003

In addition, the following TSS/360 publications can be used to supplement an understanding of the interrelationships described in this manual:

IBM System/360: Model 67 Functional Characteristics, Form A27-2719

IBM System/360 Time Sharing System: Command System User's Guide, Form C28-2001

IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032

IBM System/360 Time Sharing System: Manager's and Administrator's Guide, Form C28-2024

IBM System/360 Time Sharing System: Assembler Language, Form C28-2000

IBM System/360 Time Sharing System: Assembler User Macro Instructions, Form C28-2004

IBM System/360 Time Sharing System: IBM FORTRAN IV, Form C28-2007

IBM System/360 Time Sharing System: Linkage Editor, Form C28-2005

IBM System/360 Time Sharing System: Operator's Guide, Form C28-2033

Third Edition (June 1970)

This is a major revision of, and makes obsolete, IBM System/360 Time Sharing System: System Logic Summary, Form Y28-2009-1, and Technical Newsletters Y28-3094, Y28-3102, Y28-3111, and Y28-3112. Changes to the text and illustrations have been made throughout and this edition should be reviewed in its entirety.

This edition is current with Version 7, Modification 0, and remains in effect for all subsequent versions or modifications of IBM System/360 Time Sharing System unless otherwise indicated. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters. Before using this publication in connection with the operation of IBM systems, refer to the latest edition of IBM System/360 Time Sharing System: Addendum, Form C28-2043, for the editions of publications that are applicable and current.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Time Sharing System/360 Programming Publications, Department 643, Neighborhood Road, Kingston, New York 12401.

PART I: GENERAL SYSTEM DESCRIPTION

Time Sharing is a logical extension of the growth in sophistication and scope of the computing environment since its beginning nearly two decades ago. In particular, the IBM Model 67 Time Sharing System serves as a logical extension for the problem solving needs that gave rise to System/360 hardware and to IBM System/360 Operating System.

The development of systems programming has been a "three-generation" process. First, there was the development of translators, starting with basic assembly languages and advancing to the higher level, problem-oriented languages and macro compilers. The major objective of this development was to reduce the time required to get an operation to a computer.

The second generation consisted of monitor systems such as 7090/7094 IBSYS and the 1410/7010 Operating System. These systems were designed to reduce the time between machine room operations by, for example, providing the ability to stack jobs and minimizing dependence on the machine operator.

IBM System/360 operating system is typical of the third generation of systems programming. This can be referred to as resource allocation. Its major objective is to maximize the use of system components during multiprogramming operations. These resources are CPU time, channel time, main storage, external storage, and programs. Under IBM System/360 operating system, each of these is scheduled and dispatched separately and asynchronously to satisfy competing demands.

Time Sharing can be described as the concurrent use of the resources of a general purpose computing system by a large number of users.

In common with other third generation multiprogramming systems, the accent in this definition is on "resource sharing" rather than time sharing; because a general purpose time sharing system must be designed to share main storage, channel facilities and direct access (disk and drum) file space among a large number of users. Strictly speaking, a single CPU does not share time; it still operates on only one task at any moment.

Unlike batch multiprogramming systems whose goal it is to maximize throughput, the goal of a time sharing system is to make it easier to use a computer while maintaining a very high degree of utilization of the computer system resources.

The reasons for wishing to make a computer system easy to use are:

- To reduce the complexity involved in preparing a program for execution on a computer. This allows scientists, engineers, and students to explore potentially rewarding bypaths which are currently too much trouble to program. This can be done by performing those clerical tasks that are so much a part of problem solving, such as routine calculations, reducing and plotting data, text editing, and information retrieval.
- To allow people to interact with the computer during program preparation and execution. (In most conventional applications, the computer is used analytically to provide numeric or textual answers to problems that have already been solved, i.e., reduced to algorithms.)
- To bring the problem solver back into an intimate association with the computer, eliminating the inconvenience and confusion involved in dealing with a computer through an intermediary, the programmer, thus making the answer returned more often appropriate to the question asked.
- To increase both the quality of program design, and the quantity of output, from the professional programmer.
- To make it more convenient for the user who has a limited knowledge of programming to use the computer.

To make TSS/360 easy to use, the following features are included:

- Remote conversational terminals
- An on-line command system
- Conversational language processors
- A conversational program execution checkout subsystem
- Dynamic execution-time program linking
- A one-level store concept
- Protection and sharing for data and programs
- Large on-line storage for libraries of programs and data

To make efficient use of computer resources, TSS/360 employs:

- System scheduled multiprogramming
- Dynamic program relocation
- Partition capabilities

Command Repertoire

The commands included in TSS/360 are described in detail elsewhere and it would serve no purpose to detail them here. However, the following is an overview of that repertoire: TSS/360 provides commands for managing tasks, managing data, using language processors, controlling program execution, and tailoring the Command System.

The TSS/360 task-management commands let the user:

- Identify himself to the system
- End his task when he is through using the system
- Switch a conversational task to nonconversational mode
- Initiate a separate, independent non-conversational task
- Request any private devices needed for a nonconversational task
- Cancel a nonconversational task he has initiated
- Set a time limit for his task
- Restore his task to its initial status
- Augment the system's initialization of his task

The TSS/360 data-management commands let the user:

- Create, modify, and delete data sets
- Edit data sets by context or by line number
- Define partitioned collections of data sets (i.e., libraries)
- Define members of a group of data sets by relative generation numbers
- Catalog data sets
- Share data sets universally or only with specific individuals
- Qualify sharing privileges by catalog index levels and by individuals

The TSS/360 language processors let the user:

- Submit programs conversationally (i.e., at the terminal with line-at-a-time syntax analysis)
- Submit programs from prestored data sets
- Submit programs in TSS/360 Assembly Language or in FORTRAN IV
- Optionally submit programs to the TSS/360 Linkage Editor
- Specify a wide variety of language processor output options, including Cross-Referenced Symbol Listings and Internal-Symbol Dictionaries

The TSS/360 program execution-control commands and macro instructions let the user:

- Dynamically link programs at execution time (no prior link edit required)
- Specify program library search hierarchies
- Interrupt program execution and return to command-mode
- Nest program interruptions to any desired level
- Dynamically insert and remove program break-points
- Dynamically display the formatted contents of data locations, instruction locations, or CPU registers using source language symbols or FORTRAN line numbers
- Direct the output from dynamic display or dump commands to the terminal or to a data set
- Modify the contents of the user's address space using either source language symbols or hexadecimal addressing
- Restart an interrupted program at any convenient location
- Establish logical (i.e., true or false) conditions that allow or inhibit the execution of other dynamic debugging commands
- Utilize arithmetic operators, logical operators, and counters in constructing dynamic debugging statements

The architecture of TSS/360 is composed of the IBM System/360 Model 67 and a comprehensive programming system.

The logic of this programming system can best be understood if the environment in which it functions is first described.

This environment can be usefully examined from three points of view:

- Operational environment
- Mutiprogramming environment
- Hardware environment

OPERATIONAL ENVIRONMENT

A conceptual overview of the operational environment of Time Sharing System/360 is presented in Figure 1 from the point of view of an observer standing outside the system.

The TSS/360 programming system consists of application programs and service routines operating under the control of a supervisory program. TSS/360 provides many users concurrent access to a general-purpose computing facility in a conversational mode, coupled with nonconversational batch and bulk data handling programs. By calling on the facilities of the system, users can compile and execute programs, manipulate data sets and perform a variety of tasks.

This conceptual overview characterizes TSS/360 in terms of:

- The categories of system users
- The privilege and authorization classes assignable to system users
- The categories of tasks that may appear in the system
- The sets of programs that constitute TSS/360
- The categories of storage and devices that are supported by TSS/360

The TSS/360 concepts relevant to this conceptual overview and the facilities available to each category of system user are described in Concepts and Facilities, and only those that have a special relevance to system logic are reviewed here.

SYSTEM USERS

The administrative structure of a typical TSS/360 installation involves five types of people. These are called system managers, system administrators, system operators, system monitors, and users.

1. The system manager has overall responsibility for his installation. There is one system manager for each installation.
2. Each system administrator has administrative responsibility for a group of users. He grants those users permission to employ the system, and may withdraw that permission when necessary.
3. The system operator is responsible for operation of the computer and its peripheral devices. Although identified as one person to the system, normally three or more individuals serve as system operator -- one for each shift.
4. System monitors maintain the system and analyze and evaluate system performance. System monitors are usually customer engineers.
5. The user is anyone who employs the system. For example, he may be a programmer submitting a program for batch execution, or an engineer typing in requests from his terminal.

In addition to these five classes, an installation may set up others, with the functions of those new classes defined by that installation. Note that any one individual may carry out the functions of several classes.

PRIVILEGE CLASSES

A privilege class designates the right to use a specified set of command system commands. The time sharing system is delivered with five such classes defined, but provisions exist for the installation to expand this number up to 26. The five defined privilege classes and the individuals designated are:

TSS/360 Auxiliary Programs

1. SYSBLD/STARTUP Prelude
2. SYSBLD
3. STARTUP

TSS/360 Independent Utilities

1. DASDI
2. Dump Restore
3. Direct Access Device Dump
4. Mod 67 Core Dump
5. Error Recording Edit Print (EREP)

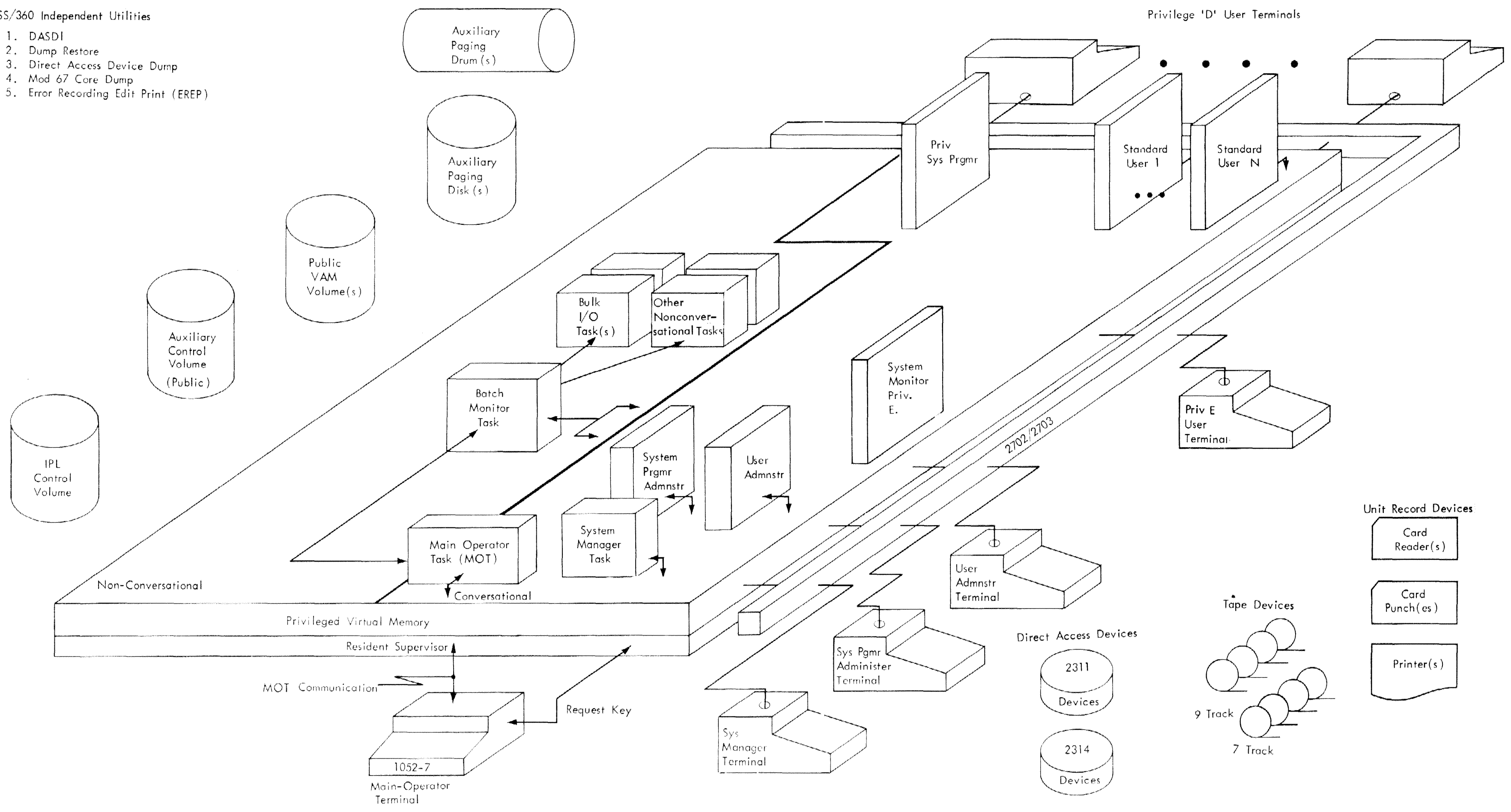


Figure 1. System Devices Available for Assignment

Privilege Class	Individual Designated
F	System manager
B	System administrator
E	System monitor
A	System operator
T	MTT administrator
D	User

The commands available to privilege classes F and B are described in _Manager's and Administrator's Guide_. Commands available to privilege class A are described in _Operator's Guide_. The commands available to privilege class D are described in _Command System User's Guide_. The privilege E commands appear in _IBM System/360 Time Sharing System: System Programmer's Guide_. The privilege class T commands are described in MTT Programming and Operation.

If anyone enters a command that is not available to his privilege class or classes, the system ignores the command and issues a diagnostic message. There are certain commands that are available to more than one privilege class.

AUTHORIZATION CODE

An authorization code designates the right to use a specified set of system programs, privileged SVCs, and macro instructions. The manager may specify an authorization code for each individual he joins. The code, however, has meaning only for privilege class D and E individuals, and designates one of the following:

1. U - a normal user; i.e., a user who has no direct access to system programs.
2. P - a system programmer, who has access to certain system programs and some privileged SVCs and macro instructions.
3. O - a privileged system programmer, who has access to all relevant system programs, SVCs, and macro instructions.

The system administrator may specify the authorization code for a normal user or a system programmer. Only the system manager may designate a privileged system programmer. A system programmer may be joined more than once in order to be able to operate at different times under different authorization codes. For a complete description of the capabilities permitted to system and privileged system programmers, see _System Programmer's Guide_.

TASKS

An operating system in which only one program at a time is executed needs relatively simple controls and concepts. However, in a multi-programming or time sharing environment, several programs may operate concurrently. The concept of "task" is introduced to provide for the orderly management of programs in this environment. For example, the association of a priority with a program would be confusing because the same program may be serving several tasks. Therefore, a priority is associated with a task instead of with a program. In TSS/360, a task may be described as an individual work requirement. For example, a task may be a terminal session in which a user compiles and executes two separate programs. Another example of a task is the compilation and execution of a program in the nonconversational mode.

A single task may call for the successive operation of several independently named programs. Also, a single program may be shared. That is, it may be used concurrently in support of two or more different tasks. Furthermore, a system user may have one conversational task and perhaps several nonconversational tasks active in the system at any one time.

Externally, a task is identified by a Task Identification Number (TID) or by a Batch Sequence Number (BSN).

Internally, the Resident Supervisor creates a Task Status Index (TSI) which contains the TID and a description of the task's characteristics and resources.

The actions that may result in the creation of a Task Status Index are:

- The LOGON command entered from a terminal
- ASNBD command
- PRINT command
- RT (read tape) command
- EXECUTE command
- WT (write tape) command
- PUNCH command
- STARTUP Routine (creates the Main Operator Task)

There are two types of tasks: conversational and nonconversational

Conversational Tasks

A conversational task is characterized by a user communicating with the system through a terminal. The user can enter his communications through the terminal keyboard or through the terminal card reader if one is available. Before a user can communicate with the system, he must have been granted access by a system administrator or system manager.

To begin using the time sharing system in conversational mode, all individuals, except the main system operator, must validate their authorization to use the system by means of the LOGON procedure.

The user dials up and presses the data button on the Data-Phone. Entry of the LOGON command from the terminal then initiates the creation of a conversational task.

As the user is identifying himself via his LOGON parameters, the system compares these parameters with those set up for him by a JOIN command previously issued by his system manager. After successful completion of LOGON, an underscore and backspace is issued at the user's terminal and the user is invited to enter his first command.

Immediately after the LOGON command routine has validated a user and before control is returned to the terminal, TSS/360 automatically invokes the ZLOGON command. As initially supplied with TSS/360, ZLOGON is a "null" command -- it does nothing. However, either an individual or his installation may redefine the ZLOGON command to perform any functions to augment the initialization of the user's task. For instance, further protection measures can be applied at this time, or a particular subsystem can be automatically invoked at this time. Thus, "null" commands are conceptually similar to the "user exits" frequently associated with general-purpose utility programs.

Each conversational task has a separate system input stream (SYSIN) and system output stream (SYSOUT). The system input stream contains the sequence of commands issued by the user. The system input stream can also include data dynamically supplied to user-written programs. The system output stream consists basically of system messages. However, it may also include messages from user-written programs or actual output data to be printed at the terminal. A terminal therefore serves as a combined SYSIN/SYSOUT device.

As an individual enters his commands (by typing them on his keyboard or by feeding them in punched-card form through the terminal card-reader) he becomes engaged in a dialog with the system. In general terms, he is told of the actions taken by the system in response to each command and, when necessary, he is prompted for additional non-defaultable information needed to complete an action, is informed of errors (if his command entry is either incomplete or incorrect), and is told of the options he may exercise in response to an error. Special care has been taken to make the types of options an individual may exercise appear consistent for all commands. Nothing, for example, is more frustrating to a user than to be required to resubmit an operand with delimiters in one situation and without delimiters in another.

The conversational task is normally terminated by the user issuing a LOGOFF or BACK command.

Nonconversational Tasks

There are many applications where dynamic communication with the system is not required. For such applications, nonconversational tasks can be set up.

Nonconversational tasks are just like conversational tasks, with the exception that the system can not directly converse with the user. However, the user can name a data set from which the system will obtain data whenever a response from a terminal would normally be required.

When a nonconversational task is executed, commands are taken from a command-procedure data set to direct program execution. Thus, the command procedure functions as the SYSIN for the nonconversational task. As such, it can also contain data required by the nonconversational task.

Most commands that can be entered from a terminal can also be invoked as macro instructions in programs or called from executing programs.

To minimize setup time, nonconversational jobs may be grouped prior to run time, and are made available via one input unit. For example, those nonconversational jobs requiring input from cards are grouped together by the Main System Operator and entered through the card reader.

To start the processing of input cards, the operator must first have created the BULKIO task. This could have been done at startup time or subsequently by means of the ASNBD command. At the same time the reader would have been added to the TSDL of the BULKIO task. Assuming that these operations have been performed, the operator need only load the cards in the hopper and start the reader.

When the reader is started, the asynchronous interrupt results in the activation of the BULKIO task which accomplishes the reading of the cards. If these cards contain a LOGON and a LOGOFF command, a SYSIN data set is created and the BULKIO task issues a request to execute the data set. When the Batch Monitor receives the request, it assigns a batch sequence number to the SYSIN.

The BULKIO task is unique in that it can recognize two types of data sets on card input. One type, signaled by a DATASET control card, is written into direct access file and cataloged. The other, which starts with a LOGON command and ends with a LOGOFF, is also written into direct access file storage and cataloged. BULKIO, additionally, will send a request to the Batch Monitor to initiate a new nonconversational task.

Optionally, nonconversational or batch jobs may be entered in EXPRESS or CONTROL mode. In EXPRESS or CONTROL mode (the terms are synonomous) several user jobs are combined and processed as one task. The total job stream is delimited by an EXPRESS control card and an END control card. Each user job or subtask is delimited by the LOGON and LOGOFF cards but full logon processing occurs only for the first subtask in the stream and full logoff processing for the last subtask. Logon and logoff processing for other subtasks in the stream consists only of minimum task cleanup and resource control and accounting.

Operation in EXPRESS mode precludes the creation of data sets by means of the DATASET control card. Any job requiring data sets while executing in EXPRESS mode can have the data sets created for them before the EXPRESS mode is entered.

EXPRESS mode also precludes the execution of jobs requiring private devices.

The actual processing of a job (such as a nonconversational FORTRAN compilation) is thus performed in another nonconversational task. The data set just read in will serve as SYSIN for the new task. This task is initiated by the Batch Monitor, assuming that a system maximum for such jobs has not been reached and the necessary system facilities are available.

Each nonconversational task has a corresponding SYSOUT. This SYSOUT must be printed at the completion of each job. The SYSIN data set created by BULKIO is assigned a data set name consisting of the USERID and the symbol SYSInnnn where nnnn is a unique identification number. The SYSOUT print is requested by the system via the PRINT macro during LOGOFF processing for the task. The PRINT macro, in turn, by means of the Bulk I/O Preprocessor routine, requests the Batch Monitor to inform BULKIO to print the SYSOUT. A PRINT task for a private data set is the only nonconversational task that does not require its SYSOUT to be listed because this would result in an endless loop.

In addition, any job requiring additional output on private unit record devices (such as Assembler output listings) requires another nonconversational task. This type of output to public unit record devices is performed by the BULKIO II task.

An overview of this relationship is presented in Figure 2.

SYSTEM PROGRAM STRUCTURE

Any control system performs two general categories of functions: (1) it provides the user with programs of general application, such as compilers and link editors, and (2) it provides services that allocate to operating programs the resources they require.

Most installation programmers may ignore the problem of resource allocation. From the point of view of the problem programmer, his own program will be executed using all those resources that he demands and, apart from restrictions and conventions imposed by the computation center, he is in no way limited as to what resources he can ask for.

It is primarily the systems designer and systems programmers who are concerned with defining the algorithms that apportion the limited real resources of the system, such as storage, devices, channels, control units, and even CPUs, among the operating tasks whose aggregate demand for resources may greatly exceed the amount of real resources available.

The intent is to enable the average user to work with an abstraction of the computer whose appearance is much simpler than the real computer.

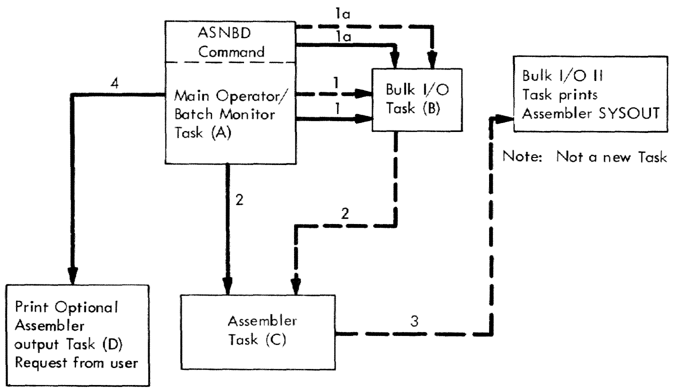

ASNBD Command
Main Operator/ Batch Monitor Task (A)
Bulk I/O Task (B)
Bulk I/O II Task prints Assembler SYSOUT
Note: Not a new Task
Print Optional Assembler output Task (D) Request from user
Assembler Task (C)
1a
1a
1
1
2
2
3
4

Figure 2. Inter-Task Relationships

Such abstractions are known as virtual machines or virtual computers. The idea behind virtual machines is not new, being the essence of all programming systems which tend to mask the real computer from the programmer. However, as computer operation becomes more complex, the virtual machine concept becomes increasingly important.

A virtual computer is an illusion, created by a combination of hardware features and programming systems.

A particular combination of hardware components and programming systems may be called a "level" of virtual computer. The proportion and sophistication of hardware components and programming systems used in creating levels of virtual computer and the degree to which any lower level is aware of higher levels may vary widely.

Within TSS/360 there are five levels of virtual computers or levels of abstraction. To each level, those levels below it appear as "hardware."

Level 4 is created when users deal with TSS/360 through the command system and language processors. Thus, for instance, it may appear to a user that he is dealing with a "FORTRAN computer."

Level 3 is the environment in which the language processors and user-written programs operate. This environment is defined as operating in the hardware problem state and the software "nonprivileged" or "user" state. A program operating in this level may address only that portion of the virtual storage assigned to his task that is assigned a user hardware storage protection key. It may execute any non-supervisory machine instruction, but only a restricted subset of the supervisor calls (SVCs) defined in TSS/360.

Level 2 is described as operating not in the software privileged state, but in the hardware problem state. A program operating at this level may execute any non-supervisory machine instruction and all supervisor calls (SVCs) defined in TSS/360. Programs operating at this level are assigned a protection key of zero, which makes them capable of accessing any allocated location in the task's virtual storage. This key differs from the key assigned to level 3 programs.

Level 1 is described as operating in the hardware supervisor state (defined as operating with bit 15 of the PSW set to zero). Programs operating at this level are capable of addressing all of real storage and any I/O device and are capable of executing any machine instruction.

Level 1 programs are not addressable from virtual storage and are not subject to time-slicing. Most level 1 programs are permanently resident in main storage.

Level 0 is the machine microprogram which operates in a special read-only storage which is not addressable by programming in normal operation.

The terms privileged state and user state deserve careful attention. Bit 15 in the PSW determines whether the CPU is in Supervisor or Problem state. In Supervisor state all machine operation codes are valid. In the Problem state an attempt to execute any of the privileged operation codes, (e.g., Start I/O, Load PSW, Set System Mask, etc.) will cause a hardware interruption. The problem state is divided into two software states - Privileged and User. Neither of these states may use the privileged operation codes.

Time Sharing System/360 is composed of four sets of programs, each designed to perform unique functions:

- The Resident Supervisor
- Task Monitor and System Service routines
- IBM-supplied problem programs
- Auxiliary programs

Resident Supervisor

The Resident Supervisor operates in level 1 in the virtual computer hierarchy and is responsible for allocating real system resources and for performing services in response to requests originating from the tasks in the system.

Each task appears to the Resident Supervisor as a virtual computer system. The Resident Supervisor is generally unaware of the fact that each of the tasks that it is managing may, itself, comprise several layers of virtual computer or program hierarchy.

The status of each task is maintained by the Resident Supervisor in Task Status records which describe the task as a virtual computer system. This includes information describing such resources as the virtual storage and symbolic devices assigned to the task.

This relationship of the Resident Supervisor to tasks and real system resources is depicted in Figure 3.

The Resident Supervisor is permanently resident in core storage after startup.

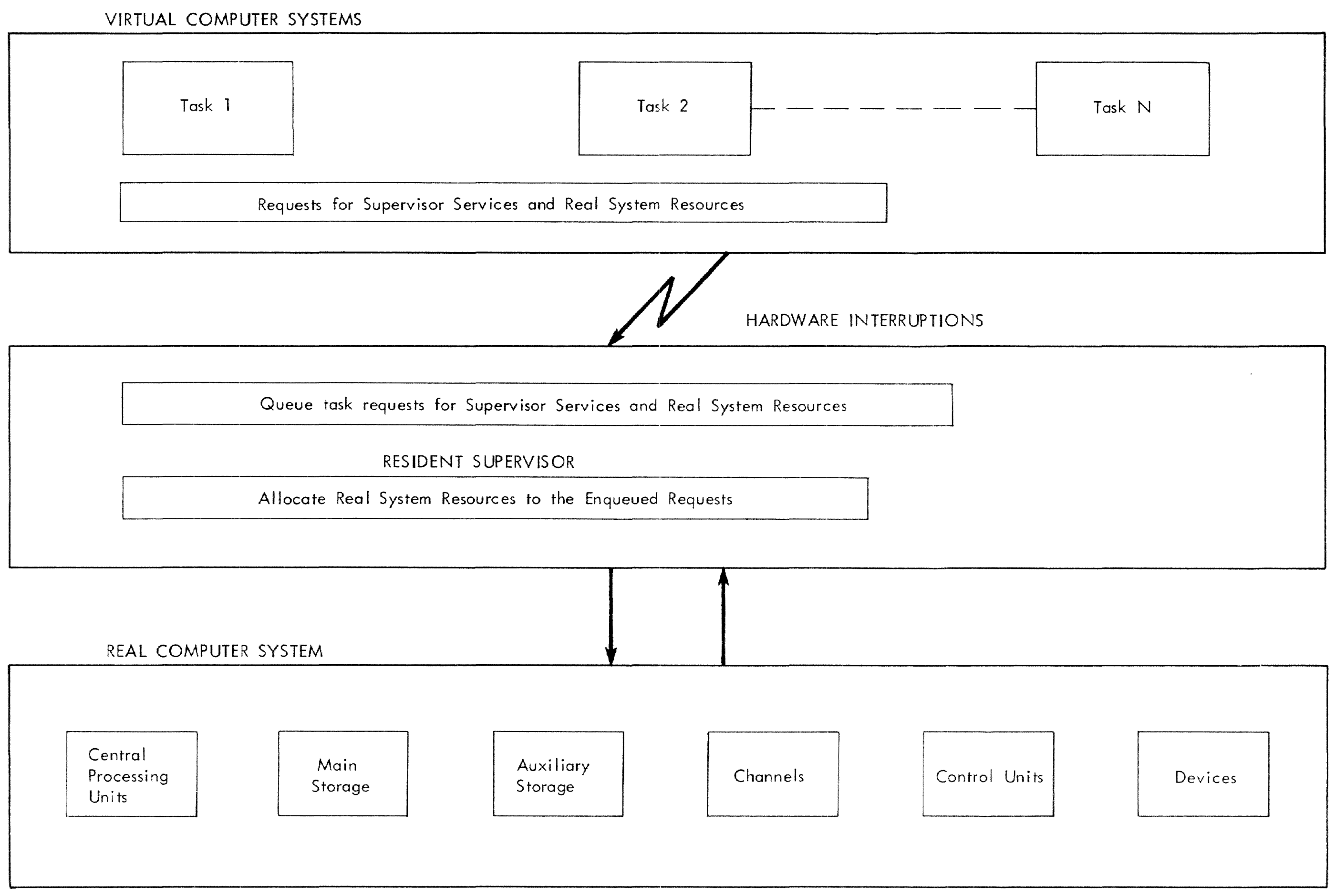


Figure 3. Relationship of the Resident Supervisor to Tasks and the Real System Resources

The Resident Supervisor is nonpageable; that is, it is not transferred back and forth between a paging device and main storage. The Resident Supervisor is nonrelocatable. Instructions within the Resident Supervisor have operands which are main storage addresses, not logical addresses. The Resident Supervisor operates in the Supervisor state; that is, the Resident Supervisor may execute privileged instructions. No location within the Resident Supervisor may be addressed by a program operating in virtual storage, because the result of dynamic address translation of virtual storage addresses will never be a main storage address within the Resident Supervisor.

The only entry to the Resident Supervisor is through a hardware interruption. The Resident Supervisor is not time sliced. Requests for Resident Supervisor services are represented by entries in queues. The Resident Supervisor runs until it can find no more work; that is, all the queues have been examined and emptied out if possible. When there is no more work that can be processed, a task is selected to be placed in execution.

Task Monitor and System Service Routines

This section discusses the reasons for including a Task Monitor, a set of system service routines, and a privileged state in TSS/360.

The basic function of a control program is tc control the real system and to provide services to tasks. The control program may itself be either entirely resident or nonresident. A nonresident control program could have been employed by TSS/360 to provide both task and system oriented services for each task that gains control. However, such a control program is inefficient because the time spent in reading sections of the control program into main storage generally would not be overlapped by processing.

On the other hand, a resident control program that provides both task and system services would permanently occupy a very large amount of main storage. Some of this storage would be occupied by infrequently used routines.

In either of these cases, it would be difficult to modify the system without simultaneously making obsolete many of the object programs contained in the system's libraries.

There are many possible resolutions to the question of which programs are to be resident and which are to be non-resident. In a paging system, an additional division is possible, for a program may operate using virtual addressing and still be locked into main storage through an appropriate use of the main storage allocation tables.

A resolution of this dilemma is to separate control program functions into a resident portion and a nonresident portion. In TSS/360 this resident portion is the Resident Supervisor, which creates a multiprocessing, multiprogramming environment and provides services for the system as a whole. The decision to make the Resident Supervisor operate in the non-relocation mode was based upon the efficiency resulting from eliminating dynamic address translation overhead and upon the increased protection resulting from the fact that no location within the Resident Supervisor may be addressed by a channel operating upon a task I/O request or a program operating in virtual storage. On the other side of the coin, the decision to operate the Resident Supervisor in non-relocation mode slightly increases the complexity involved in making certain portions of the supervisor nonresident.

The nonresident or pageable portion consists of a collection of modular virtual storage service routines under the control of the Task Monitor. This portion provides task oriented system services; that is, those services that are not immediately dependent upon the hardware resources of the system.

The Task Monitor and most task service routines can each be separated into a processing part that is common for all tasks and a part containing data that is unique to each request for service. All tasks share the common processing parts (reentrant code). Each task operates with unique copies of the data parts of service routines which the task invokes.

The Task Monitor and its associated service routines act as if they constitute a Resident Supervisor for each task. That is, they process requests for task services (such as data management services) and requests for virtual system resources (such as virtual storage or symbolic devices).

This analogy is quite extensive. For example, the Task Monitor receives control through what appears to it to be a hardware interruption complete with simulated old and new Program Status Words and gives control to programs in the User state by executing a macro instruction that is the equivalent of a Load PSW instruction. In both cases, this illusion is created by the Resident Supervisor. Figure 4 shows schematically the relationship between the Task Monitor and the Resident Supervisor in creating this illusion.

The analogy which compares the interface between the Resident Supervisor and the System/360, on the one hand, and the interface between the Task Monitor and the Resident Supervisor on the other, is quite extensive, but it is not exact.

Because programs operating in virtual storage may use full-word virtual addresses, it was not possible to make the Resident-Supervisor/Task-Monitor interface look like a System/360 Model 65. It would have been possible to make the interface look like a System/360 Model 67 so that, for example, the Task Monitor could run as a stand-alone program on a Model 67. However, because the Model 67 is not entirely compatible with other System/360 CPUs when operating with an Extended PSW, it was felt that there was an advantage in customizing the interface to facilitate, among other things, internal sharing of programs and data and intertask communication.

This design has led to the creation of a software state called the Privileged state (level 2). The Task Monitor and most System Service routines operate in the Privileged state. The Privileged state was created to prevent the Task Monitor and System Service routines from accidental destruction and thus to prevent the ordinary system user (authority code U) from interfering with other users on the system. Programs belonging to the ordinary system user execute in the nonprivileged state (level 3).

The system functions that support the privileged state are:

- In a user's virtual storage those pages that are allocated to privileged routines and their associated tables and work areas are assigned a storage protection key that differs from that assigned to problem programs (i.e.,

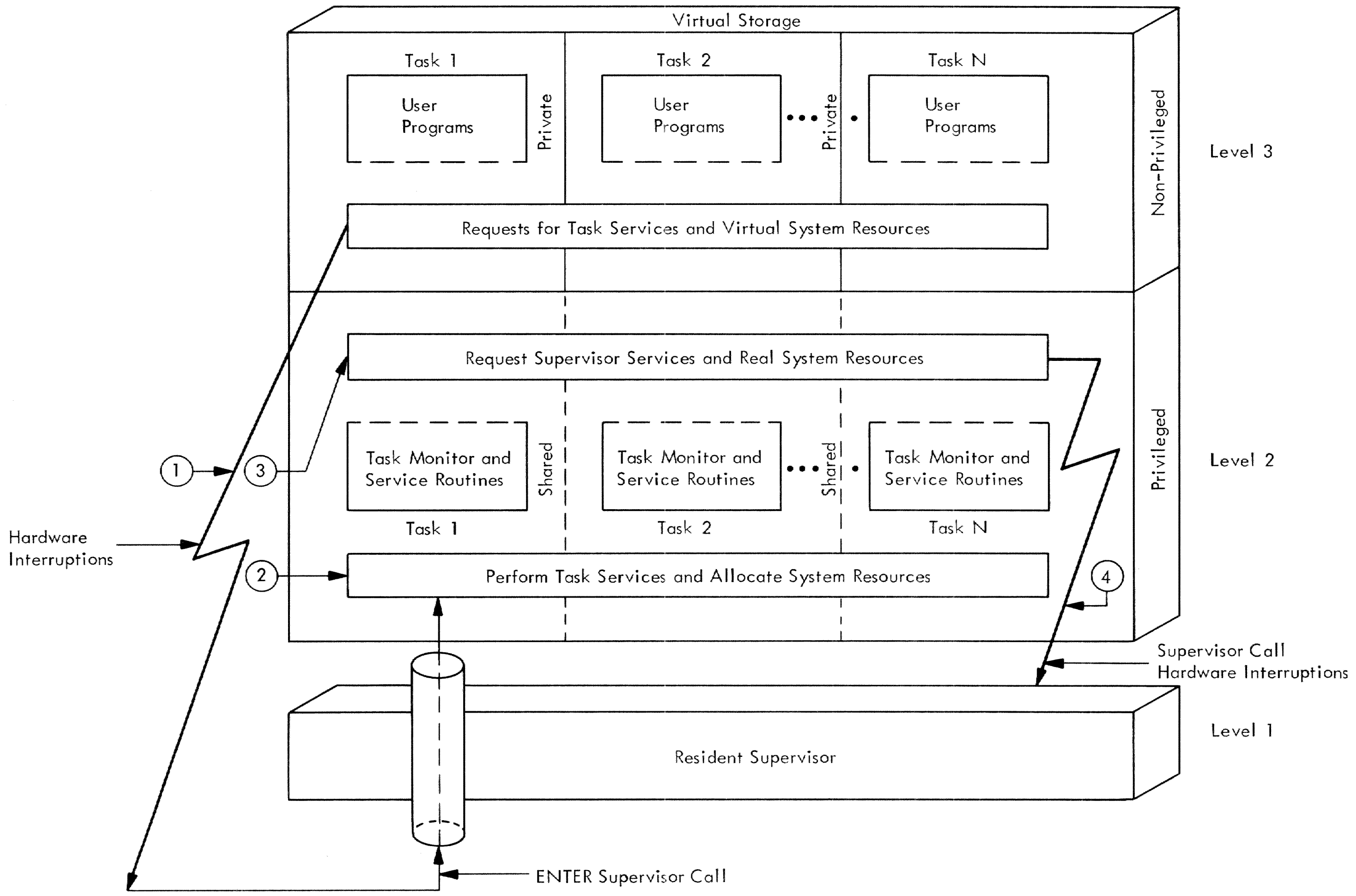


Figure 4. Relationship of the Task Monitor and System Service Routine to Nonprivileged Machines and the Resident Supervisor

level 3 programs). This key will cause a storage protect interruption if the privileged part of a task's virtual storage is addressed by a problem program. Privileged routines, on the other hand, can address all of the task's virtual storage.

- The Dynamic Loader will not treat modules from a problem programmer's library as privileged routines. In this fashion, a problem programmer cannot cause his own version of a system routine to be loaded and executed as a privileged routine.

- A problem program normally requests system services through Supervisor Calls (SVCs) which are contained in macro instruction expansions. In TSS/360 these macro instructions collectively are called the Extended Instruction Set. Many TSS/360 Supervisor Calls affect data in system tables. Erroneous information in system tables may cause incorrect system operation. Therefore, when a TSS/360 problem programmer requests virtual system services his macro instruction issues an ENTER SVC. In response to the supervisor call, the Resident Supervisor will create a simulated interruption that will cause a privileged system service routine to be invoked. The privileged routine can then determine if the user's request is valid. If it is valid, the privileged routine may then invoke other TSS/360 supervisor calls while in the process of performing services. If the request is not valid, the request will be rejected, thus preventing a nonprivileged routine from causing incorrect system operation.

The reason for communicating between problem and privileged state via the Resident Supervisor is that only the Resident Supervisor can execute the privileged instruction that alters the PSW Protection Key field.

The virtual storage system services provided by TSS/360 fall into two general categories: (1) shared by all tasks, and

(2) located in independent tasks. The distribution of system services between these two categories has been made on the basis of considerations such as frequency of use. When a function is shared by all tasks, a delay may occur while one task awaits the resetting of a programmed interlock set by another task. When a task uses a function located in a separate task, a delay may occur if the task must await the completion of that function.

System Services shared by all tasks are:

Data Management services
Dynamic Loader services
Catalog Management services
Virtual Storage Allocation
External Storage Allocation
Command System
Program Control System
Private Device Management services
Servicing of task-oriented interrupts by the Task Monitor

The System Services that operate in independent tasks are:

Main Operator Control Program
Batch Monitor
Bulk I/O service routines
System Edit program
On-Line Test System

IBM-Supplied Problem Programs

IBM-supplied problem programs also reside in virtual storage and are time-sliced. The only major distinction between these programs and programs written by any nonprivileged user of the system is that he cannot modify these programs (as he can his own programs); he can only transfer control to them.

IBM-supplied problem programs are:

- Language processors, such as the FORTRAN Compiler and the Assembler.
- Linkage Editor, which allows users to combine and delete portions of program modules.

Auxiliary Programs

These programs, with the exception of the Time Sharing Support System, run as stand-alone programs and are primarily designed to aid in the creation and maintenance of a running TSS/360.

Time Sharing Support System

TSSS runs with minimum TSS/360 support and provides system analysis facilities for the system programmer.

Utility Programs

Direct Access Device Initialization (DASDI)
Dump and Restore a direct access device [DUMP/RESTORE]
Print the contents of a direct access device [DADUMP]
Stand-alone Core Dump

System Build Program

SYSBLD is a resident, standalone utility which operates outside the TSS/360 environment.

STARTUP Program

Startup performs the initial allocation of main storage and, thereafter, the Supervisor controls its allocation (see "Main Storage Allocation").

System Storage

The three categories of storage are main storage, auxiliary storage, and external storage.

Main Storage: Main (core) storage is the only storage in which programs can be executed. It is initially allocated in 4096 byte units called pages, although programs (including the Resident Supervisor) may subdivide pages.

Auxiliary Storage: Auxiliary storage is storage that is set aside primarily for the temporary storage of main storage pages during system operation.

The devices used for auxiliary storage are specified during System Generation and must be on-line during TSS/360 operation. The devices may be either IBM 2301 drum storage or IBM 2311/2314 disk devices and fall into two categories that form an auxiliary storage hierarchy during normal operation: primary paging device and auxiliary paging devices.

Auxiliary storage is arranged in a modified virtual access method format and contains a bit directory, which describes the availability of each page.

Auxiliary storage is allocated initially by Startup and thereafter by the Resident Supervisor (see "Auxiliary Storage Allocation").

External Storage: External storage falls into three categories, which are designated during system generation and system startup: public, private, and system.

Public Storage

Public devices are those devices set aside for public storage contained on public volumes.

Public devices are direct access devices which are defined at system startup time. Most direct access storage devices, such as 2311s, have removable packs. In TSS/360, the number of concurrent users is much greater than the number of separate volumes available. Consequently, most of the available direct access drives are designated as public storage devices; their packs are not removed during normal system operation. Each public device on the system is made available to each user's task when his task is initialized. Each public volume is formatted to contain data sets that can be processed by any of the Virtual Access Methods (VAM).

Public storage contains only cataloged data sets. These data sets may be private or sharable. Public storage is allocated by the External Storage Allocation system service routines.

Private Storage

Private devices are devices that are set aside for mounting private volumes. Private volumes need not be on-line when requested. The use of private volumes, with associated mounting and demounting, plays a minor role in TSS/360 operations. A private volume may be formatted to contain data sets that can be processed by either the Virtual Access Methods or the Sequential Access Methods.

Private devices are requested by the conversational user through the DDEF command and by the nonconversational user additionally through the SECURE command. Private volume mounting is requested by the DDEF command. The allocation of private devices and the management of requests for private volume mounting and demounting is performed by the Device Management system service routines (see "Device Allocation"). Storage allocation within volumes is performed, when necessary, by the External Storage Allocation system service routines (see "External Storage Allocation").

Private storage may contain cataloged or uncataloged data sets. These data sets may be private or sharable (see "Sharing" for qualifications).

System Storage

System storage consists of two devices designated during System Generation as containing those system data sets required for starting up and running TSS/360. The two system volumes are Initial Program Load (IPL) Control Volume and Auxiliary Control Volume.

Both volumes must be in a VAM format. After System Generation, the contents of these volumes may be modified only by the SYSBLD (System Build) program, or System Maintenance. Unused space on the Auxiliary Control Volume is available for allocation as part of the system's public storage.

The contents of these two volumes are as follows:

IPL Control Volume: The first three records of track 0 must contain the IPL Control Record, TSS/360 Startup Prelude, and the Volume Label. The system data sets that normally reside on the System Residence or IPL Control Volume are:

TSS*****.SYSCCB.GxxxxVyy (System Configuration Control Block) - A data set used by Startup containing description of the system configuration.

TSS*****.STARTUP.GxxxxVyy (System Startup Program) - A data set containing the text of the Startup Program. Organized as a partitioned data set, but with a single member. Partitioned Organization Directory (POD) is ignored by the Startup Prelude which reads in Startup (see "System Generation and Maintenance").

TSS*****.SYSBLD.GxxxxVyy (System Build Program) - A data set containing the text of the SYSBLD program. Organized in a fashion similar to TSS.STARTUP (see above).

TSS*****.RESSUP.GxxxxVyy (System Resident Supervisor) - A partitioned data set whose members are the modules of the Resident Supervisor. These modules are link-loaded by Startup into available main storage. Presence in this data set is a necessary but not sufficient condition for such link-loading because Startup will load only those modules whose names are included in a special member of RESSUP named LOADLIST and which have not been loaded during a STARTUP library search.

TSS*****.SYSIVM.GxxxxVyy (Initial Virtual Memory) - A data set consisting of those system modules that are automatically provided for each user's virtual storage at task initiation time. These modules are link-loaded by Startup and written onto the primary paging device. Startup will load only those CSECTs whose names are included in a special member of SYSIVM named LOADLIST (see "Initial Virtual Memory").

TSS*****.APGENX - A data set consisting of commands to the linkage editor. These commands cause the linkage editor to replace skeletal control blocks built by STARTUP with new versions containing installation parameters defined by means of SYSGEN macro instructions.

Auxiliary Control Volume: The system data sets that will normally reside on the auxiliary control volume are:

TSS*****.SYSCAT (System Catalog) - A partitioned data set (one member per system user) containing pointers to cataloged data sets and information on ownership and sharing privileges for such data sets (see "Catalog").

TSS*****.SYSLIB.GxxxxVyy (System Library) - A collection of public modules that are automatically eligible for loading into each user's virtual storage by the dynamic loader (see "Object Program Libraries").

TSS*****.SYSMAC.GxxxxVyy (System Macro Library) - An index sequential data set containing definitions of the necessary system macros to support normal nonprivileged user assemblies (see "Symbolic Libraries").

TSS*****.MACNDX.GxxxxVyy (System Macro Index) - data set that is an index to TSS.SYSMAC (see "Symbolic Libraries").

TSS*****.ASMMAC.GxxxxVyy (System Assembler Macro Library) - An index sequential data set containing definitions of all system macros not defined on SYSMAC. This includes system macros available only to system programs as well as nonprivileged macros limited to system programmer operations because of their function. This data set must be specified as the second macro library when assembling system modules (see "Symbolic Libraries").

TSS*****.ASMNDX.GxxxxVyy (System Assembler Macro Index) - A sequential data set that is an index to ASMMAC (see "Symbolic Libraries").

TSS*****.USERLIB (System User Library) - The user library for the privileged system programmer user with identification "TSS". A user library is created for each authorized user of the system at JOIN time and is organized as a virtual partitioned data set.

TSS*****.SYSUSE (System User Table) - An index sequential data set containing the user ID, password, charge number, control counts governing the limit of system resources available to the user, accounting statistics, and privileged attributes for each currently authorized user of the system. This data set is maintained by the system administrator JOIN and QUIT commands (see "System Operator and Administrator Services" and "Resource Allocation and Control").

TSS*****.SYSLIB (SYSMLF) (System Message Table) - An index sequential data set, with keys identical to message numbers, containing system diagnostic/prompting messages.

SYSOPER0.SYSLOG.GxxxxVyy (System Operator Log) - A generation data group in which each virtual sequential data set contains a record of system-to-operator and operator-to-system communications for a startup to shutdown session. At each startup, a new SYSLOG data set is defined. Any extra SYSLOG data sets (above a maximum of 10) will be erased.

MULTIPROGRAMMING ENVIRONMENT

The TSS/360 multiple access environment differs from a conventional batch multiprogramming environment in significant ways.

In contrast to conventional systems where work is performed on only a few tasks in a specific period of time, and where only one system input (SYSIN) need be defined in a multiple access system at any one time, many such tasks are concurrently processed, and many such SYSIN sources are defined.

Because of this environment, it is essential that a time sharing system be able to multiprogram effectively among a large number of tasks.

The features that assist TSS/360 in multiprogramming among a large number of tasks are time-slicing and dynamic address translation.

TIME-SLICING

TSS/360 allows many users concurrent access to the system by granting each user, at frequent intervals, a portion of computer time called a time-slice. During his time-slice, the user has the resources of his virtual computer made available to him. Because these time slices occur frequently, the user can generally operate as though he alone is using the system.

The idea of providing users with intervals of computer time one after the other is not new. Conventional multiprogramming systems allow a task to be active in the system until it must wait for system address segment 3, page 5, byte 256. At the beginning of a time slice, the segment and page tables which describe that user's virtual storage will be loaded into main storage by the Resident Supervisor. The location of the first segment table entry is stored into a special register, called the table register, by the Resident Supervisor. This table register is one of 16 control registers provided in the Model 67 hardware.

In the example in Figure 5, the table register has been loaded with the number 8,192 (2000_{16}), representing the main storage location of the beginning of the user's segment table. The Dynamic Address Translation (DAT) unit then locates the segment table in main storage using this address and, since we are searching for segment 3 and each segment table entry is four bytes, the hardware can automatically go to core location 8204 ($200C_{16}$) to find the location of the page table for segment 3. The example shows that the page table is at main storage location 16,384 (4000_{16}). The DAT unit thus can go to location 16,494 ($400A_{16}$) to find the fifth entry. This is the entry for page 5 of segment 3. The entry in the page table shows the beginning of that page is in main

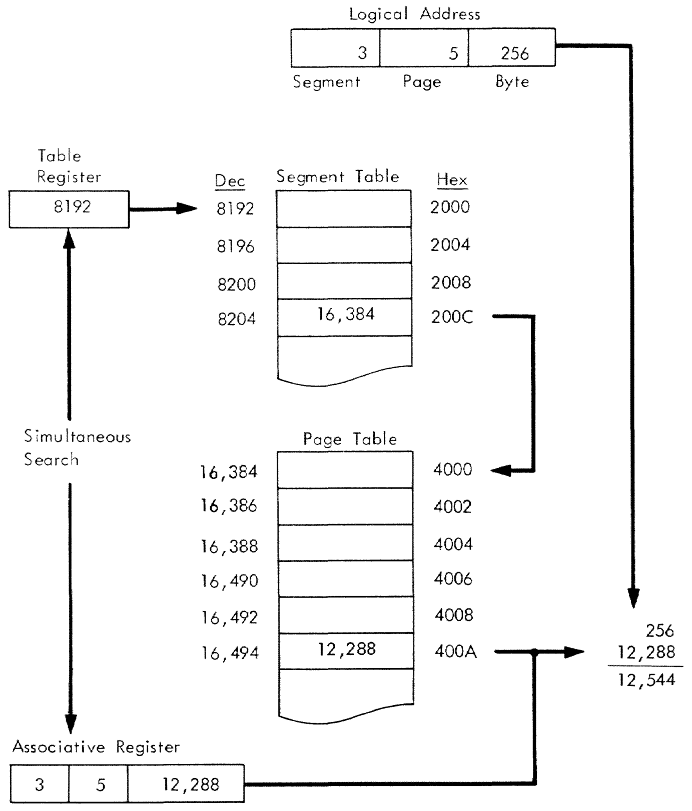


Figure 5. Example of Dynamic Address Translation

storage location 12,288 (3000_{16}). For byte 256 (100_{16}) of that page, the hardware has only to add that to the main storage location to find the actual data location 12, 544 (3100_{16}).

Figure 6 shows the general scheme of dynamic address translation, which follows the same logic as described in this example.

In performing dynamic address translation, the hardware uses binary arithmetic. Thus the finding of the third 4-byte entry in the segment table, for example, involves a binary shift of two rather than multiplication by 4. In a similar fashion, page table locations are found by a binary shift of one, and the actual address is constructed simply by appending the displacement to the block address obtained from the page table.

Even though such binary shifts are employed, it can be seen that this double table look up would be time consuming if it were necessary to do this for every address. However, it is likely that once having found the actual core location of segment 3, page 5, the user's program will reference this again, perhaps many times, during the time slice. The full translation time of 2.1 microseconds then can be avoided by making a "scratch pad memorandum" of the fact that segment 3, page 5 is actually located at position 12,288. This memorandum is made in one of eight associative registers. The registers are termed associative since they are content addressed.

For example, the next time the system attempts to translate this address, the DAT unit will begin a simultaneous search of all eight associative registers in addition to the regular table search outlined above. A successful search for segment 3, page 5 can be accomplished in just 150 nanoseconds regardless of which of the eight registers might contain the address. In general, the associative registers will hold the addresses of the last eight translations and a table lookup translation of logical addresses referencing these pages will not be necessary.

Because instructions tend to be executed in linear sequences, the Model 67 maintains both a translated and an untranslated Instruction Counter. Whenever the sequence of instructions crosses a page boundary or whenever a branch type instruction is executed, the DAT unit is used to obtain a main storage address from the virtual storage address in the untranslated Instruction Counter. Otherwise, the main storage address in the translated Instruction Counter is utilized, thus minimizing

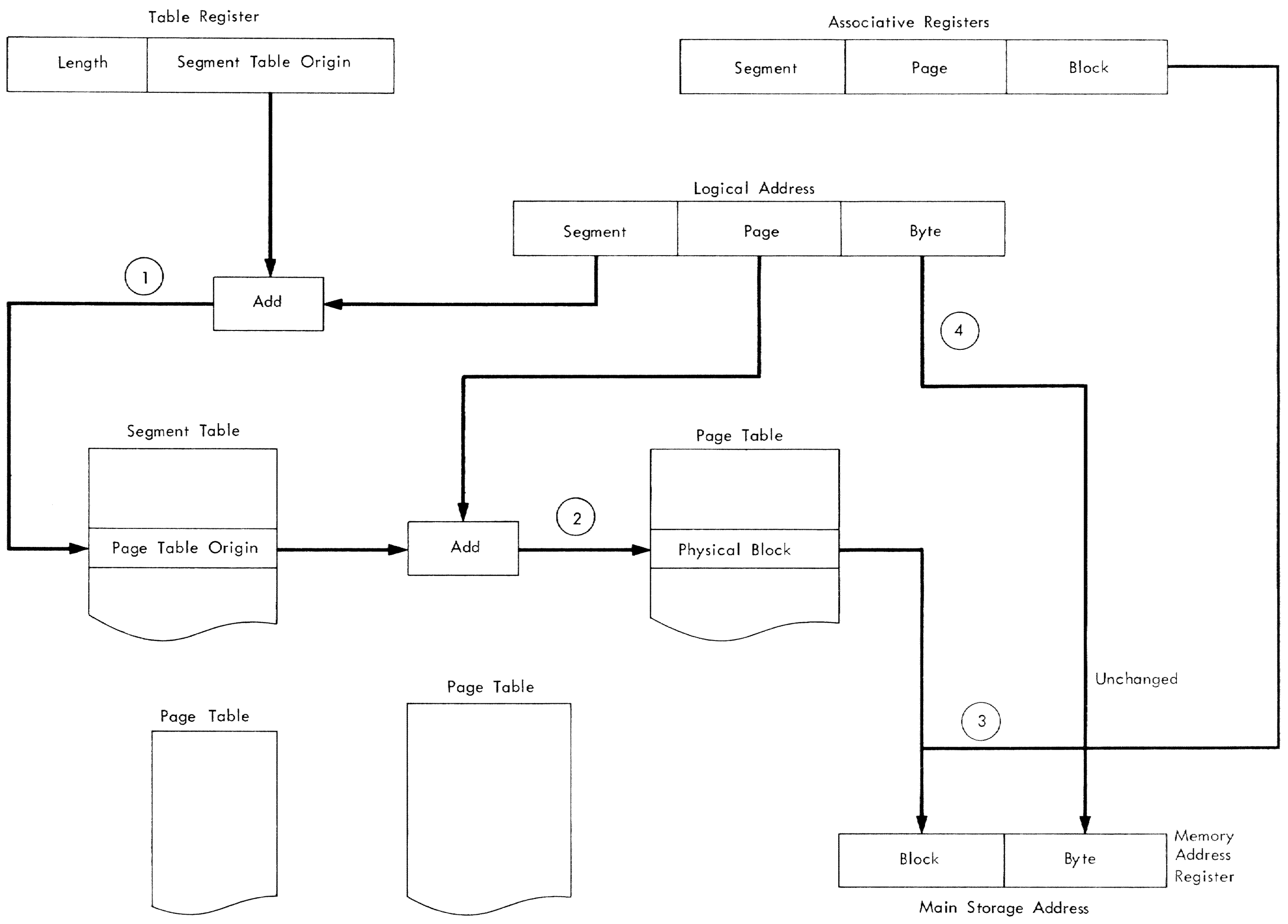


Figure 6. Dynamic Address Translation Process

the time spent in translating the Instruction Counter.

The mapping of virtual storage to main storage just described also permits the relocation of a program between time-slices. In the Model 67 hardware, additional bits in the storage protection keys provide the ability for the system to determine whether pages have been referenced or changed. At the end of a time-slice, all of the task's changed pages are written out onto the auxiliary drum or disk. Both changed and unchanged pages are also retained in main storage where they may be overwritten should some other task need more pages than the supervisor has available. This paging operation may take place in reverse when a task's new time-slice begins. When a page that has previously been written out is returned to core, there is no necessity for the page to be returned to exactly the same place it was before. The page may be relocated by having the supervisor program post the new location in the task's page tables. In the preceding example, segment 3 page 5 may be returned to main storage location 24,576 rather than 12,288 and the supervisor has only to change the entry for page 5 of segment 3 of the page table illustrated.

Thus, any future translations will map segment 3, page 5 into location 24,576. Main storage thus becomes fragmented -- at any one time holding a number of pages from several recently or currently active programs. However, the ill effects resulting from main storage fragments of unusable size is avoided, because fragments in main storage, auxiliary storage, and external storage are all of a uniform size.

Implied by the above description of the paging process is the ability of the system to keep track of pages that have currently been written out to auxiliary storage or that have never been brought into the system at all. Associated with each page is an external page table which, for each page of the user's virtual storage, holds the drum or disk address associated with it. Thus, if page 5, segment 3 were to be written out of storage location 12,288, the associated external page table entry would be noted with the temporary drum or disk location of that page. The page table entry itself would be marked to indicate that a good copy of the page may no longer be available in main storage.

When the next time-slice begins and the first reference is made to segment 3, page 5, the dynamic address translation mechanism above would find from the page table that this page was not available. A program interruption is then generated so that the Resident Supervisor may take care of the situation. The supervisor, in turn, examines the external page table associated with segment 3 to find the present location of the page, and initiates a paging operation to bring that page back into main storage so that the task may proceed.

During this period of time the task is said to be in "page wait" and another task is given control of CPU, just as in a conventional multiprogramming environment. Page wait is actually just another kind of I/O wait. When the page is brought into main storage, the page may be relocated as previously described. With the completion of the paging operation the task is now ready to proceed again.

During the initiation of a paging operation, it may be found that the main storage image of the referenced page has not been overwritten or reassigned. In this case, the main storage page is reclaimed by the task and no page-in is necessary.

The technique of grouping pages into segments has been used to provide:

- A convenient way of sharing programs or data among tasks. A common page table can be pointed to by segment table entries for several different tasks and each such page table can be restricted to be shared among different groups of users.

- A way of reducing the amount of <u>contiguous</u> main storage needed by the relocation tables. The Segment Table must reside in a contiguous block of main storage, and each Page Table must reside in a contiguous block of main storage. The Segment Table and each Page Table, however, may reside in different blocks of main storage.

- A way of reducing the amount of main storage required to contain page tables. The Dynamic Address Translation unit requires that a task's Segment Table resides in main storage while the task is executing. However, the page tables for a task need not be kept in main storage. A page table for a segment could be brought into main storage when a virtual storage address within the segment is referenced.

- A convenient way to allocate a contiguous data area of unknown length. Only the area at the beginning of a segment need be allocated. Then data and page tables can dynamically expand and fill the segment as required.

A segment contains 256 pages. This number of pages was selected in response to several considerations.

Since there are 4096 bytes in a page, 12 bits are required for addressing bytes within a page. The remaining 12 bits of the logical address may be divided between segments and pages. Because data is transferred within the Model 67 and its Dynamic Address Translation Unit in multiples of four bits, the only possible choices in the 24-bit version are to divide the remainder of the logical address into either 16 segments of 256 pages each or 256 segments of 16 pages each. Then the 32-bit version would have a maximum of 4096 segments in the former case and a maximum of 65,536 segments in the latter case. A large segment size minimizes the possibility that a contiguous data area of unknown length will grow to exceed the space reserved. On the other hand, a large number of small segments would act to increase significantly the amount of main storage occupied by tasks' segment tables.

The former alternative was chosen; that is, each segment contains 256 pages. Thus, the 24-bit version has a maximum of 16 segments, and the 32-bit version has a maximum of 4096 segments. The way in which these segments are allocated is discussed in "Virtual Memory Allocation."

In summary, the virtual storage concept of the Model 67 is implemented by a paging mechanism. This mechanism combines a hardware device called the Dynamic Address Translation Unit with programming in the Resident Supervisor to map virtual storage addresses into main storage addresses. The programming also keeps track of the original or the temporary drum or disk locations of each page and permits the relocation of pages as they are written out and then read back in again. The net effect is to ensure that generally only the pages demanded by a task are actually in main storage at any one time, and that many tasks may have such pages in main storage at one time. The entire paging mechanism is not apparent to the programmer or user. He deals only with a virtual machine and is not directly concerned with what the hardware and software do to translate the virtual machine into the real machine.

In a time sharing environment it is expected that users will most frequently be concerned with solving a problem easily and will only infrequently be concerned with improving the efficiency of object program execution. However, from the point of view of efficiency, it is well to remember that virtual storage is not the equivalent of main storage in the sense that a program is completely resident during its execution. As the preceding example of dynamic program relocation has shown, virtual storage is implemented by an "automatic" or system controlled overlay scheme based on the movement of page size fragments into and out of main storage. Therefore, when efficiency of execution is an important consideration, the TSS/360 user must take into account this characteristic of virtual storage just as the user of a conventional system must plan overlays for the efficient execution of large programs.

CONFIGURATION

Configuration refers to the number and interrelationship of the various components in a computer system. The time sharing system has been designed to operate in a wide variety of configurations so that the requirements of many different types of installations may be satisfied. The simplest type of installation may have only a few users and it therefore requires only one processor and a few input/output devices. In such an installation, changing processing requirements may be met by increasing or decreasing the number of processors or storage units or input/output devices. An installation having a large number of users may require two processors as well as several storage units and many input/output devices. Changing processing requirements in this case may cause the number of processors, in addition to the number of storage units and input/output devices, to change. The interconnection of system components has been designed so as to make the Model 67 easily adaptable to any required configuration. Configurations may be controlled by means of a configuration console as described in the section on system partitioning.

Interconnection of System Components

A simple interconnecting structure for a series of processing units and associated storage units is the crossbar switch arrangement shown in Figure 7.

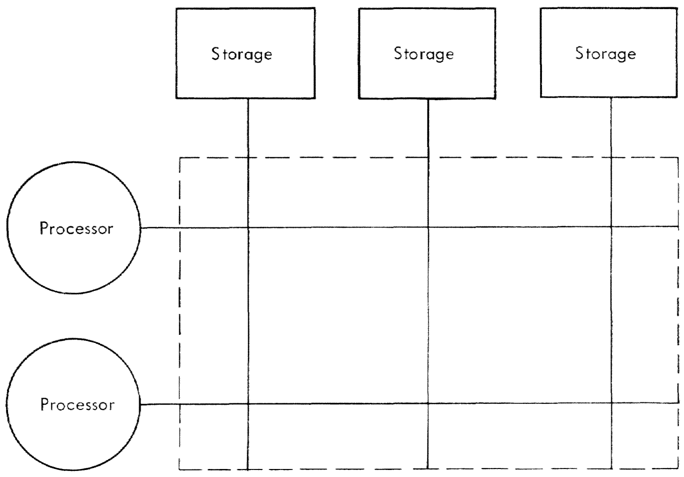


Figure 7. Crossbar Interconnections of System components

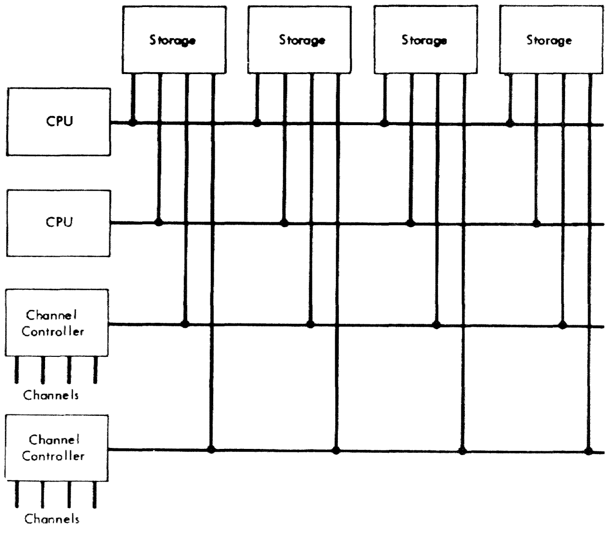


Figure 8. Distributed-Switching Interconnection of System Components

For the time sharing system, however, the switching arrangement is not a centralized switching point, but is distributed among the various components that form the system (see Figure 8).

The reasons for adopting the distributed approach are improved availability and flexibility. A centralized piece of equipment represents a crucial link within the system, since its failure would cause the entire system to fail. It would be costly of System Components to remedy this by duplicating equipment. The distributed approach is more flexible than the crossbar approach and facilitates extension of the system by addition of more processors or storage units.

The interconnecting structure shown in Figure 8 shows that the necessary switching equipment is distributed among the system components, central processor, channel controller, and core storage units. Drivers are added to the processors as required, and the equipment needed for selection of any of the processor bus systems is added to the core storage units. Instead of the single bus connection (tail) of the simplex system, multiple tails are provided for a multiprocessing system. The design of the circuits that select from among the tails is such that if a storage unit should fail (including power failure), the other storage units connected to the storage bus would remain operative. When the bus control unit within a processor fails, the entire bus driven by this unit is, of course, inoperative. This bus will not, however, prevent proper functioning of the storage units with the remaining processors.

I/O channels within a multiprocessor system are flexible in the System/360 design. The channels perform the transmission function. All controlling functions for I/O devices are placed within the control units for these devices. Thus, the channel design is general and applies to all I/O devices.

As with the processors and core storage units, a control unit can be attached to more than one channel by means of multiple tails. The switching equipment is modular, a design that avoids the problem of cost and poor reliability of centralized switches.

A channel operates concurrently with the processing unit and may be treated as an independent entity within the system. As shown in Figure 8, the channels may be grouped into two sets, each provided with its own channel controller; thus there are two systems for I/O operations. Storage units are equipped with the necessary tails to accomodate the additional busses. Channels are addressable by either processor and can return their interruptions to either unit.

System Partitioning

There are three major machine configurations in which the time sharing system may operate: simplex, duplex, and half duplex. A simplex configuration is characterized by the presence of one processor. A duplex configuration is characterized by the presence of two processors, a configuration console and one or more channel controllers. A half duplex configuration is characterized by the presence of one processor, a configuration console and one channel controller. A half duplex configuration may be part of a duplex configuration that is separated or partitioned by means of the configuration console.

The configuration console has been designed to allow a duplex configuration to be partitioned into a variety of half duplex configurations. The design is based on the distributed interconnection method. Configurations are determined by the setting of switches. A maximum of eight core storage units may be connected to a maximum of two channel controllers and a maximum of two processors. The address range (multiple of 256K bytes) of core storage units connected to the same processor may be set at the console. The connection of input/output control units to channel controllers may be set. A switch is included which indicates which channel controller is to be addressed by a processor during initial program load (IPL). This switch is necessary because IPL sets the processor to standard PSW mode and in this mode the processor addresses channel controller 0. If channel controller 1 is to be addressed by a processor during IPL, the switch must be set accordingly for the processor. (See section on "Processor Time Sharing Features".)

Figure 9 shows a sample simplex configuration. Figure 10 shows a sample duplex configuration. Each circled X represents a partition point. A partition point is controlled by a switch on the configuration console. The setting of a configuration switch is indicated in a control register and may be sensed by a program using the Store Multiple Control instruction.

In a duplex configuration each channel controller may be controlled by any processor in the system via busses connecting the two. A partitioning switch either permits this control or causes the channel controller to ignore any attempt by this CPU to initiate an I/O function. When the processor-to-channel controller path is inactive, commands to the channels and devices attached to that channel controller are not executed and causes condition code 3 to be set in the PSW. This indicates to the processor that the channel is not operational.

The control units for the high-speed drum and direct access files can have two interface connections (tails) each, thus permitting each control unit to be physically attached to two channels. The logical connections between the control units and the channels is under program control, thus providing the programming system with the facility to connect the control unit to any of its channels. Once a connection is established, it is preserved until the control unit is released by a command from the connected channel. Release causes the control unit to assume the neutral state in which it is available to any channel.

In the IBM 2702 Transmission Control Unit, which provides for attachment of communication lines to the multiplexor channel, the switch connecting the control unit with either of the channels is placed in the neutral position upon resetting the control unit. The first programmed selection of a communication line subsequently causes the control unit and all associated communication lines to be switched to the selecting channel. This connection is maintained until the control unit is reset or a release function is performed.

When a control unit has been disconnected from a channel by the partitioning switch, that channel does not have access to the control unit and all devices on the control unit appear to the channel to be non-operational. The not-operational state of a control unit or device is indicated by condition code 3 being set in the PSW. To restore switching under program control, the control unit must be reconnected to the system by the partitioning switch.

MULTIPROCESSING FEATURES

The time sharing system may operate in an environment containing more than one processor. When this is the case, consideration must be given to the interaction of processors. Since processors may access storage, provision must be made to prevent interference when several processors attempt to access a common storage medium at the same time. This safeguard is provided by interlocks. Communication between processors is accomplished by the signalling method described below. When a processor does not function properly it may cause the entire system to operate incorrectly. Therefore, the malfunction alert feature is provided. A malfunctioning processor may be taken out of the system and then reintroduced when repaired. The system then starts the processor in operation, as described below in the section on programmed initialization.

The necessity for interlocking arises when different processors share a common storage medium. Interlocks prevent one processor from interfering with another processor in the manipulating of shared data or programs. When a serial input/output device is the common storage medium, interlocks are provided by the operating nature of the device. Such a device can only maintain one data transfer operation at a time to or from the storage medium. This serial transfer of data provides the required interlock. When a direct access input/output device is the common storage medium, interlocks are not provided by the operating nature of the device. For a direct access device, several transfer operations may occur during the time between the completion of a seek command and the read or write command. In this case, interlocks may be provided by programming. The Resident Supervisor provides programmed interlocks through the use of pathfinding tables. The method is described in "Pathfinding."

The necessity for interlocking also arises when the shared storage medium between processors is core storage. For example, each processor may simultaneously attempt to update the same record in a table. Without interlocking, the results of the simultaneous update may be erroneous. With interlocking, one processor may reserve the table containing the record, perform the update, and then release the table. The second processor may then perform its update. Thus, there is no logical interference between processors.

When two processors simultaneously request access to core storage, the granting of access must be done in such a way that one processor does not lock out the other. Otherwise, the locked out processor will be effectively halted. A workable interlock method is as follows: in core storage, electrical interlocks go into effect for the period of one storage cycle. When two processors simultaneously request access to core storage, a tie breaking priority circuit grants access to one processor, then gives the next cycle to the other processor. Input/output requests are granted higher priority than processor requests. By this method, both processors receive access to core storage and neither may be locked out by the other.

A further interlock is necessary to prevent interference between processors. Even though access to core storage is granted for the duration of one storage cycle, the execution of an entire instruction usually requires several storage cycles. Therefore, two processors may write into the same location on alternate storage cycles. To prevent such an occurrence, a programmed interlock is provided in the form of the Test and Set instruction. This instruction sets the condition code according to the state of the leftmost bit of the addressed byte and then sets the entire byte to all ones. The byte in storage is set to all ones as it is fetched for the bit test. No other access to this byte is permitted between the moment of fetching and the moment of storing the ones.

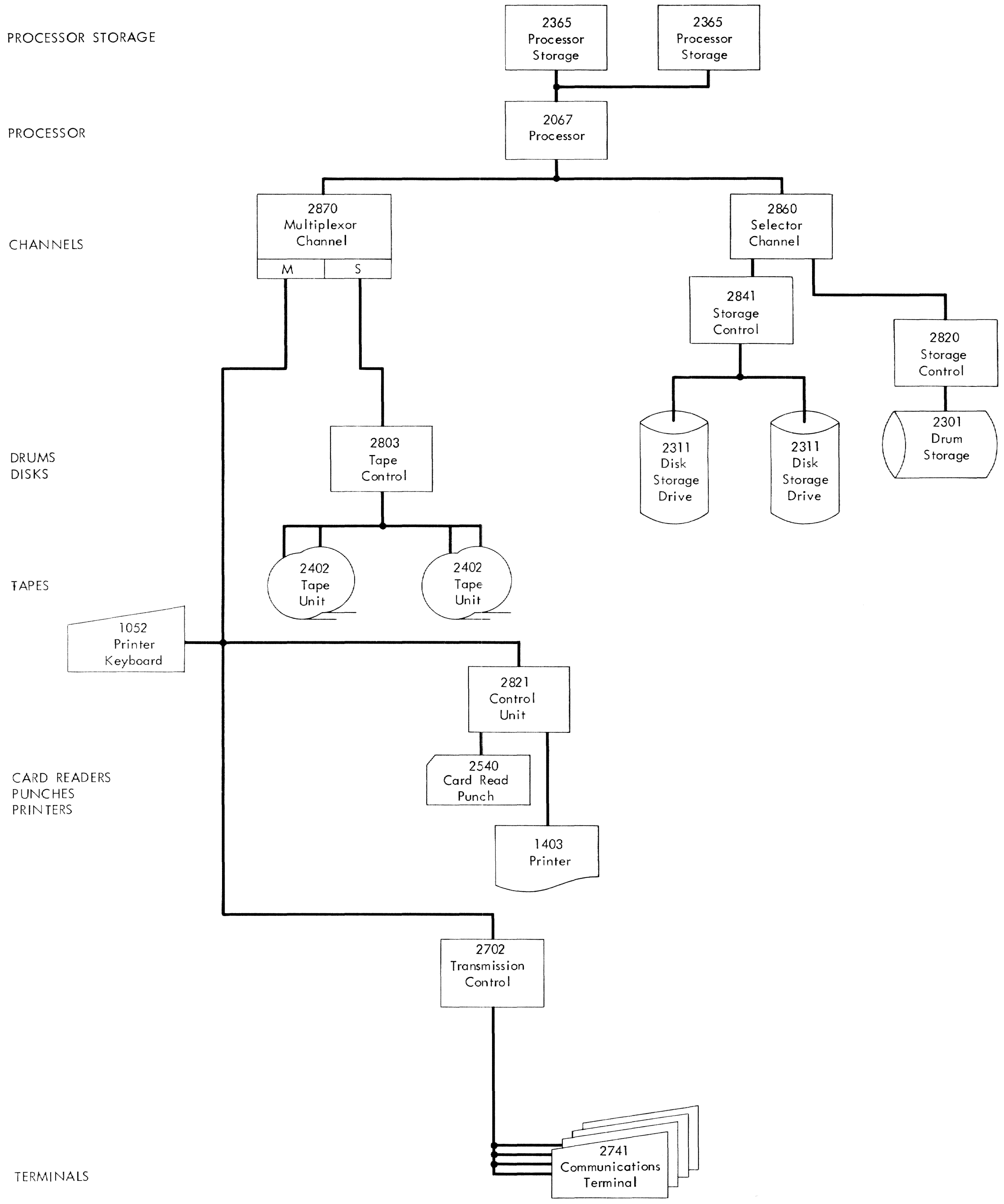


Figure 9. Sample Simplex IBM System/360 Time Sharing System

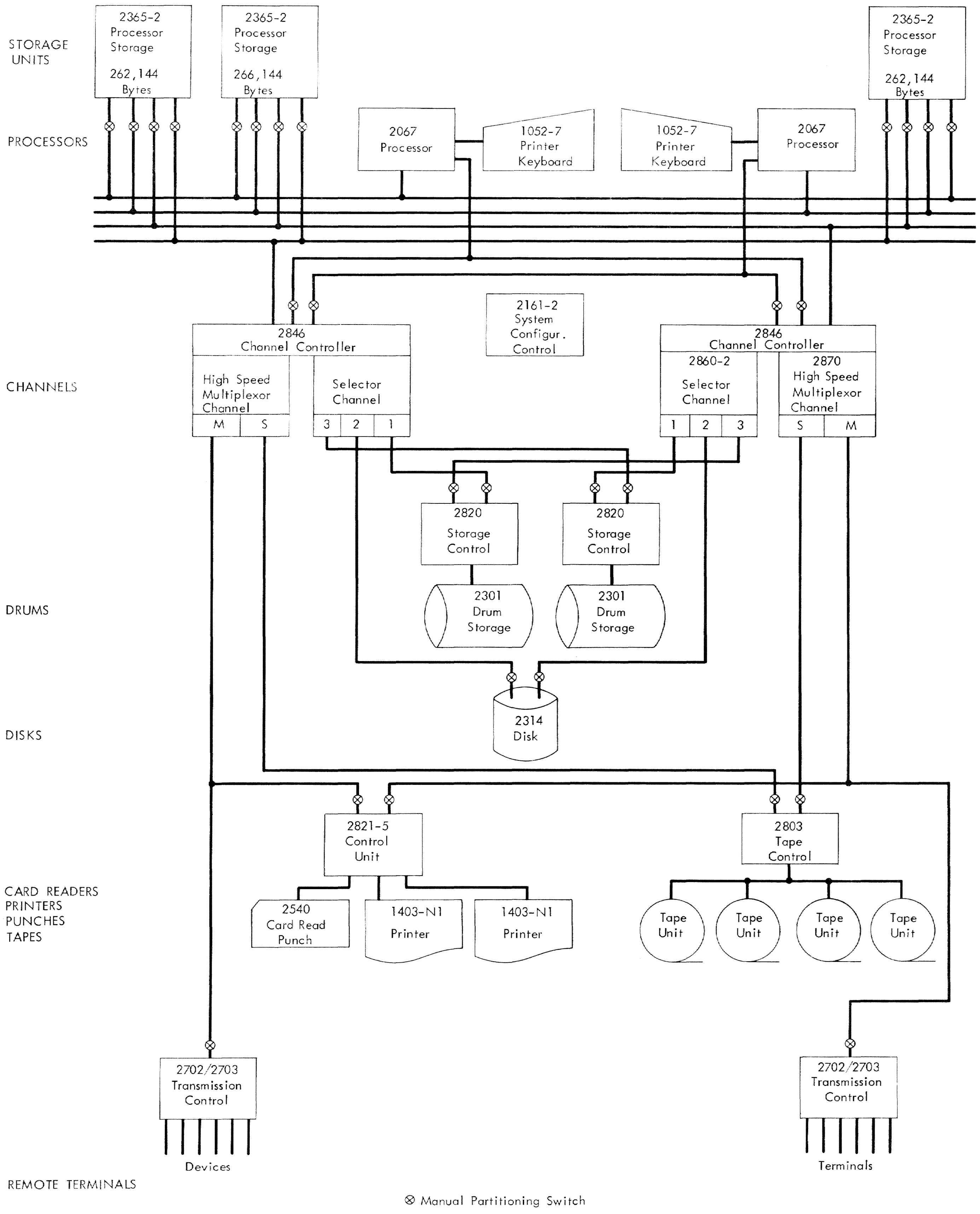


Figure 10. Sample Duplex IBM System/360 Time Sharing System

By using this instruction, a program in one processor may know that a program in another processor is updating a shared data area. The first program should then wait until the condition code of the byte indicates that the second program has released the area. This instruction also provides interlocking between programs in a time sharing environment. By means of the Test and Set instruction a program may interlock a data area even at the very end of the program's time-slice. No other program may then interfere with the data area until the interlock is released by the original program.

Signaling

The Model 67 is provided with a feature that allows processors to communicate through a common storage facility. For example, a processor may be alerted when a message has been prepared for it by another processor. This feature consists of extended direct control and external interrupt lines. Extended direct control refers to the direct control feature augmented by the ability to mask external interruptions. Associated with the direct control instructions is an interface at which eight signals are made available. A signal from one processor is connected to one of the external interruption lines of another processor. By means of the Write Direct instruction, the program in one processor may cause an external interruption in another processor.

Malfunction Indication

In a multiprocessing system it is necessary for processors to be informed when any processor in the system is not performing properly. In the Model 67 a malfunction signal is issued when a processor malfunction is detected. This signal is similar to the direct control signal and is transmitted to another processor in the system using the external interruption inputs of the processor.

The extensive checking included in all System/360 equipment is useful not only in error detection but also in the improvement of fault location. A high degree of checking makes it possible to recognize malfunctions on short notice and thus preserve the state of the processor for later diagnosis. Furthermore, the detailed error information made available to the customer engineer can reduce the repair time and contribute to the overall system availability.

Programmed Initialization

Each IBM System/360 processor uses permanently assigned storage locations (0-127) for program status words, channel address and status words, the timer, and initial program loading. During program switching, such direct addressing may also be necessary when the supervisor must store the general purpose registers. In a TSS/360 multiprocessor system, if these locations were common, they would be shared by both processors, and interference among processors would result. To provide each processor with separate assigned storage, a quantity called a prefix is automatically used by the Model 67 for relocation of addresses referring to the first 4096 storage locations. In a multiprocessor system, each processor is normally assigned a different prefix, and the sharing of these preferred locations is therefore avoided.

The prefix relocates all storage references that can be directly addressed (using zero-base specifications) by the displacement. Thus, main storage locations 0-4095 are not used and the prefixed preferred storage locations can be directly referenced by their actual storage addresses and indirectly referenced by the automatic prefixing of addresses in the range 0-4095.

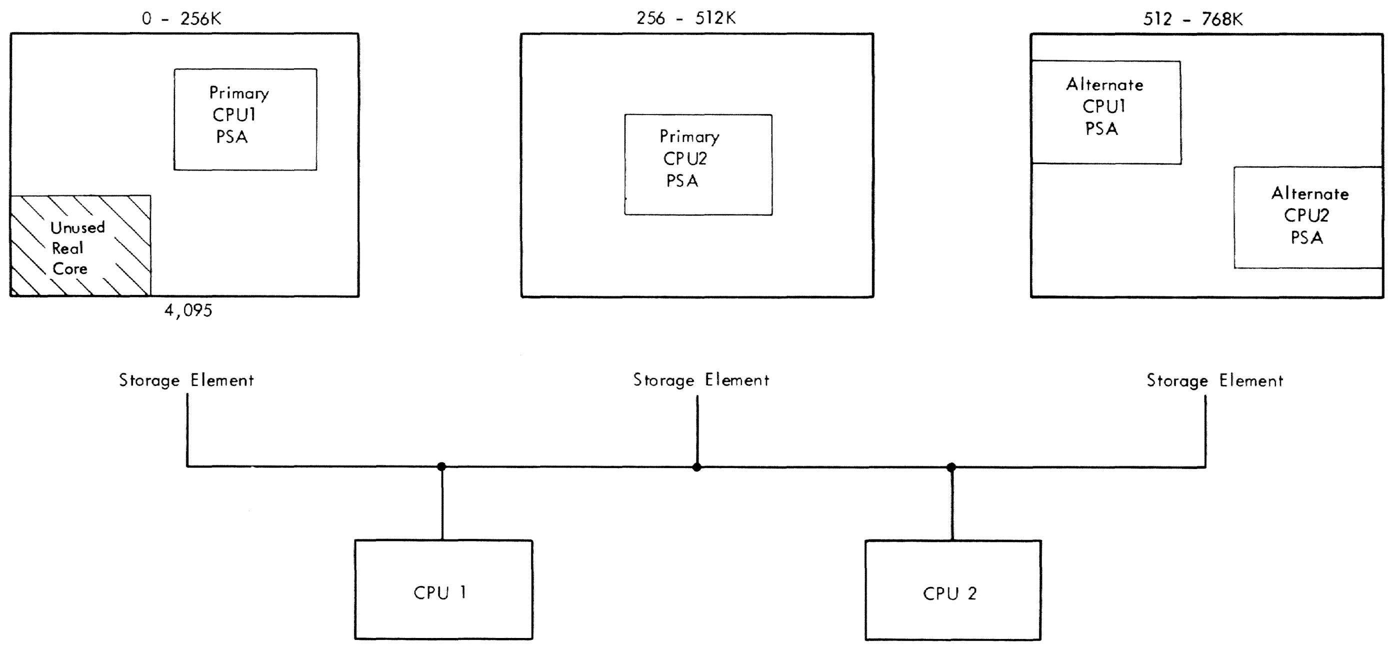
If a partitioned storage element contains the prefixed storage locations for the processor, new locations can be made available by introducing an alternate prefix. For this reason, a second (i.e. alternate) prefix quantity is supplied for a processor.

Normally, the two prefix quantities (i.e. primary and alternate) relocate the preferred storage locations to different storage units; the processor therefore becomes independent of a specific storage unit for its operation. This relationship is depicted in Figure 11.

The fact that the preferred locations can be normally addressed is useful in connection with start up or reinitialization of the system.

When a processor is starting up the system, it may determine its own identity by inspecting a specified location in its prefixed storage area. The active CPU accomplishes this by placing into the prefixed storage area of each CPU in the system a value which uniquely identifies that respective CPU and prefixed storage area. This is done through normal addressing. Then the active CPU inspects the specified location using an address in the range 0-4095 and identifies itself and the prefix currently being used.

When a processor is reintroduced into a multiprocessor system, operator action should be minimized. Introduction of a new program status word and the corresponding



•Figure 11. A Sample Relationship Among Processors, Storage Elements and Prefixed Storage Areas (PSAs)

instructions may best be performed by the still-operating part of the multiprocessor system. For this reason, means are provided for one processor to start another processor. Before the external start, the still active CPU places a PSW into relative location zero of one of the prefixed storage areas of the inactive CPU. It then issues an external start signal to the inactive CPU.

In this case, the external start consists of loading an initial PSW from location 0 and performing the necessary system reset.

This signaling again has been defined consistent with the signals of the direct-control circuits. There are two signal inputs, each of which causes an action similar to initial program loading. The choice between the two signals determines which prefix is used, and hence the location of the prefixed storage area.

Prefixed Storage Areas

The set of 4096 bytes that is directly addressable by a CPU (using the low-order 12 address bits plus a hardware prefix) is called a Prefixed Storage Area (PSA).

A prefixed storage area contains data and programs that are unique and private to each CPU. While preventing interference between processors, the PSA also functions as a logical extension of the general registers that makes it possible for more than one CPU to execute a supervisor component at the same time. The general format of a PSA is described in Figure 12. The first 128 locations of the PSA are reserved for status words, timer, interruption indicators, etc. The next 200 locations are permanently assigned to hardware diagnostic logouts. The CPU's private working storage area is assigned to selected supervisor programs. Some of the private area is used for temporary storage of general registers, without requiring a base register for

Hex		Dec
0	PSW Area	0
80	CPU Logout	128
130	CHANNEL Logout	304
148	CPU Private Working Storage	328
1C8	CPU Status Table	456
200	CPU Private Working Storage	512
228	RESERVED	552
800	INTER-COM RESIDENCE	2048
BE8	SERR DAMAGE REPORT	3048
C00	ERROR RECOVERY CONTROL TABLE (SERR)	3072
E2C	CPU Work Area	3628
E38	RESERVED	3640
1000		4096

•Figure 12. Prefixed Storage Area

generation of the address of the register save area. A special location within the PSA is reserved for the inter-CPU communication subroutine for use as an incoming message drop area from a calling CPU.

Also contained in the PSA is the Error Recovery Control Table which provides the Recovery Nucleus and the SERR Bootstrap with dynamic work areas. It also contains the SERR Common pool of adcons, the physical device address, and control data needed by the various SERR transient modules. This area is not to be confused with the SERR communication area located in the last 256 bytes of the SERR operating area.

PROCESSOR TIME SHARING FEATURES

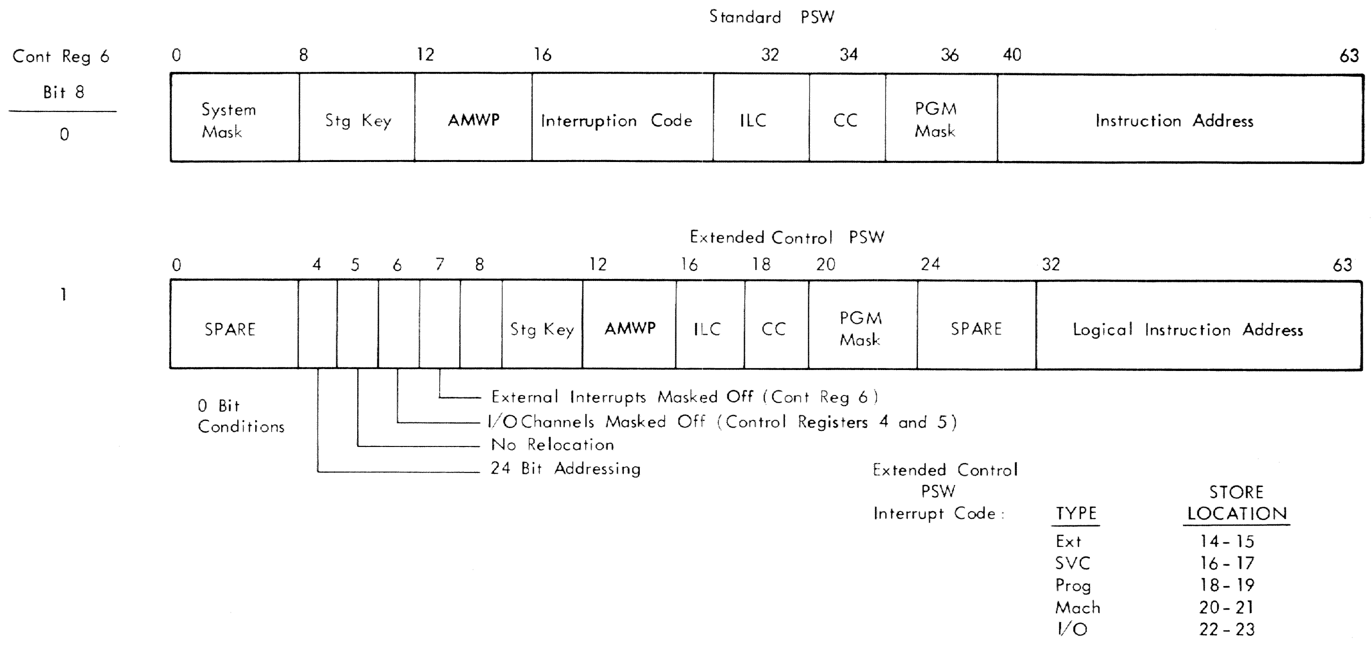
Extended Mode

In System/360, a program status word (PSW) is used to reflect instruction sequencing and to hold and indicate the status of the system in relation to the program currently being executed. The PSW contains information such as the instruction address, condition code, storage protection key and program mask.

By loading a new PSW, the state of the CPU can be initialized or changed.

The Model 67 has been designed to operate under the control of a PSW in two possible modes. The first mode, standard PSW format, is defined in Principles of Operation. The second mode, extended control PSW format, is defined in Model 67 Functional Characteristics. A comparison of the PSW formats for the two modes is shown in Figure 13. The purpose of introducing a second mode is to extend the capability of the processor to handle the time sharing environment. The environment includes dynamic address translation, extended channel masking, and external signal masking. The mode is determined by the setting of a bit in a control register. Information relevant to time sharing contained in the PSW in extended mode operation includes the following: an indicator enabling relocation (dynamic address translation), an indicator enabling I/O channel masking by control registers, an indicator for external interrupt masking by a control register, and the logical instruction address. The Extended Control PSW does not contain an interrupt code field. When an interrupt occurs, the interrupt code is stored in a specified storage location as shown in Figure 13.

TSS/360 operates in the extended control mode. When the Model 67 is operating in the extended mode, the operation of the following instructions is slightly modified: Load PSW, Set Storage Key, Translate and Test, Edit and Mark, Load Address, Supervisor Call, Branch on Index High, Branch on Index Low or Equal, Set System Mask. For a discussion of the modified



TYPE	STORE LOCATION
Ext	14-15
SVC	16-17
Prog	18-19
Mach	20-21
I/O	22-23

•Figure 13. PSW Formats

functioning of these instructions, see Model 67 Functional Characteristics.

The Model 67 contains a set of control registers to assist in time sharing and multiprocessing operations. A maximum of sixteen control registers may be included, but at present only twelve have assigned functions. The functions are:

- Table register (for dynamic address translation)
- Relocation exception address register
- Two extended mask registers for masking I/O channels
- A register containing machine check mask extensions for channel controllers, extended mode indicator, configuration control indicator, external interruption mask
- Two registers containing the status of main storage partioning switches
- A register containing the main storage addresses assigned to logical processor storage units
- A register containing the states of channel controller partitioning switches and channel address assignments
- Two registers containing the states of control unit partitioning switches
- A register containing the states of direct control partitioning switches and the states of prefix deactivation switches

The following new instructions are included in the instruction set of the Model 67: Branch and Store, Load Real Address, Load Multiple Control, and Store Multiple Control. For a discussion of these new instructions see Model 67 Functional Characteristics.

Storage Protection Extension

Additional capability has been implemented in the storage protection circuitry. By extending the field of the storage protection key from 5 to 7 bits, TSS/360 can evaluate the utilization of storage. If a reference is made to any location within a given block of storage or if the contents of the block are modified (changed) these facts are recorded by the hardware. Using these facts, the programming system may later analyze the activity of storage blocks.

This section describes the general logic of the Resident Supervisor and the Task Monitor. A schematic view of the relationship between these components and the rest of the system is shown in Figure 14. Examples of the use of these components within the system are described in other sections, particularly "Interruption Handling."

RESIDENT SUPERVISOR

The main function of the Resident Supervisor is to process interruptions and in so doing provide management of the system facilities. The system facilities include processors, main storage, input/output devices, and paging devices.

The specific functions performed by the Resident Supervisor can be categorized as follows:

- Interception and initial handling of all interruptions
- Concurrent control of a variable number of tasks

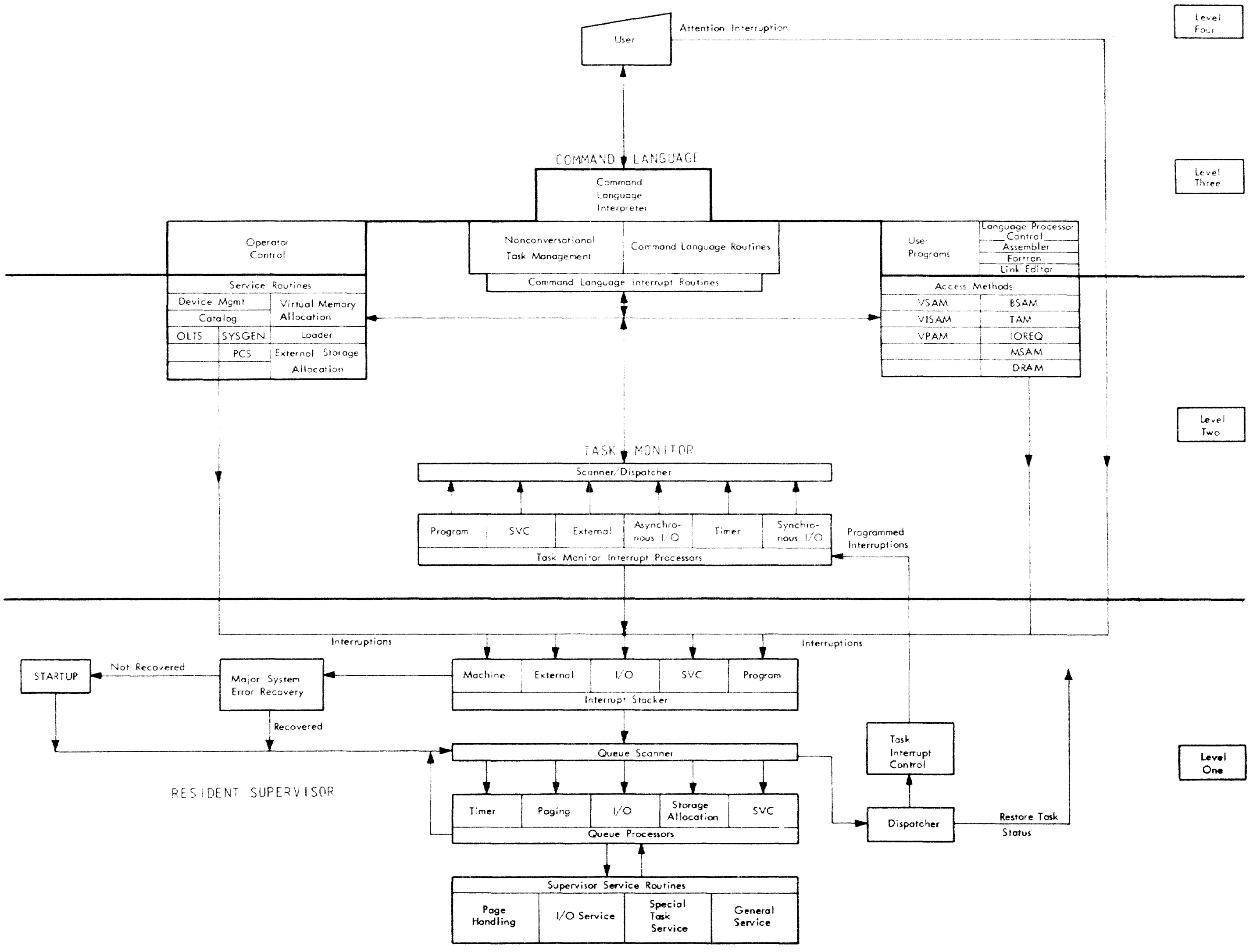


Figure 14. TSS/360 Program Structure

- Time slicing and task dispatching on the basis of a scheduling algorithm
- Main storage allocation
- Auxiliary storage allocation
- Accumulating task usage of CPU time for accounting
- Reading and writing pages
- Performing non-paging I/O in response to requests from tasks
- CPU and Paging I/O error retry and recovery procedures
- System Restart on a non-correctable error

The Resident Supervisor has the following components:

- Interrupt Stacker
- Queue Scanner
- Dispatcher
- Processors
  - Interrupt Processors
  - Request Processors
  - SVC Processors
- Major Error Recovery Routines
- Supervisor Service Subroutines

An overview of this component structure is presented in Figure 15.

INTERRUPT STACKER

The Interrupt Stacker is the only entry point to the Resident Supervisor. The function of the Interrupt Stacker is to classify interruptions and either enter them in queues or pass them to the Time Sharing Support System. That is, the Interrupt Stacker permits incoming interruptions to be stacked while previously received interruptions are being processed. A queue entry contains a forward and backward pointer to entries in the same queue. When an entry is to be moved from one queue to another, only the pointers are changed. Thus, the entry itself does not have to be physically moved. Quite frequently, one control block may belong to several queues and contain forward and backward pointers to each of them. In processing these multithreaded lists, the supervisor becomes, in effect, a list processor.

The Interrupt Stacker performs an initial analysis of the incoming interruptions in order to distinguish between software defined "emergency" interruptions which require immediate processing and "standard" interruptions which do not require immediate processing and also to distinguish between TSS/360 and TSSS interruptions.

These are the types of TSS/360 interruptions and their initial classification:

Interruption	Classification
Machine Check	emergency
External	
Timer	standard
Operator's Console Key	standard
Write Direct	emergency
Malfunction Alert	emergency
SVC, issued in:	
Supervisor State	emergency
Problem State	standard
PROGRAM, issued in:	
Supervisor State	emergency
Problem State	standard
I/O	
From paging drum	standard
From any other device	standard

The Time Sharing Support System services a similar set of interruptions (with the exception of Machine Check) with its own set of interruption processors. These interruptions are initially received by the Interrupt Stacker and, in the case of the Virtual Support System, are eventually enqueued on the task's TSI or, in the case of the Resident Support System, passed directly to the appropriate TSSS interrupt processor.

For emergency interruptions, error control routines are called (see "Error Procedures").

For most standard interruptions, the Interrupt Stacker builds a record called a General Queue Entry (GQE), which describes the interruption. The GQE is attached to the appropriate interruption processor queue. The GQE is described in a separate section. These queues of stacked interruptions are processed in a logical order, essentially on a first in first out basis within each queue. The queues themselves are processed on a priority basis.

Most supervisor call (SVC) and non-paging program interruptions originating in the software non-privileged state are simply transferred to the Task Monitor, by appropriately manipulating Program Status Words, with immediate return to the task.

Interruptions are masked during processing in the Interrupt Stacker. However, in contradistinction to most conventional systems, the Resident Supervisor generally executes with interruptions unmasked. This facilitates the processing of interruption queues on a priority basis.

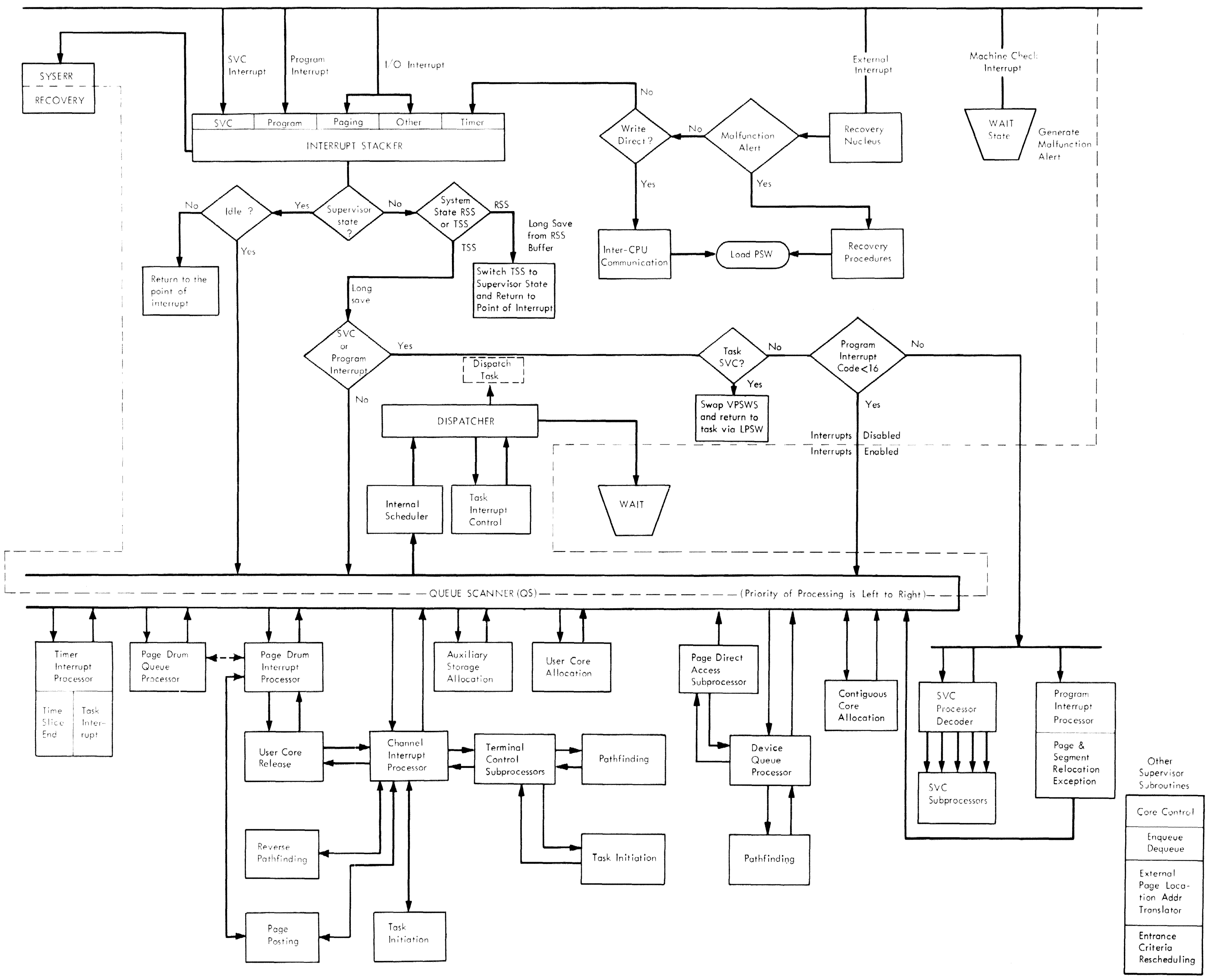


Figure 15. Resident Supervisor Component Structure

In addition to creating a GQE and placing it on the proper interrupt queue, the Interrupt Stacker maintains a log of the last 100 interruptions.

After completing its processing, the Interrupt Stacker saves the complete status of the interrupted task in the extended portion of the table's Task Status Index, unmasks interruptions, and generally transfers control to the Queue Scanner if the CPU was executing in the problem state when the interrupt occurred. Control goes directly to the SVC or Program Interrupt Queue Processors in the case of an SVC or program interruption for faster processing of these types of interruptions which occur quite frequently.

If the CPU was executing in the supervisor state, the Interrupt Stacker returns to the point of interruption using the old PSW, so that the interrupted supervisor routine can complete in an orderly fashion the work it began.

The information that the Resident Supervisor needs to describe a task may be separated into two portions. The first portion consists of the information which is needed immediately and must therefore be always resident in main storage. This portion is called the Task Status Index (TSI), and is described in a separate section. The second portion consists of information which is only needed after processing for a task commences. This information need not be resident but may be read into main storage when needed. This portion of information is called the Extended Task Status Index (XTSI). The XTSI must be in main storage during a task's time slice, but is not necessarily resident between a task's time slices. The page or pages occupied by an XTSI could have been made addressable by both the Resident Supervisor and by privileged service routines operating in Virtual Storage. However, in TSS/360 these pages do not appear in the task's virtual storage. That is, the XTSI is not addressable by a dynamic address translation. This serves to protect the Resident Supervisor from being over-written by a user program.

QUEUE SCANNER

Every system needs some facility for sequencing the work to be performed by the control program. In systems which operate with interruptions masked, the hardware priority interruption system provides this function for the interrupt handling routines and some control program routine provides a similar function for the system's resource allocation routines. Within TSS/360, these two functions have been combined into one centralized Queue Scanner.

The purpose of the Queue Scanner is to provide a sequencing mechanism responsible for deciding the order in which individual Queue Processors are to be executed. To fulfill this purpose, the Queue Scanner uses a Scan Table, whose entries are in priority order.

Because the Queue Scanner is a central facility within the Resident Supervisor, it must operate efficiently if the Resident Supervisor is to operate efficiently. To achieve this efficiency, the sequencing of entries in the Scan Table was planned to minimize the number of entries that must be inspected. Moreover, the design of the Scan Table reflects an awareness of the possible interactions among queues, so that an exit is not made to a processor only to find that a needed facility (such as an I/O path) has been allocated to some other queue.

There is one Scan Table entry and a corresponding queue for each Queue Processor, with the exception of the Paging Drum and Device Queue processors. There may be many Scan Table entries, each having corresponding queues, for the Device Queue processor. These queues are for the devices on the system, each device having a separate queue. The paging drum queues are not included in work for the Device Queue processor. There is a separate processor for the paging drum queues. There is a paging drum queue for each paging drum in the system. The order in which device entries appear in the Scan Table and hence their priority is specified from the Symbolic Device Address assigned to each device during System Generation.

The Scan Table is further described in "Scan Table."

If available work is found, the Queue Scanner passes control to the appropriate Queue Processor.

If the Queue Scanner determines that there is no available supervisor work, either because there are no more GQEs to process or because all appropriate processors are "busy", control is transferred to the Internal Scheduler which calls the Dispatcher.

DISPATCHER

The purpose of the Dispatcher is to select a task to be given CPU control and to place that task in execution. The actual scheduling of the task, the determination of the length and the frequency of its time slices, and the priority a task has relative to other tasks in the system are factors governed by parameters contained in a schedule table and also are governed by the processing characteristics of the task as interpreted by the Internal Scheduler.

To select a task to be given CPU control, the Dispatcher scans a chained list of control blocks. This list is called the Dispatchable TSI list. The Dispatchable list is a subdivision of the active list. The active list consists of all tasks which are eligible to use the CPU. The Dispatchable list consists of only those eligible tasks which have pages in main storage. The eligibility of a task is in part determined by its "priority level." The assignment of priority levels and the scheduling algorithm that determines what task, if any, is to be given control of the CPU is discussed in "Scheduling Algorithm." If a task is selected for getting CPU control, the Dispatcher enters the Task Interrupt Control routine.

Upon return from Task Interrupt Control, the Dispatcher gives CPU control to the selected task by the following operations:

1. sets the task status to "in execution"

2. sets a pointer in the Prefixed Storage Area to identify the current task

3. sets the interval timer to a value determined by the scheduling algorithm

4. Loads the General Purpose, Floating Point and Extended Control registers from the task's Extended Task Status Index (XTSI), and then loads the task's current PSW. This information was saved in the task's XTSI by the Interrupt Stacker.

If the Dispatcher scans the entire Dispatchable TSI list without finding a ready task, the Dispatcher places a value in the Interval Timer and the CPU is placed in the wait state. Then an exit from the wait state occurs upon the next hardware interruption intercepted by that particular CPU.

The Dispatcher may exit to the Queue Scanner when a condition called forced time slice end occurs. This is discussed further in "Scheduling Algorithm." The Dispatcher puts a GQE indicating forced time slice end in the Timer Interrupt Queue and then exits to the Queue Scanner.

The list of dispatchable TSIs is maintained in proper order by the Internal Scheduler which is the interface between the Queue Scanner and the Dispatcher. If a flag in the TSI indicates that the XTSI is not in main storage, the Internal Scheduler initiates a paging operation. A GQE and an associated Page Control Block are created and placed in the User Core Allocation Queue to obtain a page of main storage for the XTSI. This GQE will eventually be placed on a device queue to initiate the transfer of the page from auxiliary storage to main storage. Eventually, the task's XTSI is paged into main storage and the Page Posting routine causes the ISA and PSW pages to be brought into main storage (see "Paging"). This is the means by which the XTSI is brought into main storage even though it is not addressable in virtual storage.

A Page Control Block requests the movement of pages between main storage and auxiliary or external storage. This movement of pages may consist of reading pages into main storage from auxiliary or external storage, writing pages out of main storage to auxiliary or external storage, and posting to the program's page tables. The Page Control Block is described in a separate section.

The purpose of the Task Interrupt Control routine is to generate programmed interruptions as required.

The need for a programmed interruption mechanism arises because the Resident Supervisor processes requests for system services in a logically independent fashion. The processing of system services is logically independent in the sense that the Resident Supervisor may be concurrently performing several services for a task. There is no way of ascertaining in what order or when the processing of each of these services will be completed.

Therefore, in order for a task to operate asynchronously with respect to the completion of system services, the need arises for a programmed interruption mechanism analagous in concept to the hardware interruption mechanism that allows the Resident Supervisor to operate asynchronously with respect to the real computer system. The programmed interruptions are similar in operation to hardware interruptions. The major difference between them is that the hardware interruptions convey a change in the status of the entire system to the Resident Supervisor, while the programmed interruptions represent a change in status of only that portion of the system currently allocated to the task which causes the interruption.

A programmed interruption is initiated by the Interrupt Stacker when it discovers an interruption that is a task's responsibility, such as a supervisor call requesting the services of a virtual storage system service routine.

In another example, the Resident Supervisor will request a programmed interruption whenever it determines that the further processing of an I/O interruption is a task's responsibility.

Because a task is not always prepared to receive an interruption and because the task for whom the interruption is destined may not be the next task dispatched, there exists a software queuing and masking facility analogous to the hardware.

A request for a programmed interruption is initially handled by enqueuing a GQE on the appropriate task's TSI. The programmed interruption is implemented by the Task Interrupt Control subroutine.

Implementation of the programmed interruption requires the definition of the following programming elements: an Interrupt Storage Area (ISA) and a virtual PSW (VPSW). Diagrams of these elements are shown in Figures 16 and 17. The VPSW is analogous to the Extended PSW operated on by the Resident Supervisor. The purpose of the instruction address, ILC, interruption code, and masking fields in the VPSW remain conceptually the same as for the PSW. These fields vary in detail due to the fact that program status has two modes - resi-

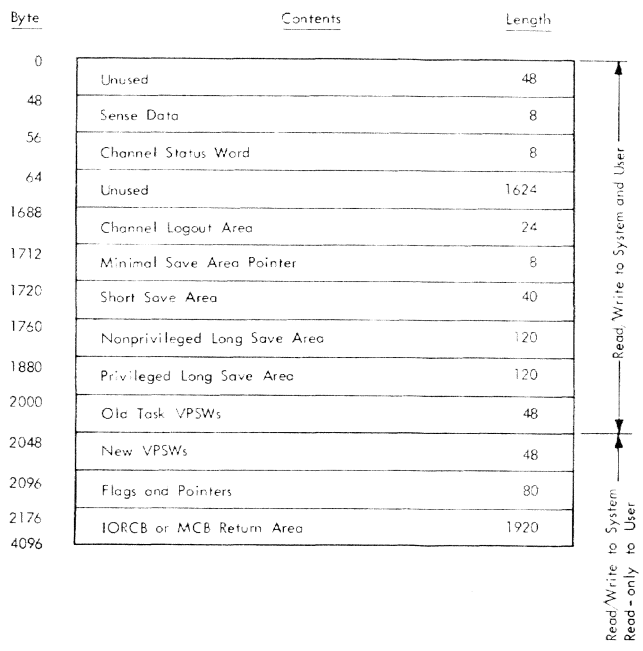


Figure 16. Interrupt Storage Area (ISA) Schematic Diagram

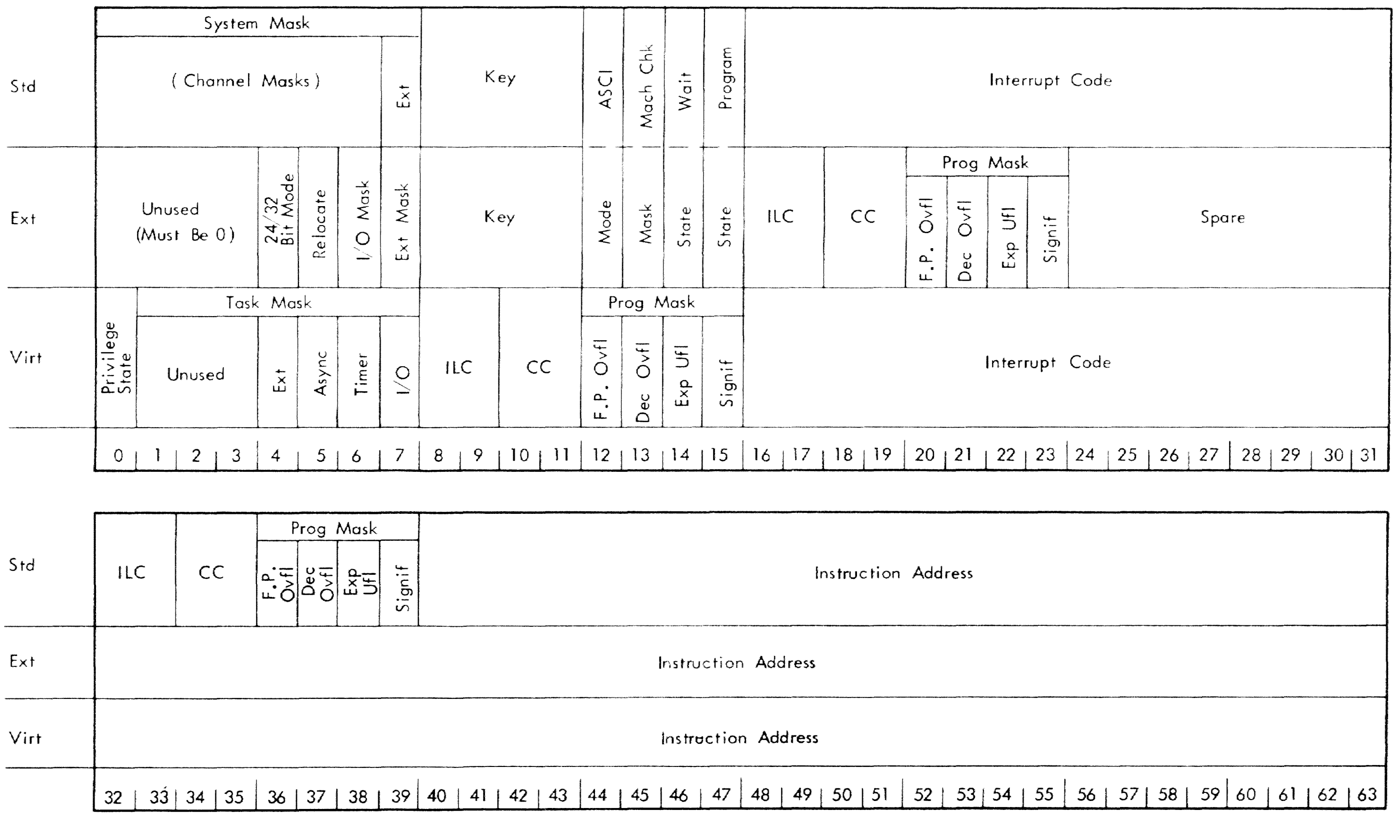


Figure 17. Comparison of Standard, Extended, and Virtual PSW Formats

dent and relocated. The VPSW represents the state of a task as it appears in a virtual machine at the moment the task is interrupted. The ISA is the first page (4096 bytes) of a task's virtual storage (i.e., segment 0, page 0) and is analogous to the Prefixed Storage Area of the Resident Supervisor. The ISA contains many analogous elements such as old and new VPSWs for programmed interruptions. The ISA is brought into main storage, along with the task's XTSI pages, before the task can be placed in execution. The Interrupt Storage Area is discussed in more detail in "Task Monitor."

The main features of a programmed interruption are depicted schematically in Figure 18. The way in which programmed interruptions are utilized is described in "Task Monitor" and "Interruption Handling."

The programmed interruption mechanism in the Task Interrupt Control routine involves the following actions:

1. The Extended PSW in the task's XTSI (which represents the status of the real system at the point when the task was interrupted) is formed into a VPSW and this VPSW is placed in the appropriate interruption old VPSW in the task's ISA.

2. The new VPSW is placed in a field in the ISA which is called the current VPSW and which is used to describe the task's status for the most recent interruption presented to the Task Monitor.

3. If the interruption is associated with Intertask Communication or is an I/O interruption, the associated control block (IORCB or MCB), if any, is placed in the ISA (see "Communication" and "Data Management").

4. The sense data, channel status word, and channel logout data are placed into the ISA, if appropriate (see "Task I/O Errors").

5. The interruption mask in the new VPSW in the ISA is used to set the programmed interruption mask in the task's Task Status Index (TSI).

6. The new VPSW, which points to an interruption processor of the Task Monitor, is obtained from the ISA formed into an Extended PSW and stored in the task's XTSI. When the Dispatcher gains control, it loads the extended PSW from the XTSI and thus gives control to the appropriate Task Monitor interruption processor.

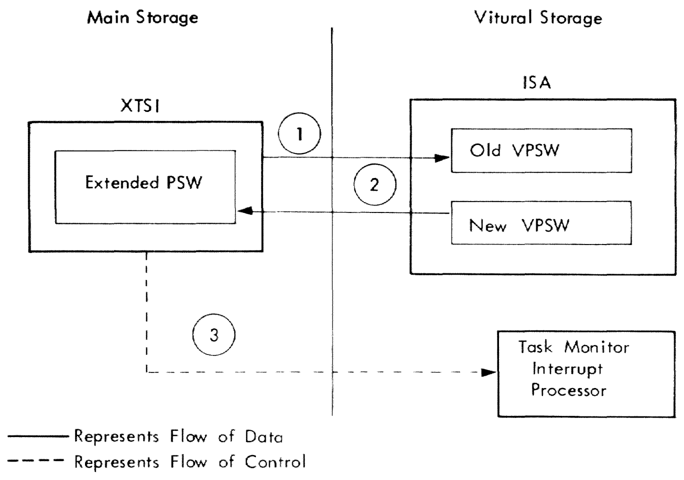


Figure 18. Main Features of a Programmed Interruption

The main features of the Task Interrupt Control routine's operation are: The Task Interrupt Pending flags in the TSI are compared with the Task Interrupt Mask in the TSI. If all pending task interruptions are masked, control returns to the Dispatcher; otherwise, a programmed interruption is generated for the highest priority unmasked pending interruption. (Note that SVC and program interruptions cannot be masked.) Control then returns to the Dispatcher.

The TSS/360 task interruption scheme is a priority interruption scheme, as opposed to a single-level "wakeup" scheme because, in part, this allows for a more graceful handling of such events as attention interruptions and abnormal task terminations.

QUEUE PROCESSORS

Each queue processor is responsible for processing the GQEs on its queue. In general, each processor operates on only one GQE and then returns control to the Queue Scanner. However, certain processors, for example the Page-Drum Interrupt Queue and the Channel Interrupt Queue Processors process all the GQEs on their queues before returning control to the Queue Scanner. The Program Interrupt Queue Processor and the SVC Queue Processor are misnamed because they do not operate on queues. They are linked to directly from the Interrupt Stacker. This allows faster processing of these frequently encountered interruptions.

In general, GQEs are only created by the Interrupt Stacker and are not destroyed until their processing (which may involve several processors) is completed. A few queue processors (for example, the Auxiliary Storage Allocation Queue Processor, the Program Interrupt Queue Processor and

I/O Device Queue Processor) may create additional GQEs.

If the processing of a GQE requires the attention of several processors, the GQE is transferred from one processor's queue to the next through the services of one of the Queue Control subroutines.

The Queue Control subroutines examine the first routing field in a GQE. This field will either contain a location-on-queue value or all ones. The location-on-queue value designates the relative location on the Queue Scanner's Scan Table of the queue to which the GQE is to be transferred. A value of all ones indicates that no further processing is to be performed for the GQE and the main storage occupied can be released.

In general, a Queue Processor locks its associated queue upon entry and unlocks the queue as soon as the processor has dequeued a GQE from the queue for processing.

In certain cases a Queue Processor may wish to lock a queue until some specific future event or condition has occurred. Each Scan Table entry has several indicators reserved for such use. These indicators are called Suppress Flags and are set or reset by the Resident Supervisor routines involved. Both the queue lock byte and the Suppress flags are used to prevent unwanted recursion.

The queue processors may be classified into two groups: interruption queue processors and request queue processors.

There are five interruption queue processors as follows:

- Timer Interrupt Queue processor
- Paging Drum Interrupt Queue processor
- Channel (non-paging-drum) Interrupt Queue processor
- Program Interrupt Queue processor
- SVC Interrupt Queue processor

The request queue processors constitute a somewhat arbitrary grouping of the remaining queue processors. A function of these queue processors is to service requests made by the interruption queue processors and other supervisor routines. The request queue processors are:

- User Core Allocation
- Contiguous Core Allocation
- Auxiliary Storage Allocation
- I/O Devices
- Paging Drums

The SVC Interrupt Queue processor calls upon a particular SVC processor when the type of SVC is identified. The SVC processors are listed below with the names of the corresponding macro instructions that issue these SVCs. These macro instructions are described in _System Programmer's Guide_.

- TSI/XTSI Modification/Extraction Group

  Create TSI processor (CRTSI)

  Delete TSI Processor (DLTSI)

  Special Create TSI processor (SCRTSI)

  Change Priority processor (CHAP)

  Setup XTSI Field processor (SETXTS)

  Setup TSI Field processor (SETUP)

  Extract TSI Field processor (XTRCT)

  Extract XTSI Field processor (XTRXTS)

- Virtual storage service group

  Add Pages processor (ADDPG)

  Add Shared Pages Processor (ADSPG)

  Delete Page Processor (DELPG)

  Set External Page Table Entries processor (SETXP)

  Move Page Table Entries processor (MOVXP)

  Connect Segment to Shared Page Table processor (CNSEG)

  Disconnect Segment to Shared Page Table processor (DSSEG)

  List Changed Pages processor (LSCHP)

  Check Protection Class processor (CKCLS)

  Load Virtual Program Status Word (LVPSW)

- Timer maintenance/task synchronization group

  Time Slice End processor (TSEND)

  Await Interruption SVC processor (AWAIT)

  Terminal I/O Wait processor (TWAIT)

  Set User Timer Interval processor (SETTU)

Set Real Time Interval processor (SETTR)

Restore Time processor (RSTTIM)

Read Time processor (RDTIM)

- System Table Modification/Extraction Group

  Setup System Table Field processor (SETSYS)

  Extract System Table Field processor (XTRSYS)

  Set Time of Day processor (SETTOD)

  Set Year, Month, and Day processor (SETYMD)

- Input/Output and Device Management Group

  I/O Call (IOCAL)

  Pageout Service processor (PGOUT)

  Reset Device Suppression Flag processor (RESET)

  Set Path processor (SPATH)

  Add Device on Task processor (ADDEV)

  Remove Device From Task processor (RMDEV)

  Set Asynchronous Entry processor (SETAE)

SUPERVISOR SERVICE SUBROUTINES

The Supervisor service subroutines are used by the queue processors and other components of the Resident Supervisor to provide various required services. There are eight groups of supervisor service subroutines:

- Queue Control
- Page Handling
- I/O Service
- Task Service
- Main/Auxiliary Storage Allocation
- Inter-CPU Communication
- Dispatcher Service
- Intertask Communication (VSEND and XSEND)

MAJOR ERROR RECOVERY ROUTINES

The relationship of the major error recovery routines to the system is discussed in the section on Error Procedures. The major error recovery routines are:

- Machine check new PSW
- Recovery nucleus
- Reconfiguration
- External machine check interrupt
- System Environment Recording and Retry (SERR) group
  SERR bootstrap
  Environment recording
  Immediate print
  Checker
  Pointer
  Restore and validate
  Instruction retry execution
  CPU/Storage checkout
- System Error Processor

CONTROL BLOCKS

This section contains fundamental descriptions of the important control blocks used by the Resident Supervisor. The way in which these control blocks are used and the significance of the various fields and flags within these control blocks that have not been explained thus far, will be explained elsewhere in this manual.

General Queue Entry (GQE)

A GQE has a fixed length of 64 bytes and contains a description of the work to be done by a device or facility controlled by the Resident Supervisor. One use of a GQE is to save information for an interruption.

The contents of a GQE depend on the type of interruption and generally consist of the following:

- Pointers to:

  Task Status Index (TSI)

  Preceding and succeeding GQEs on the same queue

  Page Control Block if the GQE is associated with a request to read or write a page in main storage

  I/O Request Block (IROCB) or Message Control Block (MCB) if the GQE is associated with non-paging I/O or intertask communication

- Flags:

  Request for a page in
  Request for a page out

- Instruction length code from Program Status Word

- GQE movement information

- Data obtained by the Sense I/O command
- Channel Status Word
- Channel logout data
- Interruption code
- Symbolic device code

Scan Table

The Scan Table is a resident control table private to the Queue Scanner. The size of the Scan Table is a function of the installation configuration and is set during System Generation. The Scan Table serves as a common anchor point for those GQEs that represent work for the Resident Supervisor. The Scan Table determines the order in which the queues are processed.

To achieve efficiency, many of the system's queues have been organized into groups called Device Interaction Groups. A Device Interaction Group (DIG) generally consists of the set of queues for all I/O devices having common I/O access paths.

The Queue Scanner inspects individual queues within a DIG only if a master count indicates there is work enqueued within the DIG, and then only if other flags indicate that the appropriate queue processor is not "busy" and that an I/O path to the device is available. Furthermore, in order to prevent one active device in a group from monopolizing an I/O path and greatly delaying the processing of other requests within the DIG, the Queue Scanner processes the queues within each DIG in a round-robin order.

There is one entry in the Scan Table for each device or supervisor facility. An entry has a fixed length of 16 bytes and contains the following information:

- Pointers to:

  First GQE on the queue
  Last GQE on the queue
  Location of the Queue Processor

- Flags:

  Queue empty flag

  Device Interaction Group identification number

  Suppress flags (prevent processing of the queue until all Suppress Flags are off)

- Queue processor lock byte

A schematic diagram of the Scan Table is shown in Figure 19.

Task Status Index (TSI)

The Task Status Index is the principal control block for a task in that it contains task information that must be permanently resident. There is one TSI for each task in the system and the TSI is resident in main storage from LOGON to LOGOFF. After LOGOFF, the storage occupied by a TSI is made available for reuse by the

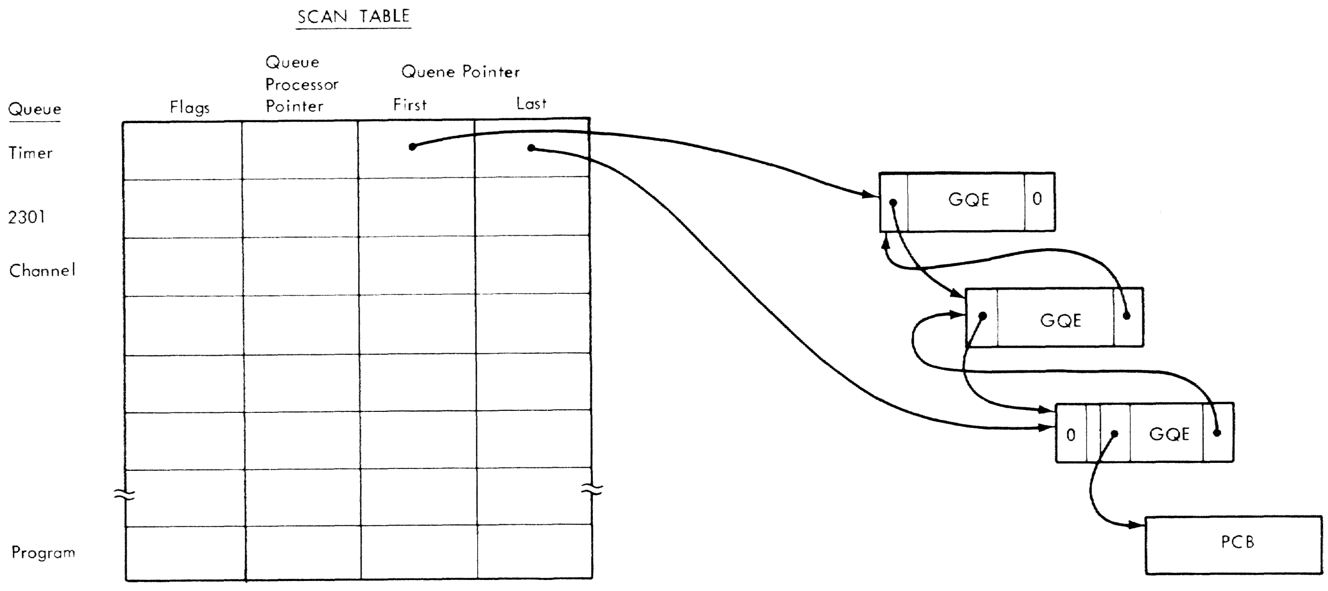


Figure 19. Schematic View of the Scan Table

supervisor. The TSI has a fixed length of 128 bytes.

The contents of a TSI include the following:

- User identification
- Task identification
- Task priority
- SYSIN symbolic device address
- SYSOUT symbolic device address
- Pointers to:

  Extended TSI

  Next TSI in the chain of active or inactive tasks

  Task interruption queue (GQEs representing interruptions that will be passed to the task)

  Task Symbolic Device List (a list of those devices on which the task can perform I/O)

- Flags for task status:

  In-execution
  Ready
  Page wait
  Time slice end
  Delay

- Lock byte
- Task interruption mask
- Task interruption pending flags (one for each task interruption queue)

Extended Task Status Index (XTSI)

The XTSI contains task information which does not have to be permanently resident. At least part of the XTSI is resident during a task's time slice. The XTSI is created at LOGON and released at LOGOFF. The XTSI consists of a fixed length of 256 bytes (of which 208 bytes are used) and a variable length portion. The total size may vary from one page upward, depending on the installation limit placed on the number of XTSI pages allowed and on whether the system is operating in 24- or 32-bit addressing mode.

The contents of the header include:

- Extended PSW save area
- Control register save area
- General purpose register save area
- Floating point register save area
- Pointer to the task's TSI
- Time slice information
- Timer information

The contents of the variable length portion include:

- Segment Table
- Auxiliary Segment Table
- Page Tables (non-shared)
- External Page Tables (non-shared)

Page Control Block

The Page Control Block is used to control the movement of virtual storage or XTSI pages between main storage and external or auxiliary storage. A Page Control Block entry represents a request for the movement of a page. Entries may be generated during any of the following events:

- Time slice end interruption processing
- Page relocation exception interruption processing (program interruption code 17)
- PAGEOUT or IOCAL SVC processing
- Dispatcher request for initial XTSI page
- Page Posting processing for the remaining XTSI pages and the ISA page

The contents of an entry include the following:

- Main storage address of page
- Virtual storage address of page
- External or auxiliary storage address of page
- Flags:

  Virtual storage or XTSI page

  Paging I/O has been completed

  Put main storage occupied by the page in the Preferred Page (XTSI-PSW) Pending queue or in the non-XTSI-PSW Pending queue

  Drum or disk preference for page residence when writing a page to auxiliary storage.

A Page Control Block consists of up to three entries plus a four byte pointer to the next block. Thus, a block may contain up to three requests to move pages. Additional blocks are needed when there are more than three requests.

TASK MONITOR

The Task Monitor handles task oriented interruptions that are passed on to it by the Resident Supervisor. To accomplish this objective, the Task Monitor consists of a group of privileged service programs that receive and process task oriented (programmed) interruptions in a prescribed sequence and on a priority basis via queueing, scanning, and dispatching mechanisms. These mechanisms are somewhat analogous to the stacking, scanning, and dispatching mechanisms of the Resident Supervisor design. As in the Resident Supervisor, a queued interruption represents an element of work. Such an element may be deferred for reasons of priority, efficiency, or protection against recursion.

The Task Monitor performs the following major functions:

- Provides an interface with the Resident Supervisor for receiving and analyzing task oriented interruptions.
- Provides linkage to required service routines either by immediate dispatching or by queueing the interruption for later dispatching in a sequence based on priority.
- Maintains the integrity of the task and service routines that are dispatched.

The Task Monitor consists of interruption processors, a Queue Linkage Entry routine, a Scanner-Dispatcher, all contained in one module, and a separate group of service routines. Interruption processing proceeds as follows: entry is made into an interruption processor from the Resident Supervisor. The interruption processor either immediately dispatches a service routine to handle the interruption or makes an entry for the interruption in an Interrupt Table. The entry is made by using the Queue Linkage Entry routine. Then control goes to the Scanner-Dispatcher which uses the Interrupt Table to select and invoke a specified service routine to process an interruption.

When the Scanner-Dispatcher determines that no more service routines can be dispatched, control is returned to the task at the point of its last interruption. An overview of the general flow of interruption processing in the Task Monitor is shown in Figure 20.

INTERFACE WITH RESIDENT SUPERVISOR

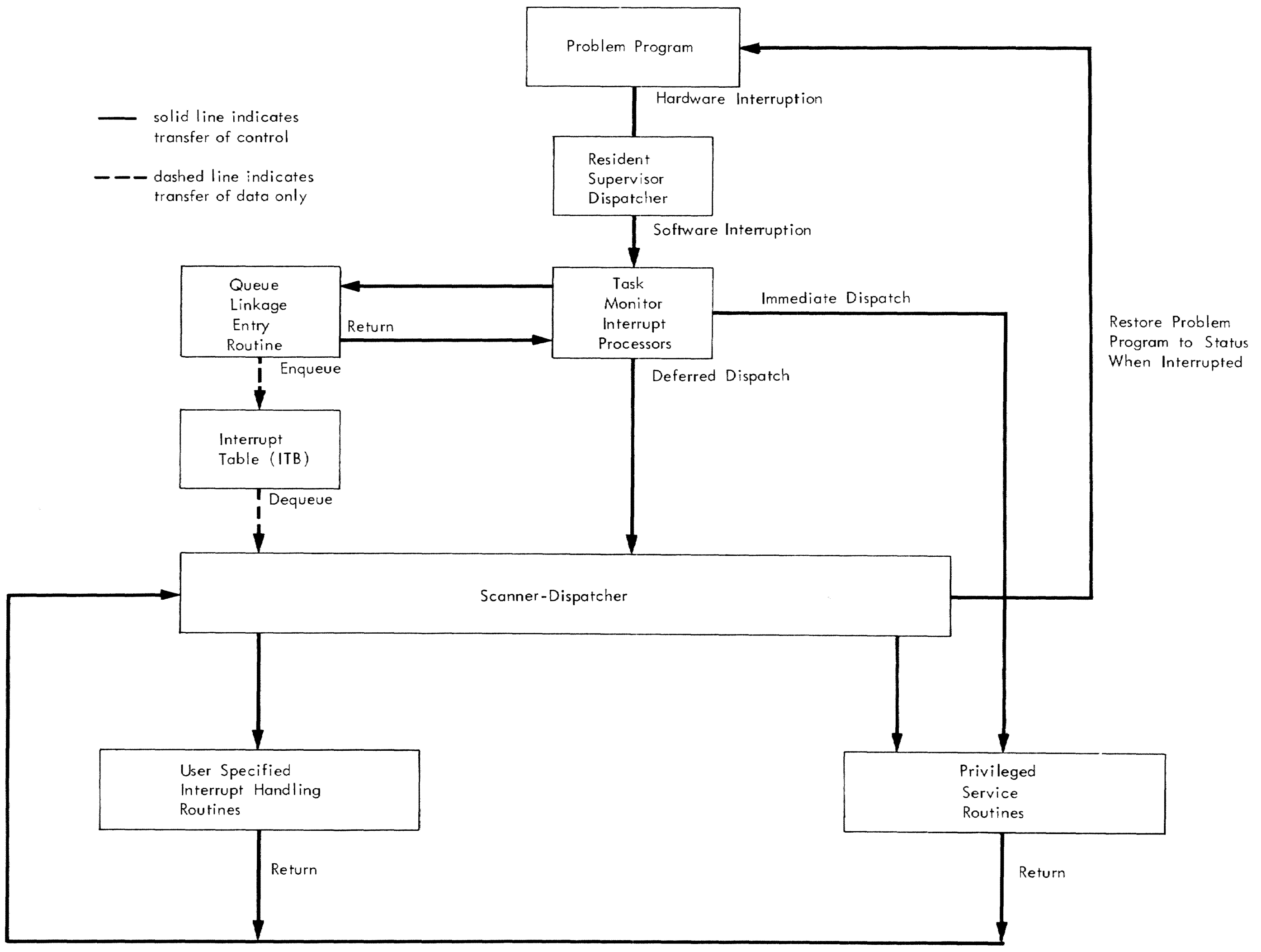
Control goes to the Task Monitor from the Resident Supervisor when a task interruption is presented. The Resident Supervisor Dispatcher always gives control to a task at a location specified in the Extended PSW saved in the task's XTSI. However, before control is given to a task, the Dispatcher transfers control to the Task Interrupt Control routine. This routine checks the task's interruption queues for unmasked pending interruptions. If none are found, control is returned to the Dispatcher. In this case, the Dispatcher gives control to the task at the point of its last hardware interruption. If a pending interruption is found, the Task Interruption Control routine changes the Extended PSW in the XTSI to point to an appropriate interruption processor of the Task Monitor. Now, when the Dispatcher gets control, it causes control to go to the interruption processor. This action of influencing the Dispatcher's transfer of control is called a programmed interruption. The Task Monitor may be considered to be a programmed interruption handler, whereas the Resident Supervisor is a hardware interruption handler.

INTERRUPTION PROCESSING

After an entry is made into one of the interruption processors, the interruption processor checks to determine whether the interruption should be processed immediately or put on a queue for later processing. (Both actions are possible for some interruptions.) The processing of an interruption entails the eventual invocation of a service routine to perform the actions required by the interruption.

An interruption is processed immediately when its nature is such that the required action to be performed is of the highest priority to the task. For example, an interruption requiring the invocation of the Dynamic Loader must be processed immediately. In the immediate processing case, the interruption processor frequently uses the Load Virtual PSW SVC to change the program mask in the PSW, to set the TSI interruption mask, and to give control to the invoked service routine with a new PSW. This process is called immediate dispatching. When the dispatched service routine finishes its processing, it returns control to the interruption processor which then exits to the Task Monitor Scanner-Dispatcher.

When an interruption is to be put on a queue, the interruption processor calls the Queue Linkage Entry routine. This routine creates a Queue Entry in an Interrupt Table for the interruption. When control returns from the Queue Linkage Entry routine, the interruption processor exits to the Scanner-Dispatcher.



solid line indicates transfer of control
dashed line indicates transfer of data only
Problem Program
Hardware Interruption
Resident Supervisor Dispatcher
Software Interruption
Queue Linkage Entry Routine
Return
Task Monitor Interrupt Processors
Immediate Dispatch
Restore Problem Program to Status When Interrupted
Enqueue
Deferred Dispatch
Interrupt Table (ITB)
Dequeue
Scanner-Dispatcher
User Specified Interrupt Handling Routines
Privileged Service Routines
Return
Return

Figure 20. Interrupt Processing General Flow

The Task Monitor uses the Interrupt Table to hold information concerning service routines and interruptions to be processed. There are two main types of entries in the table: Request Entries and Queue Entries. There must be a Request Entry for every service routine that may be dispatched. The Request Entry contains an activity indicator, a priority code, a pointer to a description of the service routine, and a pointer to its Queue Entries (if any exist). A Queue Entry generally represents an interruption that has not yet been processed. A Queue Entry is chained to a Request Entry and contains information about the interruption. The activity indicator in a Request Entry is off when there are no Queue Entries attached to the Request Entry, and is on when there is at least one Queue Entry attached. A set of Request Entries is entered in the Interrupt Table to provide for the dispatching of service routines supplied with the system. Request Entries may also be added to the Interrupt Table by a user issuing a Specify Interrupt Routine (SIR) macro to provide for the dispatching of user supplied routines.

Queue Entries are entered in the Interrupt Table and attached to particular Request Entries by the Queue Linkage Entry routine (which sets the activity indicator in the Request Entry). This routine is normally called by a Task Monitor interruption processor. However, a privileged service routine may call the Queue Linkage Entry routine in order to cause a Queue Entry for a service routine to be enqueued. Either another service routine or the original service routine may be dispatched in this way. The Request Entries are used by the Scanner-Dispatcher to select a service routine for dispatching. To facilitate the selection, the active Request Entries are chained together in priority order. Request Entries may be deleted by a user issuing a Delete Interrupt Routine (DIR) macro when he no longer wants the corresponding service routines to be dispatched. The information contained in Queue Entries may be inspected by either a privileged or a nonprivileged routine through the use of the Interrupt Inquiry (INTINQ) macro instruction. After a service routine is dispatched, the Queue Entry is removed from the chain attached to the Request Entry for the service routine.

The purpose of the Scanner-Dispatcher is to select a queued interruption or linkage request and to dispatch a service routine. The selection is made from active Request Entries which are chained together in order of priority. After the service routine is dispatched, the Queue Entry for the interruption or linkage request is removed from the Request Entry. If no more Queue Entries are attached to the Request Entry, the activity indicator is set to indicate that the Request Entry is inactive and the Request Entry is removed from the active chain. When the Scanner-Dispatcher determines that no more service routines can be dispatched, control is returned to the point of the last interruption. The Scanner-Dispatcher always transfers control by means of the Load Virtual PSW (LVPSW) SVC. The LVPSW SVC is used because it permits the Task Monitor to specify the task's PSW program mask, the TSI task interruption mask, and the PSW storage protect key.

The dispatching of service routines using the Queue Linkage Entry routine to enqueue a request rather than by direct calls to the service routines serves three purposes. First, the Task Monitor takes care of linkage conventions which involve providing save areas and return points. Second, by this method a service routine will be able to gain control at a future time. Third, a service routine is able to release save areas by returning to the Task Monitor.

In order to eliminate the overhead that would be incurred if the Task Monitor had to build a Request Entry to handle each such linkage request, the Task Monitor maintains two general-purpose Request Entries upon which all such linkage requests can be enqueued. There is one Request Entry for privileged routines and one with a lower priority for non-privileged routines. Because these Request Entries are general-purpose in nature, the Queue Entry for a linkage request must contain the address constants for the routine to be dispatched.

Examples of service routines provided with the system are Command System routines and the Data Management access methods. Examples of service routines which may be supplied by the user include: program interrupt handling routines, routines for handling SVC codes not recognized by the system, routines for handling task time interruptions, and routines for handling special task I/O interruptions.

An example of the use of user specified interruption handling conditions is as follows: if a problem is to process program interruptions resulting from decimal overflow, the user must issue a Specify Program Interrupt Entry Conditions (SPEC) macro instruction to build an Interrupt Control Block (ICB) that names the routine and identifies the interruption type, in this case Program interruption code 10. The user then issues the Specify Interrupt Routine (SIR) macro instruction for this ICB to enter a Request Entry into the Task Monitor tables with the priority the routine is to have relative to all the other interruption handlers that he may have specified. If a program interruption of code 10 occurs, the Task Monitor queues the element in the Task Monitor Interrupt Table on the Request Entry that represents the user's ICB, so that the specified routine can be dispatched by priority. At any time, the user may delete this routine by using the DIR macro instruction and specify another routine in its place.

TASK INTEGRITY

Task integrity refers to the preservation of information which might otherwise be lost due to recursive or sequential task interruptions. One of the major functions of the Task Monitor is to maintain the integrity of service routines. This is accomplished through the management of program and machine status data, linkage conventions, and save areas.

Furthermore, the Task Monitor is provided with a programmed interruption masking capability analogous in concept to the hardware interruption masking capability of the Resident Supervisor.

Most task interruptions are independent of one another but are not necessarily mutually exclusive. That is, several task interruptions may be in process simultaneously. For example, a nonprivileged program may wish to send a message to the task's SYSOUT. The Gate Write macro will cause an ENTER supervisor call to be executed in order to eventually invoke the privileged GATE routine. The GATE routine will obtain the output line from somewhere in the nonprivileged program. If this page has never yet been referenced, the Resident Supervisor may pass a task program interruption to the Task Monitor which, in turn, will immediately invoke the Dynamic Loader. While the Dynamic Loader is processing, it is possible that an I/O interruption (signaling the completion of some previous I/O operation for the task) may be presented to the Task Monitor, which will immediately dispatch the appropriate access method posting routine. Thus, three task interruptions have occurred concurrently. That is, one occurred before the processing of another was completed. The Task Monitor has to hold the program and machine status of three service routines.

Since service routines may cause recursive interruptions but are themselves not necessarily written to be recursive, the following protection mechanisms are used when there is a possibility of a loss of information. Before passing an interruption to the Task Monitor, the Resident Supervisor checks the task interruption mask field in both the TSI and the ISA when one of the four maskable task interruptions is pending. If either mask field indicates inhibit, the task interruption is not presented to the Task Monitor. The external, asynchronous I/O, timer, and synchronous I/O interruptions must be maskable because the time of their occurrence is unpredictable and the task might have a vulnerable status at the time. Program and SVC task interruptions cannot be masked because by definition they mean that the issuing task cannot or does not wish to continue until the interruption has been processed.

Another situation in which interruptions may be inhibited occurs during scanning and dispatching in the Task Monitor. A routine dispatched by the Task Monitor has the capability of preventing the Task Monitor from dispatching other service routines until a previously dispatched routine has completed its processing and returned. This capability is referred to as enabling and disabling the Scanner-Dispatcher. Interruptions may still be presented to the Task Monitor (for either immediate dispatching or for queueing), but another service routine cannot be dispatched until the dispatched routine returns control to the Scanner-Dispatcher.

Since task interruptions may be concurrent and in some cases may occur recursively the Task Monitor preserves task integrity by managing save areas. Whenever there is a possibility that further task interruptions may occur before the current task interruption is completely processed, the appropriate Task Monitor Interruption processor saves the program and machine status data that existed when the interruption occurred. This process is called a long save and the appropriate long save area in the ISA or Task Monitor's PSECT is utilized. The non-privileged long save area is generally utilized when the interruption occurred in the User State. The privileged long save area is generally utilized if the interruption occurred in the Privileged state.

If there is no possibility of further interruptions, the interruption processor utilizes only a short save area in the ISA in order to free up some registers. For example, an access method posting routine is dispatched directly by the Task Monitor Synchronous I/O processor with no long save because interrupts remain masked.

The interruption processors also provide any routine that they call or dispatch with a Type I linkage save area (19 words). These save areas are located in the PSECT. Type I linkage and other linkage types are described in the section on Linkage Conventions. An interruption processor may immediately dispatch a service routine or it may exit directly to the Scanner-Dispatcher.

It is possible for task interruptions to be presented to the Task Monitor before an immediately dispatched routine completes processing. In general, this could result in a problem of save area availability because there are only two long save areas in the ISA. However, in practice there are only a few cases in which it might be necessary to perform a long save when the appropriate long save area might not be available. These cases are associated with supervisor call and program interruptions which can not be masked. The two principle situations are associated with a Type III linkage and a task page-relocation-exception interruption.

If an immediately dispatched service routine (such as CHECK) wishes to transfer control to a non-privileged routine (such as a user's label handling routine), a Type III linkage is employed. On Type III linkages, the Leave Privilege routine of the Task Monitor is invoked to (among other things) save the non-privileged long save area thereby making it available for another task interruption such as an ENTER supervisor call generated by the non-privileged routine.

Any routine, including the Dynamic Loader can cause a task page-relocation-exception interruption. This interruption causes an immediate dispatch to the Dynamic Loader. Special handling and save area management is provided in the case of such interruptions (see "Dynamic Loader").

An immediately dispatched routine always returns eventually to the Task Monitor interruption processor that dispatched it.

A Task Monitor Interrupt Processor always exits eventually to the Scanner-Dispatcher. If a long save was performed, the information in the appropriate long save area is moved to the Scanner-Dispatcher's long save area in the PSECT. If there are no further routines to dispatch, the Scanner-Dispatcher restores the task to its status at the time of interruption. If there are service routines to dispatch, the Scanner-Dispatcher dynamically allocates long and 19 word save areas for routines that it dispatches. This is done to free up the long save areas used by the Task Monitor interruption processors and the Scanner-Dispatcher in anticipation of further task interruptions that may occur before the processing of all pending task interruptions is completed.

Both privileged and non-privileged routines are dispatched via the Load Virtual PSW supervisor call. However, privileged and non-privileged routines are provided different save areas and return points.

The save area for a privileged routine is located in privileged virtual storage and a return is normally made directly to the Scanner-Dispatcher.

The save area for a non-privileged routine is located in non-privileged virtual storage so that the non-privileged routine can access the save area. The general purpose register used for returns is set to point to a location in the ISA that contains a Restore Privilege (RSPRV) supervisor call. The execution of the RSPRV supervisor call results in an indirect return to the Scanner-Dispatcher. The RSPRV supervisor call is used because a direct return from a non-privileged routine to a privileged routine would cause a storage protection exception interruption.

The Scanner-Dispatcher may concurrently dispatch a number of routines in the sense that a second routine may be dispatched before the first routine has returned. The Scanner-Dispatcher maintains order within both the non-privileged and privileged save areas through a push-down list which reflects the order of dispatch.

The basic function of both the Resident Supervisor and the Task Monitor is to process interruptions. The purpose of this section is to provide an overview of the various categories of interruptions processed by the system and to designate the appropriate sections of this manual which discuss the processing of some specific interruptions associated with each of the categories.

There are five classes of System/360 hardware interruptions:

- Machine Check
- External
- Program
- I/O
- Supervisor Call (SVC)

MACHINE CHECK INTERRUPTION

The Machine Check interruption indicates a CPU, Storage Element or Channel Controller error and is discussed in the section on Error Procedures

EXTERNAL INTERRUPTION

External interruptions are initially accepted by the Recovery Nucleus routine, which saves and resets the timer location in the PSA, and checks for and processes any malfunction alert interruptions (see "Error Procedures").

If the interruption is the result of the interrupt key on the operator's console being depressed, this routine loads the RSS External Interrupt PSW from the system table. This action results in an entry to the Resident Support System and the subsequent dedication of the system to the activity of the master programmer at the operator's console.

If the interruption is not the result of a malfunction alert or the interrupt key, the Recovery Nucleus transfers control to the External Interrupt routine of the Interrupt Stacker.

This routine further classifies an external interruption as a Timer interruption or a Write Direct interruption.

Timer Interruption

This interruption occurs as a result of the contents of the timer cell becoming negative. The contents of the timer are decremented by the hardware approximately every 13 microseconds. When the timer contents change from a positive (including zero) to a negative number, an interruption occurs. The primary functions of this interruption are to signal time slice end of a task, to force a timer task interruption or to activate a CPU that was placed in the wait state by the Resident Supervisor's Dispatcher.

The processing of a time slice end interruption is discussed in the section "Time Slice End Processing Example."

The conditions under which the Dispatcher will place a CPU in the Wait state and TSS/360 timer functions, in general, are discussed in "Timer Services Allocation."

Write Direct Interruptions

The CPU instruction WRITE DIRECT is used for inter-CPU communications. The WRITE DIRECT instruction forces an external interruption in the destination CPU. A special routine (called the Inter-CPU Communication subroutine) is invoked to process all inter-CPU communications. This routine is used when one CPU wishes to issue an External Start (simulated IPL) to another CPU. See section on Error Handling. Another example of the use of this routine is to reset the associative registers in another CPU to prevent residual references to a shared page whose main storage block has just been released.

PROGRAM INTERRUPTION

There are seventeen program interruption codes generated by the System/360 Model 67. If any of these program interruptions are caused while a CPU is operating in the Supervisor state, i.e., caused by the Resident Supervisor, an error is indicated and the Interrupt Stacker invokes the System Error Processor (see "Error Procedures").

Fifteen of these 17 interrupt codes are common to all System/360 systems and are specified in Principles of Operation.

If a program interruption with a code 0-15 occurs from the problem state, the Interrupt Stacker creates a program interrupt GQE and enqueues this GQE on the task's TSI. Just before the task next receives control, the Task Interrupt Control subroutine creates a programmed interruption which results in the task's being dispatched at the entry point of the Task Monitor's Program Interrupt Processor.

If the program interruption occurred in a privileged routine, the Task Monitor Program Interrupt Processor issues a System Error (SYSER) SVC which will cause the Resident Supervisor's Interrupt Stacker to invoke the System Error Processor. Upon return the task is abnormally terminated.

If the program interruption occurred in a problem program, two situations are possible.

First, the problem program may have specified a routine to handle this type of program interruption through use of the Specify Program Entry Conditions (SPEC) and Specify Interrupt Routine (SIR) macros.

In this case, the Task Monitor Program Interrupt Processor invokes the Queue Linkage Entry subroutine to activate the Request Entry for the routine that is to handle the interruption. Later, the Task Monitor Scanner - Dispatcher transfers control to the routine.

In the second situation, the problem program has not specified a routine to handle the program interruption. In this case, the Task Monitor activates a Request Entry for the Command Language DIAGNO routine. The DIAGNO routine will use the Command System User Prompter routine to obtain an appropriate error message from the System Message (SYSMSG) dataset which is then placed on the task's SYSOUT using the GATE Write (GATWR) macro. If the task is conversational, the user is prompted for corrective action. If the task is non conversational, the task is abnormally terminated (i.e., the ABEND routine is invoked). (See "Command Controller.")

Program interruption codes 16 and 17 are unique to the System/360 Model 67.

Program interruption code 16 indicates that a page table is unavailable. Program interruption code 16 may signal that a Shared Page Table is unavailable. This is discussed in "Internal Sharing."

Program interruption code 17 indicates a page relocation exception. That is, the Dynamic Address Translation unit found the page "unavailable" when the Page Table entry associated with a virtual storage address was inspected. The processing of a page relocation exception interruption within the Resident Supervisor is discussed in "Page Relocation Exception Example."

If the Page Posting routine of the Resident Supervisor finds that the External Page Table entry for a page that has just been brought into Main storage has an "Address Constants Unprocessed by Dynamic Loader" flag turned on, a program interruption GQE (with an interruption code of 17) is enqueued on the task's TSI. This results in a software interruption being passed on to the Task Monitor's Program Interruption Processor which will, in turn, immediately dispatch the Dynamic Loader. The reason for this flag and the processing of this software program interruption, are discussed in the section on the Dynamic Loader.

If a user attempts to reference a virtual storage address not allocated to him, either a code 16 or 17 interruption occurs. The Resident Supervisor detects this error and enqueues a program interruption GQE on the Task's TSI.

The interruption code field of an extended or virtual PSW permits the specification of 65,535 interrupt codes. Within TSS/360, the program interruption codes in the range 32 to 65,535 are used in a virtual PSW to designate additional program errors and are designated as extended interruption codes. Interruption codes 18 through 31 are reserved for future hardware interruption expansion, and codes 65,280 through 65,535 are set aside for temporary definitions for use in the development or testing of TSS/360. The extended interruption codes can be placed into a virtual PSW by either the Resident Supervisor or the Task Monitor.

The Resident Supervisor uses extended program interruption codes when it discovers errors associated with a Supervisor Call or permanent hardware errors associated with task operations. The Task Monitor will place an extended program interruption code into a virtual PSW whenever the Task Monitor discovers errors associated with a Supervisor Call that represents a request for the services of a privileged routine. If the Task Monitor discovers a virtual PSW containing an interruption code greater than 31, a SYSER SVC will be issued or the Command System DIAGNO routine will be invoked as described for program interruption codes 1 to 15. A complete listing of the extended program interruption codes defined in TSS/360 is contained in System Programmer's Guide.

I/O INTERRUPTION

The I/O interruption is the normal method used by the input/output hardware to communicate the termination status of an input/output operation to a CPU.

For efficiency, the Interrupt Stacker places interruptions associated with paging drums on the Paging Drum Interrupt Processor queue and all other I/O interruptions on the Channel Interrupt Processor queue.

If the appropriate Interrupt Processor discovers an I/O error, processing will proceed as described in "Error Procedures."

Apart from error interruptions, an interruption GQE on an interrupt processor queue can represent either a synchronous interruption or an asynchronous interruption.

As defined in TSS/360, a synchronous interruption is an interruption resulting from an I/O operation initiated by the Resident Supervisor. Examples of the processing of synchronous interruptions are presented in "Example of BSAM Processing," and "Paging."

An asynchronous interruption is any interruption that is not synchronous and generally is an interruption resulting from a user pressing the attention button on his terminal. When the Channel Interrupt Processor encounters an initial attention interruption from a terminal, the Task Initialization Routine is called to create a new task. Then an asynchronous interruption GQE is enqueued on the created task's TSI.

This causes a software interruption to invoke the Task Monitor's Asynchronous Interrupt Processor which, in turn, invokes the Command System Virtual Memory Task Initiation and Intitial Attention Interrupt Processing routines to supervise initialization of the new task. This process is described in "Creation of a Conversational Task."

In the case of other asynchronous I/O interruptions, the Resident Supervisor ignores the interruption unless a task has issued a Set Asynchronous Entry supervisor call to direct asynchronous interruptions from that device to a particular task.

If the interruption can be associated with a task, the interuption is passed to the Task Monitor as an asynchronous interruption. The Task Monitor ignores the interruption unless a Request Entry can be found for asynchronous interruptions on this device.

In the case of asynchronous interruptions from a SYSIN terminal (attention interruptions caused by depressing the attention key on the terminal), there will always be an appropriate Request Entry.

Normal processing of attention interruptions is the responsibility of the Command System Attention Handler.

LOGON2 receives control during task initialization and issues a SIR macro, which points to a prebuilt Interrupt Control Block that identifies the Attention Handler as the routine to receive control when future attention interruptions are received from SYSIN.

Under these conditions, any attention or pseudo-attention interruption causes the Task Monitor to invoke the Queue Linkage Entry routine to enqueue a linkage request on the Request Entry for the Command System Attention Handler. When the Attention Handler is dispatched by the Task Monitor Scanner-Dispatcher, the Attention Handler will invite the user to enter a command by issuing an exclamation symbol to the terminal.

It is desirable to allow a problem program to dynamically pause and seek a new command from the task's SYSIN. This can be accomplished through the Read Command from SYSIN (CLIP) and Read Command from Conversational SYSIN (CLIC) macro instructions.

The CLIP macro instruction issues a supervisor call which is passed to the Task Monitor as a task SVC interruption. The Task Monitor creates a pseudo-attention interruption by invoking the Queue Linkage Entry routine to enqueue a linkage request on the Command System Attention Handler Request Entry.

The CLIC macro instruction causes the same processing except that the SVC generated by the macro instruction is ignored if the task is nonconversational.

A user may establish routines to process attention interruptions. The routines are established through the use of the SAEC and SIR macro intructions. The user then issues a User Attention (USATT) macro instruction. This causes the Attention Handler to deactivate its Request Entry for attention interruptions, thus leaving the problem programs Request Entry as the only active Request Entry for SYSIN attention interruptions. All subsequent attention interruptions are processed by the specified routine. If the Attention Handler did not deactivate its Request Entry, the Attention Handler would receive attention interruptions instead of the user's routine because the Attention Handler's Request Entry has a higher priority than a problem program's Request Entry. When the user wants processing of attention interruptions to be resumed by the system he issues a Clear Attention (CLATT) macro instruction.

Use of the CLIP or CLIC macros after a USATT macro will cause the Attention Handler (not the user's routine) to receive control from the Task Monitor.

User routines to process attention interruptions may also be handled by the AETD macro instruction. The AETD macro instruction allows a user to interrupt his programs during execution via the attention key, and thereby enter a predefined user routine for processing of the attention interrupt. The execution of the AETD macro instruction generates a table (AET) containing the addresses of routines which are to be given control when a user presses the attention key a specified number of times. When the attention key is pressed, the Attention Handler first determines whether the current value of the attention count has a corresponding entry in a connected Attention Entry Table (AET). If it does, the appropriate user's attention routine is executed. If there is no corresponding entry in the AET, the Attention Handler performs various tests (see "Attention Handler").

The user might employ the AETD macro instruction to pass control to any user-provided control systems, or to provide partial backup in a current task so that a bad error situation does not cause the task to require total reconstruction. The AETD macro instruction can be used to predefine simple automatic debugging procedures by using PCS commands in the user-coded attention handling routines.

When the AETD macro instruction is issued with no operand, the AET, if one was previously defined, is disconnected, and the Attention Handler will be invoked for subsequent handling of attention interrupts.

If a user presses the attention key while a message is being printed at his terminal, the transmission is terminated. An attention interruption which interrupts an active channel program for a SYSIN device is not an asynchronous interruption. It is passed to the Task Monitor as a synchronous interruption (see "Command Controller").

SUPERVISOR CALL INTERRUPTION

The supervisor call (SVC) is the normal method of communication between a task and the supervisor. The interruption is caused by the execution of an SVC instruction which is usually imbedded in the expansion of a macro instruction. An SVC instruction has a one byte variable field which is used to contain a code (0 to 255) to indicate the operation to be performed. In TSS/360, an SVC interruption can occur under two general conditions:

- The SVC issued from a task in the problem state.
- The SVC occurred while the CPU was operating in the Supervisor state.

If the interruption occurred in the problem state, the interruption code in the PSA is examined to determine the type of request represented:

- A request for problem program services (SVC codes 0-63)
- A request for Time Sharing Support System Services (SVC codes 64-95)
- A request for privileged program services (SVC codes 96-127)
- A request for Resident Supervisor services (SVC codes 128-255)

The SVC codes from 0 to 63, reserved for requests for the services of problem programs, are not defined at present.

When one of the SVC codes from 96 to 127, reserved for requests for the services of routines operating in privileged virtual storage, or SVC 254 (LVPSW) is encountered, the SVC-interrupt routine of the Interrupt Stacker checks for the existence of other pending interruptions. If no interruption exists, the Interrupt Stacker places the address of the appropriate processor in the current PSW location of the ISA and issues the LPSW instruction which causes the interruption to be immediately serviced. If another interruption is pending, the Interrupt Stacker switches addresses as above and transfers control to the Queue Scanner.

Later, the Task Interrupt Control subroutine of the Dispatcher will create a software interruption from the GQE and the Dispatcher will place the task in execution at the entry to the Task Monitor's SVC Interrupt Processor.

Several of the supervisor calls in this range are pre-defined and thus are not available for dynamic interruption specification by the task.

These pre-defined supervisor calls are:

Macro ID	SVC Code	Macro Name
DLINK	127	Dynamic Linkage Request (i.e., load a program module)
PCSVC	125	Program Checkout Subsystem Call

DELET	123	Delete a program module
RTRN	122	Return to calling program
ENTER	121	Enter a Privileged Service Routine
RSPRV	120	Restore Privilege
CLIC	119	Read Command from SYSIN (conditionally)
CLIP	118	Read Command from SYSIN
RAE	117	Restore and Enable task interrupts
EXIT	116	Writes a message to SYSOUT. If the task is conversational, the task is placed in command mode. If nonconversational, the next command is read from SYSIN.

The DLINK and DELET Supervisor calls are discussed in the section "Dynamic Loader".

The PCSVC is discussed in "Program Control System."

The RTRN supervisor call is discussed in "Example of System Operation - Nonconversational Processing."

The RSPRV supervisor call was discussed in "Task Integrity" and "Linkage Conventions."

The ENTER Supervisor call is used to request the Task Monitor SVC Interrupt Processor to immediately dispatch a privileged service routine.

The Task Monitor uses as an argument to inspect two Enter Tables an Enter code which the execution of the Enter macro caused to be placed in general purpose register 15.

The first Enter Table contains entries which either point to corresponding entries in the second Enter Table or indicate that the Enter code is unassigned. If the Enter code is unassigned this constitutes an error and the Request Entry for the Command Language DIAGNO routine is activated in the Task Monitor Interrupt Table. DIAGNO will subsequently be invoked by the Task Monitor Scanner-Dispatcher. If the ENTER supervisor call was issued by a privileged routine, this constitutes an error and a SYSER supervisor call is issued by the Task Monitor. When control is returned to the Task Monitor, the task is abnormally terminated.

The second Enter Table contains the address constants used in invoking the service routine. The Appendix section of the Task Monitor Program Logic Manual contains a table of system enter codes.

The DDEF and CATALOG routines are examples of the type of routine dispatched in this way. This second Enter Table also specifies whether the routine to be entered (or a routine the entered routine might call) may cause another task interruption.

Another flag in each Enter Table entry specifies whether the entered routine can be recursively interrupted. Currently, all TSS/360 service routines specify that they cannot be interrupted by interruptions that would affect the operation of the interrupted routine.

An example of the processing of an Enter SVC used to invoke the Basic Sequential Access Method Read/Write routine is described in "Example of BSAM Processing."

In the case of SVC codes from 128 through 255, requests for the services of the Resident Supervisor, the Interrupt Stacker will invoke the SVC Queue Processor. This direct linkage is possible because the processor is never interlocked.

The Resident Supervisor SVC Processor will use the SVC interruption code to inspect an entry in the processor's SVC Flag Table. Each Flag Table entry contains information specifying the privilege class necessary to invoke this SVC. If this information is not compatible with the authority code specified in the task's TSI, a task Program Interrupt GQE is placed on the TSI. If the SVC was issued from a routine with the proper authority, the SVC Queue Processor uses the SVC interruption code to obtain a pointer from its SVC Address Table. The SVC Queue Processor then invokes the proper SVC subprocessor to initiate the processing of the supervisor call.

A list of SVC subprocessors was presented in the section on "Resident Supervisor." Examples of the processing of SVC requests for Resident Supervisor services are presented throughout this manual. An especially illustrative example is presented in the section, "Example of BSAM Processing."

If there is no SVC subprocessor associated with the SVC interruption code, a program interruption GQE is enqueued on the task's TSI.

There is at least one example of a SVC that requires processing by both the Resident Supervisor and the Task Monitor. This is the Virtual Memory to Virtual Memory Send [VSEND] SVC. Whenever one task wishes to send a message to another task, the sending task issues a VSEND SVC. After the Resident Supervisor's VSEND SVC subprocessor has completed processing this Supervisor Call, an External Interrupt GQE is enqueued on the receiving task's TSI. This Supervisor Call is described in "Communication."

It is possible for the Interrupt Stacker to receive an SVC interruption which originated while the CPU was in the Supervisor state, i.e., the SVC instruction was executed by the Resident Supervisor itself. This is generally an ERROR SVC that was issued by the Resident Supervisor in order to invoke the System Error processor. Even if the SVC was not an ERROR SVC, the Interrupt Stacker issues an ERROR SVC because any other SVC issued in the supervisor state is an error.

The processing of such interruptions is discussed in "Error Procedures."

The Resident Terminal Access Method (RTAM) provides a resident method for communication with terminals, reducing the amount of paging, and therefore the amount of time, necessary for terminal I/O operations. RTAM is used by both TSS/360 and Multiterminal Task (MTT) operation, the latter allowing several terminals to use an application program simultaneously. The term 'system' is used to indicate TSS mode operation; the term 'application' is used for MTT mode operation.

RTAM Control Blocks

The two most important control blocks used with RTAM will be mentioned several times throughout this document:

- Multiterminal Status Control Block (MTSCB) -- The MTSCB is a resident table which services TSS/360 or MTT. When used in TSS mode, there is a single system MTSCB used to service all terminals (See Figure 21). The system MTSCB contains the virtual storage address of the system TCT and buffer pages, along with other information. The system MTSCB is addressed with an EXTRN.

  When used in MTT mode, there is one application MTSCB created for each MTT task (see Figure 22). The application MTSCB contains the virtual storage address of the application TCT and buffer pages, along with other informa-

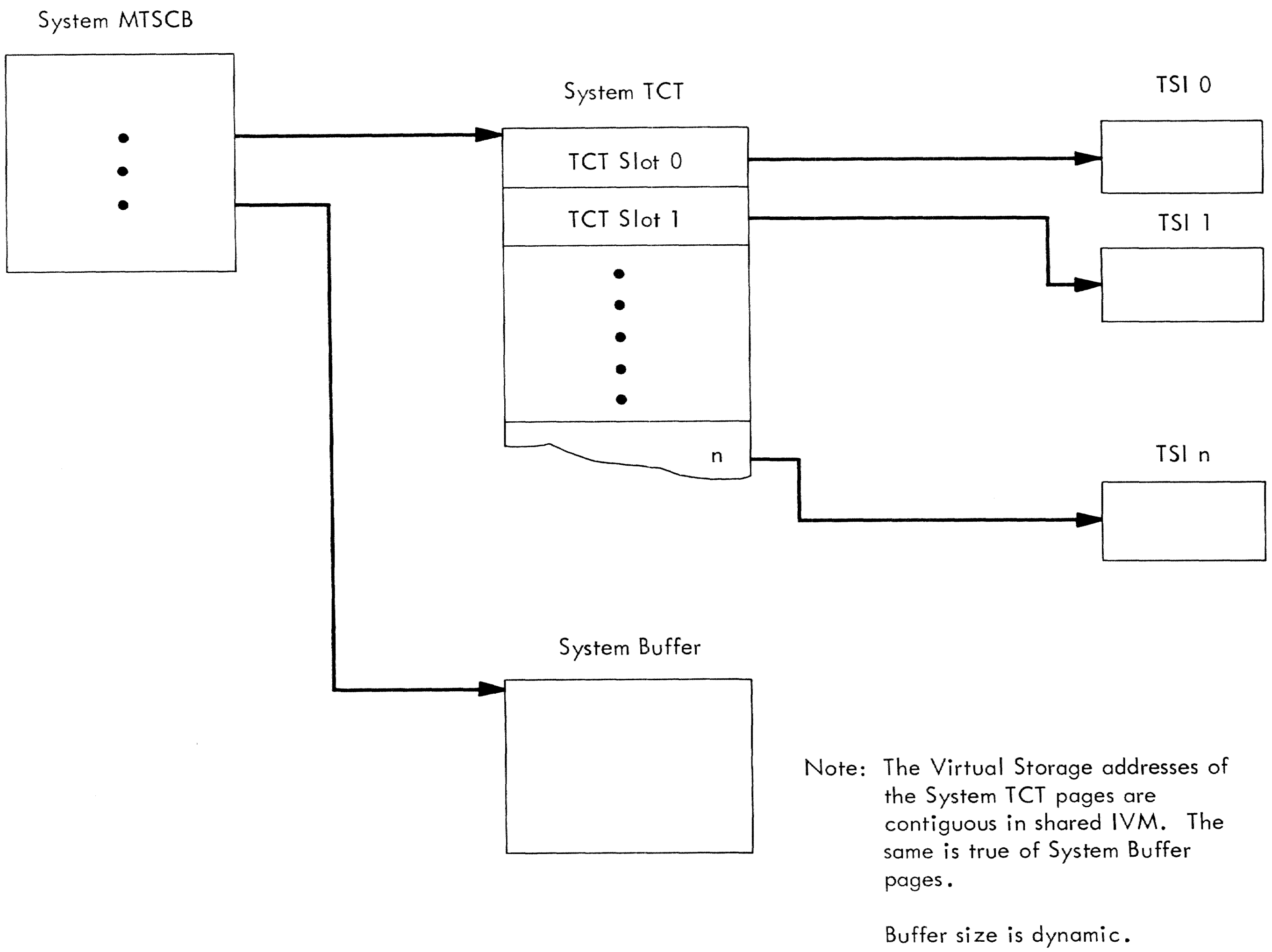

System MTSCB
System TCT
TCT Slot 0
TCT Slot 1
n
TSI 0
TSI 1
TSI n
System Buffer
Note: The Virtual Storage addresses of the System TCT pages are contiguous in shared IVM. The same is true of System Buffer pages.
Buffer size is dynamic.

Figure 21. RTAM System Control Block Relationship

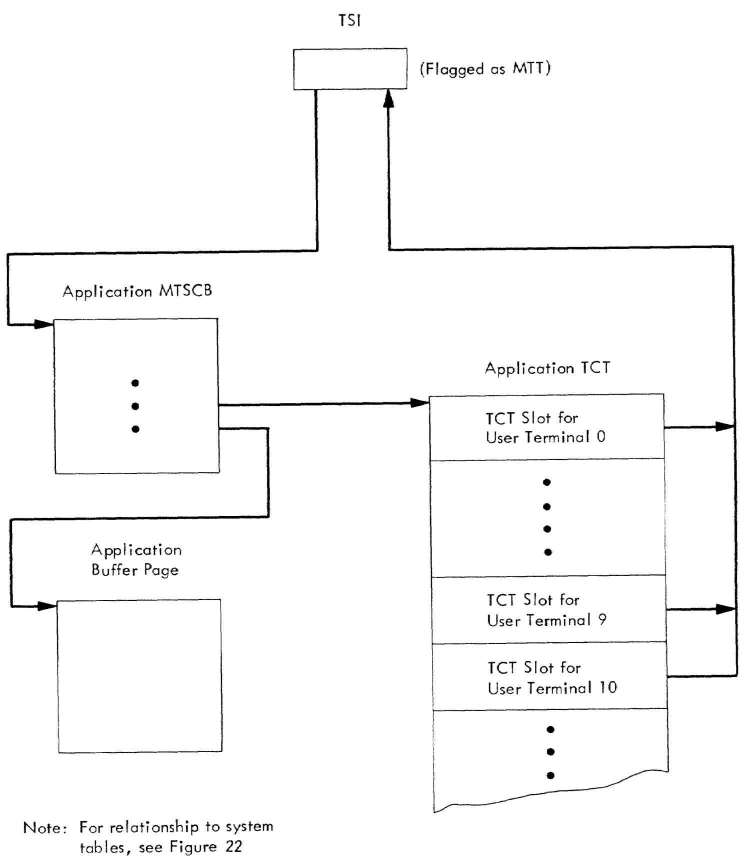

TSI
(Flagged as MTT)
Application MTSCB
Application TCT
TCT Slot for User Terminal 0
Application Buffer Page
TCT Slot for User Terminal 9
TCT Slot for User Terminal 10
Note: For relationship to system tables, see Figure 22

Figure 22. RTAM MTT Control Block Relationship

tion. The address of an application MTSCB is contained in the TSI of the task that issued the MTT command.

- Terminal Control Table (TCT) -- In TSS mode, there is one system TCT with a slot containing a TSI pointer for each TSS mode task attached to the system. In MTT mode, the application TCT contains one slot for each user terminal, each slot pointing back to the TSI of the task which issued the MTT. The TCT is posted with pointers to the buffer pages and the TSI, and control information for the terminal.

RTAM TSS Initialization

During the initialization procedure, when the Read placed on the line by TCS finds LOGON, the Terminal Communications Subprocessor (TCS) will call the TCT Entry Allocation Subprocessor to assign a system TCT slot and buffer pages to the task. (For detail on the initial interrupt, see 'Creation of a Conversational Task' in this book.)

RTAM TSS Mode Operation

The LOGON processor issues the ATTACH macro instruction to find the system TCT entry (assigned during initialization) and locate the virtual storage buffer address of the LOGON parameters. If the parameters are valid, a GATWR macro instruction informs the user that his LOGON was successful. In response to this GATWR issued by LOGON, the GATE processor sets the Read/Write I/O flags in the system TCT slot and issues the ATCS macro instruction to call TCS. TCS builds appropriate CCWs for read and write requests to the terminal, based on system TCT settings established by GATE, and returns to GATE before completion of the CCW execution. Completion of the CCW caused a synchronous I/O interrupt, which allows TCS to post the completion in the terminal's system TCT slot. Control is eventually returned to the task.

RTAM MTT Initialization By Administrator

To establish MTT mode, an MTT Administrator with O or P authority code and T privilege class, issues a LOGON command and becomes a system user, as shown under 'Creation of a Conversational Task' in this book. The Administrator subsequently enters the MTT command. The TSI is flagged MTT, the application program is loaded, the CONN SVC is issued to build the application MTSCB and allocate storage sufficient for application TCT and buffer pages, and the application program is explicitly called (see Access Methods PLM, module CZCTC). This task is now in MTT mode (see Figure 23).

RTAM MTT Initialization By User

If the initial interrupt from a terminal is for the purpose of using an MTT application, the BEGIN command will be entered. TCS will search the application MTSCBs for the module named in the BEGIN parameters, and assign an application TCT slot to this MTT user terminal.

The application TCT slot will point back to the TSI of the task that originally issued the MTT command. In fact, this TSI will be used by all terminals attached to this application program (see Figure 23).

RTAM MTT Mode Operation

The MTT application program that is activated by the MTT command will service many remote terminals simultaneously; but within the time slice of a single task, and with the system overhead of a single task.

Within the application program, any standard TSS/360 commands may be entered, plus a special set of MTT macro instructions (READQ, WRITEQ, FINDQ, etc.) which allow the application to communicate directly with the user terminals. (See Multiterminal Task Programming and Operation and the System Programmer's Guide SRLs, and the Access Methods PLM.)

The user connects to the application program with the BEGIN command, using that format defined for the application program. (See the Command System User's Guide SRL.)

Any subsequent communication between the application user terminal and the application program must be with commands defined in the application program. The creator of the application program must therefore provide documentation support for the user.

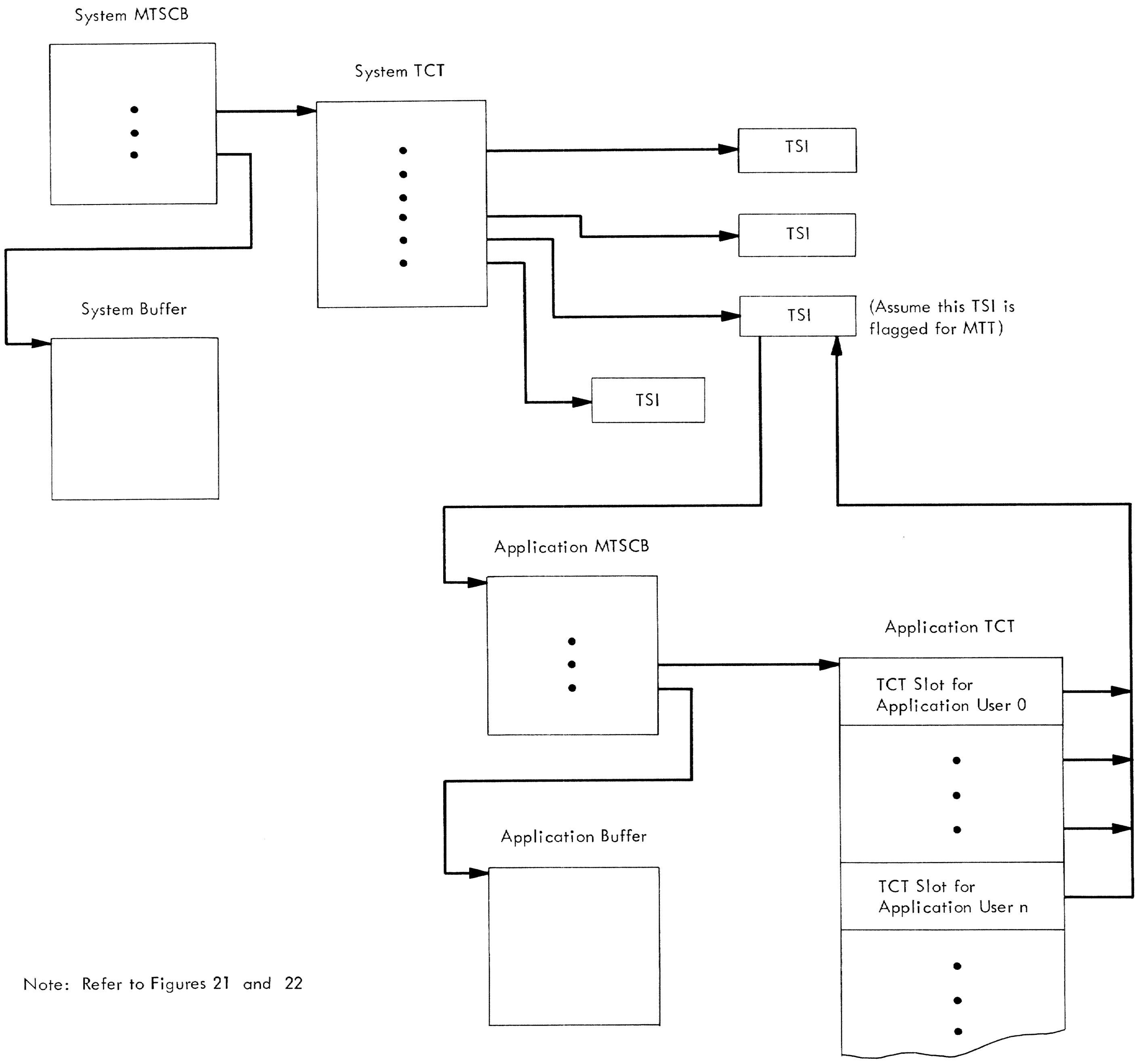

System MTSCB
System TCT
TSI
TSI
TSI
(Assume this TSI is flagged for MTT)
TSI
System Buffer
Application MTSCB
Application TCT
TCT Slot for Application User 0
TCT Slot for Application User n
Application Buffer
Note: Refer to Figures 21 and 22

Figure 23. RTAM Application/System Control Block Relationship

DATA MANAGEMENT ACCESS METHODS

Traditionally, Data Management access methods have been composed of routines to perform two logical functions:

- Effective handling of data structures
- Effective handling of physical devices

In TSS/360 there are two categories of access methods:

- Virtual Access Methods (VAM)
- Sequential I/O Access Methods (SAM)

The Virtual Access Methods have been specifically designed for a time-sharing environment and present a clear division between data set and physical device management. There are three Virtual Access Methods each of which provides access and processing capability for a specific type of data set organization:

- Virtual Sequential Access Method
- Virtual Index Sequential Access Method
- Virtual Partitioned Access Method

In all three of these access methods, data set management is performed in virtual storage and all physical device management (i.e., I/O and error recovery) is performed by the Resident Supervisor.

The Sequential I/O Access Methods are all characterized by the fact that the access method specifies the appropriate channel program and controls the logic of error recovery in addition to performing data set management. These access methods call on the Resident Supervisor to perform the actual execution of the Channel programs. The sequential I/O access methods are:

- Basic Sequential Access Method (BSAM)
- Queued Sequential Access Method (QSAM)
- Multiple Sequential Access Method (MSAM)
- Terminal Access Method (TAM)
- Resident Terminal Access Method (RTAM) (see section on RTAM/MTT)
- I/O Request Facility (IOREQ)
- On-Line Test System Access Method (OLTAM)
- Drum Access Method (DRAM)

The Catalog Services Routines OBTAIN and RETAIN which, in effect, make up an access method, are discussed in "Catalog Service Routines." A summary of the macro instructions available for the major TSS/360 Access Methods is presented in Figure 24. An overview of these macros is presented in _Concepts and Facilities_. These macros are described in detail in _Assembler User Macro Instructions_ and _Programmer's Guide_.

VIRTUAL ACCESS METHODS

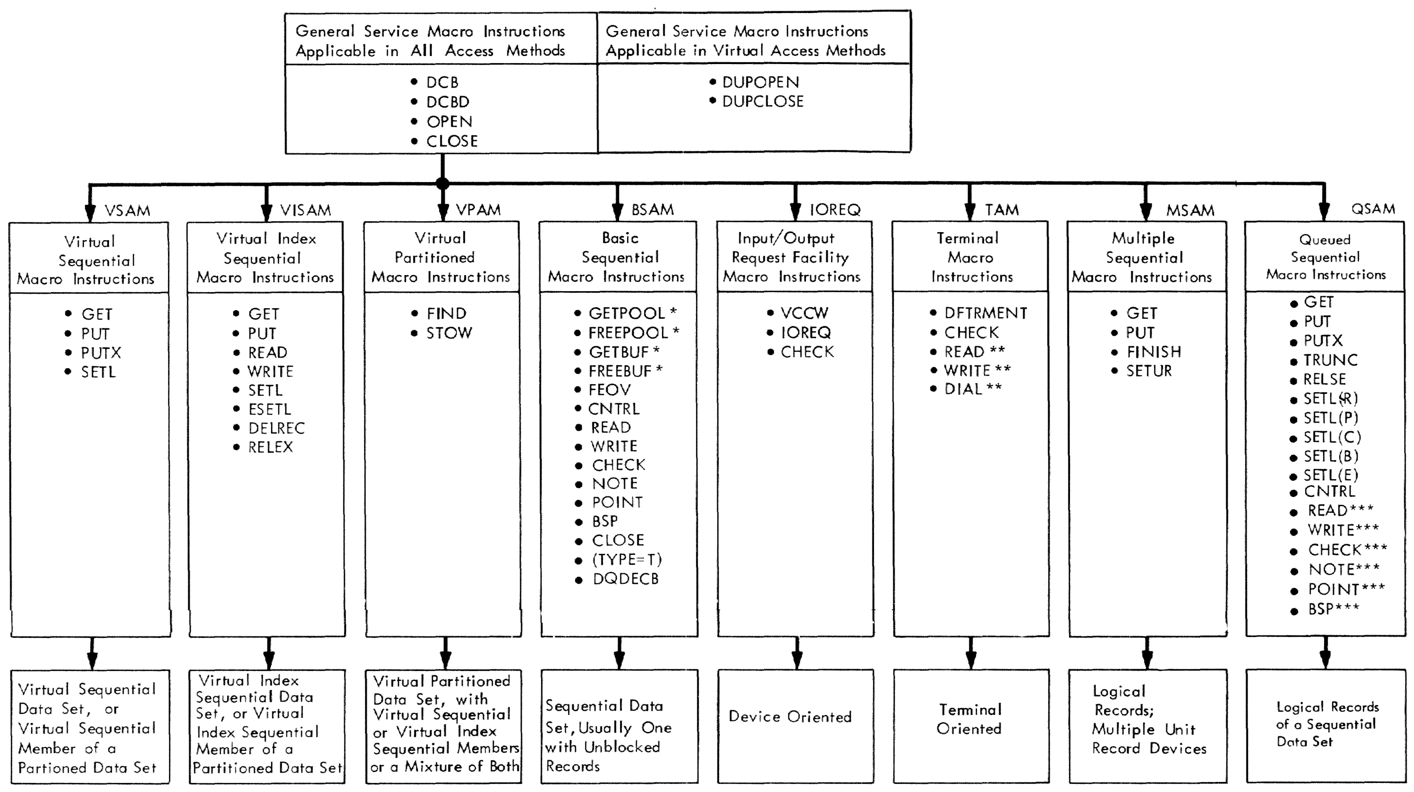
The Virtual Access Methods comprise routines especially designed for TSS/360. Data sets with a virtual storage organization reside only on direct access volumes; however, VAM data sets may be copied to tape via the VT command to free public storage or create backup copies, and may then be restored to direct access storage via the TV command when required for processing (see "Command Routines"). Users create, read, and process these data sets on the basis of the logical records they contain. TSS/360, however, organizes these data sets by pages and uses these pages as the unit of transfer between the direct access device and main storage.

The name "virtual" was given to these access methods to reflect the fact that they utilize only one physical block size; that of a page.

The direct access volumes, on which TSS/360 virtual organization data sets are stored, have fixed-length, page size data blocks. No key field is required. The record overflow feature is utilized to allow data blocks to span tracks, as required. The entire volume, with the current exception of part of the first cylinder, which is used for identification, is formatted into page size blocks.

The page-sized block for data storage was selected for a number of reasons. It is as small as the smallest unit of main storage allocation. It is large enough so that direct access throughput is high. Rotational delay is a significant factor in direct access throughput, since it cannot be overlapped as mechanical seek time can. Any block size significantly smaller than a page would be extremely wastefull of total direct access capacity unless elaborate strategies were utilized to avoid rotational delay.

The need for a large block size is also apparent when the simultaneous direct access activities of multiple users are considered. Due to conflicts in demands for access arms, a mechanical seek may frequently be required before accessing a data block. The larger block size makes better use of the total access cycle while, at the


General Service Macro Instructions Applicable in All Access Methods
• DCB
• DCBD
• OPEN
• CLOSE
General Service Macro Instructions Applicable in Virtual Access Methods
• DUPOPEN
• DUPCLOSE
VSAM
Virtual Sequential Macro Instructions
• GET
• PUT
• PUTX
• SETL
Virtual Sequential Data Set, or Virtual Sequential Member of a Partioned Data Set
VISAM
Virtual Index Sequential Macro Instructions
• GET
• PUT
• READ
• WRITE
• SETL
• ESETL
• DELREC
• RELEX
Virtual Index Sequential Data Set, or Virtual Index Sequential Member of a Partitioned Data Set
VPAM
Virtual Partitioned Macro Instructions
• FIND
• STOW
Virtual Partitioned Data Set, with Virtual Sequential or Virtual Index Sequential Members or a Mixture of Both
BSAM
Basic Sequential Macro Instructions
• GETPOOL *
• FREEPOOL *
• GETBUF *
• FREEBUF *
• FEOV
• CNTRL
• READ
• WRITE
• CHECK
• NOTE
• POINT
• BSP
• CLOSE
• (TYPE=T)
• DQDECB
Sequential Data Set, Usually One with Unblocked Records
IOREQ
Input/Output Request Facility Macro Instructions
• VCCW
• IOREQ
• CHECK
Device Oriented
TAM
Terminal Macro Instructions
• DFTRMENT
• CHECK
• READ **
• WRITE **
• DIAL **
Terminal Oriented
MSAM
Multiple Sequential Macro Instructions
• GET
• PUT
• FINISH
• SETUR
Logical Records; Multiple Unit Record Devices
QSAM
Queued Sequential Macro Instructions
• GET
• PUT
• PUTX
• TRUNC
• RELSE
• SETL(R)
• SETL(P)
• SETL(C)
• SETL(B)
• SETL(E)
• CNTRL
• READ***
• WRITE***
• CHECK***
• NOTE***
• POINT***
• BSP***
Logical Records of a Sequential Data Set

*These routines are primarily designed for use with BSAM; however, they may be used with any Access Method.

**In TAM, only class E privileged programs may issue the READ/Write macro instructions. Nonprivileged class D programs issue GATRD/GATWR macro instructions.

***These BSAM routines are invoked by QSAM to perform various operations.

Note: For RTAM, See Resident Terminal Access Method/Multiterminal Task

Figure 24. Summary of Data Management Macro Instructions and Data Set Organizations

same time, reducing the frequency of access requests by each user.

The direct access volume-packing efficiency is also quite high for page-sized blocks. First, the data recording space is utilized at better than 90% of its theoretical capacity (if cylider-length blocks were written). Second, the smallest external storage allocation unit is a single page; so a large number of small data sets can be kept on one volume. Furthermore, the freedom from requirements for physically contiguous external storage space leads to higher volume packing efficiency.

The Virtual Access Methods are device independent across the range of direct access devices. That is, it is perfectly feasible for a VAM data set to have physical records recorded on, say, both 2311 and 2314 devices in any mixture. Furthermore, user information is referenced by its location relative to the beginning of the data set, never by its location with respect to external storage. As a result, it is entirely practical within VAM operations to move data sets, either in part or in total, among a hierarchy of devices.

The dataset copy routine of the command system is able to convert information simply from any VAM organization to any other VAM data set organization with a small number of instructions. This is because the record formats, contents and control fields are identical between all organizations.

The Virtual Access Methods do not use the hardware data-searching facilities of the direct access control units. The operation of these searching facilities would lengthen the data access cycle and thereby reduce direct access throughput. In general, it is better to conduct a programmed search in virtual storage. Conceptually, this amounts to substituting auxiliary storage for external storage. This concept of programmed searches can be extended to secondary indexes. For instance, the TSS Assembler Macro Library is maintained as a line data set for maintenance purposes. Major activities against this file are, however, based upon an alphabetic search based upon macro name. There are two specific routines that exemplify the type of secondary index processing that VAM supports. The first routine, called Build Index, is used to scan the entire macro file, selecting information as to the first occurrence of a given macro name and its line number within the file. This information is then placed within a page-oriented data image, sorted by alphabetic name. The second routine, called Search Index, uses VAM to bring this entire page-oriented image into Virtual Storage and conducts a binary search against the secondary index, to find the location of a macro expansion within the line data set.

The utilization of the Resident Supervisor's page-oriented I/O facility significantly simplifies the implementation of the VAM access methods. This is due to the elimination of device-dependent operations (with complex CCW lists), standardization of block size, and elimination of such exceptional procedures as end-of-volume operations.

TSS/360 assures that only those pages of a data set that are actually required are brought into main storage and that only those pages containing updated information are written back onto external storage. VAM organizes data sets by relative page number. That is, as each page of a data set is created it is assigned a page number relative to the beginning of the data set. On external storage these relative page numbers are related to the external storage addresses where the pages reside. This information is stored in the Data Set Control Blocks (DSCBs) residing in the volume.

In virtual storage, the relative page numbers are related to external storage addresses through a table which is created from the data set page entries contained within the DSCBs. This table is called the Relative Page/External Page Correspondence Table (RESTBL), and is maintained by VAM routines (1). As a record is desired, for example, using a locate mode GET in the most straightforward case, the appropriate external storage address of the page in which the record is contained is obtained from the RESTBL and passed to the Resident Supervisor which will place this address in an external Page Table (XPT) entry which is associated with a virtual storage page-sized buffer (2). Note that the page itself is not read into main storage at this time.

When a user addresses a record in his virtual storage buffer, a paging relocation exception interruption occurs and the Resident Supervisor paging processors proceed to bring the page into storage from external storage (3).

Frequently, when updating a data set, only a portion of the logical records are actually updated, although the entire data set is read.

In VAM, a buffer page is written onto external storage only if that page has been changed. When it is necessary to write a buffer page back onto external storage, the appropriate VAM routine obtains the external storage address of the page from the RESTBL and passes the virtual storage address of the buffer along with this external storage address to the Resident Supervisor. The appropriate Resident Supervisor paging processors then proceed to write the buffer page back into the data set on external storage.

From this it can be seen that the RESTBL represents, conceptually, a level of paging control above the relocation tables. The function of the relocation tables is the mapping of pages that have been allocated virtual storage. The prime function of the RESTBL is to map the pages of a data set into virtual storage.

This is accomplished by mapping the data set extents describing the external storage occupied by the data set into a table that is constructed in the same order in which pages are associated with the data set. This table is then used to map the external storage locations of a given portion of the data set into a virtual storage buffer. The size of the buffer controls the amount of virtual storage allocated to the data set at any point in time. This second level of mapping allows the user to process a VAM formatted data set that can be as large as 65,000 pages, which is a great deal larger than 16 million bytes directly addressable by the 24 bit system.

VAM brings into the buffer only those pages of the data set which are currently needed. The size of this buffer need not be limited to one page, but may be as large as 256 pages, thereby allowing a user to efficiently process a sequentially organized data set containing records that are a great deal larger than one page.

Figure 25 shows the overall concept of the Virtual Access Methods.

IMPLEMENTATION

The data set structures that each of the Virtual Access Methods supports are described in Concepts and Facilities and are briefly summarized here.

Virtual Sequential Data Sets

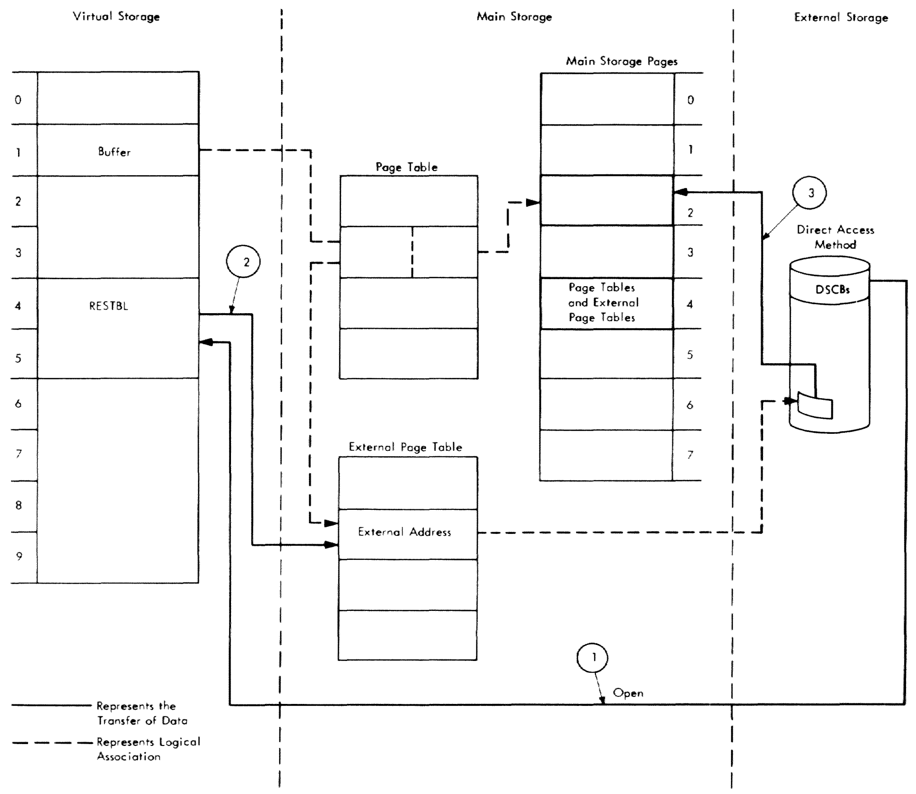
In a virtual sequential data set, the order of the logical record is determined solely by the order in which the records are created. In creating this type of data set, the user provides the system with a stream of logical records. The system concatenates the records, organizes the records into pages, and stores the data set page by page on a direct access device. After each record is stored, the system makes its retrieval address available to the user's program. Users employing assembler language can form another virtual sequential or virtual index sequential data set containing these retrieval addresses. After the data set has been created, if the user wishes to make an orderly sweep through it, the records can be read back in the order in which they were created merely by the user requesting one logical record after the other. An assembler user can also read and update logical records nonsequentially by providing the required address of each record involved.

Virtual Index Sequential Data Sets

A virtual index sequential data set is one in which the logical records are organized into an ascending collating sequence, based on a data key associated with each record. The data key may be a control field that is an actual part of the record itself, or it may be an arbitrary identifier (such as line number) which is the beginning of each logical record, and is added to each record to give it a unique key. A virtual index sequential data set that is organized by line number is called a line data set.

In addition to the logical records, virtual index sequential data sets contain a page directory and locators that relate the keys and physical address of the record in the data set.

The page directory is initially set up when the data set is created. The page directory gives the value of the key for



•Figure 25. Relationship Among RESTBL, Virtual Memory, and Main Storage the first record in each data page. There is no entry in the page directory for the first data page. The page directory can consist of one or more pages, depending upon the size of the data set.

In each page of the data set is an ordered set of locators, one locator per record. Each locator specifies either the physical location of the record in the page, or the position of a corresponding locator in an overflow page. Overflow pages are provided automatically by the system when it is necessary to logically insert a record between two existing records after the data set has been created and there is not enough room to place the records on the page on which the record belongs. The record is thus available in proper logical sequence even though it is not physically located in sequence.

Virtual Partitioned Data Sets

A virtual partitioned data set is used to combine individually organized data groups into a single data set. Each group of data is called a member, and each member is identified by a unique name. The partitioned organization allows the user to refer to either the entire data set or to any member of that data set.

References to individual members may be made through a directory called the partitioned organization director (POD). When a partitioned data set is created, a POD is set up to keep track of each member. As members are added, deleted, or changed, the directory information is automatically updated.

The first entry in the data set is the partitioned directory, which is used to locate the member of the data set. Each member begins on a new page; any space remaining on the previous page is unused.

Provision is made for users to assign additional names, called aliases, to each member, and for the location of each member on the basis of the member name of any of its aliases. The partitioned data set organization is thus ideally suited for storage of libraries of program or other groups of data that are referred to frequently.

SEQUENTIAL I/O ACCESS METHODS

BASIC SEQUENTIAL I/O ACCESS METHOD

The Basic Sequential Access Method (BSAM) is a Sequential I/O Access Method and performs two major functions in TSS/360.

First, BSAM provides a limited data set compatibility with the Operating System/360 by supporting the direct access or unlabeled or standard labeled magnetic tape data set formats (except for the direct access split cylinder format) that are produced by the OS/360 Basic Sequential and Queued Sequential Access Methods.

In this same vein, TSS/360 will accept or create data sets recorded using American National Standard Code for Information Interchange (ASCII) formats. Translation tables are used, allowing the internal processing to be handled in EBCDIC.

Secondly, BSAM is the primary means within TSS/360 of accessing magnetic tapes. BSAM creates the channel programs for sequentially accessing tapes or disks, and passes a control block called an I/O Request Control Block (IORCB) containing the Channel program and buffer information to the Resident Supervisor through a Supervisor Call. The Resident Supervisor, in turn, executes the channel program, records any pertinent error information and passes the IORCB control block back to BSAM which then attempts error recovery if necessary, and informs the user of the results of the I/O operation by posting the information in a Data Event Control Block (DECB). The IORCB format is shown schematically in Figure 26.

Since BSAM is a basic access method, the user must determine the outcome of his request before he can do any processing dependent on that request. The DECB provides a means for making the determination.

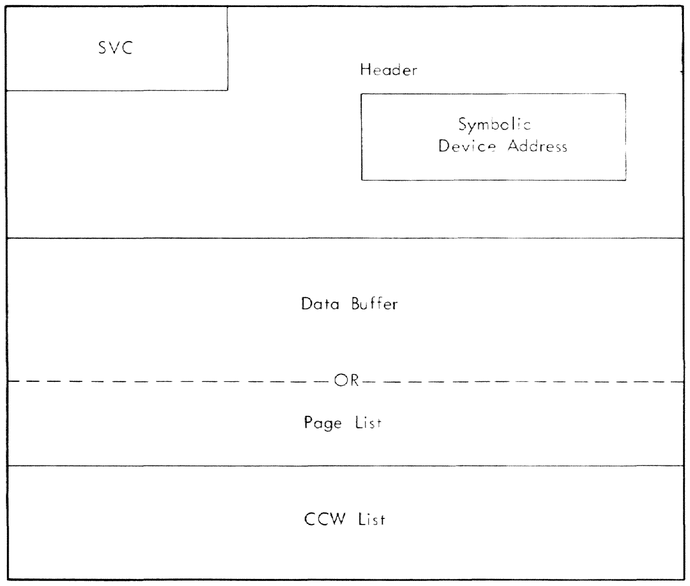


Figure 26. Input/Output Request Control Block (IORCB)

The test for completion is made by issuing the CHECK macro. If the I/O operation has ended satisfactorily, control is given to the next sequential instruction following the CHECK macro. If the request resulted in an error or a special condition, control is passed to the user's SYNAD routine if it

is specified, otherwise the Abnormal Task Termination (ABEND) routine is invoked. If the operation is not complete when the CHECK is issued, the task will wait until it is complete.

The Resident Supervisor performs the actual I/O, because the Model 67 channels and storage busses operate only on real 24-bit addresses. The access methods are located in virtual storage and utilize only virtual storage addresses. The Model 67 channels do not handle virtual storage addresses, because channels normally cannot stop and wait for Dynamic Address Translation when an I/O operation references a new buffer page.

Thus, the Resident Supervisor must either be invoked to translate the virtual storage addresses used in creating channel programs in virtual storage, or the Resident Supervisor must itself create channel programs, and, thus, lose a great deal of device independence.

The first alternative is preferable and is implemented by the creation of an I/O Request Control Block (IORCB).

The IORCB provides a way to reduce the amount of main storage and paging required to perform an I/O operation. The need of the IORCB is based on three observations:

- Because the channels do not perform relocation, all buffer areas that are to be referenced during the execution of a channel program must be in main storage during the entire I/O operation. However, the task that initiated the operation has, frequently, finished its time slice and thus the pages may no longer be available.
- There are a number of control blocks (originally defined in OS/360 BSAM) involved in a TSS/360 BSAM operation and they are usually located in different pages.
- Most BSAM buffers are expected to be much less than a page in length so that frequently, an entire page would be required to remain in main storage for a relatively long period of time when only a few hundred bytes of buffer space are being used.

By collecting all the pertinent information required to perform an I/O operation and the I/O buffer itself into one control block, the IORCB, and having the Resident Supervisor copy IORCBs (whose maximum size is 1920 bytes) into Supervisor storage which is allocated in 64 byte increments, a saving in paging overhead and main storage use is obtained.

The maximum size of an IORCB is 1920 bytes because it must fit into the area of the Interrupt Storage Area (ISA) that is used for passing information from the Resident Supervisor to virtual storage.

If a buffer is too large to be contained within the IORCB, BSAM places pointers in the IORCB to the page or pages containing the buffer.

Error recovery for any of the sequential access methods is performed in virtual storage for two major reasons:

- Error recovery procedures are device dependent.
- Recovery procedures differ depending on the over-all intent involved in a series of I/O operations. The Sequential Access Methods construct channel programs and thus this information is available only in virtual storage.

QUEUED SEQUENTIAL ACCESS METHOD

The Queued Sequential Access Method (QSAM) permits the programmer to store and retrieve the records of a sequential data set without coding blocking/deblocking and buffering routines. The programmer can, therefore, concentrate all his efforts on processing the data he reads and writes. Another major feature of this access method is that it provides two buffering techniques, allowing the programmer to choose the one most suited to his application.

QSAM, as designed for use in TSS/360, provides the following expanded facilities:

- Both locate and move mode macro instructions can be intermixed on the same data set, except when a printer is in use and CNTRL is specified in the MACRF parameter of the DCB macro instruction.
- Variable record formats are allowed on a data set opened for RDBACK.
- A SETL routine is provided to alter sequential processing of a QSAM data set.

QSAM can be divided into four basic functions:

- Blocking logical records.
- Deblocking logical records.
- Buffering blocks of data.
- Issuing I/O requests, checking, and repositioning for blocks of data.

Blocking, deblocking, and buffering are performed by QSAM internally. I/O operations such as reading, writing, checking, and positioning for access to data are performed by the Basic Sequential Access Method (BSAM).

Blocking Logical Records

QSAM will place logical records in a block where the maximum block size is specified in the DCB. The user issues a PUT macro instruction for each logical record he wishes to include in the output data set. It is the function of the PUT subsection to determine whether or not each record will fit within the current buffer, and, if it will, to add add the logical record to the block. If not, the block is considered complete, and the record for which the PUT was issued will be treated as the first record of a new block. The user may cause a block to be regarded as complete prematurely by issuing a TRUNC macro instruction.

Deblocking Logical Records

It is the function of the GET subsection to return to the user a single logical record each time he issues a GETmacro instruction. When a block of records has been read and checked, the address of a logical record within the buffer is returned to the user if the GET macro instruction was in locate mode. If it was in move mode, the logical record is moved to his work area. When the current block is completely processed, the next GET issued will cause the buffer to either be refilled if the data set was opened for INPUT or RDBACK, or to be written back, if required, to an update data set and then refilled. The user may cause processing on a buffer to be regarded as complete at any time by issuing a RELSE macro instruction.

Buffering Blocks of Data

The normal buffering facility of QSAM is known as double buffering, which involves the use of two buffers, one of which will be currently in use while I/O activity is being performed on the other. Thus, on a normal input or readback data set, while logical records from one buffer are being supplied to the user, the other buffer will be refilled. On a normal output data set, QSAM will continue adding logical records to one buffer while the other is being written out.

The decision to use double or single buffering is based upon the OPEN option of the data set, or upon the combination of device type and macro reference option specified in the DCB. Double buffering will be done in all cases except:

- When the data set is opened for UPDATE.
- When SETL is specified in the macro reference field of the DCB.

Single buffering must be done on an update data set to allow the user to update one block of records at a time. No reading ahead can be done until it is determined whether or not the current block of records must be updated, since an update write can only return the last block read.

When SETL is utilized, various control operations are possible. Moveover, SETL, type C, can only be valid for single buffering.

Issuing I/O Requests, Checking, and Positioning for Blocks of Data

The internal functions of QSAM are performed entirely within storage. Any I/O requests for transfer of data between storage and any I/O device, or requests for repositioning a data set, are passed on to BSAM. The BSAM modules invoked by QSAM are READ/WRITE, CHECK, POINT, CNTRL, NOTE, and ESP.

MULTIPLE SEQUENTIAL ACCESS METHOD

The Multiple Sequential Access Method (MSAM) is designed for use in TSS/360 to provide a fast and efficient mechanism for simultaneously driving several card readers, card punches, and printers under the control of a single task. MSAM is currently used by the TSS/360 Bulk I/O routines and may be used by any other user with the Command Language privilege class E. This privilege class allows the user to secure unit record equipment through the Command Language.

The user interface with MSAM is the GET, PUT, DCB, DCBD, OPEN, CLOSE, FINISH, and SETUR macro instructions. The SETUR macro is used for setting up specific forms on the unit record equipment. The use of the DCB, DCBD, OPEN, and CLOSE macro instructions under MSAM is generally consistent with the other access methods. An automatic error retry option is available to the user under the control of the DCB macro instruction. For instance, the DCB may specify that a print error be handled by striking out the erroneous line and repeating the line.

MSAM supports both fixed (F) and variable (V) format records and is consistent with other Sequential Access Methods in this respect.

Like BSAM, MSAM builds the channel program to control a data transfer in an IORCB and then passes the IORCB to the Resident Supervisor.

MSAM differs from the other sequential access methods (such as BSAM and TAM) in several significant ways. First, for each MSAM I/O request, the system processes a buffer group of physical records, while for each BSAM and TAM I/O request the system processes only one physical record. Considerable processing is required in the supervisor and the access methods for each I/O request regardless of buffer size. Usually MSAM will make an I/O request only once for processing each buffer, thus minimizing system processing overhead when using unit record equipment. For example, when generating output in the form of card images or listings, the user processes logical records which become separate blocks or physical records (i.e., cards or print lines) on the unit record equipment. MSAM routines buffer these logical records from their data set into system provided buffers, each of which resides in a separate page of virtual storage. Each MSAM buffer page contains a small fixed portion of control information. The remaining portion of the page is packed with format F or format V logical records.

Another way in which MSAM differs from the other sequential access methods is as follows: Several data sets may be grouped together on any one device, allowing the user to process all of them under the same Data Control Block without having to issue an OPEN and CLOSE for the DCB each time a data set with different characteristics is to be processed. Each of the separate data sets is referred to as a data group. Input data groups may be separated by control cards. MSAM will recognize these control cards and notify the user that a control card has been read, allowing him to take whatever action is necessary. Output data groups on the card punch may be separated by special cards from the card reader by specifying the COMBIN option in the DCB macro instruction, or they may be removed from the stacker by the operator who may be instructed to do so when a FINISH macro is issued. The FINISH macro instruction allows the task to avoid much of the overhead involved in closing a data set.

Also provided by MSAM is the capability of efficient accessing of multiple devices within one task. While this is possible with other sequential access methods, the MSAM macro instructions are designed in such a manner that the system service routines need not put the task in delay status while waiting for an event, such as I/O completion, to occur. This efficient device management is accomplished by defining the macro instructions to provide a return code to inform the invoking routine that a delay is necessary before the request (such as GET, PUT, or FINISH) can be completed for this DCB. This transference of responsibility of waiting from system service routines, such as the BSAM check routine, to the invoking routine provides the ability for the task to process all its opened DCBs until all DCBs accessed require waiting. At this time the task may wait for the first I/O interrupt for any DCB in the task.

All messages written to the operator from MSAM service routines are of the WTO macro form (see section on Communication). The WTO macro does not put a task in delay status. A WTOR macro is not used as it would put the task in delay status even though there may be opened DCBs which may be processed. For example, if a message must be provided to the operator of the on-line unit record devices to make a specific device ready, a WTO is issued and the Task Monitor is notified to continue with the task programs and provide an interrupt when that specific device goes from the not-ready to the ready condition.

MSAM processes from one to a maximum of 40 buffer pages based on an installation parameter specified during system generation. This parameter is set in the Symbolic Device Allocation Table (see section on Device Allocation) and may be different for each device. For example, the value for a device can be adjusted so that the device will be driven full speed for the maximum length of time between two consecutive time slices.

TAM

The purpose of the Terminal Access Method (TAM) is to provide an interface with IBM 1050, 1052-7, 2741 or TTY35 terminals attached to the IBM/360 Model 67 through either a 2702 Transmission Control Unit or directly attached to a multiplexor subchannel.

As with all access methods, the means by which a user invokes TAM is to issue macros of a prescribed form. At the present time, however, these macros are not directly accessible by the nonprivileged user who must use the GATE macros to communicate with the task's SYSIN/SYSOUT terminal. TAM can be used directly only by privileged routines.

The Terminal Access Method belongs to the Basic Sequential class. Buffering may either be handled by the user or dynamic buffering may be employed as an alternative. Macros are of the READ-WRITE type, and interrogation by means of a CHECK macro is necessary to determine the completion of the operation being performed.

Like BSAM, TAM builds the channel program to control the data transfer in an IORCB. Like BSAM, it also locates the data which will be transferred in a given operation. However, if the data record is too long to be contained in a buffer within the IORCB itself, the action taken by BSAM and TAM is different. In the case of BSAM or MSAM, a pointer is provided within the IORCB which locates separate pages to carry the data. In TAM, long records are handled by constructing additional IORCBs, and performing separate I/O operations on each -- all data is carried within IORCBs. This difference in methods of dealing with long data records is one primary difference between BSAM and TAM.

Because TAM supports devices and not data structures, it does not utilize many of the fields contained within the control blocks utilized by BSAM, MSAM and IOREQ. However, as a matter of design convenience, these control blocks have not been redefined or consolidated for TAM. TAM supports the Define Data, DCB, OPEN, CLOSE and CHECK general service macros.

TAM provides its own error recovery and posting routines.

IOREQ

The Input/Output Request Facility (IOREQ) is the access method provided by TSS/360 for handling unsupported devices or for handling supported devices in a non-standard way.

Since the user of IOREQ can have complete control over a device and, perhaps monopolized the channel to which the device is attached, the use of IOREQ is restricted to devices defined as private in the Symbolic Device Allocation Table.

In addition, only the BULKIO task and users with Privilege class "E" can request the allocation of a specific private device through a Symbolic Device Address (SDA), and only the "E" class user can request the allocation of unit record devices.

Unlike the preceding access methods, IOREQ has no knowledge of the data set organization or, perhaps, of the device being used.

To use IOREQ the user must:

- Be thoroughly familiar with how the device interfaces with a channel through its control unit.
- Handle all exceptional conditions through his SYNAD routine.
- Re-issue all outstanding requests if an I/O request is unsuccessful. (In BSAM, MSAM, or TAM, the access method would re-initiate all I/O requests which were issued after the one which was unsuccessful).
- Issue the SAEC and SIR macros to handle asynchronous interrupts from the device if such interrupts are possible and if the user wishes to handle such interrupts in a non-standard way.
- Not exceed the maximum number of concurrent I/O requests for this device as specified in the Symbolic Device Allocation Table (SDAT).

IOREQ utilitizes the DDEF, DCB, OPEN, CLOSE and CHECK macros.

Like BSAM, the appropriate channel program and buffer information is specified in an IORCB but, unlike BSAM, the user must specify the buffer and channel program to be used by building a set of Virtual Channel Command Words through the VCCW macro. The buffer may be contained in the IORCB or in user pages, as the user desires. The user requests that this channel program be executed by issuing the IOREQ macro. The ability to create his own channel programs and specify his own buffers allows for greater flexibility than is found in BSAM. For instance, channel programs can be longer and scatter-read or gather write may be used.

An additional feature of IOREQ is that channel programs may be command chained in the channel. This means, for example, that when the channel completes the channel program in one IORCB, the channel will (if command chaining was specified) immediately begin executing the channel program established in a second IORCB that has been made available.

OLTAM

The On-Line Test Access Method (OLTAM) is used in conjunction with the On-Line Test System (OLTS) and is available only to a user with Privilege class "E".

OLTAM is similar to IOREQ in that the user must create his own list of virtual CCWs which are then passed to the Resident Supervisor in an IORCB and in that he must specify his own Attention interrupt handling routines. It differs from IOREQ in that:

- The DDEF, DCB, OPEN, CLOSE and CHECK macros are not supported. The routine invoking OLTAM provides the appropriate parameters directly.
- OLTAM permits a test program to specify an I/O path of its own choosing rather than accepting an arbitrary path as provided by the other access methods. If the specified path is not available, the request will be queued by the appropriate Supervisor SVC processor until the path becomes free.
- The use of a channel program controlled interruption flag within a CCW is not restricted and the reservation of a malfunctioning device is not prohibited.
- Information describing the results of the I/O operation is placed in an OLTAM defined Test Event Control Block (TECB), instead of the Data Event Control Block (DECB) used by BSAM, QSAM, MSAM, TAM, and IOREQ.
- The WAITIO macro is used to test for completion of an I/O operation, instead of the CHECK macro used by BSAM, QSAM, MSAM, TAM, and IOREQ.

DRAM

The Drum Access Method (DRAM) is a special access method used by Virtual Memory Error Recording (VMER) to write error information into the short records that separate each page-sized record on a paging drum. This information can then be obtained by a system monitor (Privilege class E) through DRAM by use of the Virtual Memory Environment Recording Edit and Print (VMEREP) program. (See "Error Procedures.")

The Drum Access Method is similar to OLTAM in that the user creates his own list of virtual CCWs which are then passed to the Resident Supervisor in an IORCB. A special DRAM flag is set in the IORCB to notify the Supervisor that the I/O call is from DRAM.

DRAM is allowed to access a paging drum even if the device is malfunctioning.

DRAM is also similar to OLTAM in that it employs a special virtual storage routine to post the results of an I/O operation.

THE CATALOG

The catalog is a virtual partitioned data set containing the following information about data sets:

- Where the data is physically located.
- Who can access the data set.
- How the data set can be accessed.

The catalog is a hierarchical structure of indexes residing on direct access external storage devices. Each index has an alphameric symbolic name (up to eight bytes) associated with it. Each index is identified by this symbolic name plus the symbolic names of each higher-level index in its structure. A data set is uniquely identified by listing the symbolic names of the index levels (separated by periods) starting with the highest level index and proceeding to the lowest. Thus, as shown in Figure 27, data set 1 can be identified by its name A.B.C. and data set 2 can be identified by its name A.B.D. All data sets in the catalog structure need not have the same level of indexing. Data set 3, for example, has only two levels of indexes in its name (A.E), whereas sets 1 and 2 have three levels.

The highest level of index in the catalog structure consists of the 8-character user-identifications, one for each authorized user of the system, which are prefixed automatically by the system to the name that the user assigns to a data set. Thus, if a user identifies a data set as A.B.C, the system will retrieve it using the name userid.A.B.C. This highest-level index is referred to as the Master Index. Using the user identification as the highest index level ensures each index below the Master Index can be identified with a user and assures uniqueness of data sets given similar names by separate users.

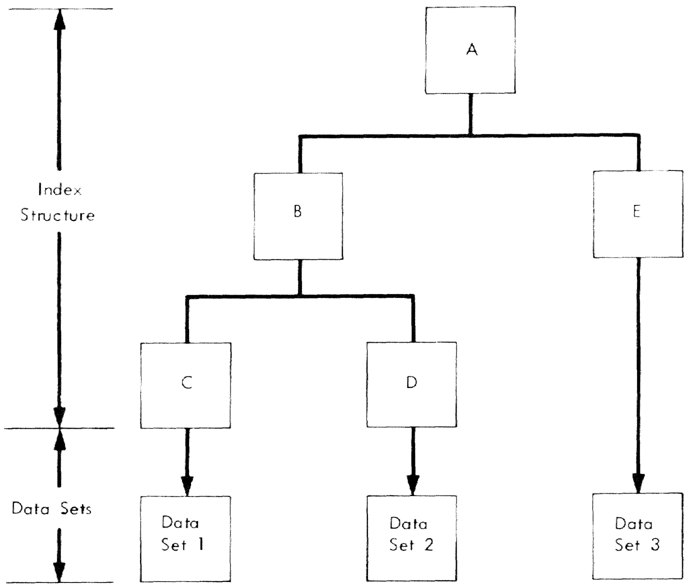


Figure 27. Catalog Index Structure Hierarchy

The collection of indexes below a user indentification is called a user catalog. Each index name is referred to as a simple name. Combining the names, as described above, produces a qualified name. If the name of each level, from the highest to the lowest, is specified, the name is called a fully qualified name and identifies a single data set. If one or more of the lower levels are not included in the name, the name is called a partially qualified name. A partilly qualified name identifies a collection of data sets. For example, with the index structure shown in Figure 24, the partially qualified name A.B identifies the data sets with the fully qualified names A.B.C and A.B.D. Including all simple names and separating periods (but excluding the user identification), the length of a data set name may not exceed 35 characters. The prefixing of the user identification gives the name, of a maximum length of 44 characters, for catalog references.

The catalog is a virtual partitioned data set. The Master Index is the Partitioned Organization Directory (POD), and each user catalog constitutes a member of the partitioned data set. The user catalog can reside on more than one volume. However, the volumes on which the catalog resides must remain on-line during system operation.

The system catalog (TSS*****.SYSCAT) is a dynamic virtual partitioned dataset -- it expands as new users log onto the system. It is also a "scratch catalog," in that only members (users) active during a session will exist in it. SYSSVCT, a virtual indexed dataset, is used by the system to keep track of the individual user catalogs (USERCATs). SYSSVCT uses the user identifications (userids) as keys, and contains pointers to the USERCATs. SYSCAT and SYSSVCT are located on the Auxiliary Control Volume. The USERCATs are virtual sequential datasets, residing on public volumes.

At startup, the SYSBUILD routine will place TSS*****, SYSMANGR, and SYSOPER0 in both SYSCAT and SYSSVCT. SYSSVCT will also contain the userids of all the users that have been joined to the system, and pointers to the DSCBs for their user catalogs.

When a user logs on, his user catalog may not be found in SYSCAT. Using SYSSVCT (where he is identified), the system will then copy his user catalog into SYSCAT. SYSSVCT is then flagged to indicate that the user catalog and SYSCAT are identical for this user. When the user subsequently references his catalog, the copy in SYSCAT is used (as when the user adds a new data set). During these operations, SYSSVCT is flagged to indicate that the copy of the user's catalog in SYSCAT is different from the copy on external storage. When the user logs off, this flag is reset, and the system copies the user's catalog member from the scratch catalog (SYSCAT) to the external residence of the user catalog.

At system shutdown, the userids in SYSCAT are searched and, if SYSSVCT indicates that the USERCAT and SYSCAT are not identical, the user catalog on external storage is updated from SYSCAT for that user.

When a task is created, a skeletal JFCB for the catalog is included in the task's Initial Virtual Memory. During task initialization, the Virtual Memory Task Initialization routine is used to open a

skeletal DCB for the catalog, which is a shared data set. This causes the RESTBL and Partitioned Organization Directory to be entered in the user's virtual storage.

When catalog data set pages are assigned to members, the pages are formatted into 64-byte blocks called S blocks. S blocks are the basic unit of storage within the catalog and are used to promote efficient storage and retrieval of information. Each S block contains information about a logical entity and contains pointers to related S Blocks. A logical entity within the catalog is composed of one or more chained S blocks. These logical entities are:

- Indexes
- Generation indexes
- Data set descriptors
- Sharing descriptors
- Sharer lists

Logical entities are chained together in groups called index levels. Within each index level are S blocks containing information about that index level and the name of each subordinate index level.

For a fully qualified data set name, the lowest level in the owner's catalog consists of one or more S blocks which constitute a "Data Set Descriptor."

The information contained in a Data Set Descriptor is included in the following fields:

- Forward and backward pointers to S blocks in the same Data Set Descriptor
- Name of the Data Set Descriptor
- Pointer to Sharer List
- Identification of volume or volumes on which the data set resides (SAM or private VAM data sets only)
- Public or private data set indicator
- Label data indicating the type of labels, if any
- Share flags indicating the extent to which the data set is shared
- Share privileges (if data set Descriptor is universally shareable) indicating the type of access to the data set allowed
- Owner access privileges
- Data set organization

Each new member of a generation data group is described by a generation index. Sharing descriptors and sharer lists are discussed in "Sharing."

Figure 28 provides a schematic of a member of the catalog partitioned data set. Any of the various catalog management operations that can be performed on a member are accomplished through the Catalog Services routines.

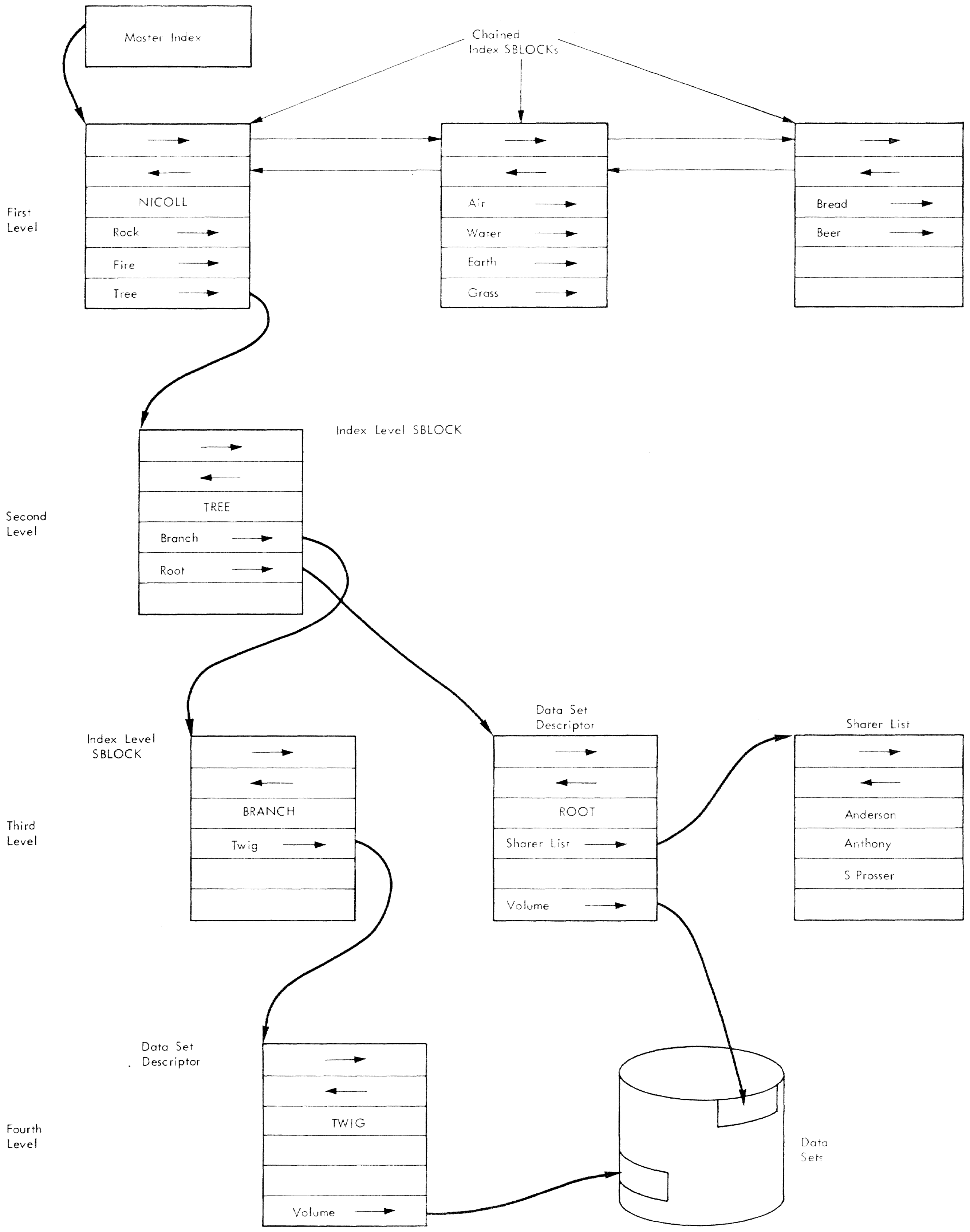


Figure 28. Catalog Member

There are two general categories of libraries in TSS/360:

- Object Program Libraries
- Symbolic (Source Statement) Libraries

OBJECT LIBRARIES

There are three types of object program libraries available to a user:

1. The System Library (SYSLIB) is the source of all standard system routines which are not contained in initial virtual storage. The System Library is opened for each user during task initialization processing.

2. The User Library (SYSULIB) is a private library created for each user when he first logs on to the system. This library is associated with the user's ID, and is opened for him during LOGON processing. A DDEF macro is issued by the Virtual Memory Task Initialization routine to make the User Library available. The SYSULIB for a user is released when a QUIT command is issued for him.

3. A Job Library (JOBLIB) is a library that the user defines by specifying the JOBLIB keyword operand in the DDEF command. The user is allowed to define any number of JOBLIBs during his task, and these are normally used for the purpose of stowing away and retrieving object program modules generated as output by the language processors. Job libraries are frequently used to contain programs that are undebugged versions of programs in an individual's user library or to contain programs that are to have sharing attributes different from an individual's user library.

To be made accessible to a task, object program modules must be contained in one of these object program libraries.

Each of these libraries must be in the form of a VPAM data set. Each program module, then, is a member of the partitioned data set, while each entry point and control section (i.e., CSECT) name is an alias for that member's name. Thus, a program module may be loaded by module (i.e., member) name, or by any alias. Object programs created during the process of assembling, compiling, or link-editing are automatically formed as members of partitioned data sets. The user's only responsibility is to issue a DDEF command or macro instruction for each Job Library that he wishes to establish for his task.

The DDEF command program creates a Job File Control Block (JFCB) and places it in the Task Definition Table (see "Data Management"). If the DDEF command contains the keyword "JOBLIB," control will pass from the DDEF command to the "LIBMAINT" module which creates a Data Control Block (DCB) from that JFCB, links it into a chain of DCBs which describes a library search heirarchy, and issues an OPEN macro to open the DCB for input.

This chain of library DCBs includes the System Library, the User Library, and any Job Libraries that a user has declared. Whenever conditions indicate that user libraries are to be searched, the search normally begins with the most recently defined JOBLIB, or with the user's SYSULIB, if no JOBLIBS have been declared. The last library to be inspected will be the System Library.

The chain of library DCBs is also used by the Language Processor Control (LPC). After a compilation or a link edit run, LPC opens the DCB at the head of the library search chain by issuing an OPEN macro with the keyword OUTPUT. The head of the chain is the most recently defined JOBLIB or is SYSULIB if no JOBLIB has been specified. LPC then issues PUT macros to place the virtual storage image of the object module into this VPAM library. LPC next issues two Stow macros. The first is a Stow macro with the module name. This macro creates a member entry. The second Stow macro creates an alias descriptor for all external symbol definitions and CSECT names within the module. LPC then closes the output DCB.

For each partitioned data set, there exists a Partitioned Organization Directory (POD). The POD for a partitioned data set relates member names to the positions of the member within the data set and defines the attributes of each member. The POD also relates aliases to members. A search of the POD is effected by the use of FIND, and entries are added to, deleted from, or changed within the POD by use of STOW.

Whenever a partitioned data set is opened, the POD for the data set is placed in an area of virtual storage protected from the user. For a shared data set, the POD is also located in pages which are shared among the sharing users. The entire POD remains in the user's virtual storage from open time until close time. For a non-shared data set, the POD is updated on the resident device at close time, if either the data set or the POD has been altered. For a shared data set, the POD is updated on the resident device only when the last sharing user closes the data set.

SYMBOLIC LIBRARIES

A symbolic library is organized into two portions: a source portion and an index portion. The source portion is a virtual indexed data set and consists of a collection of named groups of data called "parcels." The source portion is organized as a line data set. The index portion is a virtual sequential data set and contains information that relates the name of each parcel to the location of that parcel within the source portion data set. The index portion consists of a single record in the Undefined (U) format.

Symbolic libraries are most frequently used as macro libraries. The macro definitions corresponding to the TSS/360 System Macro Instructions, together with any parcels to be accessed by means of the COPY assembler instruction, form the source portion of the TSS/360 system macro libraries, TSS*****.SYSMAC and TSS*****.ASMMAC.

If a user declares a user macro library in response to the prompting from a Language Run command, his macro library is searched first to find macro definitions or COPY code.

The Command System DATA and MODIFY commands can be used to create or modify the source portion of a Symbolic Library.

The index can be created either by using the Command System RUN command to execute SYSINDEX or by issuing a CALL macro for SYSXBLD in a program.

Through the facility of the VT (Copy VAM to Tape) command, a seldom-used macro library may be removed from public storage. If a system macro library is removed from public storage, it must be restored via the TV (Copy Tape to VAM) command routine before performing system maintenance (see "Command Routines").

As presented in this manual, the operation of TSS/360 involves a single Resident Supervisor and a set of tasks which operate concurrently and which share CPU, channel, and storage facilities.

There are protection considerations associated with each of these areas:

- CPU references to main storage
- References through channels to external or auxiliary storage
- Channel references to main storage

CPU REFERENCES

Protection requirements associated with CPU references can be categorized as follows:

- To protect the Resident Supervisor from actions originating within a task.
- To protect each task from all others.
- To provide protection among various portions of a single task.

The basic mechanism for protection of the Resident Supervisor is provided by the fact that a task generates virtual addresses and cannot, therefore, directly address main storage.

An additional level of protection is provided by allowing only privileged routines to issue supervisor calls which may affect system operation.

The SVC Queue processor checks whether the routine which issued the SVC has sufficient privilege, and the SVC processors of the Resident Supervisor check the validity of the supervisor calls they process. Some checking is also done on the correctness of the SVC parameter list or control block involved. However, even with this checking, a privileged routine may issue a supervisor call which adversely affects system operations. The Resident Supervisor does not completely check the correctness of SVCs in order to allow an installation some flexibility to modify virtual storage system services without making corresponding modifications to the Resident Supervisor.

The basic mechanism that protects each task from all others is effected through the virtual storage concept.

Each task has its own set of relocation tables (except in connection with shared virtual storage). These relocation tables cannot be addressed from virtual storage. The Resident Supervisor controls the allocation of main storage to the various tasks. Thus, unless the system makes an error, destructive inter-task interference will not occur in main storage.

To the extent that tasks share relocation tables and thus share virtual storage, it is possible for one task to affect another task.

The considerations involved in sharing are discussed in the section on "Sharing." Two aspects deserve further mention in connection with protection:

1. Shared main storage is normally assigned a read-only protection key which eliminates the possibility of nonprivileged routines in one task affecting other tasks. No such protection can be afforded against privileged routines because they use a protection key of zero.
2. A task can only symbolically reference those shared routines whose Program Module Dictionaries have been loaded into its Task Dictionary. (See section on Dynamic Loader).

The concepts that create the privileged state are the mechanisms provided for intra-task protection. These concepts are described elsewhere in this manual and are a consideration whenever a program module is inserted into a TSS/360 library, whenever a program module is to be loaded from a library, whenever linkages are generated, and whenever main storage references are generated.

EXTERNAL AND AUXILIARY STORAGE REFERENCES

The basic protection mechanisms for external or auxiliary storage are also described elsewhere in this manual, and are mainly provided by the Catalog Services, Device Management, External Storage Allocation, and access methods routines.

CHANNEL REFERENCES TO MAIN STORAGE

The I/O channels operate with the storage protection feature in the same way as CPUs. The protection key for each channel is obtained from the first four bits of the Channel Address Word (CAW) when a Start I/O (SIO) is issued. This key value is held by the channel and is applied to all storage references. By proper assignment of I/O protection keys in CAWs and storage keys in storage, protection against erroneous modification of storage areas due to a channel malfunction or a programmed error can be provided.

Classes of I/O Operation

- I/O to Virtual Storage areas
- Paging to Virtual Storage areas
- I/O to IORCB Buffers

Each of these I/O classes involves a different requirement for protection and will involve different CAW protection keys.

Classes of Storage

Main Storage can be grouped into four protection classes:

- Task Virtual Storage
- Pages in transit (output)
- IORCB Buffer Areas
- Resident Supervisor

Each of these memory classes is distinct and will involve one or more different storage keys.

Assignment of Keys

The assignment of storage keys to various storage blocks is as follows:

Task Virtual Storage	
User State Read/Write	- Key = 1
User State Read Only	- Key = 2
Privileged State	- Key = 2 (Fetch Protected)
Pages being written out	- Key = 3
IORCB Buffers	- Key = 4
Resident Supervisor	- Key = 5

The assignment of PSW protection keys is as follows:

Programs operating in the User State	- Key = 1
Programs operating in the Privileged state	- Key = 0
Programs operating in the Supervisor state	- Key = 0

The assignment of CAW protection keys for various I/O operations is as follows:

I/O to Task Areas	- Key = 1 or 2 (as appropriate)
Paging	- Key = 3
I/O to Buffers	- Key = 4

No I/O operations after system start up are allowed to reference the Resident Supervisor (Key = 5).

Operation

The operations of the storage safeguards during I/O operations are as follows:

I/O TO TASK AREAS: I/O operations to task areas are performed as a result of an IOCAL SVC issued by an access method privileged routine. When an I/O buffer is not contained within an IORCB, it is normally the responsibility of the access method to assign the task area involved. The access method sets the protection key field of the IORCB to 1 or 2, depending on whether the task area being used as a buffer is a user or privileged area respectively. The IORCB key field is used by the Resident Supervisor in specifying the CAW protection key for the I/O operation. The task area being used as a buffer is not assigned a protection key other than user or privileged because it is desirable to allow the task to reference other data or instructions that may be located in the page containing the buffer.

Note: Buffer contents containing constants (such as SEEK Addresses) can be read for I/O purposes although the keys do not match.

PAGING: Paging operations will always be performed with a protection key of 3. It is the responsibility of the supervisory routine which requested the paging operation to set the storage key to 3 for all storage blocks involved in the operation. At the completion of an input paging operation, the page posting routine will be required to set the storage keys to their operational value (1, 2, or 2 with fetch protect).

Note: This special treatment of paging is necessary due to the drum queuing strategy. Only one SIO is issued (setting the protection key) for a series of drum paging operations which could involve both storage keys 1 and 2. The only protection key that could write in both areas is zero which is ruled out. Therefore, the only feasible solution is to use another key for all paging operations.

INPUT TO IORCB BUFFERS: This operation is much like the operation of I/O to task areas with one important exception - it is operating with supervisor allocated main storage. The areas which hold IORCBs are assigned storage key 4. It is the responsibility of the access method preparing an IORCB for buffered operation to set the protection key field of the IORCB to the value 4.

The result of this assignment of storage keys is as follows:

- No I/O operation can modify the Resident Supervisor storage.
- No I/O operation can inadvertently modify a privileged task area utilizing an erroneous user supplied address.
- The probability of detecting I/O addressing errors is increased.

PROGRAM MODULE STRUCTURE

The primary output of all the language processors operating in the TSS/360 system environment is the program module. The input consists of a stream of source language statements which may be converted to output program modules. The input to the Linkage Editor consists of directive statements and a set of program modules to be combined into a single program module.

PROGRAM MODULES

A program module generated by a language processor or the Linkage Editor resides as a member of a partitioned data set before being loaded, and in this state consists of three parts: a Program Module Dictionary (PMD), hexadecimal text, and an optional Internal Symbol Dictionary (ISD). Figure 29 illustrates this structure of object program modules.

The ISD contains information which describes the values and other attributes of the internal symbols (or FORTRAN statement numbers) used in source language statements or generated by the FORTRAN compiler.

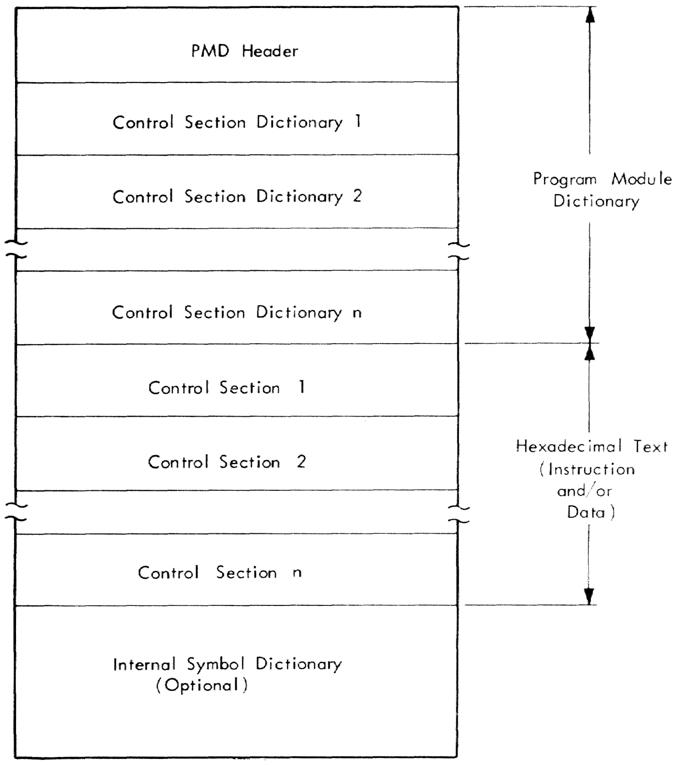

PMD Header
Control Section Dictionary 1
Control Section Dictionary 2
Control Section Dictionary n
Control Section 1
Control Section 2
Control Section n
Internal Symbol Dictionary (Optional)
Program Module Dictionary
Hexadecimal Text (Instruction and/or Data)

Figure 29. Format of an Object Program Module

If a program module contains an ISD, a user can refer to internal symbols when using the Program Checkout Subsystem (PCS). Otherwise, he can only reference external symbols in PCS statements.

An external symbol is a symbol defined in one program module which can be referenced by a separately assembled program module. An external symbol can (usually) also be referenced as an ordinary symbol within the module in which it was defined. Each external symbol has associated with it a V-value and an R-value.

The Program Module Dictionary consists mainly of a group of Control Section Dictionaries. There is a Control Section Dictionary for each control section of the program module. A Control Section Dictionary contains information describing a control section such as its length, attributes, external symbol references and definitions and information to be used in relocating address constants contained within the control section. Collecting all linkage data for a module into one Program Module Dictionary allows the TSS/360 Dynamic Loader to dynamically calculate linkage addresses without having to bring the text portion of the module into main storage.

The text is divided into sections called control sections. The control sections, PMD and ISD are all allocated external storage beginning on a page boundary and for an integral number of pages. However, they are not necessarily allocated full pages in main storage.

The way a program module is divided into control sections is determined by the source language statements in the case of output generated by the Assembler and by source language statements and the language processor in the case of output generated by the FORTRAN compiler. The TSS/360 language processors assign each control section a unique location counter during compilation.

For the system, the purpose of having control sections is to allow a program to be divided into sections whose virtual storage locations can be adjusted (independently of other sections) by the Linkage Editor or Dynamic Loader without altering or impairing the operating logic of the program. In System/360, this relocation involves only the adjustment of address constant values and does not require the direct modification of machine instructions.

For the user, a control section represents a segment of coding or data that can be replaced or modified (within certain limitations) without having to re-assemble an entire program. A control section also represents a segment of coding or data that can be independently assigned attributes.

At the time the user creates a control section, he may assign to it a variety of attributes. These attributes are: fixed length, variable length, read-only, privileged, system, public, common, and prototype. Only the prototype attribute (PSECT) will be discussed in this section. Further information about control sections may be found in "Dynamic Loader" and in <u>Assembler Language</u>.

<u>Prototype Control Sections and Reenterable Code</u>

PSECTs are generally used in programs which are designed for simultaneous sharing by more than one task. Such programs are termed reentrant and are characterized by the fact that the shared portions of the program do not change in any way during execution.

It is not necessary to use a Prototype control section when composing a program to run under TSS/360. In special cases, it is possible to write reenterable programs without using PSECTs. Such programs do exist in TSS/360. They may hold all their working data in registers or in the directly addressable ISA or they may acquire virtual storage working space dynamically. In general, however, any programs which are designed for simultaneous sharing by more than one task will contain PSECTs. All of the TSS Compilers produce such output modules, and all of the privileged TSS programs, and the nonprivileged system programs use reenterable coding with PSECTs.

When a reenterable program is composed, all modifiable data, work areas, and address constants may be placed within a PSECT.

Allowing the composer of a reentrant program to create the PSECT relieves the caller of that program of the requirement to know precisely what address constants the called program requires or, alternatively, of having to ensure that the shared programs occupy identical virtual storage locations within each sharing task.

Then, whenever a TSS/360 user loads a reenterable routine, a copy of the reenterable routine's Prototype control section is mapped into the user's private virtual storage. On the other hand, all concurrent users share a single copy of the program's reenterable control sections. The use of PSECTs has the following effect on the structure of programs within TSS/360. Program sharing is implemented in such a way that the PSECT and the reenterable portions of the called routine are separately mapped into a task's virtual storage. This means that in order to perform linkage to a reentrant routine, two virtual storage addresses must be supplied.

The first virtual storage address specifies the location at which execution of the object program module is to begin when control is transferred. This is the commonly understood, conventional external symbol value and is called the V-value. A V type address constant is often used for this purpose.

The second virtual storage address can be used to specify where the prototype control section has been mapped within the task's virtual storage. If this pointer were not supplied, the reentrant module would have no way of knowing where its modifiable data, etc., are located. This second value is called the R-value. An R type address constant is often used for this purpose.

The reentrant module itself cannot set up in a reenterable control section an address constant pointing to its PSECT. The reason that this cannot be done is that the PSECT may be mapped into different virtual storage locations for each concurrent user. An address constant contains data that is to be used to form a virtual storage address. Because all concurrent users share the same physical copy of the re enterable control section, there would be only one copy of the address constant which, of course, can not simultaneously point to different locations.

Because virtual storage is allocated dynamically, a reentrant control section that is not contained in Initial Virtual Memory may also be mapped into different virtual storage locations for each concurrent user, even though there is only one physical copy of the control section.

This explains why all address constants must be treated as modifiable data when creating a reentrant program and, as such, must be placed in the program's PSECT.

Because Initial Virtual Memory modules occupy identical virtual addresses in every task, they may treat their internal address constants as unmodifiable or "read-only" data.

Putting all address constants, modifiable instruction sequences, etc., into one or more PSECTs still does not guarantee that the resulting routine will be reenterable under all conditions. This certainly takes care of intertask reenterability, that is, the sharing of a program by many different tasks. However, there is another type of reenterability to be considered: intra-task reenterability. This refers to a single task reentering the same program. As discussed in "Task Monitor," this can occur when a task receives a programmed interruption while executing some system routine or when a routine is recursively called.

In such a situation, the PSECT will not protect task integrity, since within a single task there is only one copy of the PSECT. This explains why the Resident Supervisor does not use PSECTs and why the Task Monitor provides either a push down save area or a means by which a routine can protect itself from unwanted intra-task reentrancy.

A program module can contain more than one PSECT as long as each is identified by a unique name. Multiple PSECTs do not appear to be absolutely necessary, but they are convenient.

The contents of a PSECT need be in no particular order and can be identical in every respect with those of a CSECT. However, unlike most other types of control sections processed by the TSS/360 Assembler, the PSECT must always be named.

A PSECT is generally used to hold the save area for a reentrant routine. The save area, and the origin of the PSECT itself, are located by separate and independent pointers. However, there are advantages in placing the save area at the origin of the PSECT, and, in fact, most system routines are written this way. Placing a save area within a PSECT, rather than a push-down stack, facilitates tracing linkages during debugging.

The Appendix section of <u>Task Monitor Program Logic Manual</u> contains a table of all system enter codes.

<u>CSTORE Macro Instruction</u>

The CSTORE macro instruction enables a user to store an area of virtual storage as a control section, thus bypassing the need to store and process this area as a data set.

The use of a CSTORE macro instruction causes a control section to be created during the execution of a program. Any set of contiguous bytes may be transformed into a control section. This can be stowed as a BLOCK COMMON module, placed on the current JOBLIB as a module, and may then be subsequently loaded as part of the originating program or as part of a later program. When such a control section is loaded, no relocation takes place. Therefore, it may not contain any relocatable items.

The resulting control section will contain an integral number of pages starting at the page boundary preceding the first byte address and terminating at the page boundary following the last byte address.

Situations can occur in a time sharing environment which necessitate communication between tasks. Such communication may be required by:

- The System Operator task to send messages to user tasks.
- Device management routines to send volume or form mounting messages to the system operator and to receive replies.
- The Batch Monitor task to initiate or cancel non conversational tasks.
- User tasks to enter messages in the system log or send messages to the system operator terminal.
- User tasks or the Main Operator Task to send requests to the Batch Monitor task for the initiation of nonconversational tasks.

A nonprivileged user has at his disposal the Write to log (WTL), Write to operator (WTO), and Write to operator with reply (WTOR) macro instructions. These macros expand into a Type II linkage to the Command Language XWTO routine.

A privileged user has at his disposal the Virtual-Memory-to-Virtual-Memory (VSEND) and VSEND-with-reply (VSENDR) macro instructions. The VSENDR macro instruction causes a Type I linkage to the Command Language XWTO routine.

The system operator has at his disposal the BCST, REPLY, and MESSAGE commands. These commands cause the Command System routines of the same name to be invoked.

Transmitting a message from one task to another involves: identifying the task to receive the message; establishing authorization between the sender and receiver; and transmitting and processing the message.

In order to communicate with another task, two important requirements must be met. First, the task to which a message is to be sent must exist at the time the message is to be sent; i.e., the task must be represented by a TSI. Second, the destination task must be identified by the Task Identification Number (TID).

The Resident Supervisor ensures that a task will not receive a message (whether meaningful or not) from another task unless the receiving task is prepared to handle the message. A flag in the TSI specifies whether incoming messages may be accepted. If the flag is on, all incoming messages except those from the System Operator or Batch Monitor tasks will be refused and the sender will be notified. If the flag is off, all incoming messages will be allowed to interrupt the task.

The process of sending a message is as follows:

An intertask message and an identifying message code are placed in a Message Control Block (MCB) and transmission is initiated by issuing a VSEND SVC.

The VSEND SVC processor of the Resident Supervisor searches the Active and Inactive TSI lists and, if the TSI with the correct TID is found, the Message Control Block is copied into Supervisor storage in the same manner as an Input/Output Request Control Block (IORCB).

A GQE is then created containing a pointer to the location of the MCB. This GQE is then enqueued on the receiving task's TSI as an External interruption request.

The VSEND SVC processor passes a return code to the sending task through a general purpose register to indicate the result of the VSEND request. The code may indicate that the destination task does not exist, that the destination task does not accept messages, or that the destination task has accepted the message.

The Task Interrupt Control subroutine eventually causes an external interruption in the receiving task and places the MCB in that task's ISA.

The External interruption will be initially processed in the receiving task by the External Interrupt Processor (XIP) of the Command System.

If the MCB message code indicates the message is to be handled by system routines, XIP obtains virtual storage, transfers the MCB out from the ISA, and establishes a linkage to the proper routine to handle the message.

If the user is expecting to receive such messages and they do not contain message codes processed by XIP, the user must provide for the processing by issuing the SEEC and SIR macro instructions.

It is possible to send a message and await a reply.

In this case, the original message will contain a flag indicating that a reply is expected and a pointer to a Message Event Control Block (MEB). The MEB is similar to the Data Event Control Block (DECB) utilized by the Sequential I/O access methods.

After the message has been sent, the XWTO routine issues an AWAIT supervisor call to place the sending task in delay status.

When a reply is received, XIP will post the receipt of the awaited reply in the MEB.

When XWTO next receives control, it inspects the MEB and finds that a reply has been received. XWTO then places the reply in a user specified area and returns to the caller.

An example of the processing of an inter-task message and reply is included in the section "Examples of System Operation, Nonconversational Processing."

Programs in TSS/360 can link to each other only in certain ways, and under certain conditions, depending on their privilege level. These design rules are called linkage conventions. They affect the setting up of save areas, the passing of parameter lists, and the setting up of entry and return registers.

Linkage conventions allow the transfer of control from one program to another in a standard way. This standardization eliminates redundant register usage, and allows linkages to be generated by means of system macro instructions.

Linking between modules in TSS/360 can be divided into two major classes:

- Linkage between modules of the Resident Supervisor
- Linkage between Virtual Storage programs

SUPERVISOR LINKAGE

At Startup, the Resident Supervisor is link-loaded and thereafter remains self-contained. Therefore, to increase efficiency, no conventions have been established to govern linkages between the modules of the Resident Supervisor. However, Resident Supervisor routines make extensive use of the standard linkage registers 0, 1, 14, 15 for linkage purposes.

VIRTUAL STORAGE LINKAGE

Within Virtual Storage, there are four main classes of linkage called Type I, Type II, Type III, and Type IV.

TYPE I

The Type I linkage is the most common and is, essentially, the traditional form of linkage. Its characteristics are:

- It uses the BASR instruction.
- The Resident Supervisor does not assist in the transfer (i.e., no interruption occurs).
- The two routines involved (i.e., caller and called) must be of the same level - i.e., two problem programs or two privileged programs.

TYPE II

Type II linkage enables a problem program to communicate with certain privileged System Service routines. Its characteristics are:

- The user generally issues a macro such as OPEN, READ, etc., which is expanded into an ENTER SVC.
- The SVC interruption is passed to the Task Monitor by the Resident Supervisor.
- The Task Monitor translates the Type II linkage into what appears to the called Privileged program as a Type I linkage.
- It is employed when a nonprivileged program uses a CALL macro to link to a Privileged program.

It should be noted that not all privileged routines can be called by a nonprivileged user. Certain privileged routines (e.g., Allocate, Extend, etc.) can only be called by other privileged routines. The called routine need not be aware of whether a Type I or Type II linkage is employed.

The Task Monitor saves the called routine from having to handle two different calling sequences by providing a save area and making the Type II linkage look like a Type I linkage. The language processors protect the caller from having to consciously set up one of two different calling sequences, by expanding system macros differently depending on whether the module being compiled is specified as Privileged or not. There is, incidentally, no need to restrict the use of the Privileged specification at compile time, because protection control is applied when the module is to be loaded.

TYPE III

Type III linkage is the reverse of Type II linkage. It is infrequently used. However, it is used whenever a privileged program invoked through a Type II linkage calls a nonprivileged program. The characteristics of a Type III linkage are:

- The transfer and save area management are assisted by the Leave Privilege subroutine of the Task Monitor.
- The actual linkage is performed via the Load Virtual PSW (LVPSW) macro which

generates a supervisor call of the same name.

- The Resident Supervisor subsequently issues the privileged instruction Load PSW and completes the transfer to the nonprivileged program.
- When the nonprivileged program effects a return, the address contained in the return register causes a Restore Privilege (RSPRV) supervisor call in the task's ISA to be executed.
- The Resident Supervisor passes the supervisor call to the Task Monitor as a task interruption.
- The Task Monitor SVC interruption processor restores the privileged routine's status from a protected save area and returns control to the privileged routine through the Load Virtual PSW supervisor call.

TYPE IV

Type IV linkage is used by TSS/360 programs under restricted circumstances for the sake of linkage efficiency. Type IV linkage is much more restricted linkage than types I, II, and III. Type IV linkage is found principally in the coding of the language processors. Type IV linkage conventions standardize the use of the general registers and the method of transferring control from the calling program to the called program. No provision is made for a standard save area in this convention. The characteristics of Type IV linkage are as follows:

- Control is transferred via a BASR instruction.
- General registers 0 through 6 are used as parameter registers.
- The calling program provides a PSECT address in general register 13.

FENCE SITTER ROUTINES

There is a small class of routines such as GETBUF that are called "fence-sitters" because they can be called through a Type I linkage by either a privileged or a nonprivileged routine. These routines are assigned a hardware storage protection key that makes them read-only to nonprivileged routines. Whenever a Type I linkage is performed, the PSW protection key is unchanged. Therefore, when called from a nonprivileged program, a fence-sitter routine takes on the characteristics of a nonprivileged routine. Whenever a fence-sitter service routine is called from a privileged routine, the PSW protection key is zero, and the fence-sitter takes on the characteristics of a privileged routine.

This convention is established for the purpose of permitting an efficient transfer of control to those system service routines which do not frequently need to link to other (privileged) service routines.

Some fence-sitter routines have initial entry point names beginning with the letters SYS. This distinguishes them from service routines that must be linked to from a nonprivileged routine through the ENTER mechanism.

Other fence-sitter routines are linked to from macro instruction expansions which utilize address constant values which were filled into a Data Control Block by a privileged access method routine. (See section on Data Management.)

If a fence-sitter routine needs to link to a privileged service routine, the fence-sitter routine utilizes either a Type I or a Type II linkage, depending upon the privilege class of the routine that invoked the fence-sitter routine.

For example, TSS/360 QSAM is designed as a fence-sitter routine, and thus will run in the same privilege status as the routine which invokes it. Since it is most often invoked by the problem program, it will generally run in the privilege of the user, and, as such, may or may not be of the same privilege as the BSAM modules which it invokes. All the BSAM modules invoked, except NOTE, are privileged routines. As NOTE is also constructed as a fence-sitter routine, and will take on the privilege status of QSAM whenever it is invoked, type-I linkage is always established to invoke NOTE.

Before establishing linkage to any of the other BSAM modules, it is necessary to determine the status of QSAM subsections. QSAM routines perform this function with respect to their BSAM counterparts by testing the first bit of the VPSW in the ISA table. If QSAM is privileged, type-I linkage is established, using the address constants defined within the data control block. If it is not privileged, type-II linkage is established via the ENTER SVC.

There are additional slight modifications of linkage types which are used in certain instances. For detailed information and examples of the rules governing each type of linkage, refer to the <u>IBM System/360 Time Sharing System: System Programmer's Guide</u>.

The Data Management facilities of TSS/360 can be invoked from both IBM-supplied and user-written programs and are used extensively by privileged system service routines, Command System routines, and the TSS/360 language processors.

Data Management facilities are called upon by macro instructions, such as GET and PUT, which are included in source programs. The TSS/360 language processors expand these macro instructions. Their expansions introduce instructions into an object module to provide for completing linkage and passing parameters to the appropriate Data Management service routines.

Some macros (such as READ/WRITE) include a Data Event Control Block (DECB) in their expansions. Each DECB contains information relative to the specific I/O operation to be performed. The access method obtains parameters necessary for the execution of the I/O operation from the DECB and, upon completion of the operation, posts in the DECB information such as the Channel Status Word (CSW) describing the results and completion status of the operation. The DECB format is shown schematically in Figure 30.

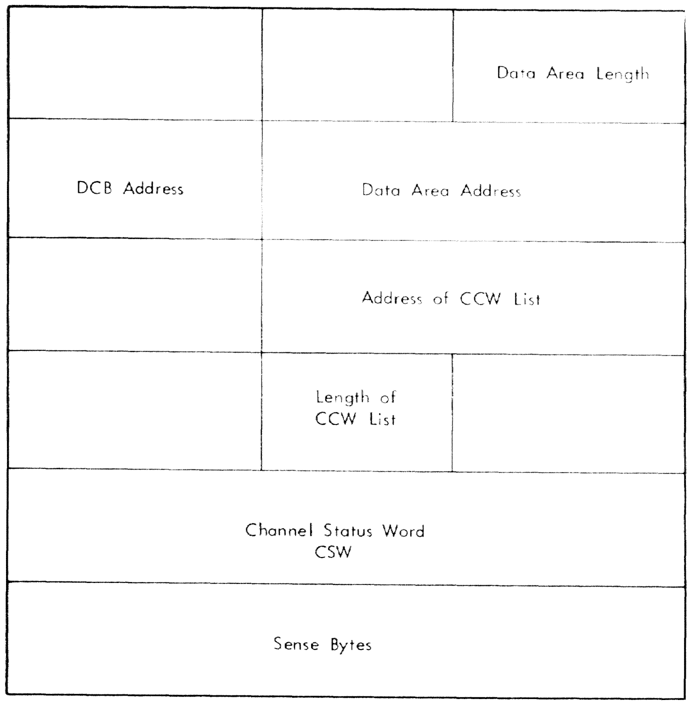


Figure 30. Data Event Control Block (DECB)

The expansion of some Data Management macro instructions results only in the creation of a control block. For instance, the Terminal Access Method DFTRMENT macro instruction expands into a list of terminal dialing, polling and addressing characters.

In order to use the facilities of Data Management, the data sets or devices involved must be described to TSS/360.

Parts of this description are generally supplied at three different times:

- When a program is composed.
- When a program is to be executed.
- When the data set or device is to be processed.

Before a data set can be processed, a Data Control Block (DCB) and a Job File Control Block (JFCB) must be created.

The DCB, when it is fully processed, becomes the principal control block used to supply information describing the data set or device and it is the control block which is referenced in all data management macro instructions.

A user or language processor creates a DCB by including the DCB macro instruction in source coding.

The Data Control Block (DCB) is created in-line wherever the macro instruction is placed. The control section containing the DCB is assigned main storage in accordance with the control section's attributes. This means, for instance, that care should be taken not to place this modifyable control block in a section with read-only or public attributes.

The DCB has a fixed length and consists of two contiguous parts: a common portion and an access method dependent portion. The DCB format is shown schematically in Figure 31.

Data set processing flexibility may be enhanced by not specifying certain DCB parameters during program composition and completing the DCB during or just before program execution. Even the data set organization (DSORG) parameter need not be specified during compilation.

The common portion of a Data Control Block contains such DCB parameters as buffer length and record format as well as address constants pointing to user speci-

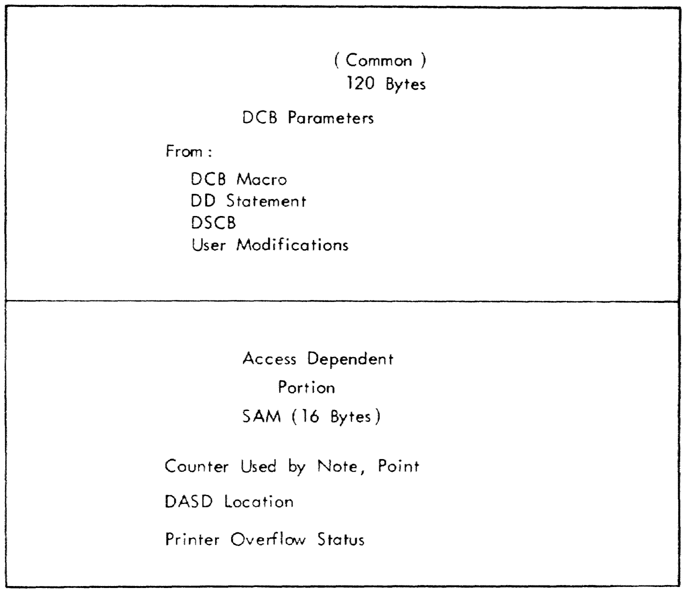


Figure 31. Data Control Block Table (DCB)

fied exit routines, such as the SYNAD address, and access method routines, such as the address of the PUT module.

Certain information describing a data set, such as the data set name (DSNAME), cannot be specified in a DCB macro but must be supplied before the data set can be processed. This information is supplied by the Define Data routine, which creates a Job File Control Block (JFCB). The JFCB format is shown schematically in Figure 32.

Privileged routines will sometimes use the Find JFCB (FINDJFCB) subroutine to locate a specific JFCB and to prompt a conversational user to issue a DD command when he has neglected to do so. Privileged routines also may use the Find Data set (FINDDS) subroutine to request the Define Data routine to directly create a JFCB for a cataloged data set if a JFCB does not already exist.

The nonprivileged user or system program may create a JFCB by issuing the DDEF command or macro instruction or by using the CDD command to cause prestored DDEF commands to be issued.

The DDEF command or macro must supply a Define Data Name (DDNAME) and a Data Set Name (DSNAME) in order that the data set may later be associated with a Data Control Block (DCB). Most other DDEF parameters may be omitted under certain conditions.

There are three classes of DD Names:

- System DD Names that begin with the characters SYS.
- DD Names generated by system routines which are formed from the characters $$$ concatenated to sequentially higher 5-digit numbers.
- Reserved DD Names, such as PCSOUT, which is used by the DUMP routine of the Program Checkout Subsystem and names beginning with the characters LPC, which are reserved for DD Names issued by the Language Processor Control module.

Within any particular task, only one JFCB can exist for a data set. If a data set name in a DDEF command matches a data set name in another JFCB for the task, the new ddname will be substituted for the old ddname and processing for that command will be considered completed.

This could occur, for instance, if during a terminal session a data set is processed as a new output data set in one program and as an old input data set in another program. In this case, the first ddname might be OUT and the second ddname IN.

For additional information concerning such data set characteristics as reserved data set names, see the Appendix section of the <u>Assembler Programmer's Guide</u>.

There are also four categories of additional information that may be supplied in a JFCB created by the DDEF routine:

- External storage space allocation parameters.
- Device Management parameters.
- Data set disposition parameters.
- DCB parameters.

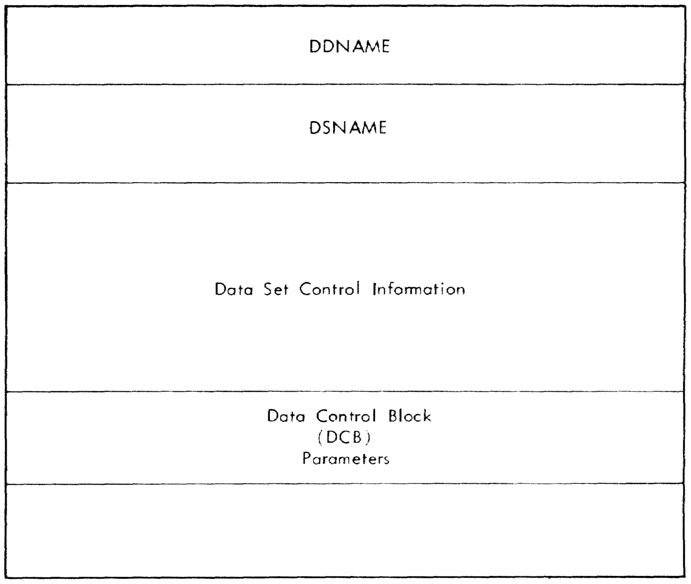


Figure 32. Job File Control Block (JFCB)

If the data set has been cataloged, the appropriate catalog services routines are invoked to obtain the data set descriptor.

The Catalog and DDEF values for data set organization, data set disposition, device class and data set affinity must agree or an error is indicated. All other catalog information (such as label type) is used in filling out the JFCB in preference to corresponding information in the DDEF command.

If a user specifies the *ddname parameter in the DDEF command (or macro) all the DCB parameters specified in the particular previously-issued DD command named by the *ddname parameter are placed into this JFCB. Any new DCB parameters submitted in the current DDEF command overlay parameters obtained from the previous DDEF command.

If conditions are such that a DDEF parameter may be omitted, the DDEF routine places standard values in the JFCB to fill out the defaulted fields. Most defaults result in placing zeros in the corresponding JFCB field. The appropriate values for the remaining fields are obtained from the System Common table. These values, such as the data set organization default value, are specified during System Generation.

If a new SAM data set is to reside on a direct access device, the ALLOCATE routine is invoked to obtain the required amount of direct access storage space. (See the section on External Storage Allocation.)

A call is made to Device Management to ensure that the proper Private devices are on-line for SAM data sets (see "Device Allocation").

When the JFCB has been completed, it is linked into a chain of JFCBs called the Task Data Definition Table (TDT). The TDT contains all the JFCBs defined for a task and resides in virtual storage obtained by the privileged DDEF routine. Thus, the TDT is protected from the nonprivileged user. During LOGOFF processing for SAM data sets this TDT chain is searched in order to prompt the user for the disposition of his new and as yet uncataloged data sets. It is also used for abnormal task termination (ABEND) processing to release interlocked facilities. No such prompting is required for VAM data sets since all such data sets are cataloged and the user specifies the deletion option by means of DDEF. As with most task chains or tables that are used by more than one routine, the TDT anchor is pointed to by a field in the Interrupt Storage Area (ISA).

If the user specifies in the DDEF command that he wishes the processing of a SAM data set concatenated with the processing of other data sets, the JFCB is chained to the other appropriate JFCBs. All members of a concatenated data set must have the same ddname. The CONC parameter of the OPTION field is used to specify the concatenation of this data set.

If a data set is declared to DDEF as a JOB LIBRARY, the data set is linked into a chain of job libraries within the TDT and the routine LIBMAINT is invoked to create a DCB from this JFCB. DCBs thus created are linked together in a chain and are used when library searching is indicated (see "Libraries").

The RELEASE macro instruction is used to release a data set, or a concatenated series of data sets, or a member of a concatenation, or a JOB LIBRARY from the program library list.

The next stage in preparing a data set for processing is initiated by the execution of an OPEN macro instruction and is called "opening a DCB."

Both a DCB and a JFCB must exist for the data set before the OPEN macro instruction is executed. Any particular DCB may be opened for only one data set or data set member at a time. In separate tasks DCBs may be opened for the same data set or data set member at the same time if the data set is sharable (see "Sharing"). If multiple DCBs within the same task are associated with the same data set, they may all use the same JFCB.

OPEN processing consists of processing that is common to all access methods (OPEN COMMON) and access-method dependent processing which is performed by one of a set of Access-Dependent Open (ADO) routines. This is schematically depicted in Figure 33.

If the data set resides on a direct access volume, the appropriate Data Set Control Blocks (DSCBs) are read into main storage from the volume on which the data set resides.

A DSCB is a control block that describes the attributes of a data set and resides on the direct access volume with the data set.

Empty fields of the DCB are filled with information obtained from the JFCB, and any remaining empty fields in the DCB are filled in with information from the DSCB. The user may create a routine to modify the DCB. This routine is given control if the routine's name was supplied as a DCB parameter.

This processing is depicted in Figure 34 and allows great flexibility in specifying

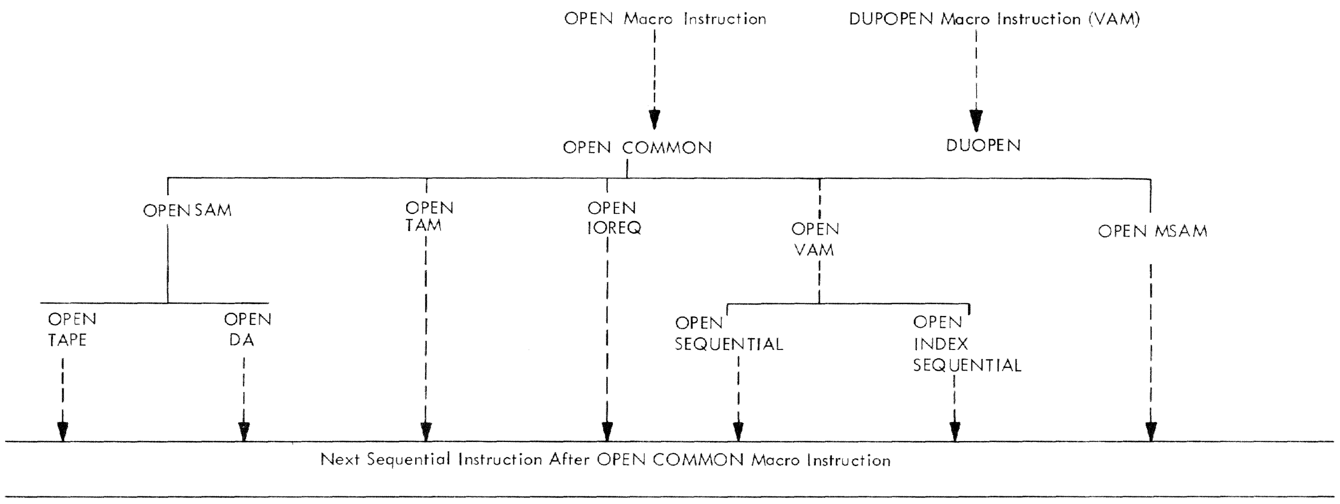


Figure 33. A General Flow of Open Processing

DCB parameters. Addresses of the various routines which will be used to process the open data set are filled into the DCB by the Access Dependent Open routines. Consequently, the proper routine will be given control when the macro instruction operator names are identical, e.g., the GET macro instruction will produce linkage to the VAM SEQUENTIAL routine if VSAM is being used, rather than the VAM INDEX SEQUENTIAL routine.

There is a field in the DCB containing the exceedingly rare combination of characters "*%*%". This field is known as the DCB identifier and is frequently inspected. If the field has been altered, this is taken as evidence that the integrity of the DCB is in doubt and a task is abnormally terminated (ABEND).

Another relevant check made during OPEN processing is to ensure that a nonprivileged user is not attempting to access a privileged system data set or device reserved for system use.

If appropriate volume and data set labels are processed, the volume is positioned. Privilege class is checked, information is obtained from the Symbolic Device Allocation Table, and various tables are built.

If the access method concerned is TAM, BSAM, QSAM, MSAM, or IOREQ, a Data Extent Block (DEB) is built.

If the access method is VSAM, VISAM or VPAM, a Relative External Storage Correspondence Table (RESTBL) is built.

Because the location of the DCB is specified by the user, it may be modified at any time. This gives added flexibility in such areas as specifying buffer lengths. However, in a multiprogramming system, no user should be able to interfere with others. For this reason, the BSAM, QSAM, MSAM, TAM and IOREQ access methods place certain information in what is essentially a protected extension of the DCB called the

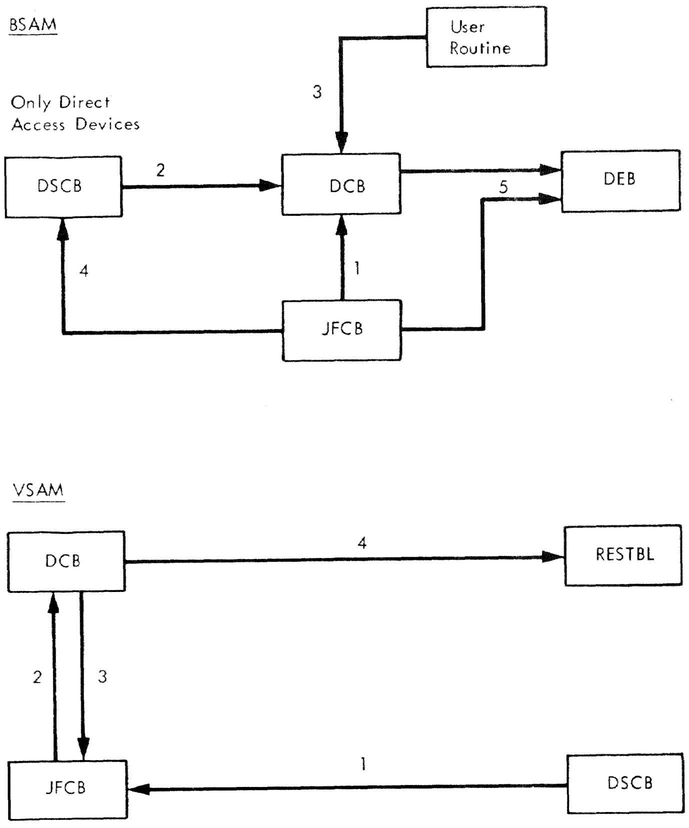


Figure 34. Data Flow During Open Processing

Data Extent Block (DEB) and the VAM access methods place functionally similar information in the RESTBL.

The Data Extent Block (DEB) is variable in length and is logically divided into three sections. The first section contains information about the data set and devices such as the number of tracks per cylinder as well as pointers to the other control blocks associated with the data set.

The second section anchors a chain of pointers which describes the location of each of the user's Data Event Control Blocks (DECBs) that have not yet been processed. Under some error conditions, succeeding I/O requests may be queued on the DEB pending a resolution of the error.

The third section exists only for direct access volumes and contains information such as the size and characteristics of each group of contiguous track or cylinders (i.e., extents) on which the data set resides and the direct access seek and search addresses used for the last read or write to this data set. The DEB format is shown schematically in Figure 35.

The Relative External Storage Correspondence Table (RESTBL) is composed of three parts - a RESTBL header, the page correspondence entries, and the DCB headers and VPAM data set member headers.

The RESTBL header, at the beginning of RESTBL, is immediately followed by the data set page/external page correspondence entries. The DCB headers and member

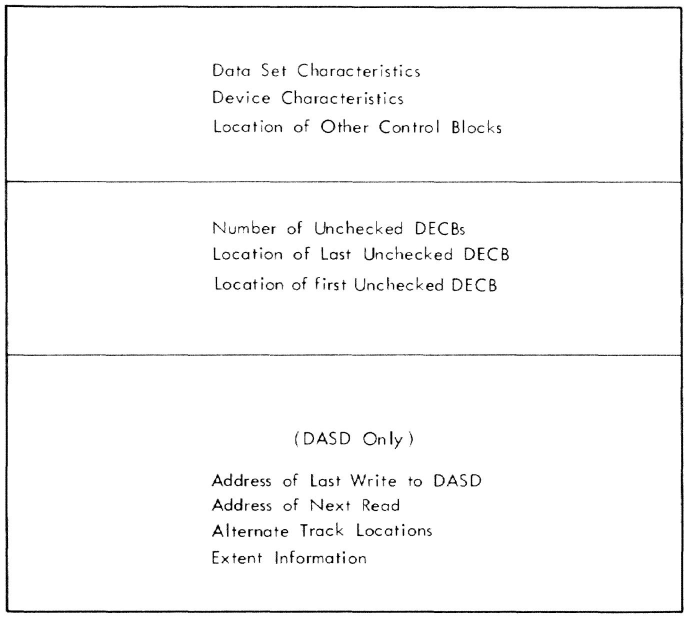

Data Set Characteristics
Device Characteristics
Location of Other Control Blocks
Number of Unchecked DECBs
Location of Last Unchecked DECB
Location of first Unchecked DECB
(DASD Only)
Address of Last Write to DASD
Address of Next Read
Alternate Track Locations
Extent Information

Figure 35. Data Extent Block (DEB)

headers originate at the end of RESTBL and expand toward the external page entries. A RESTBL thus has the following organization:

```
┌─────────────────────────────────────────┐
│              RESTBL HEADER              │
├─────────────────────────────────────────┤
│              Dataset Page               │
│                   vs                    │
│                EXTERNAL                 │
│                  PAGE                   │
│                ENTRIES                  │
├─────────────────────────────────────────┤
│             Up to one page of           │
│                AVAILABLE                │
│                  SPACE                  │
│             (all zero bytes)            │
├─────────────────────────────────────────┤
│              DCB HEADERS AND            │
│              MEMBER HEADERS             │
└─────────────────────────────────────────┘
```

The RESTBL header entry contains general information about the pages of a data set (such as the number of pages occupied by a VISAM data set directory) and describes the content of the remainder of the RESTBL.

The DCB header entries summarize DCB information for the VAM access methods in much the same way that the Data Extent Block (DEB) does for the sequential access methods. The header contains information such as the options the user specified in the OPEN macro and the location of the buffer.

Because a VPAM data set is really a collection of independently organized data groups, there must be a header for each member. These headers are in the same format as the data set information in the RESTBL header.

There is an entry in the middle section of the RESTBL for each data page assigned to the data set (including all data set members). Thus, the fifth data page assigned to the data set is the fifth entry.

The general format of each one word RESTBL entry is as follows:

```
 0    2              15                   31
┌────┬──────────────┬─────────────────────┐
│Flag│Relative      │External Storage     │
│    │Volume Number │Page Number          │
└────┴──────────────┴─────────────────────┘
```

FLAG: 00 - An External storage page has been assigned and there is a copy of this data set page on External storage.

01 - Not-in-use. An External storage page has been assigned to the data set, but the user

has not yet placed a logical record in this page. This page can, at the user's option, be released during CLOSE processing.

10 - Data has recently been written into a record located in this page (which is now in main or auxiliary storage), but the page has not yet been written to external storage. If a logical record spans several pages, each page will carry this flag, even though the user has not necessarily written into that portion of the logical record continued in this page.

11 - A permanent error was encountered when an attempt was made to write on this external storage page. The data was placed on a substitute page and this page is not used although it still belongs to a DSCB extent allocated to this data set.

Relative Volume Number: This identifies the relative number of the volume within the PVT. The PVT (public or private volume table) contains a list of the symbolic device addresses, volume serial numbers, and device codes of all devices on which the data set may reside.

External Storage Page Number: The physical address of VAM pages on external storage can be calculated from knowledge of the device type and the relative location of the page on the direct access device. A Format E or F DSCB is thus able to describe external storage by relative page number as well as describe a VAM data set by relative page number.

If the data set is sharable, there will be another full word entry for each data set page containing interlocks. VAM sharing is discussed separately in "Sharing."

When processing of a data set has been completed, the data set must be closed. The close processing, initiated by the execution of a CLOSE macro instruction, may be performed any time after open processing has been completed. Usually, however, a data set is closed after all I/O operations have been completed. Close processing is initiated when a task is to be abnormally terminated (ABEND) or as a result of a LOGOFF command.

Closing a data set may be thought of as a reversal of open processing. Closing a data set includes restoring the Data Control Block (DCB) to its original condition, i.e., as it was before open processing. Data Set Control Block (DSCB) processing, label processing, disposition of the data set volume, completion of all outstanding DECBs and deletion of the DEB are also done during closing. TAM CLOSE disables any relevant 2702 lines, removes the terminal from the task and then enables the 2702 line once again so that the terminal may be used again.

When using the Basic Sequential Access Method (BSAM), a CLOSE TYPE=T macro instruction may be issued for data sets residing on magnetic tape. This causes a bit to be set in the parameter list which is passed to the CLOSE COMMON routine indicating that a temporary close is to take place. The temporary close executes like a normal close, except the DCB is not restored and the DEB is not deleted so that I/O may be continued without a new OPEN macro instruction being issued. It is also used for convenient repositioning of a volume. Figure 36 indicates the overall flow of close processing.

The CLOSE COMMON Routine performs those close processing functions which are necessary no matter which access method is being used. CLOSE COMMON branches to the appropriate Access Dependent Close (ADC) Routine. The MAINLINE EOV routine, indicated under CLOSE SAM is quite extensive and its main components are shown in Figure 36.

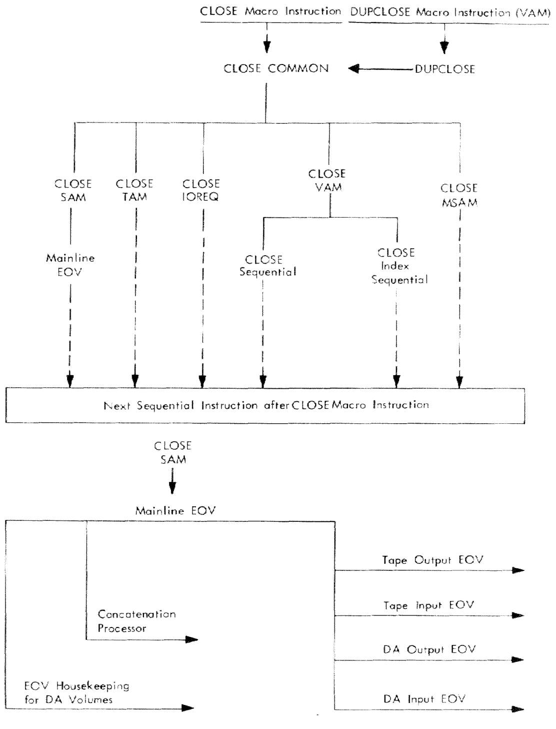


Figure 36. CLOSE Processing

MAINLINE EOV may also be called by the BSAM CHECK and Force End-of-Volume (FEOV) routines.

The overall logic of I/O processing can best be described through two illustrative examples. An example of the processing involved in executing a Basic Sequential Access READ macro instruction is presented in "Example of BSAM Processing." The action of the system in the creation and processing of a three record Virtual Sequential data set is described in "Example of Virtual Sequential Processing."

The READ example will also give a first opportunity to tie together the parts of TSS/360 already described.

EXAMPLE OF BSAM PROCESSING

This example describes a read operation.

In order to read a record from an already existing BSAM data set, the programmer must issue a DDEF command and execute the OPEN macro instruction for input. This processing is summarized in Figure 37.

The step by step processing in the system is shown in Figure 38. The processing is described below and is keyed to the figure.

1. When a program executes the READ macro instruction a Type II linkage is performed to invoke the BSAM READ/WRITE routine because it is a privileged routine. If the READ macro instruction had been generated in a control section with the "Privileged" attribute, a Type I linkage would be performed for a transfer within the same privilege level.

   The Type II linkage consists of placing a code, established by convention, in Register 15, which specifies a link to the BSAM Read/Write routine, and then executing an ENTER Supervisor Call which generates a hardware interruption which will cause the Interrupt Stacker module of the Resident Supervisor to be invoked.

   When an interruption occurs, the PSW Register contains the address of the instruction that would have been executed next if the interruption had not occurred. This extended PSW is moved by the Model 67 from the PSW register into the appropriate old PSW area in the Prefixed Storage Area (PSA). The new extended PSW is fetched by the Model 67 from the corresponding new PSW area in the PSA and becomes the machine PSW.

   The address contained in this new PSW is the Interrupt Stacker entry point which corresponds to the type of interruption.

2. The Interrupt Stacker is the module of the Supervisor which receives control when any hardware interruption is taken. The Interrupt Stacker identifies the interrupt type and creates a generalized queue entry (GQE), which represents a unit of work for other parts of the Supervisor. In this case the stacker is entered at its SVC interrupt entry point. The entry point address is carried in the instruction counter field of the new SVC PSW. The Interrupt Stacker temporarily saves General Registers 0 to 4, 14, and 15 in the PSA since each type of interrupt routine has its own unique processing before it can enter a subroutine to save the status of the machine at the time of the interrupt.

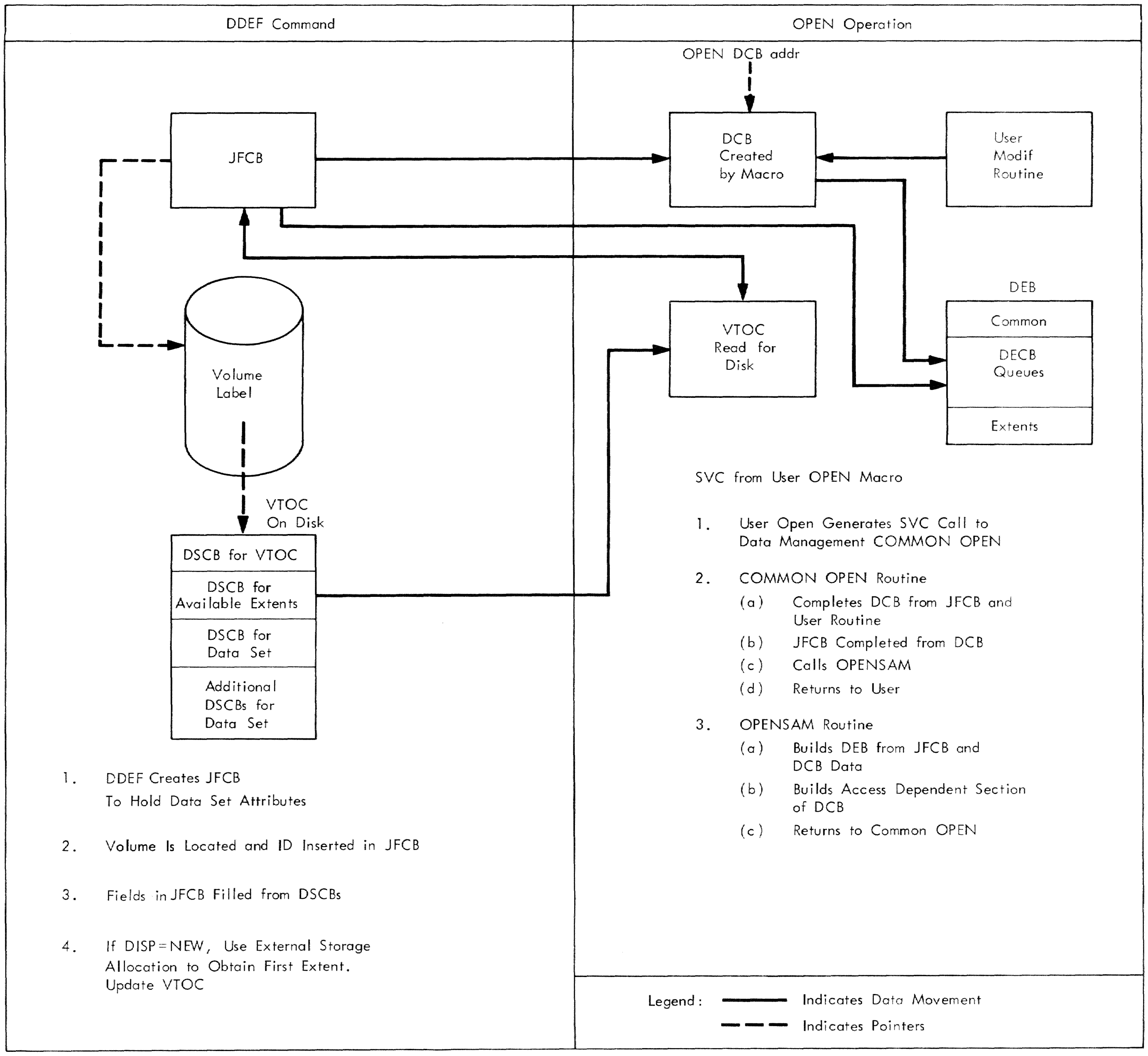
   Supervisor Core Allocation (SCA) is called by the Interrupt Stacker to get 64 bytes in which to build the GQE for this interruption.

   Then the Interrupt Stacker proceeds to build the GQE. The GQE is initialized with the following information:

   - The TSI address and the SVC interrupt code from the PSA.
   - The Instruction Length Code (ILC) obtained from the Supervisor Call old PSW.
   - The symbolic designation (Loc-on-Q) of the SVC queue.
   - The virtual storage address of the SVC instruction if the SVC was the subject of an execute instruction.

3. At this point a check is made to determine if the request is for services of a ~~privileged~~ routine in virtual storage (i.e., SVC<128). In this example it is and a software interruption is enqueued for the task by calling the Queue GQE on TSI routine which adds the GQE to the TSI's SVC queue and turns on the SVC interruption pending bit in the TSI. The Interrupt Stacker next saves the old extended PSW (located now in the PSA) in the appropriate location in the task's Extended TSI (XTSI). In addition, all General Purpose, Control, and Floating Point registers are placed in the XTSI. In this fashion, the complete status of the task is preserved. The status will be restored at some later time.

Bounce P[illegible] to VPL [illegible]
change [illegible] in XTSI put SVC [illegible]



1. DDEF Creates JFCB To Hold Data Set Attributes
2. Volume Is Located and ID Inserted in JFCB
3. Fields in JFCB Filled from DSCBs
4. If DISP=NEW, Use External Storage Allocation to Obtain First Extent. Update VTOC

SVC from User OPEN Macro

1. User Open Generates SVC Call to Data Management COMMON OPEN
2. COMMON OPEN Routine
   - (a) Completes DCB from JFCB and User Routine
   - (b) JFCB Completed from DCB
   - (c) Calls OPENSAM
   - (d) Returns to User
3. OPENSAM Routine
   - (a) Builds DEB from JFCB and DCB Data
   - (b) Builds Access Dependent Section of DCB
   - (c) Returns to Common OPEN

Figure 37. DDEF and OPEN Processing

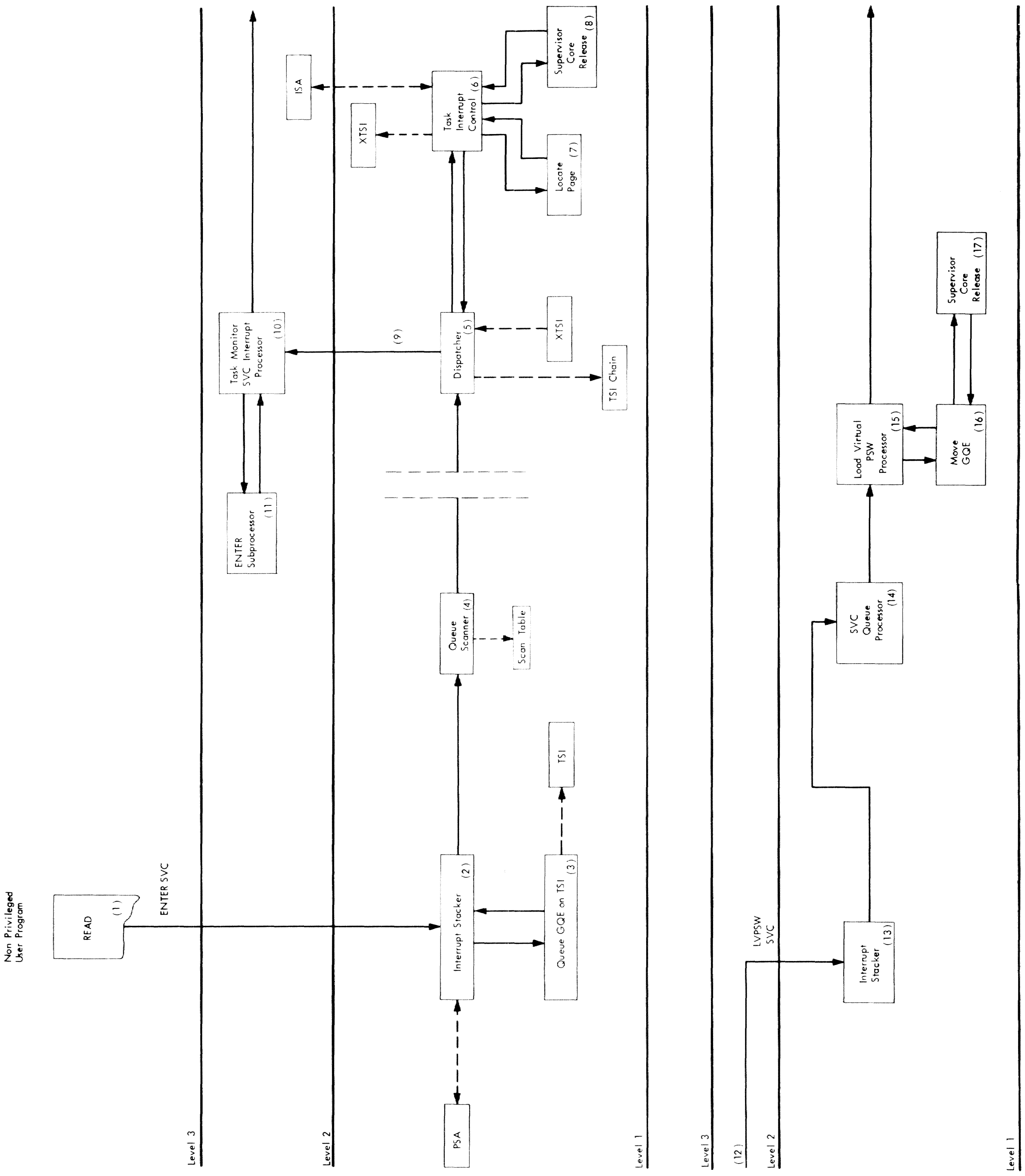


Figure 38. BSAM Read Walkthrough (Part 1 of 3)

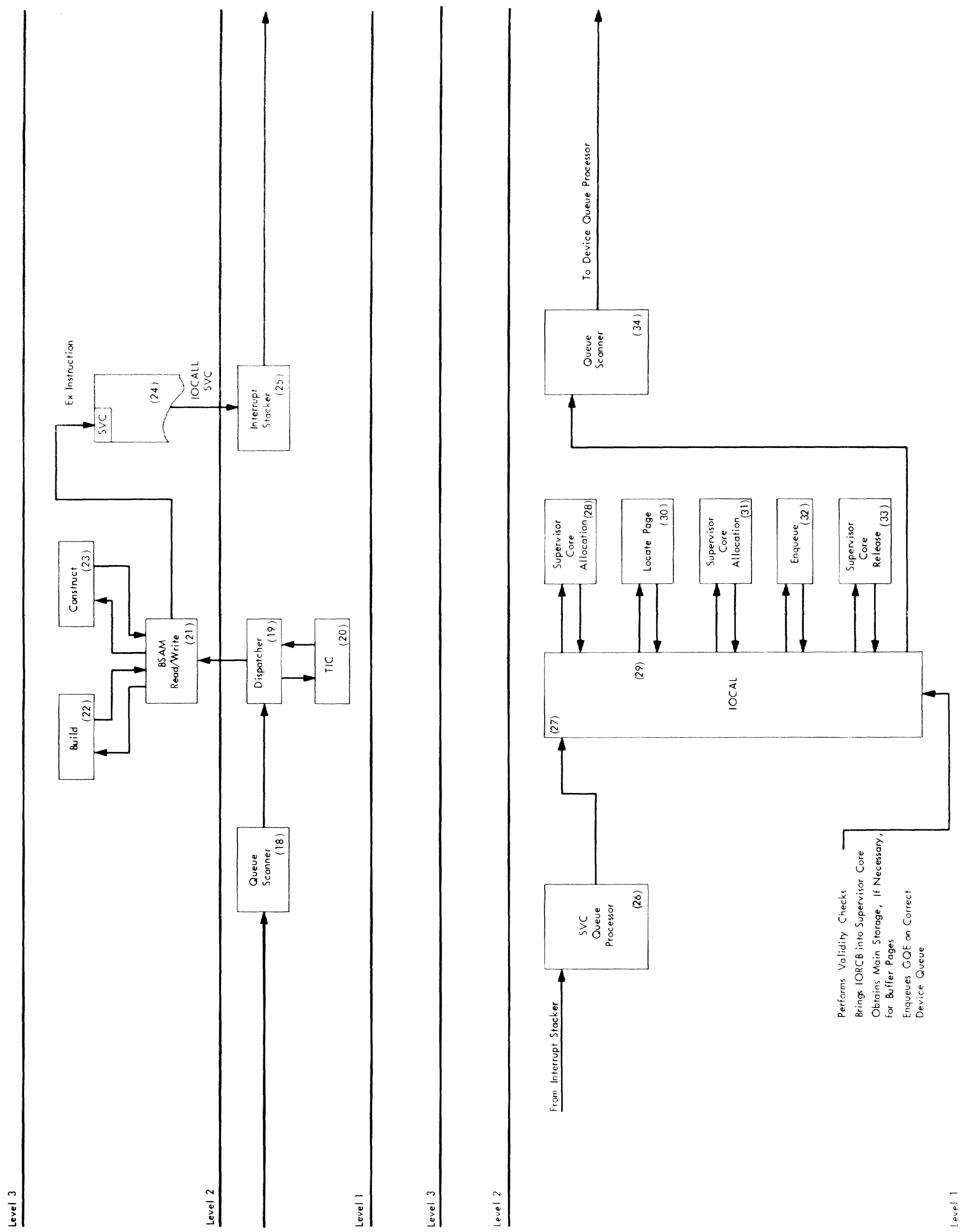


Figure 38. BSAM Read Walkthrough (Part 2 of 3)

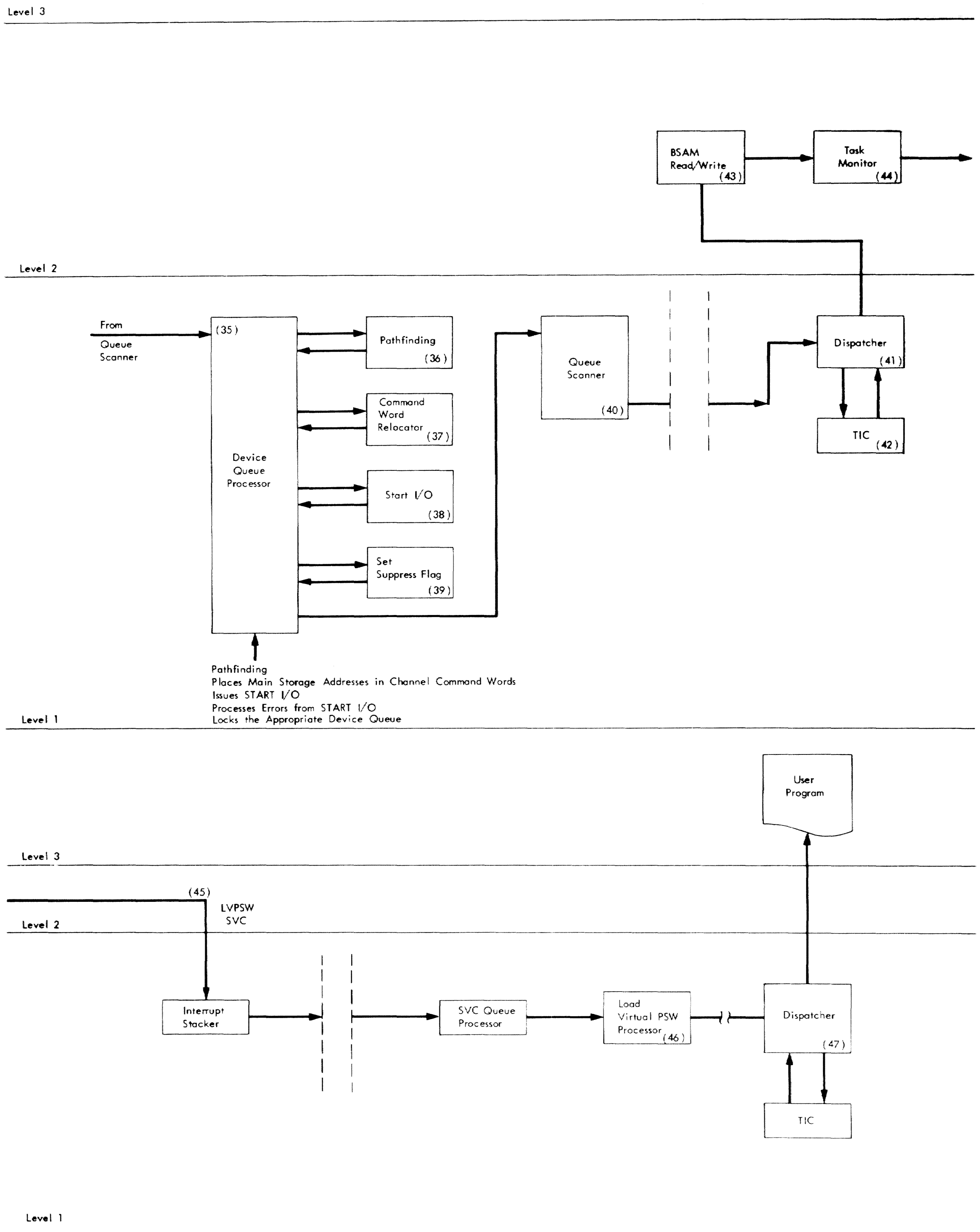


Figure 38. BSAM Read Walkthrough (Part 3 of 3)

Having completed its processing, the Interrupt Stacker unmasks interruptions and exits to the Queue Scanner.

4. The function of the Queue Scanner is to look for work which the Supervisor can do. The Queue Scanner searches the various queues for GQEs which can be processed. When one is found, the appropriate Supervisor module (Queue Processor) is invoked.

   In this example, the program has not created any work for a Queue Processor. However, there may be quite a bit of previously queued work.

   When the Queue Scanner finds that there are no more GQEs that can be processed, it exits to the Internal Scheduler which calls the Dispatcher.

5. The Dispatcher applies its scheduling algorithm to select the next task to be placed in execution, since a CPU is now available to do work in the Problem state.

   The Dispatcher executes with interruptions masked because it normally does not return to the Queue Scanner, but exits to the Problem state. Operating with interruptions masked prevents an interruption from being serviced until the next interruption from the Problem state.

   The task described in this example may not be immediately selected for execution by the Dispatcher, but it will eventually be considered.

   When the task is selected for execution, the Dispatcher wishes to determine if there is a task interruption pending for this task. Task Interrupt Control (TIC) is called to make this determination.

6. TIC provides an important link between the Resident Supervisor and the Task Monitor. Its function here, since there is a pending task interruption, is to make sure that when the task is placed in execution, it is started at the entrance to the appropriate interrupt processing routine in the Task Monitor. In order to do this TIC must set up the correct data in the first double word (PSW location) in the XTSI. This data is obtained from the new SVC Virtual PSW in the ISA, which is located in a fixed location in virtual storage, segment 0, page 0.

7. TIC invokes the Locate Page routine to determine the location of the ISA page. In this example, the ISA is still in main storage. Finding that the ISA is in main storage, TIC obtains the task's PSW from the XTSI and places it in the old SVC Virtual PSW location in the ISA.

   Next, TIC translates the new SVC Virtual PSW information obtained from the ISA into extended PSW format, places it in the XTSI and sets the TSI interrupt mask field. It also copies information from the GQE into the ISA.

8. Supervisor Core Release is then called to release the space used by the GQE. Task Interrupt Control then returns to the Dispatcher.

9. The Dispatcher now sets the task in execution by loading the PSW and the General Purpose, Floating Point and Control registers from the XTSI. The instruction counter in the PSW points to the Task Monitor's SVC Interrupt entry point. The Task Monitor is already addressable in virtual storage because it is included in each task's Initial Virtual Memory. (See Virtual Memory Allocation.)

   However, it is quite possible that the referenced virtual storage page is not in main storage and must be paged into main storage. In this example, the processing involved in paging is ignored. (See Paging.)

10. When the Task Monitor's SVC Interrupt Processor receives control it will detect that the VPSW interrupt code specifies an ENTER SVC.

11. The Task Monitor SVC Interrupt Processor then links to its own Enter Subprocessor.

    The BSAM Read/Write module is part of IVM, so the Task Monitor will pick up the appropriate V-type and R-type address constants from an Enter Table in its PSECT. The Task Monitor will then place the status of the task in a save area in the ISA. This long save is necessary, because BSAM Read/Write may cause more task interruptions before Task Monitor processing is complete.

    Finally, the Task Monitor will provide a save area for BSAM Read/Write and execute a Load Virtual PSW supervisor call.

    The LVPSW SVC allows the Task Monitor to change the hardware interruption mask and storage protection key in the task's PSW (if needed) as well as setting the software interruption mask in the TSI to that setting which BSAM Read/Write requires.

12. The LVPSW SVC will cause a hardware interruption just as the ENTER SVC did.

13. The Interrupt Stacker will again process as in step (2). However, this time the Interrupt Stacker will continue processing by placing a pointer to the GQE in a register, saving the task's status in the XTSI, unmasking interruptions, and then exiting directly to the SVC Queue Processor. This processing reflects the fact that this supervisor call is requesting the services of the Resident Supervisor.

14. When the SVC Queue Processor receives control from the Interrupt Stacker, the interruption code in the GQE is inspected. Based on the SVC number, the SVC Queue Processor links to the LVPSW subprocessor.

15. This routine, using as input the VPSW furnished to it, sets the appropriate task interruption mask bits of the TSI and places the VPSW (translated into Extended PSW format) into the first double word of the XTSI. The XTSI now points to the virtual address of the BSAM Read/Write routine. to task
exits by bounce pass

[illegible] GQE

16. The GQE is sent to MOVEGQE to be removed from the system. MOVEGQE inspects a flag in the GQE which specifies that we have finished processing this GQE. MOVEGQE links to Supervisor Core Release (SCR).

17. SCR reclaims the 64 bytes of core.

18. The LVPSW sub-processor exits to the Queue Scanner.

19. The Dispatcher ultimately selects the task to be put in execution. It routinely calls on TIC to set up for task interrupts, if any exist.

20. TIC finds none pending for this task and returns.

21. The Dispatcher blindly picks up the PSW Save field from the XTSI and loads it as current PSW causing the desired transfer back to the problem state at the BSAM Read/Write entry point. The overall processing of this Read request is depicted in Figure 39.

The BSAM Read/Write routine uses two major subroutines to perform its processing, a BUILD subroutine and a CONSTRUCT subroutine.

22. The BUILD subroutine builds a skeletal Input/Output Request Block (IORCB).

There is an IORCB built for every I/O request that requires a channel program. The IORCB contains an IOCAL SVC; the address constants for the BSAM Posting routine; and the channel program, in addition to other data.

23. The CONSTRUCT routine generates the required channel programs in the IORCB. The first decision CONSTRUCT must make is whether to use the IORCB as a buffer. For records longer than 1800 bytes, a flag is set indicating that the IORCB is not to be used as a buffer and a list of up to eight entries is built which provides the virtual storage address of each page of the buffer.

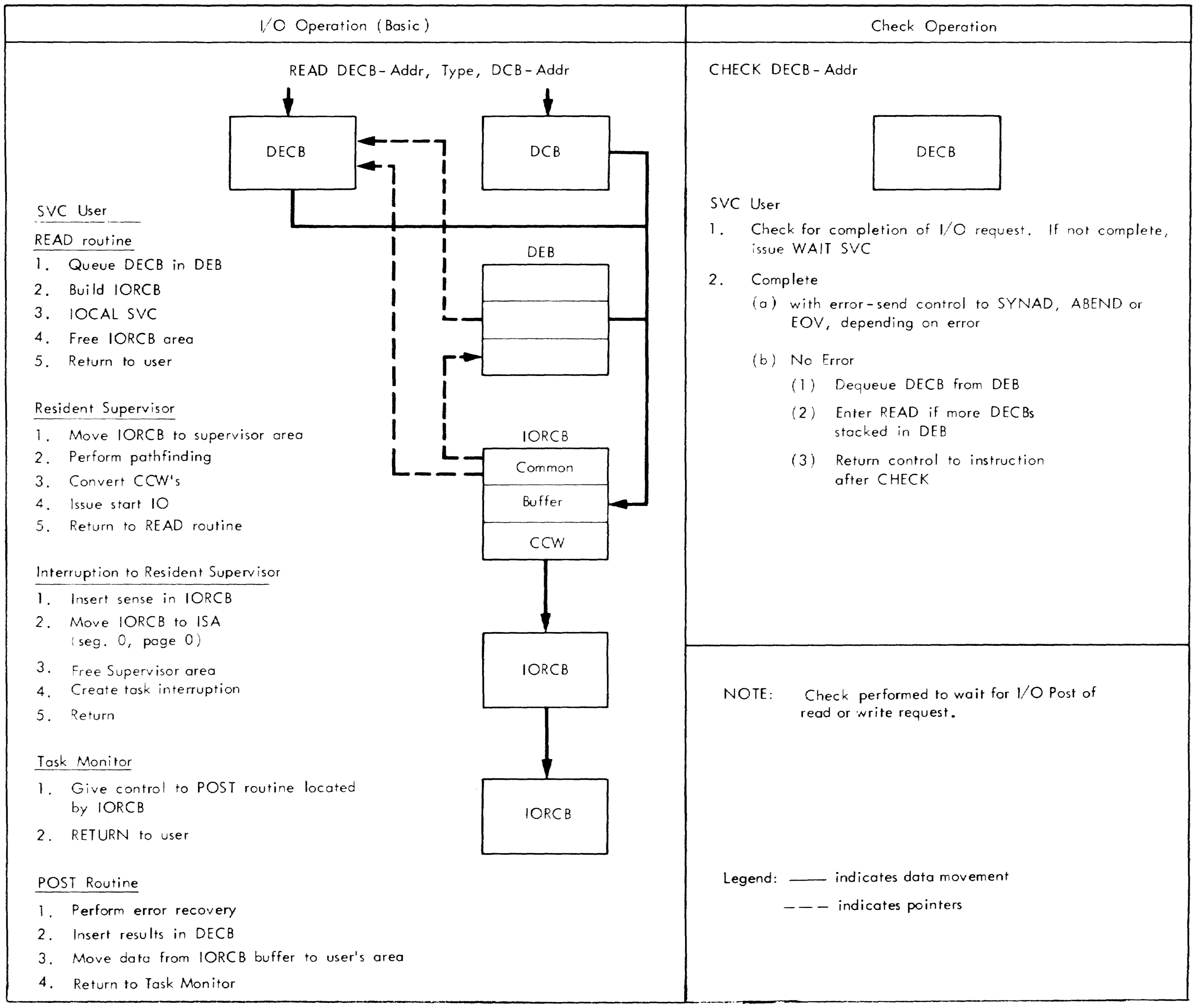
A channel transmits data to or from main storage, but pages which appear contiguously in virtual storage are not necessarily contiguous in main storage. Therefore, whenever a read or a write would span pages, data chaining must be used.

Notice that the page list allows I/O from non-contiguous virtual storage, since the page list can be any eight pages. However, there is no scatter-read/gather-write facility, unless IOREQ is used (scatter-read is the technique of reading selected blocks of a data set into non-contiguous storage locations; gather-write allows writing from selected locations in core, skipping unwanted, in-between areas of core). Under IOREQ, the programmer writes his own Virtual CCW list, and so data chaining is possible.

Next the channel program is built. The channel control words in the channel control program are not true CCWs because they contain virtual storage addresses. In point of fact, they only contain the low order bits of the virtual storage address or a displacement relative to the beginning of the IORCB buffer because there is only room for a 24-bit address in a real CCW. Using a larger address would distort the CCW format.

The high order portion of each address can be obtained from the page list or is implied from the virtual storage address of the IORCB itself.

The CONSTRUCT routine uses several subroutines to generate the channel program.



SVC User

READ routine

1. Queue DECB in DEB
2. Build IORCB
3. IOCAL SVC
4. Free IORCB area
5. Return to user

Resident Supervisor

1. Move IORCB to supervisor area
2. Perform pathfinding
3. Convert CCW's
4. Issue start IO
5. Return to READ routine

Interruption to Resident Supervisor

1. Insert sense in IORCB
2. Move IORCB to ISA (seg. 0, page 0)
3. Free Supervisor area
4. Create task interruption
5. Return

Task Monitor

1. Give control to POST routine located by IORCB
2. RETURN to user

POST Routine

1. Perform error recovery
2. Insert results in DECB
3. Move data from IORCB buffer to user's area
4. Return to Task Monitor

SVC User

1. Check for completion of I/O request. If not complete, issue WAIT SVC
2. Complete
   (a) with error-send control to SYNAD, ABEND or EOV, depending on error
   (b) No Error
      (1) Dequeue DECB from DEB
      (2) Enter READ if more DECBs stacked in DEB
      (3) Return control to instruction after CHECK

NOTE: Check performed to wait for I/O Post of read or write request.

Legend: —— indicates data movement
— — — indicates pointers

Figure 39. Overall Processing of Read Request

If the programmer had been using the Terminal Access Method (TAM) instead of BSAM, a table called the Terminal Control Program Library (TCPL) would specify the channel program. This is schematically described in Figure 40.

A final consideration concerning the construction of the IORCB is the fact that a Time Slice End could occur at just the wrong time. (That is, after the IORCB has been completed but before the IOCAL SVC has been executed.) This would mean that, at the beginning of the next time slice, the IOCAL SVC would be executed but the IORCB would quite likely not be in main storage. To avoid this difficulty, the IOCAL SVC is placed in the first half-word of the IORCB and is executed out-of-line by using the System/360 Execute instruction. This accomplishes two things. First, the virtual storage address of a SVC is saved by the Interrupt Stacker when the SVC is the subject of an Execute instruction. Thus the virtual storage address of the IORCB is passed to the Resident Supervisor in a convenient way. Second, the relocation mechanism is invoked if the SVC is not in main storage. This assures that the IORCB will be in main storage at the time the SVC is executed and will never have to be paged in after the SVC has been performed.

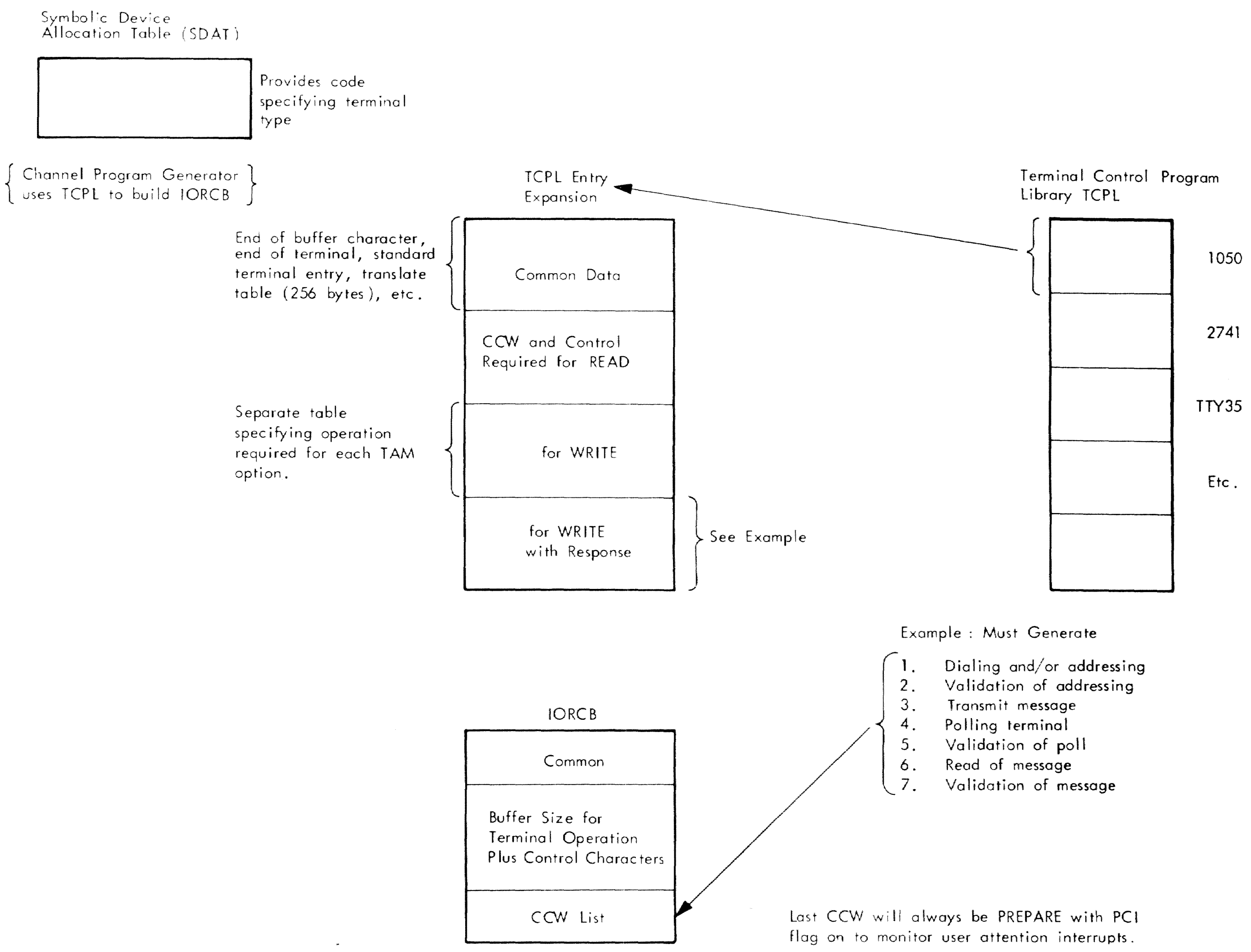

Symbolic Device
Allocation Table (SDAT)
Provides code specifying terminal type
Channel Program Generator uses TCPL to build IORCB
TCPL Entry Expansion
Terminal Control Program Library TCPL
End of buffer character, end of terminal, standard terminal entry, translate table (256 bytes), etc.
Common Data
CCW and Control Required for READ
Separate table specifying operation required for each TAM option.
for WRITE
for WRITE with Response
See Example
1050
2741
TTY35
Etc.
Example : Must Generate
1. Dialing and/or addressing
2. Validation of addressing
3. Transmit message
4. Polling terminal
5. Validation of poll
6. Read of message
7. Validation of message
IORCB
Common
Buffer Size for Terminal Operation Plus Control Characters
CCW List
Last CCW will always be PREPARE with PCI flag on to monitor user attention interrupts.

Figure 40. TAM IORCB Generation

24. When the IOCAL SVC is executed, the Interrupt Stacker is entered at its entry point for SVC interrupts.

25. The Interrupt Stacker calls Supervisor Core Allocation to get 64 bytes for a GQE.

    The GQE is then filled in with the interrupt code and the virtual storage address of the SVC instruction. Processing continues as in step (13).

26. The Interrupt Stacker gives control to the SVC queue processor.

27. On the basis of the SVC code, the SVC Queue Processor makes a selection of the appropriate SVC Subprocessor to continue the processing of the GQE. In this case, the IOCAL SVC Subprocessor is called.

28. IOCAL first executes a Test and Set instruction to determine if another CPU is currently executing in the subprocessor and thus using the subprocessor's one permanent work area. In this example, this is the case and IOCAL invokes Supervisor Core Allocation to obtain a temporary work area. This type of processing is common to most Resident Supervisor routines.

29. Then, since IOCAL will be working on the IORCB, IOCAL must first find out where the IORCB is located in main storage. The virtual storage address of the IOCAL SVC is available in the GQE, and this address is also the virtual storage address of the IORCB. Since the supervisor operates in nonrelocation mode, this address cannot be used directly. The virtual storage address must first be translated into

a main storage address. This function is performed with the help of the Locate Page routine.

30. Locate Page is called and returns the main storage address of the Page Table entry for the IORCB page. From the Page Table entry, the main storage address of the IORCB itself is determined.

31. The IOCAL Subprocessor then gets main storage from Supervisor Core Allocation and moves the IORCB into the supervisor main storage area. If the record was so large that BSAM used a page list instead of using the IORCB as a buffer, the IOCAL Subprocessor would have additional duties, as follows.

    Although BSAM provided the virtual storage address of each page list entry, IOCAL is the routine which must obtain a real address for the page. If any page is not in main storage, IOCAL initiates paging operations to bring the page into main storage. If this buffer has never yet been written into, User Core Allocation will supply a zeroed-out block of main storage and no page in will be necessary.

32. If the Task Symbolic Device List indicates that the task is permitted to perform I/O on this symbolic device, Enqueue is called to place the GQE (which now points to the IORCB in supervisor storage) on the proper device queue. (The access method has specified the device by furnishing its symbolic device address in the IORCB.)

33. IOCAL invokes Supervisor Core Release (SCR) to return the temporary work area space.

34. The standard exit is taken to the Queue Scanner.

35. Since there is still work to be done to process the I/O request, the Queue Scanner will find the GQE on the appropriate device queue and call the Device Queue Processor to initiate the I/O operation.

    Recall that for each and every device attached to the system, there is an entry on the Scan Table. Each entry has a pointer to the first GQE, which represents a request for I/O to or from that device. Succeeding requests - GQEs - are chained from the first request. The Device Queue processor processes these requests for I/O, and does this for all of the devices on the system except the paging drums.

    The primary function of the Device Queue Processor (DQP) is to initiate a successful I/O operation (Start I/O) given a GQE and an IORCB. The DQP also processes requests to Halt I/O and to initiate sense operations. The function of the Channel Interrupt Processor (CIP) is to process the GQE which is created when an I/O interruption occurs. The DQP and CIP can affect each other's operations greatly by communication with each other.

    The Device Queue Processor must create the actual channel program. The Pathfinding routine is called for this purpose.

36. Pathfinding converts the symbolic device address from the IORCB to an available hardware path (Channel Control Unit - Channel - Control Unit - Device) and marks those elements busy in the various pathfinding tables (see "Pathfinding").

37. The Device Queue Processor invokes the Command Word Relocator routine to complete the channel program by placing the proper main storage addresses into the CCWs. CCWs created by access methods contain components of virtual addresses.

38. The START I/O routine is now called to issue the Start I/O instruction to initiate the channel program. Upon return from Start I/O, the Device Queue Processor checks to make sure of successful initiation of the I/O operation. In this example, a successful initiation is assumed.

39. Set Suppress Flag is called to prevent the Device Queue Processor from being invoked again to process a GQE on this device queue (until the I/O operation just started is complete).

40. Note that the original GQE still remains on the device queue as exit is made to the Scanner.

41. The Queue Scanner, when it ultimately finds no GQE's that can be processed, calls the Dispatcher.

42. The Dispatcher may choose this task to be placed in Execution. Before placing a task in execution, the Dispatcher calls Task Interrupt Control (TIC) to check for pending task interrupts.

43. Since none is found for this task, the Dispatcher returns control to the task (BSAM READ/WRITE) at the point after the EXECUTE instruction which invoked the IOCAL SVC.

44. BSAM Read/Write has finished its processing and so it returns to the Task Monitor SVC Interrupt Processor.

[illegible] set inhibit task interrupt (TI)

The Task Monitor SVC Interrupt Processor merely exits to the Task Monitor Scanner-Dispatcher to look for further work.

The Task Monitor Scanner-Dispatcher is analagous to the Resident Supervisor's Queue Scanner. It scans the Interrupt Table looking for work to do and dispatches the appropriate routine when work is found.

45. Task Monitor Scanner-Dispatcher, finding no further work for the Task Monitor, restores the task's registers from the ISA and issues a Load VPSW SVC to return control to the last interrupted routine.

46. Processing continues as in steps 12-18. The LVPSW subprocessor will set the PSW in the task's XTSI to point to the first instruction following the READ macro the program executed.

47. Eventually, the Dispatcher picks up the PSW save field from the XTSI and loads it as the current PSW for a CPU and thus returns control to the program.

Some time later, the program executes a CHECK macro to synchronize the READ operation, since it can proceed no further without the data. (See Figure 41).

Although not explicitly shown in the figure, the CHECK routine is a privileged routine and as such, is invoked through an ENTER supervisor call just as the BSAM Read/Write routine processing described in steps 1-21.

The CHECK routine tests the DECB associated with the READ macro expansion and determines, in this case, that the I/O operation is not complete. CHECK then issues an AWAIT SVC to take the task out of execution until a software

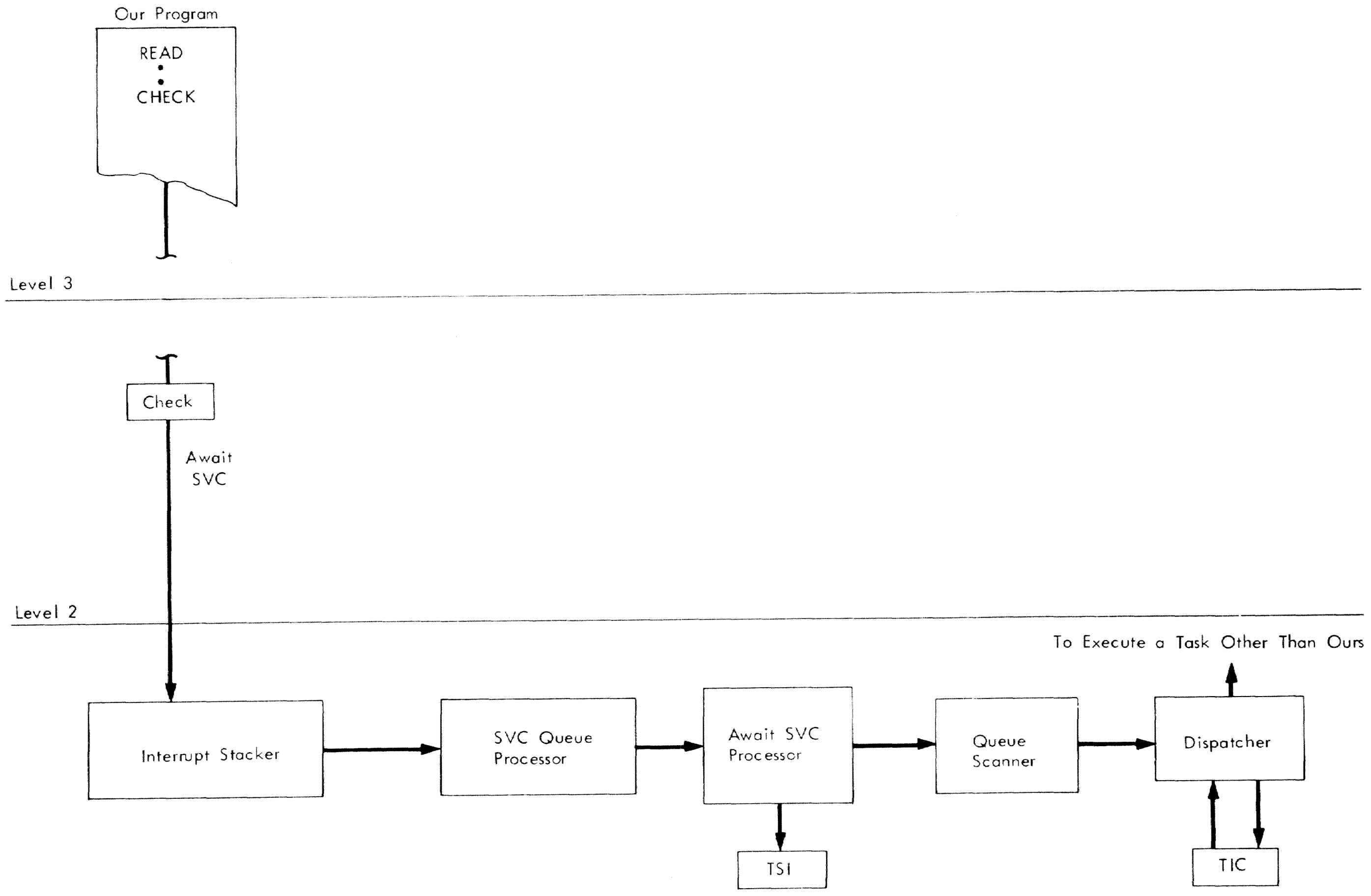


Figure 41. CHECK Macro Processing

interruption is presented to the task. Depending on the complexity of the operations, the next interruption presented to the task need not signal the completion of this channel program. In such a case, the CHECK routine would reissue the AWAIT SVC when the Task Monitor returns control to the program at the point in the CHECK routine following the AWAIT SVC. However, in this case, the next interruption presented to the task will signal that the channel program has completed operation.

The AWAIT SVC will cause the Await SVC subprocessor of the Resident Supervisor to be invoked.

The Await subprocessor will place the task in the Inactive list by calling Rescheduling and places the task in delay status (see "Scheduling Algorithm").

Many milliseconds later, the I/O operation is completed and a channel-end, device-end condition causes a hardware I/O interruption. The flow of control is now shown schematically in Figure 42, which is keyed to the following description.

1. Recognition of this interruption causes entry to the Interrupt Stacker at its I/O interruption entry point.

   The Interrupt Stacker performs standard functions for an I/O interruption. The Interrupt Stacker creates a GQE and inserts the hardware device address (automatically stored with the I/O interrupt as the interrupt code), the Channel Status Word (CSW), and the symbolic designation of the queue on which the GQE is to be placed for later processing. A test is made to see if the interruption came from a paging drum since such interruptions are handled differently from other I/C interruptions. Since this is not a paging drum interruption, the Interrupt Stacker designates that this GQE is to be placed on the Channel Interrupt Queue.

2. The Interrupt Stacker ultimately exits to the Queue Scanner. If the CPU which took the interruption was operating in the Supervisor state prior to the interruption, the Interrupt Stacker causes execution to resume at the point of interruption. The Queue Scanner will be entered later, when the Queue Processor currently in control finishes its work. If the CPU was interrupted from the problem state, the Interrupt Stacker will transfer directly to the Queue Scanner.

3. A GQE is found on the Channel Interrupt Queue by the Queue Scanner. The Channel Interrupt Processor (CIP) is invoked.

4. The CIP calls Set Suppress Flag to prevent a second CPU from being invoked to process the same queue.

5. The Channel Interrupt Processor then calls Reverse Pathfinding to convert the actual hardware device address to the symbolic device address.

   CIP looks at the device queue whose symbolic number has just been returned by Pathfinding. CIP finds at least one GQE on the device queue as well as a flag set (in the Scan Table entry for that queue) indicating that an I/O operation was in progress for that device. On this basis CIP decides that the interruption is associated with a previous IOCAL SVC, and thus distinguishes between synchronous and asynchronous I/O interruptions. CIP then proceeds to call Reverse Pathfinding again.

6. The function performed this time is to remove the busy indication in the pathfinding tables and thereby free the hardware path to the device.

   The Channel Interrupt Processor finds through a series of tests that this interruption is to be given to the task for further processing, i.e., returned to virtual storage as a task interruption. Status information is copied from the interruption GQE into the device GQE.

7. Dequeue is called to detach the original GQE from the device queue.

8. Queue GQE on TSI is then called to attach the original GQE to the TSI and turn on the TSI's Synchronous I/O Interrupt Pending bit. Finding that the task that Queue GQE on TSI has been called to process is in Delay status, Queue GQE on TSI resets the task to Ready status and activates the task. That is, it places the task's TSI on the Active List.

9. CIP calls Dequeue to detach the interruption GQE from the Channel Interrupt Queue and calls Move GQE (which uses Supervisor Core Release) to remove this GQE from the system.

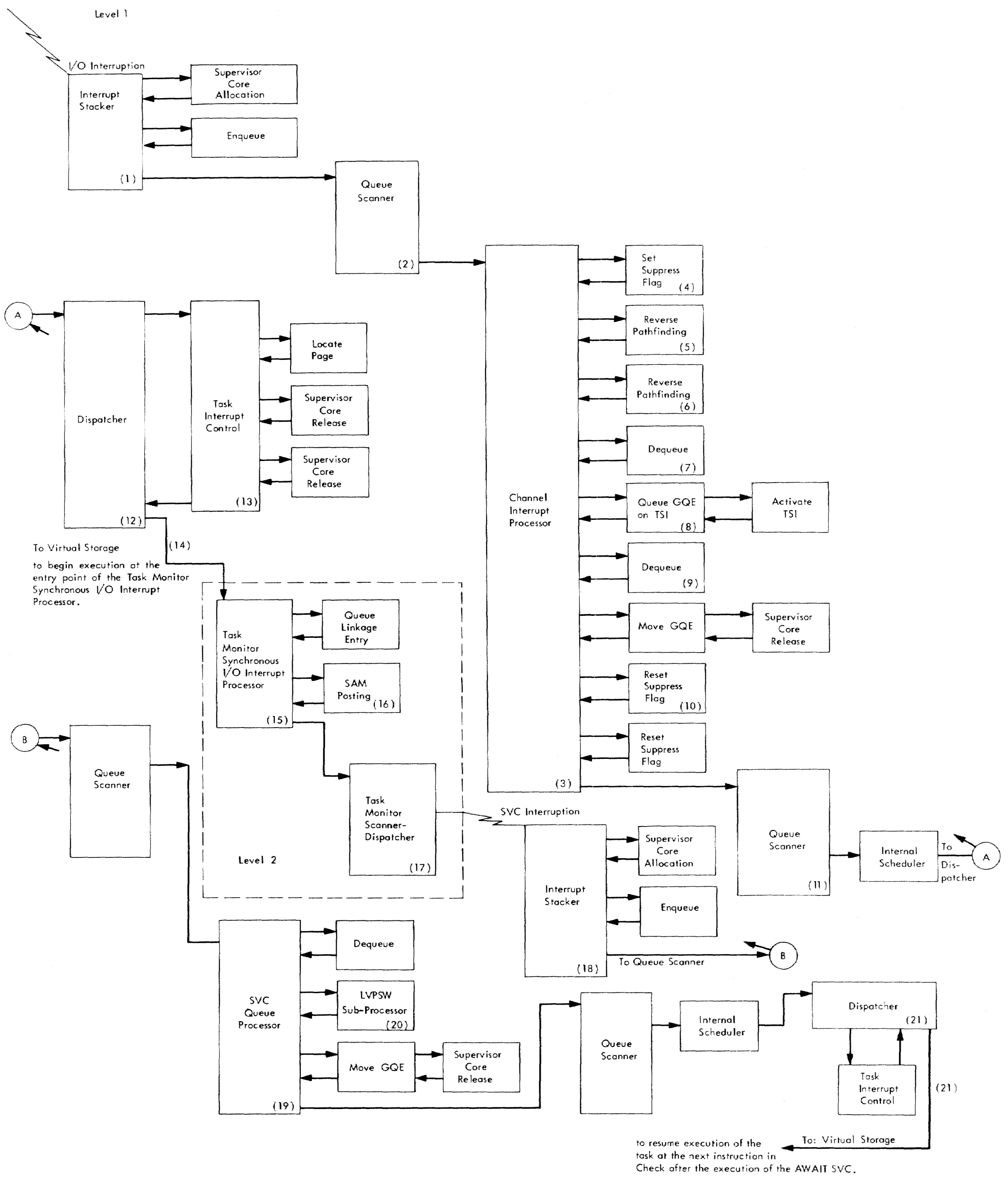


Figure 42. Resident Supervisor Task Monitor Synchronous I/O Flow (Non-Terminal)

10. CIP then calls Set Suppress Flag twice, once to free the device queue and again to free the Channel Interrupt Queue. The device queue in question must be freed because it has been locked during the I/O operation. Control is then given to the Queue Scanner.

11. When no further work is found, the Queue Scanner exits to the Internal Scheduler which, in turn, calls the Dispatcher.

12. The Dispatcher calls TIC.

13. TIC, after verifying that the ISA page is in main storage via Locate Page, moves information from the GQE to the ISA. TIC saves the XTSI's PSW data in the Synchronous I/O Old VPSW and replaces the PSW with the Synchronous I/O New VPSW (translated into real extended PSW format). The instruction counter of the new PSW points to the Task Monitor's Synchronous I/O Interrupt Processor entry point. TIC then removes the GQE from the TSI. Supervisor Core Release is called to free the GQE space. TIC moves the IORCB into the ISA and again goes to Supervisor Core Release to free IORCB space before returning to the Dispatcher.

14. The Dispatcher now loads the registers and the PSW from the XTSI, thereby giving control to the Task Monitor Synchronous I/O Interrupt Processor (TMSYNCH).

15. This routine first calls Queue Linkage Entry (QLE) to see if the user wished to have control sent to a routine he has specified after the BSAM access method Posting routine has processed the interruption. In this case none is found and QLE returns.

16. TMSYNCH then gives control to BSAM Posting (whose address constants are obtained from the IORCB). The Task Monitor directly calls the BSAM Posting routine, rather than executing a Load Virtual PSW supervisor call, because this routine runs with the same interruption mask as TMSYNCH.

    BSAM Posting performs standard posting functions, including marking the I/O event complete in the Data Event Control Block (DECB) associated with the READ request in our program. Control returns to TMSYNCH.

17. TMSYNCH has finished processing the interruption and exits to the Task Monitor Scanner-Dispatcher.

CHECK returns to Task Monitor
Queue Scanner

18. The Task Monitor Scanner-Dispatcher, finding no further work for the Task Monitor, restores the task's registers from the ISA and issues a Load VPSW SVC to return control to the last interrupted routine. The Load VPSW SVC is used to effect the transfer of control, in part because it is also desired to change the protection key in the task's PSW.

19. Eventually, the Resident Supervisor SVC Processor invokes the LVPSW Subprocessor.

20. The LVPSW subprocessor will set the PSW in the task's XTSI to point to the first instruction following the AWAIT SVC that the CHECK macro in the program executed.

21. Eventually, the Dispatcher picks up the PSW save field from the XTSI and loads it as the current PSW for a CPU and thus returns control to the program. Control is returned to the CHECK routine which, having nothing further to do, returns to the Task Monitor. If there is no further work for the Task Monitor scanner-dispatcher to dispatch, the Task Monitor restores control to the problem program.

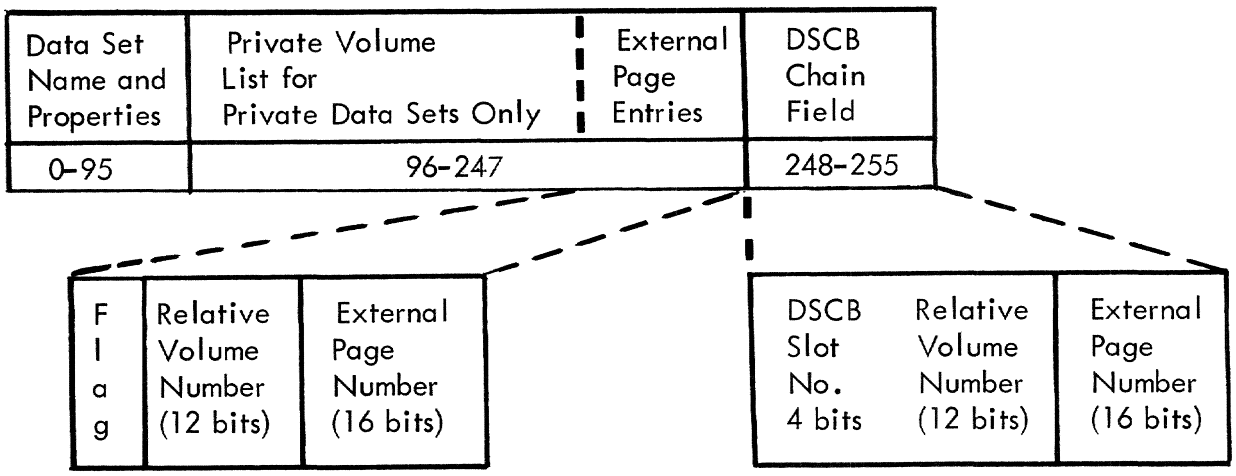
    Upon return to the program, a code of zero in Register 15 signifies a normal return, and the program continues with the instruction following the CHECK macro instruction. A nonzero code indicates an exception condition and a link to either SYNAD or EODAD occurs.

    The BSAM READ processing is now finished.

EXAMPLE OF VIRTUAL SEQUENTIAL PROCESSING

In this example, the program is going to create a new virtual sequential data set consisting of three 11,000 byte logical records. At this time, the program is going to place some master information in the logical records. In processing the data set, the program will use the OUTPUT option in the OPEN macro instruction. The data set will be created using Locate Mode PUT macro instructions. During the OPEN processing, primary space allocation (specified in the JFCB) will be assigned using a format E DSCB. In this example, the data set is to reside on a 2311 Disk volume. The pages will be flagged as assigned to the data set, but not in use. A schematic of the FORMAT E DSCB is shown in Figure 43.

When the Virtual Sequential data set is opened, space will be allocated and this



Data Set Name and Properties
Private Volume List for Private Data Sets Only
External Page Entries
DSCB Chain Field
0-95
96-247
248-255
Flag
Relative Volume Number (12 bits)
External Page Number (16 bits)
DSCB Slot No. 4 bits
Relative Volume Number (12 bits)
External Page Number (16 bits)

Figure 43. Format E DSCB

DSCB will be written to external storage and the page entries will be used in creating RESTBL entries. These pages will appear in the RESTBL as not in use pages of the form 10 xxxx yyyy where 10 is the flag, xxxx is the Relative Volume Number of the volume on which the data set resides, and yyyy is the relative location of this page on the direct access volume. At this time, the external storage pages contained in the DSCB have been assigned to the data set, but the pages do not contain any logical records.

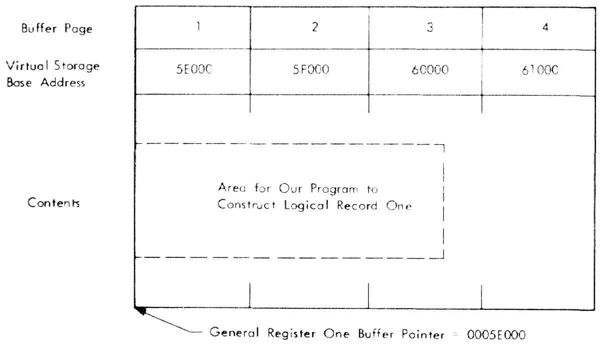
GETMAIN (see Virtual Memory Allocation) is used to obtain a virtual storage buffer area whose size is, in general, dependent on the mode of the PUT macro used and on the record length. The buffer consists of an integral number of pages and controls the amount of virtual storage used by the data set. Because the buffer is obtained by the GETMAIN macro, the External Page Table (XPT) entries for the buffer pages are initially zeros. In this example, each logical record is 11,000 bytes long. The buffer is four pages long because the logical records can cross page boundaries and thus require an overflow buffer page.

When the first PUT (Locate mode) is given, a Type I linkage is made to the VSAM PUT routine which will return, in register one, a pointer to the buffer. In this example, it is assumed to be virtual storage address 0005E000 hexadecimal. (See Figure 44.)

In this case, the program is creating information in the 11,000 byte logical record. It is the program's responsibility, when using Locate mode PUT macro instructions, to place the master information in the buffer. This is done utilizing the Move Characters machine instruction.

Invisible to the program, a page relocation exception is caused the first time each buffer page is referenced. Because the buffer pages were just obtained by GETMAIN, the Resident Supervisor directly assigns a zeroed out main storage page to the task, and no page-in operation is necessary. (See "Main Storage Allocation").

On the occurrence of the second PUT (Locate mode) the buffer is full and the VAM FLUSHBUF routine will be invoked to arrange to have the first two buffer pages processed before returning control to the PUT routine. The third buffer page will be held, because it is only partially full. An overview of this processing is schematically described in Figure 45. A more detailed schematic of the VAM routines involved in this processing is presented in Figure 46. This schematic emphasizes the levels of linkage involved. These routines have additional responsibilities for setting and releasing interlocks when a data set is sharable. However, these functions are not described here.



Buffer Page
1
2
3
4
Virtual Storage Base Address
5E00C
5F000
60000
6100C
Contents
Area for Our Program to Construct Logical Record One
General Register One Buffer Pointer = 0005E000

Figure 44. VSAM Buffer Page After Processing of First Locate Mode PUT

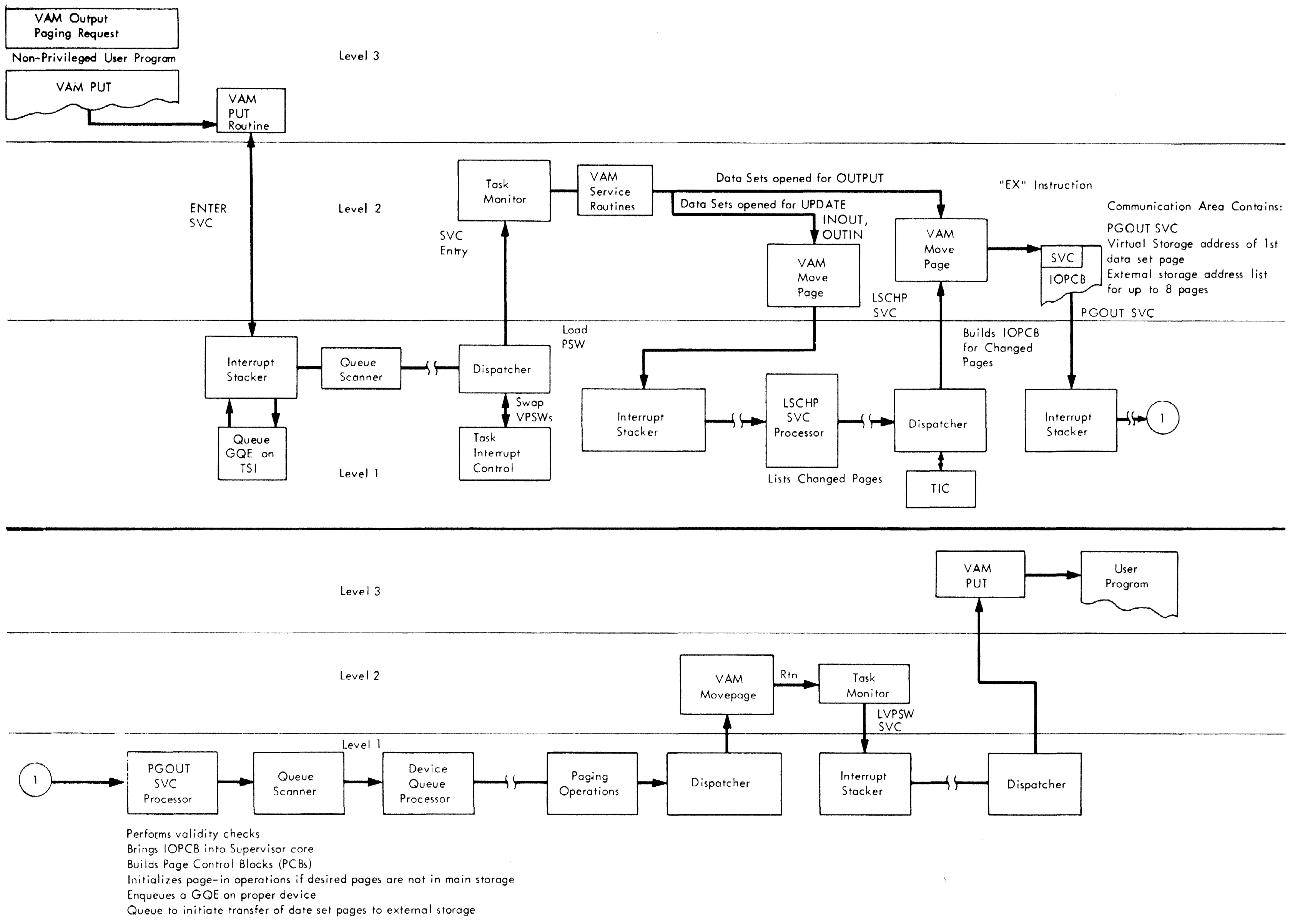


Figure 45. VSAM Processing

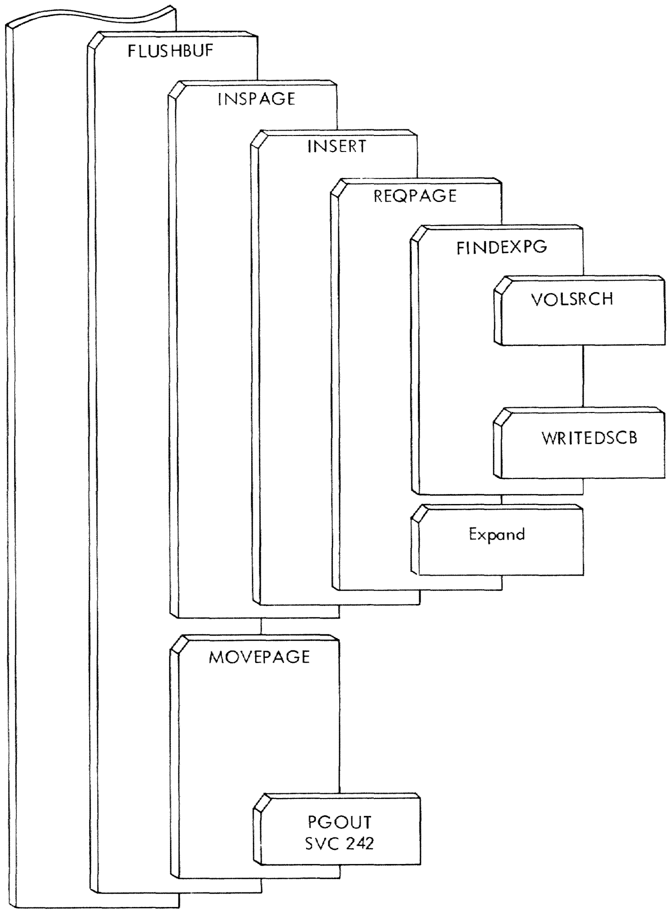


Figure 46. Schematic of VAM Routines

FLUSHBUF will call Insert Page/Delete Page (INSPAGE/DELPAGE) to supervise the creation of data set relative page entries for these two pages. After determining that the request is legitimate, INSPAGE will call Insert (INSERT).

INSERT will shift the RESTBL entries if pages are being inserted into a data set as opposed to being placed at the end of the data set. The former would be the case if the program were adding data set pages to a member of a VPAM data set. In this case, the program is placing the records at the end of the data set (PUT defines the end of a data set) so INSERT merely calls Request Page (REQPAGE).

REQPAGE determines if the pages can be assigned from the not in use RESTBL entries created from the primary space allocation DSCB. In this case, this request is easily met and REQPAGE merely changes the RESTBL entry flag for the two pages to an intermediate setting that indicates that these two pages are now in use and may have been written on. The RESTBL entries thus flagged are later processed by the MOVEPAGE routine. If the program could not satisfy the request from the DSCB extents currently assigned to the data set, REQPAGE would have called the External Storage Allocation routine named EXTEND to obtain a secondary allocation of external storage on some direct access device (see External Storage Allocation). If the RESTBL should then not have room to hold these new entries reflecting the newly allocated external storage, REQPAGE would call Virtual Memory Allocation at EXPAND to move the RESTBL into a larger block of Virtual Memory (see "Virtual Memory Allocation"). However, the request was satisfied from the current extent. The Move Page (MOVEPAGE) routine is invoked next. A flow of the entire processing is shown in Figure 46.

For data sets opened for UPDATE, INOUT, and OUTIN, MOVEPAGE implements one of the more important principles of VAM; namely, that no page will be written back out to external storage unless the user has updated that page. For data sets opened for OUTPUT, as in our case, this test for the presence of updated data is bypassed, as the assumption is made that all pages destined for output contain data.

MOVEPAGE constructs a small communication block, called an I/O Page Control Block (IOPCB), which contains a Page Out (PGOUT) Supervisor Call followed by a parameter list containing the virtual storage and external storage addresses of the buffer pages to be transferred to external storage. This communication block is somewhat analogous to the I/O Request Control Block (IORCB) used by other access methods. However, the IOPCB does not contain any channel program or buffer space. The paging procedures of the Resident Supervisor are to be used to write the two data set pages onto external storage.

After the PGOUT SVC is issued, control will eventually be passed to the Page Out SVC processor of the Resident Supervisor. The Page Out SVC processor will copy the IOPCB parameter list into supervisor storage and determine if the two buffer pages are in main storage or on auxiliary storage. The buffer pages could be on auxiliary storage if the task completed its time slice sometime between the time the buffer pages were written into and the time MOVEPAGE issued the PGOUT SVC.

If the pages are not in main storage, they will be paged into main storage and then paged from main storage to external storage. The action taken by the Resident Supervisor to perform paging operations is discussed in "Paging."

Sometime after the paging is complete, the Dispatcher will return control back to MOVEPAGE.

In this example, the Resident Supervisor encountered a permanent error when attempting to write out the first buffer page. The Resident Supervisor passes this information back to MOVEPAGE. MOVEPAGE flags the old page in the form "11" and, through the services of REQPAGE, assigns a new external storage page to the data set page. The PGOUT SVC is issued repeatedly until MOVEPAGE is informed that the pages are successfully written.

The buffer has now been almost completely processed. When the PUT routine once more gets control, it will complete the buffer processing by transferring the last section of logical record one to the top of the buffer (using the Move Characters machine instruction). PUT will then return with a buffer pointer in Register one for use in placing the second logical record in the buffer. See Figure 47.

When the program issues the third PUT the remainder of the first logical record and part of the second logical record will be processed. (See Figure 48.) However, this time three buffer pages will be processed and the fourth buffer page will be held because it contains part of the third record.

The part of the logical record contained in the fourth page is moved to the top of the buffer and an updated buffer pointer is returned by PUT.

After placing the third logical record in the buffer, the program decides to CLOSE the data set. The VAM CLOSE and VAM SEQCLOSE routines will be invoked. The remainder of the second logical record and the third logical record are still in the buffer and are automatically processed by the above methods.

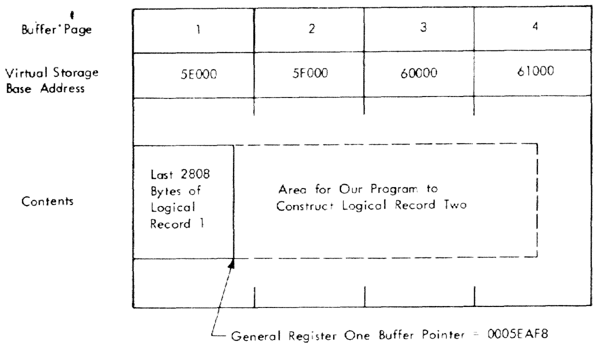

Buffer Page
1
2
3
4
Virtual Storage Base Address
5E000
5F000
60000
61000
Contents
Last 2808 Bytes of Logical Record 1
Area for Our Program to Construct Logical Record Two
General Register One Buffer Pointer = 0005EAF8

Figure 47. Appearance of the Buffer After the Second PUT Macro

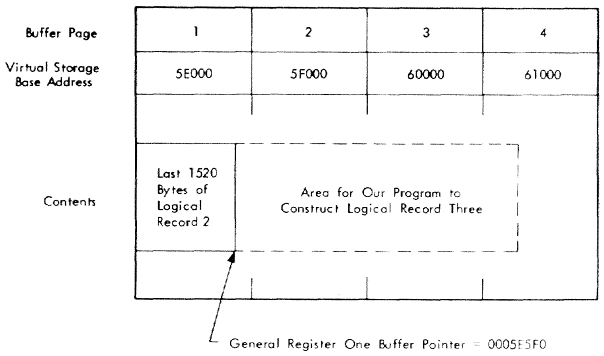

Buffer Page
1
2
3
4
Virtual Storage Base Address
5E000
5F000
60000
61000
Contents
Last 1520 Bytes of Logical Record 2
Area for Our Program to Construct Logical Record Three
General Register One Buffer Pointer = 0005F5F0

Figure 48. Appearance of the Buffer After the Third PUT Macro

Since the program has not specified the DDEF HOLD option, the "in use" pages are used to form new Type E DSCB page entries.

The "not in use" pages are made available in the Page Assignment Table.

The bad page that was encountered disappears. It is dropped from the DSCB and flagged in error in the Page Assignment Table. In this way, it will cause no more trouble.

Had the program specified the HOLD option, the unused pages would occupy space on external storage and both groups would appear in the page entries of a Type E DSCB with the appropriate "Use" flags set. It is assumed that the external storage is located on an IBM 2311 disk pack. Figures 49 and 50 describe the format of records on this disk pack.

The Virtual Access Method CLOSE routine will complete its processing by performing the necessary housekeeping of releasing the RESTBL and restoring the DCB to the state it was in just before OPEN processing, etc.

During a subsequent run a requirement may occur to update the data set by updating the information in the last two thousand bytes of the first logical record. In order to determine which information needs updating, some data located in the second four thousand bytes of the logical record must be read. Thus, that part of the logical record contained in the first buffer page is not read. That portion contained in the second page is read but not updated, and that portion contained in the third buffer page is updated but not read.

Therefore, the UPDATE option would be used in the OPEN macro instruction and the Locate mode GET macro instruction would be used to obtain the first record.

GET processing can be considered a reverse analogy to PUT processing.

IPL Records

Cylinder 0
Track 0
Record 1, - 24 Bytes, Record 2, 144 Bytes

Standard Volume Label

Cylinder 0
Track 0
Record 3, 80 Bytes

Volume Table of Contents (VTOC)

Cylinder 0
Tracks 1 to 9
Record 1 to End-of-Cylinder - 140 Byte Data Set Control Blocks
Room for Approx 170 DSCB's.

Page-size Physical Records

Cylinders 1 to 202
With 8 Pages per Cylinder = 1616 pages on Disk

Figure 49. VAM Format for the IBM 2311

The program is assigned a three page buffer because the largest logical record spans close to three pages.

When the first GET is issued, the VAM service routines will obtain the external storage page numbers assigned to the data set pages and will use the Set External Page Table Entry (SETXP) Supervisor Call to instruct the Resident Supervisor to place these values in the External Page Table (XPT) entries which map the buffer pages. At this point in time, no main storage has been allocated to the buffer and the logical record has not been read from external storage (see Figure 51).

The program now proceeds to update the first logical record. As the program references the second and third pages of the logical record buffer, page relocation exception interruptions occur. The Resident Supervisor processes these interruptions by causing the second and third pages of the data set to be read into main storage from external storage. The first page will not be read into main storage because it was not referenced.

When the program has finished updating the logical record, it executes a PUTX macro instruction. When this happens, the processing is similar to the processing performed upon creating the data set.

If a CLOSE macro instruction were issued at this time, the system would then write out this page. However, if the program were to issue a second locate mode GET, the contents of this buffer page would be written into the first buffer page, and the External Page Table (XPT) entries mapping the buffer would be changed so that the buffer will map the external storage pages containing the second logical record.

For purposes of exposition, consider that no further processing is done on the data set and that this VAM processing example is completed.

Cylinder 3 Track No.	Data Bytes/ Record	Record Identification CCHHR	External Storage Page No.	Data-Set Page No.
0	3625	CC001	25	
1	471	CC011		Not
1	3069	CC012	26	Assigned
2	1027	CC021		to
2	2486	CC022	27	Data Set
3	1610	CC031		
3	1875	CC032	28	
4	2221	CC041		
	(1234 Unused Bytes on Track 4)			
5	3625	CC051	29	Bad Page
6	471	CC061		
6	3069	CC062	30	1
7	1027	CC071		
7	2486	CC072	31	2
8	1610	CC081		
8	1875	CC082	32	3
9	2221	CC091		
	(1234 Unused Bytes on Track 9)			

Figure 50. Track Formats for Page-Size Records on Symbolic Devices No. 7 -- 2311 Disk Pack

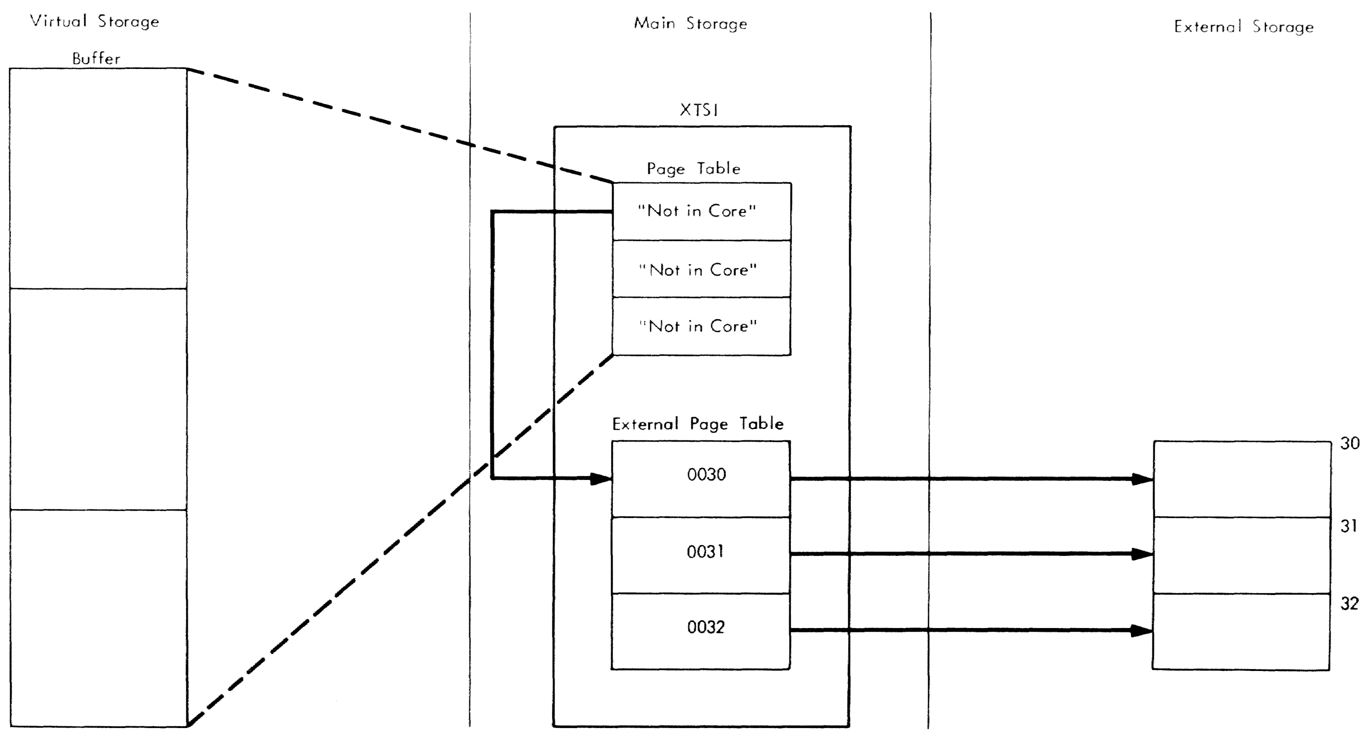


Figure 51. Relationship Between Virtual Storage Buffer and External Storage

PART II: EXTENDED SYSTEM DESCRIPTION

Because of the potentially large number of users concurrently operating on the system and the limited availability of resources, it is desirable and, in some cases necessary, to limit the amount of a given resource which a user may have at his disposal at any given time. The system exercises control over the user's access to these resources by means of the RCR macro instruction, a privileged macro instruction, and a portion of the user table entry (TSS*****.SYSUSE) which indicates the maximum amount of a resource which is available to the user at any one time.

Within the data set TSS*****.SYSLIB) is an indexed sequential member called the User Limits Table data set (SYSULT). This member contains sets of parameters which list the maximum amount of each resource which is to be allowed to each user assigned to that set of parameters. Each set of parameters or entry is a 64 byte long keyed record and contains these resource limits:

1. Maximum CPU time -- this is the total amount of CPU time allowed the user for all tasks he runs during one accounting period.
2. Maximum connect time -- the total terminal time allowed for all the user's conversational tasks during one accounting period.
3. Task count -- the total number of non-conversational tasks a user may execute concurrently.
4. Maximum auxiliary storage allowed the user at a given time.
5. Total pages of temporary storage allowed a user at a time.
6. Total pages of permanent storage allowed the user during one accounting period.
7. Total number of direct access devices that may be allocated to a user at one time.
8. Total number of tape drives allowed a user at one time.
9. Total number of high speed printers allowed a user at one time.
10. Total number of reader-punches allowed a user at one time.

Several sets of parameters may exist in the User Limits Table for any installation and each user is assigned one set or entry at JOIN time based on the key in the JOIN command. The entry assigned to the user becomes a part of his User Table entry and governs the allocation of the various resources during his execution time.

In addition to these maximum limits or rations, the user table contains allocation count fields which are maintained with the current total allocation of each resource for all the user's tasks. Note that for devices, this represents, for example, the total number of tape drives the user has assigned to all his active tasks. The permanent storage field represents the total number of external storage pages allocated to the user's data sets. These totals may not exceed the maximum limits indicated in the ration fields. For other resouces such as CPU time, this count field contains the accumulated time since the last accounting period. The product fields contain the accumulated time product that a resource has been used by the user during the accounting period. The length of these accounting periods varies from one installation to another.

When a user logs on the system, his user table entry is located and read into shared virtual storage. If he already has a task active in the system, the new task will simply be connected to the user table in shared virtual storage. Each task in the system is also assigned an Active User List entry. This entry is analogous to the device count fields of the User Table entry but keeps separate counts of device allocation for each of the user's tasks. The device counts in the Active User List entry enable the system to maintain proper counts of the devices allocated to all of a user's tasks when one of the tasks is abnormally terminated. Under certain conditions it is possible that the count of devices allocated to the user's tasks will not be updated to reflect the release of the devices used by that task being terminated. If this occurs, ABEND will find a nonzero count in one or more fields in the Active User List entry for the task. ABEND can then decrement the corresponding count in the User Table entry so that the user will not continue to be charged for the device.

For accounting purposes, a count is also kept of the total time each resource is used. The time is accounted for in units of resource seconds. That is, if a user has access to three tape drives for six minutes he is charged with 1080 resource seconds for the devices. CPU time is accounted for in milliseconds.

Allocation control and accounting is accomplished by means of the privileged macro instruction RCR which is executed by the system routines which actually allocate resources. This instruction operates in six distinct modes which are:

1. OPEN
2. CLOSE
3. UPDATE
4. RATION
5. VACATE
6. LOGOFF

The OPEN mode of RCR finds the User Table entry or provides one for the user in shared virtual storage. If the user already has an active task, the entry exists in shared virtual storage and need only be found; if there is no active task, the entry must be retrieved from the User Table data set and read into shared virtual storage.

The CLOSE mode updates the User Table entry to reflect the resource usage, writes the entry from shared virtual storage to the User Table, and either disconnects the task from the entry if other tasks are active or releases the shared virtual storage if no other tasks are active.

The UPDATE mode calculates the products which are added to the accounting fields in the User Table. These products are the resource second counts described above. The RCR macro instruction in the UPDATE mode is used by the CLOSE mode prior to updating the User Table entry.

The RATION mode is used to determine the right of the user to the requested resource and, if he is entitled to it, adds the new allocation to the total current allocation in the User Table entry and, in the case of private devices, to the Active User List entry. The determination of the right of the user to a resource consists of adding the new request to the total current allocation of that resource and comparing the new total to the maximum allowable limit. If the new allocation would exceed the maximum allowed, control is transferred to a specified error routine.

The VACATE mode is used to decrement the count of resources currently allocated to the user and the task. The count fields in the User Table entry and, for private devices, in the Active User List entry are updated by this mode.

The LOGOFF mode extracts information from the XTSI for statistical analysis and writes it to the System Log. The information which may be extracted is:

- The number of TWAITs issued
- The number of AWAITs issued
- The number of time slices used
- The number of page-in operations from auxiliary storage
- The number of page-in operations from external storage
- The number of page-out operations to auxiliary storage
- The number of page-out operations to external storage
- The maximum number of pages used on auxiliary disk

After extracting the information, the LOGOFF mode updates the total CPU time and, for conversational tasks, the total terminal time, in the User Table entry and decrements the device allocations in the User Table entry and the Active User List entry. This latter function is performed by issuing the RCR macro instruction in the VACATE mode. Finally, the RCR CLOSE mode is issued to update the User Table entry.

The USAGE command and macro instruction are provided to enable a user to read the data on system resources. The USAGE command writes this information to SYSOUT while the macro instruction must be provided with an area into which the data may be read. A nonprivileged user may only read data pertaining to his own tasks but a privileged user has access to all such data in the system.

This command and macro instruction may be used for accounting purposes or, for an individual user, to determine the advisability of requesting more of a given resource.

An additional command, the UPDTUSER command, is provided for privileged users. This command enables the user to update the count of external storage pages in use by all users. The information concerning usage is extracted from the format E DSCB for each data set and is used to update the page counts in the User Table for each user.

Scheduling Overview and the Schedule Table

The scheduling of jobs in a time sharing system is somewhat more complex than in a simple batch-processing system. Numerous good arguments can be advanced for various scheduling algorithms. Ultimately, only continued use of the time sharing system in an installation's own environment will determine what scheduling algorithm is best adapted to the particular installation.

The scheduling of tasks in TSS/360 is governed by a group of rules, which constitute the scheduling algorithm, and several sets of parameters known as the schedule table. Each task in the system is assigned a set of parameters based on the user's priority (USEPRI) and the task type (batch or conversational). Various components of the Resident Supervisor make use of these parameters and rules to determine the frequency with which the task executes and the duration of each execution. The set of parameters may be varied according to the amount of paging or execution a task does but the rules remain constant. A given installation may achieve optimum system performance by the judicious alteration of parameters based on an analysis of its own job mix and the quality of performance achieved with previously tested sets of parameters.

The ultimate purpose of the TSS/360 scheduling algorithm is to select a task to be placed in execution in an available CPU. The algorithm attempts to allocate CPU time to tasks in the most efficient manner as defined by the algorithm.

The schedule table consists of a variable number (256 maximum). of 28-byte schedule table entries. Each entry consists of 26 parameters which govern the frequency and length of time a task is given a CPU. Figure 52 depicts the format of the schedule table entry. The figure and the following description detail the contents of the schedule table and the function of each parameter. For the current format of each entry see <u>System Control Blocks</u> PLM.

<u>Level</u>: This field indicates the relative location of the given schedule table entry (STE) within the schedule table. The first entry is assigned level zero and the level is incremented by one for each succeeding entry. This field is the level number to which other fields, such as Pulse level and Time Slice End (TSE) level, refer but the level plays no part in the scheduling of tasks.

<u>Priority</u>: This field contains the priority assigned to all tasks using this entry as its set of scheduling parameters. The priority associated with each STE, as well as all remaining parameters, is specified by the system administrator in the schedule table CSECT which is loaded at STARTUP. It may have any value from 0-255. The priority determines the position a task assumes within a list of eligible tasks; low priority numbers are given preference over higher priority numbers.

<u>Quanta Count and Quantum Length</u>: These two parameters determine the duration of the time slice for tasks assigned to this entry. Time slice duration = Quanta count X Quantum Length X 3.33 milliseconds. Each time a task is placed in execution, the value Quantum length X 3.33 is computed and added to the current time. A timer interrupt is then created which will cause the task to be interrupted and examined. This procedure is repeated the number of times indicated in Quanta count after which the task will be brought to time slice end.

<u>Maximum Pages Allowed</u>: This field contains a count of the maximum number of pages a task may have in main storage at one time (maxcr). If the task exceeds this value, it is forced to time slice end or page stealing occurs.

1 byte	1 byte	2 bytes	1 byte	1 byte	2 bytes	2 bytes	2 bytes	1 byte	1 byte	2 bytes	1 byte	1 byte	1 byte
Level (STELEVEL)	Priority (STEPRIOR)	Quantum Length (STESVAL)	Max Quanta Count (STEQUANT)	(DTR) Delta to Run (STEDELTA)	Max Relocations Per Quantum (STEMRQ)	Max Pages Allowed (STEMAXCR)	Max Disc I/O (STEMAXRD)	Scan Threshold (STEST)	Pulse Level (STEPULSE)	AWAIT Extension (STEAWTEX)	Time Slice End Level (STETSEND)	Max Pages TSE Level (STEMPRE)	AWAIT Level (STEAWAIT)

1 byte	1 byte				1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	2 bytes
	1 bit	1 bit	1 bit	1 bit								
TWAIT Level (STETWAIT)	Recompute Flag (STERCMP)	Preemot Flag (STEPRMPT)	Steal Request (STESRI)	Subtract DTR (STESDTR)	Holding Interlock Level (STEHLCK)	Low Core Holding Interlock Level (STELCHL)	Waiting on Interlock Level (STEWLCK)	Conversational Write Only Level (STECWO)	Low Core Force TSE Level (STELCF)	Maximum behind Schedule 6.7 seconds (STEMBS)	Next Steal Level (STENSL)	Drum Share (STEDSH)

Figure 52. Contents of the Schedule Table Entry

Maximum Disk I/O: This field represents the maximum disk reads or writes a task will be allowed before a time-slice end will occur. Reads and writes as a result of migration, purging shared pages, or time-slice ends do not count toward this limit.

Scan Threshold: If the Steal Request Flag is on, the Resident Supervisor will release some of a task's pages when the page count equals MAXCR. The Scan Threshold is the percentage of MAXCR pages to be retained. This percentage is specified in hex (80%= 80=X'50'). When stealing occurs, the task is not time sliced, but stays in the dispatchable list. However, the Schedule Table Entry in the TSI is changed to the value in the Next Steal Level field.

Pulse Level: This field contains a level number which will be assigned to a user when he issues the PULSE macro instruction. It may be determined at an installation that one group of parameters provides maximum throughput for an I/O-bound task but that a second set is better for compute-bound tasks. The user can be given the facility to select the best set of parameters by use of the PULSE macro instruction each time the processing characteristics of his task change. An installation may deprive certain classes of users this facility by setting the Pulse Level field equal to the Level field thereby locking him into one schedule table entry.

AWAIT Extension: This field contains a value which represents a period of time a task may remain in the list of dispatchable tasks while awaiting the completion of an I/O operation. A task in AWAIT status is normally removed from the list of dispatchable tasks. Since this causes a delay in redispatching the task, it may be desirable to permit high priority tasks performing I/O on a high speed device to remain on the dispatchable list. This can be accomplished by making the AWAIT extension large enough to allow for the completion of the I/O operation.

Delta to Run (DTR): This field contains a factor which is used in calculating a new scheduled start time for a task as it moves from one state to another (i.e., as the task becomes ready, in AWAIT, in TWAIT). The value in this field is multiplied by 852.5 milliseconds and may be combined with the master clock or the old scheduled start time (SST) if old SST is negative to determine the task's new SST. These other factors and the manner in which they are combined will be discussed later. If this field is zero, tasks are serviced on a LIFO (last in-first out) queue within the priority level.

TSE Level: This field represents the schedule table level entry to be used when a time-slice end occurs because of the maximum number of quanta or maximum disk I/O being reached.

Maximum Pages TSE Level: This field contains the level number of the STE which will be assigned to the task when the number of pages the task has in main storage exceeds the value contained in the Maximum Pages Allowed field.

TWAIT Level and AWAIT Level: These fields contain the level numbers of the STEs which will be assigned to the task when it leaves TWAIT or AWAIT status respectively.

Recompute Flag: If the Recompute byte is X'80' and a task enters the eligible list, then:

SST = PSAETM + STEDELTA

where SST = Scheduled Start Time

PSAETM = Current Clock Value

STEDELTA = Delta-to-Run from Schedule Table

If the Recompute byte is X'00' and a task is coming from the inactive list to the eligible list, then:

SST = PSAETM + STEDELTA - (amount behind schedule)

or

SST = PSAETM + STEDELTA , if ahead of schedule

Preempt Flag: If the byte value is X'40' for a task in the dispatchable list, and a behind schedule task of higher priority resides in the eligible list, the task in the dispatchable list can be preempted by forcing it prematurely to time slice end.

Steal Request Flag: A task on the dispatchable list with this flag set (X'20'), will have pages released when its private pages in core reach the MAXCR limit. If pages are brought in faster than they can be released so that the MAXCR limit is exceeded, the task will be time sliced.

Subtract DTR Flag: If this flag (SDESDTR) is set (X'10') when the task is being placed in the eligible list, delta-to-run (STEDELTA) is subtracted from Master Clock (PSAETM) in the calculation which sets the Scheduled Start Time. This allows FIFO (first in, first out) ordering, with a negative DTR.

Maximum Page Relocations per Quantum: The value contained in this field is the threshold value which is used to distinguish between execute-bound and paging-bound tasks. If a task has more than this number of page relocations in one quantum it is paging-bound, otherwise it is execute-bound.

Holding Interlock Level and Low Core/ Holding Interlock Level: These two fields contain the level number of the STE which is assigned to a task which has placed an interlock on a shared data set or data set page when being time-sliced under normal or low core conditions respectively. The purpose of changing the level under these conditions is to expedite the release of the interlock.

Waiting on Interlock Level: This field contains the level number of the STE which is assigned to a task which is waiting for the release of an interlock placed on shared data by another task. This change in level enables a task to check the interlock from a level of lower priority than the task which holds the interlock.

Conversational Write Only: This field represents the Schedule Table level to be used when a write without response message is sent to the terminal. The level change occurs without a time slice end.

Low Core Forced Time Slice End: This field represents the Schedule Table Entry to be used when a task is forced to time slice end for low core and it is not holding an interlock.

Next Steal Level: This field represents the Schedule Table Entry to be used when stealing (releasing of pages) occurs. The task is not time sliced.

Drum Share: This is the number of drum pages reserved for a task. There are about 500 pages available after startup in a one drum system, and 1400 pages in a two drum system. In general, the number of a task's private pages on drum is a function of the number of tasks logged on, the number of drums, and the time from last time-slice. If the number of unassigned drum pages falls below a predetermined limit, some pages are moved from drum to disk. Each task receives a system calculated minimum drum space. The Drum Share field allows a task to keep a larger drum share. A zero value defaults to the system calculated minimum.

Task Scheduling: The rescheduling of a task has a precedence order for assigning Schedule Table exit. Some of the following can occur simultaneously, some are mutually exclusive:

1. TWAIT
2. AWAIT
3. Low core holding interlock
4. Holding interlock
5. Low core
6. Waiting on interlock
7. Maximum disk I/O
8. Maximum pages in core
9. Maximum quanta
10. Preempt

Conversational write-only and Next Steal Level are two fields which provide a level change without time slicing. In both cases, the task remains in the dispatchable list until time sliced for some cause listed above.

Each task which enters the system is represented by a Task Status Index (TSI). These TSIs remain resident in main storage and contain the nucleus of information required to keep track of the task and its current status in the system. Among other things, the TSI contains the task identification number (TID), a pointer to the Extended Task Status Index (XTSI), which contains more voluminous task information, the task's current schedule table entry (in the field TSISTE), and the task's scheduled start time (TSISST). These latter two fields contain the information required to schedule and dispatch the task.

The Active and Inactive Lists

All TSIs in the system are chained together on one of two lists (see Figure 53), the Active list and the Inactive list. The Active list is subdivided into the Dispatchable list and the Eligible list. The Dispatchable list consists of tasks which are in main storage attempting to compete for CPU time and, in most cases, whose scheduled start time is less than the master clock. (The master clock (MC) is defined as bytes 3-6 of PSAETM.) When the SST of a task is less than the Master Clock, the task is said to be behind schedule. Under certain circumstances it is possible for a task which is not behind schedule to be placed on the Dispatchable list. This situation arises when there are no behind schedule tasks awaiting entry to the Dispatchable list but there is room in main storage for another task or when the number of tasks on the Dispatchable list is below the system minimum.

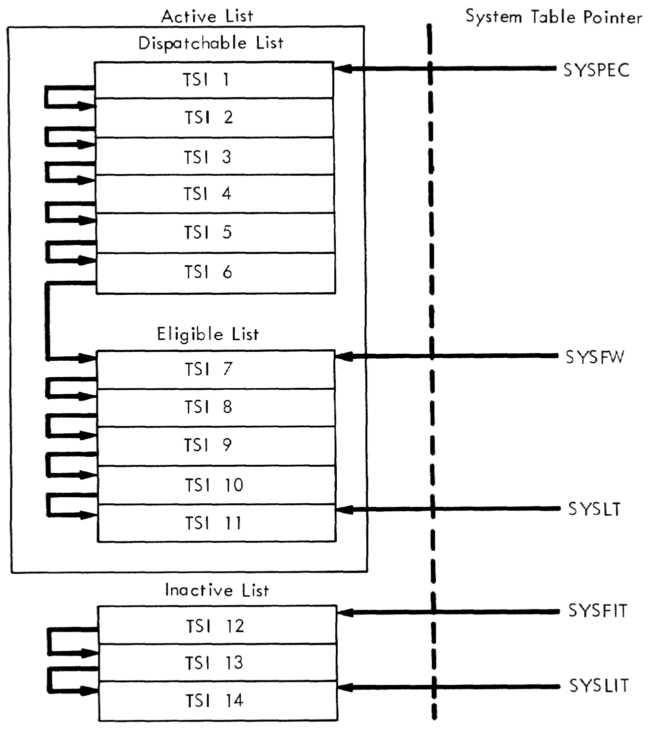


Figure 53. TSI Lists

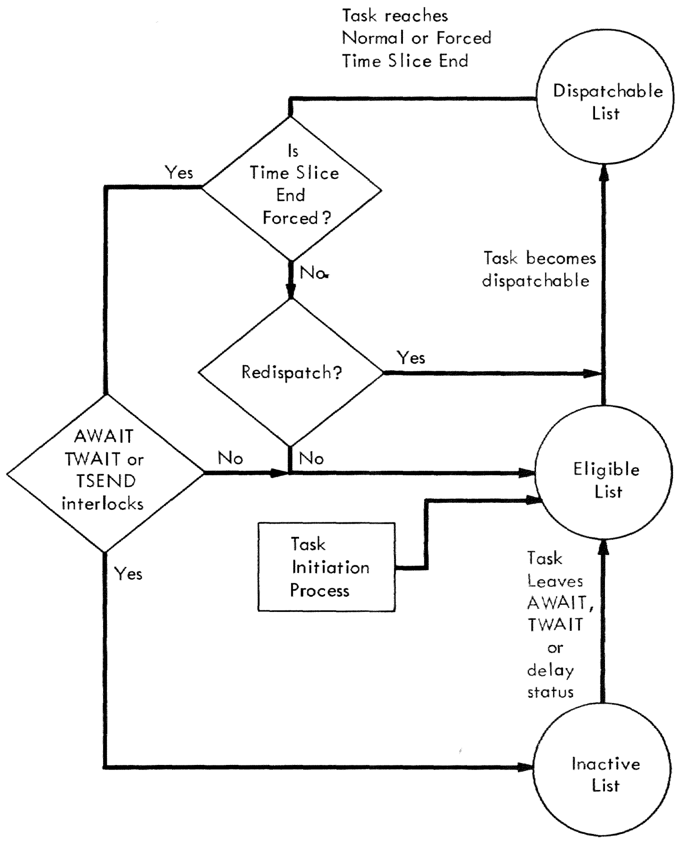
The Eligible list consists of tasks which are ready to execute but have not yet been brought into main storage. Tasks on the Eligible list are ordered by priority with the lowest priority number first on the list. Tasks with the same priority number are ordered by SST with tasks furthest behind schedule (i.e., lowest SST) having priority. Tasks on the Eligible list are moved to the Dispatchable list when conditions permit and so become candidates for use of a CPU. The manner in which this is done will be discussed later.

The Inactive list consists of tasks which are in AWAIT or TWAIT status or have issued a TSEND SVC. These tasks are incapable of continuing execution until a particular interruption occurs. When the awaited interruption occurs, the task is moved from the Inactive list to the Eligible list in its proper order as explained above. Figure 54 depicts the movement of tasks among these three lists.

Task Scheduling

When a task first enters the system it is assigned a schedule table entry (STE) which is the Level number. This value is stored in the task's TSI (field TSISTE). The Bulk I/O task is assigned STE level 10 and all other tasks are assigned to the level which equals the decimal sum of their user priority (USEPRI) and either 0 for conversational tasks or 10 for nonconversational tasks. Note that USEPRI cannot be 0 since this would result in the task being assigned the STE reserved for the operator task or the BULKIO task. The new task is then filed in the Eligible list on the basis of its priority (as shown in its STE) and its scheduled start time (SST). All newly created tasks are given an SST equal to the sum of the master clock (bytes 3-6 of PSAETM and the delta to run (DTR) parameter from the STE. If the DTR is 0, the SST will be 0 regardless of the master clock. The manner in which this is accomplished is described in the section "Examples of System Operation -- Creation of a Conversational Task."

When the Queue Scanner finds that there is no work which it can perform, it calls the Internal Scheduler. This module is responsible for moving tasks from the Eligible list to the Dispatchable list and maintaining the Dispatchable list in its proper order. In this case, the Internal Scheduler starts at the beginning of the eligible list (SYSFW) and scans all TSIs looking for behind-schedule tasks (i.e., tasks for which SST<MC). As each task is examined, the system table (SYSLSST) is updated to reflect the lowest SST ahead of schedule (i.e., the lowest SST>MC).



•Figure 54. Maintenance of TSI Lists

Once a task falls behind schedule, there is a limit on how long it must wait to be dispatched. When it exceeds this "maximum-behind-schedule" value (computed from STEMBS), it will be submitted to the Entrance Criteria module regardless of its normal priority. If no task has exceeded maximum-behind-schedule, the highest-priority behind-schedule task is submitted to the Entrance Criteria module which determines whether or not the task may actually be placed on the Dispatchable list. If the task in question is waiting for the completion of a paging operation, it cannot be placed on the Dispatchable list. If the task is ready to execute, a comparison is made between the pages used last time slice (TSIPTS) and the system estimate of available core blocks (SYSECB). If the comparison shows that room exists in main storage, the task is placed on the Dispatchable list. If the comparison shows that main storage space does not exist but there are fewer than the system minimum number of tasks on the Dispatchable list, the results of the comparison are ignored and the task is moved to the Dispatchable list.

The Internal Scheduler continues to scan the Eligible list looking for behind schedule tasks. As each such task is found, its TSI is placed on the chain of Dispatchable TSIs ahead of all TSIs belonging to execute-bound tasks, and behind the TSIs of tasks in page wait, and the task's XTSI page is brought into main storage. If the task's current SST=0, it remains 0. If the task's SST≠0, a new SST is computed as the difference between old SST and MC. The scan of the Eligible list continues until the Entrance Criteria module rejects a task or until the end of the list is reached.

In the latter case the scan begins again at the beginning of the Eligible list and each task is submitted to the Entrance Criteria module regardless of its SST. This second scan continues until a task is rejected by Entrance Criteria or until the Eligible list is exhausted.

When a task is rejected by Entrance Criteria before the first scan of the Eligible list is completed, the task is flagged as the first task to be considered the next time the Internal Scheduler is entered and the Dispatchable list is searched for a lower priority task with its STE preempt flag on. When such a task is found, time slice end is forced on it, the task is filed in the Eligible list, and its pages are removed from main storage. This ensures that there will be room when the task just rejected is resubmitted to Entrance Criteria.

If a task is rejected during the second scan, no attempt is made to add more tasks to the Dispatchable list and the first phase of internal scheduling terminates.

During the first scan, all possible behind-schedule tasks have been placed in the Dispatchable list. If a behind-schedule task is rejected, it receives preference during the next pass through the Internal Scheduler because it will be the first task checked. Tasks following the last rejected task will be examined next followed by tasks at the head of the Eligible list. This first phase of the Internal Scheduler places the maximum possible number of tasks on the Dispatchable list thereby decreasing the possibility of the Dispatcher running out of dispatchable tasks.

The second phase of the Internal Scheduler is devoted to ordering tasks in the dispatchable list. Tasks on this list are classified as "paging-bound" or "execute-bound." Paging-bound tasks are those which, in one quantum, cause more page relocation exceptions than they are allowed, as indicated in the field Maximum Page Relocation Exceptions per Quantum in the STE. Tasks which do not exceed this limit are classified as execute-bound. The second phase of the Internal Scheduler moves all execute-bound tasks to the end of the Dispatchable list. This ordering of tasks on the Dispatchable list improves multiprogramming by causing tasks with high paging requirements to be dispatched first. This increases the overlap of CPU and channel operations. When the Internal Scheduler concludes its work it exits to the Dispatcher.

Application of the Scheduling Algorithm

As previously stated, the scheduling algorithm consists primarily of a group of formulae and a set of parameters contained in the schedule table entry. In addition, certain system constants, which may be varied when the schedule table is constructed, are used to govern the length and frequency of a task's time slice. The STE parameters which are used to govern a task's scheduling are priority, delta-to-run and recompute. The various level fields in each entry dictate changes in the STE assigned to a task because of certain characteristics of its execution, such as excessive use of main storage, excessive paging, and performance of I/O. The remainder of the parameters are maximum values permitted for certain operations.

The existence of level change fields in the STE makes it possible for an installation to vary the scheduling algorithm for a task as its performance characteristics dictate. The assignment of a schedule table entry on the basis of task priority and task type has the effect of creating several scheduling algorithms for tasks entering the system. An installation has the capability of constructing a similar number of sets of variable scheduling algorithms based on a task's priority. For example, conversational tasks with user priority of 3, are assigned STE 3 on entry to the system. The installation may reserve entries 30 through 39 for this type of task. By setting all level fields in STE 3 to point to a level in the range 30-39 and levels in the range to point to level 3 or another level in the range, these tasks are locked into the eleven entries 3 and 30 through 39. Once this is done, scheduling of tasks of one type can be controlled in one manner and tasks of other types in other manners. High priority, conversational tasks might be allowed more main storage each time slice than nonconversational tasks and penalized less when they exceed it. The following paragraphs list the formulae which are used in calculating the task's SST and the routines which use them.

When a task first enters the system, the Task Initiation routine calls the Rescheduling routine which enters the task in Delay status and sets SST=0. When Task Initiation regains control, it assigns STE 20 to the new task. This entry remains in effect until the LOGON process is completed and governs task activity in the early stages of its existence.

The Channel Interrupt Processor calls the Queue-GQE-on-TSI routine to queue an asynchronous I/O interruption on the task's TSI. When this interruption is processed by the Task Monitor, the Command System will be invoked to complete the task initiation processing. When Queue-GQE-on-TSI is called, it finds the task on the Inactive list and calls the Rescheduling routine.

The Rescheduling routine recomputes the SST using one of three formulae:

1. In all cases where DTR=0, SST is set to 0.

2. If DTR $\neq$ 0 and the old SST<0, the new SST=DTR+MC+(1-R)SST_0 where R is 1 if the Recompute flag is on and 0 if it is off and SST_0 is the old scheduled start time.

3. If DTR$\neq$0 and $SST_0 \geq 0$, the new SST=DTR+MC.

In the case of this newly created task, formula 3 is used and the new SST is stored in the TSI (TSISST). Rescheduling then places the task on the Eligible list in its proper order according to priority and SST.

At some later point in time the Queue Scanner will find that there is no more work which it can process and will call the Internal Scheduler. Depending on how heavily the system is loaded, the new task may be behind schedule (SST<MC). Eventually, the Internal Scheduler and Entrance Criteria modules determine that the new task should be added to the Dispatchable list. At this time, the Internal Scheduler recalculates the SST according to one of two formulae:

1. If $SST_0 = 0$, $SST_1 = 0$

2. If $SST_0 \neq 0$, $SST_1 = SST_0$-MC

For this new task assume formula 2 is used. If the current system load is light, SST will be greater than MC and SST_1 will have some small positive value. If the system load is heavy, SST_1 will be negative. The Internal Scheduler then adds the new task to the beginning of the Dispatchable list and enters the sort phase of its operation. New tasks are classified as paging-bound (TSIEB=0) and so this task will remain first on the Dispatchable list and be the first to be dispatched when a CPU is available.

The next change which occurs is a change in the task's STE. When the Command System processes the LOGON command, it issues the SCHED macro instruction. This macro instruction assigns STE 10 to the BULKIO task and calculates a new STE for all other tasks by taking the decimal sum of the external priority (USEPRI) and 0 for conversational tasks or 10 for nonconversational tasks. The task operates with this STE until it causes one of the level parameters to be employed to change the level number.

As the task receives time slices and performs various operations its scheduling is affected by further changes to its SST, by changes in the STE level assigned to it, and by the list to which it is moved as various events occur. The following paragraphs list the events which can cause these changes and describe the manner in which the task is affected by each.

Time Slice End (TSE): When the task attempts to exceed its allotted time (quantum length X quanta count), a normal TSE occurs. All other TSE conditions are considered forced.

When normal time slice end occurs, the quanta count (TSIQCT) is decremented by one. This field is initialized to the value in the Quanta count field in the task's STE. If the count has not reached zero, the task is given another quantum of CPU time. The task is left on the Dispatchable list and no change is made to its SST or STE. If the count does reach zero, the Rescheduling routine is called. Rescheduling changes the STE level (TSISTE) to the level indicated in the TSE level field of the old STE and recomputes the SST according to one of three formulae:

1. If DTR = 0, $SST_1 = 0$

2. If DTR ≠ 0 and $SST_0 < 0$, $SST_1 = DTR+MC+SST_0$

3. If DTR ≠ 0 and $SST_0 \geq 0$, $SST_1 = DTR+MC$

Each time the quantum count is decremented, a comparison is made between the number of page relocations which occurred during the quantum and the maximum allowed the task per quantum. If the task has exceeded the maximum, it is classified as paging-bound (TSIEB=0). Since this test is made and the field updated at the end of each quantum, only the paging history of the last quantum is significant.

If time slice end is forced on a task via a TSEND issued for interlocks, the task is placed on the Inactive list and in the delay state for the period of time specified in the system table field SYSDLY. The length of this delay can be altered at an installation in order to achieve better scheduling. The end of the delay is signalled by a timer interruption which results in a call to Queue-GQE-on-TSI. The Rescheduling routine is called to move the task to the Eligible list and the task's SST is recomputed using one of the three formulae listed above in the description of the task initiation procedure.

If a task timer interruption occurs, it is enqueued on the TSI; no change occurs to the STE or SST and the task is filed on the Eligible list.

AWAIT: When a task enters AWAIT status, pending the completion of an I/O operation, the AWAIT extension in the task's STE is used to create a timer interruption for some future time. When the interruption occurs or if the extension is zero, the task is placed on the Inactive list until the I/O operation is completed. If the I/O operation is completed before the timer interruption occurs, the pending interruption is cancelled and the task is allowed to remain on the Dispatchable list.

If the completion of an I/O operation is signalled for a task on the Inactive list, Queue-GQE-on-TSI calls Rescheduling which substitutes the AWAIT level from the current STE in the task's TSI and recomputes the SST according to one of the three formulae used in the task initiation procedure. For a task which has been granted an AWAIT extension, the AWAIT level is also substituted on I/O completion.

TWAIT: Since no TWAIT extension exists, a task which enters TWAIT status is moved directly to the Inactive list. The completion of the operation results in the movement of the task to the Eligible list and the recomputation of the SST in the same manner as for tasks in AWAIT status. The current STE is replaced with the entry indicated in the TWAIT level field of the current STE.

Table 1 summarizes these stimuli and the changes they effect in the scheduling of the task.

Task Scheduling Walkthrough

The following discussion and diagram depict the movement of tasks among the lists of all tasks in the system and the changes in their scheduled start times as various phases of the scheduling algorithm are applied. For the sake of simplicity only three tasks are shown and only three schedule table entries are used. Also, the tests for level changes are only mentioned.

Figure 55 shows the content of the schedule table. The values shown have been arbitrarily chosen and should not be interpreted as recommended or suitable values in any real environment. Note that gaps have been left in the sample schedule table. This cannot be done in an actual working table.

In this example (see Figure 56), task A is the BULKIO task and is assigned STE 10. Since the DTR in this entry is zero, the SST is always zero. Task B is a nonconversational task with user priority 3 and is assigned STE 13. Task C will be shown logging on as a conversational task with user priority 3; it will be assigned STE 3.

In the initial system state, task A is the only task on the Dispatchable list and is in execution. Task B is the only other task in the system and is on the Eligible list, ready, and behind schedule (SST<MC).

The first event which occurs is a timer interruption which occurs to signal the end of task A's first quantum. The interruption is handled by the Resident Supervisor which updates task status information and determines that the task should be given another quantum. The task is left on the dispatchable list and the Internal Scheduler is entered.

The Internal Scheduler finds task B on the Eligible list and moves it to the Dispatchable list. At this point task B's SST is recomputed and set to -2. Assuming that task A has been classified execute bound and task B has been classified paging bound, task B will become first on the Dispatchable list and state 2 in the diagram will exist.

• Table 1. TSI List and Parameter Changes

List	Stimulus	Special Conditions	Move TSI to	Recompute SST	Change STE to
None	Async. Interrupt unowned terminal or CRTSI SVC	None	Inactive list	Do not recompute	Do not change
Inactive	Queue Asynch. Interrupt GQE on TSI	None	Eligible list	SST=0	20
	Channel Interrupt	Task in AWAIT status	Eligible list	If DTR=0, SST=0 If DTR≠0 & SST<0 If DTR≠0 & SST≥0 SST_1=DTR+MC	AWAIT level
Eligible	Task becomes dispatchable	TSI moved by Internal Scheduler	Dispatchable list	If DTR = 0, SST_1= 0 If SST_0=0, SST_1= SST_0-MC	Do not change
Dispatchable	Time slice end	Forced TSE SVC	Inactive list with delay= SYSDLY	Do not recompute	Do not change
		Normal, TSIQCT≠0	Do not move	Do not recompute	Do not change
		Normal, TSIQCT=0	Eligible list	If SST_0< 0, SST_1=DTR+MC+SST_0 If $SST_0 \geq$ 0, SST_1=DTR+MC	TSE level
	LOGON	BULKI/O task	Do not move	Do not recompute	10
		Conversational task	Do not move	Do not recompute	USEPRI
		Nonconversational task	Do not move	Do not recompute	USEPRI+10
	Timer interrupt	AWAIT extension expired	Inactive list	Do not recompute	Do not change
	Task enters AWAIT	AWAIT extension≠0	Do not move	Do not recompute	Do not change
		AWAIT extension=0	Inactive list	Do not recompute	Do not change
	Task enters TWAIT	None	Inactive list	Do not recompute	Do not change

Level	Priority	Quanta Count	Quantum Length	Maximum Page Reads	Pulse Level	Await Extension	DTR	TSE Level	Recompute	Preempt	All Other Levels
03	4	1	20	70	13	0	5	03	0	0	03
≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	
10	5	2	20	70	10	0	0	10	0	0	10
≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	
13	10	4	20	70	13	0	5	13	1	1	13

Task A - STE 10
Task B - STE 13
Task C - STE 3

Figure 55. Sample Schedule Table Entries

	DISPATCHABLE LIST						ELIGIBLE LIST						INACTIVE LIST					
System State	1st on list		2nd on list		3rd on list		1st on list		2nd on list		3rd on list		1st on list		2nd on list		3rd on list	
	Task	SST	Task	SST	Task	SST	Task	SST	Task	SST	Task	SST	Task	SST	Task	SST	Task	SST
Initial MC=100	Task A in execution	0					Task B ready	98										
2 MC=102	Task B ready	-2	Task A ready	0														
3 MC=103	Task B ready	-2	Task A ready	0									Task C					
4 MC=104	Task B ready	-2	Task A ready	0			Task C ready	0										
5 MC=106	Task C page wait	0	Task B ready	-2	Task A ready	0												
6 MC=110	Task C page wait	0	Task A ready	0			Task B ready	113										
7 MC=120	Task A ready	0					Task B ready	113					Task C TWAIT	0				
8 MC=122	Task B ready	-9	Task A ready	0									Task C TWAIT	0				
9 MC=124	Task A ready	0					Task C ready	129					Task B AWAIT	-9				

Figure 56. Scheduling Walkthrough

At this point, an asynchronous interruption is received from an unowned terminal and a new TSI is created and placed on the Inactive list placing the system in state 3. The Channel Interrupt processor enqueues an asynchronous interrupt GQE on the new TSI. Among other things, this results in a call to Rescheduling which places the new TSI on the Eligible list with SST=0 and STE=20 as shown in state 4 in the diagram.

Once this initiation procedure is completed, the Queue Scanner is entered and it passes control to the Internal Scheduler. The Scheduler finds task C on the Eligible list and, after determining that there is room for the task in main storage, places the task at the head of the Dispatchable list. The task is placed in page wait status, its SST remains 0, and a request is enqueued to bring the task's XTSI page into main storage. At the end of this procedure, the system lists are in state 5.

Following the initiation of the paging operation, the Queue Scanner gets control and, finding no work, calls the Internal Scheduler. The Scheduler finds no tasks on the Eligible list and the Dispatcher is called.

The Dispatcher begins at the head of the Dispatchable list looking for a task to place in execution. It bypasses task C which is in page wait and selects task B which is ready. Assuming that the paging operation is not completed, Task B will execute for one quantum. It will then be classified as execute bound and the Queue Scanner, Internal Scheduler, and Dispatcher will be called. The Dispatchable list is rearranged just before the Dispatcher is called and task B, which is now execute bound, is placed at the bottom of the Dispatchable list. Task A will now be placed in execution, and tasks B and A will alternate until one of them has received its full quanta. On the fourth interruption the task will have reached normal time slice end. Assuming this was task B, it will be removed from the Dispatchable list and placed on the Eligible list in ready status. Its SST will be recomputed according to the formula $SST_1=DTR+MC+SST_0=113$. At this point, the system lists are in state 6.

At this point the first XTSI page has been brought into main storage. Other initialization procedures are required but eventually task C is placed in execution and the user issues the LOGON command. Part of the system processing of LOGON is a recomputation of the STE assigned to the task. In this case the new STE is 3 and replaces the 20 assigned to the task initially.

Task C is using this terminal session to run a prestored program. In these initial stages the task goes through several checks and changes which are being omitted from this example for purposes of simplicity. The next event of significance is the occurrence of a TWAIT. When this SVC is issued, the task is placed on the Inactive list with its SST and STE unchanged. The system lists now appear in state 7.

Task A has now worked its way to the top of the Dispatchable list. Before the Dispatcher is entered, the Internal Scheduler receives control and begins scanning the Eligible list. It finds task B which is now ready and behind schedule and moves it to the head of the Dispatchable list (A task is always entered at the head of the dispatchable list.) Task B's SST is recalculated as -9 and the system lists are in state 8.

When the Dispatcher gets control it places task B in execution. Task B performs an I/O operation, followed by an AWAIT which causes task B to be placed on the Inactive list. Task C receives an interruption indicating the completion of the operation for which it was waiting. Task C is moved to the Eligible list and its SST is recalculated at 129. Assuming that no conditions exist which will force task C to be placed on the Dispatchable list, the lists will be in state 9 and task A will again be dispatched.

MAIN STORAGE ALLOCATION

The allocation of main storage is controlled by the following Supervisor routines:

- User Core Allocation Queue Processor.
- Contiguous Core Allocation Queue Processor.
- User Core Release Subroutine.
- Supervisor Core Allocation Subroutine.
- Supervisor Core Release Subroutine.

The allocation of main storage is recorded in a Core Block Table which is created during System Generation and initialized by STARTUP. The Core Block Table contains one entry for each page of main storage in the system. There are three major categories of pages:

- Assigned for user core allocation.
- Assigned to the Resident Supervisor.
- Non-operational pages (i.e., defective or partitioned pages).

STARTUP initializes the Core Block table by testing main storage and marking any partitioned or defective pages as "non-operational". It then assigns the pages occupied by the following to the Resident Supervisor:

- Active Prefix Storage Areas (PSAs).
- SERR/RECONFIGURATION buffers (two pages for each active PSA).
- Resident Supervisor.

In addition, five pages immediately following the Resident Supervisor are assigned to the Resident Supervisor. They constitute a Reserve List and are used as an initial pool of pages for satisfying Resident Supervisor requirements for temporary storage (such as space needed for TSIs, GOEs, and PCBs). All other pages (called User Core) are made available to tasks.

User Storage Allocation

User storage is requested in connection with page-in operations or when any virtual storage page allocated by GETMAIN is referenced. The former case includes any paging-in operation associated with processing the TWAIT and PGOUT SVCs or with paging in XTSI pages. The processing of the TWAIT SVC may require obtaining a 3 page buffer for use in moving pages from drum to disk auxiliary storage (see "Paging").

The two major categories of user storage are <u>in use</u> and <u>not in use</u>.

The Core Block Table (CBT) entries representing not-in-use user storage are organized into three lists:

- Available list -- not currently assigned to a task.
- Pending list -- those pages assigned to tasks that are in Time Slice End, TWAIT or AWAIT status.
- Preferred Page-Pending list -- those pages assigned to tasks in Time Slice End, TWAIT or AWAIT, that a task must have in main storage to begin executing again when the task next becomes active. Currently this is defined as consisting of only the particular page in which a task was executing when interrupted (PSW page) and those XTSI pages which are currently necessary for task processing.

It should be noted that a page is not returned to the not-in-use lists until all user I/O or paging operations on it are completed. In addition, a task's XTSI pages are not returned until all paging or user I/O is completed on the rest of the task's pages.

Requests for user storage pages are of two types:

- A request to reclaim a specific page previously assigned to a task.
- A request for an initial assignment of a page to a task.

If the request is to reclaim a page that previously belonged to the same task, the Core Block Table entry for the page is checked to see if the page has, in the meantime, been assigned to another task.

If it has not, the page (i.e., CBT entry) is removed from the not-in-use list to which it is attached and is assigned to the current task. Reclaiming the page avoids the necessity of having to read in the page from drum or disk storage.

On the other hand, the page may have already been assigned to another task or the request may be for the initial assignment of a core block (i.e., page) to a task.

In this case, the first page in the "Available" list is assigned to the task. If this list is empty, the first page in the "Pending" list is taken; or, if that list is also empty, the first page in the "Preferred Page" (XTSI/PSW) list is assigned.

If a page is found, it is removed from the "not-in-use" list to which it is attached and assigned to the requesting task. In addition, if this page represents storage assigned to a virtual storage page obtained by GETMAIN, the page is cleared to zero and no page read in is required.

If the pages just assigned reduce the amount of assignable storage blocks below the value of the "high" Low Core Threshold Parameter, the Low Core Indicator is set (see "Scheduling Algorithm"). This indicator will be reset when the number of assignable storage blocks again rises above the value of the "high" Low Core Threshold Parameter. Values are assigned to the threshold parameters at startup.

If no page is available or a "low-core-low" condition exists, the request for user storage (represented by one or more Page Control Blocks attached to a GQE) is saved and user core allocation is suppressed. An attempt will be made to process the request again after some in-use pages are reclaimed by releasing shared pages or, as a last resort, by forcing time-slice-end on the task nearest the commutator that is not already in time-slice-end status.

Whenever a page is released, it is placed in the appropriate not-in-use list unless the Resident Supervisor Reserve List has decreased below five pages. In this case, either the returned page or a less preferred not-in-use page is transferred to the Reserve List. Thus, in effect, the Reserve List is the highest priority not-in-use list.

Whenever a page is returned to one of the User Core not-in-use lists, the suppression of user core allocation is turned off, if allocation was suppressed.

When a user's task is sufficiently large, it is possible for the segment tables to exceed one page. When this occurs, it is necessary for the several segment table pages to be allocated contiguous pages in main storage. This restriction arises from the fact that the Dynamic Address Translation unit locates relative page table entries in the segment table by simply indexing into the table from its base address.

If the task is starting on a new time slice, contiguous pages are allocated from previously owned pages, if possible, or new pages are assigned. If the task is dynamically expanding its segment table size, the page following the current segment table page, the page preceding the current segment table page, or a sufficiently large block of new pages is allocated in that order of preference. In all cases except for the page following the current segment table page, the segment table pointer in the XTSI is updated.

Supervisor Storage Allocation

The only Supervisor storage that is allocated after Startup is obtained from the Reserve List.

The Reserve List itself is composed of unfragmented pages. Requests for storage in less than page size blocks are sub-allocated from pages obtained from the Reserve List. Once a page is obtained from the Reserve List for sub-allocation, the page is no longer chained to the Reserve List.

Pages used for sub-allocation are chained together and are divided into 64-byte blocks. The first block of each page contains an "available block counter" and a bit map which indicates which 64-byte blocks within the page are available.

Requests for Supervisor storage fall into three categories:

1. Requests for one 64-byte block.

2. Requests for several contiguous blocks.

3. Requests for a full page.

Requests for one block (for instance, to build a GQE) are expected to be the most common type of request. Therefore, to speed up the allocation of a single block of storage and to reduce the fragmentation of the pages obtained for sub-allocation, six "quick cells" are maintained. They point to the most recently returned single blocks.

When these cells are empty, the first available blocks found are assigned.

If the request is for contiguous blocks, the bit maps of those pages whose available block counters indicate a good probability that the request can be satisfied are searched before the bit maps of the other pages.

If the request is for an entire page, an unfragmented page is allocated from the Reserve List, if possible.

If a request for Supervisor Storage cannot be satisfied, an unavailable indication is returned if the request specifies that it would accept such an answer. If the request specifies that an unavailable indication is not acceptable, an in-use page is borrowed from user core. An example of such a case is a request for storage during interruption stacking. The page tables of the XTSIs in main storage are searched rather than the not in use user core allocation lists in the Core Block Table. This is because the Reserve List is automatically replenished from these "not in use" lists and thus, whenever the Reserve List is empty, there is a high probability that the "not-in-use" lists are also empty. The search for a user page to borrow involves up to three passes over the XTSIs, looking for:

- First Pass - An unreferenced page belonging to a task that is not "in execution"

- Second Pass - A page that has been referenced but not changed belonging to a task that is not in execution

- Third Pass - An unchanged page belonging to a task that is in execution. (The dynamic address translation unit's associative registers are reset in this case.)

Whenever a full page is returned or whenever all the blocks in a page have been returned, the page is placed in the User Core Available sublist.

VIRTUAL STORAGE ALLOCATION

In TSS/360 each task operates in a unique virtual storage. That is, (except in the case of shared virtual storage) each user's virtual storage references correspond to storage locations within pages unique to the user and which, at the time of the references, may be located in core, on auxiliary storage or on external storage.

A discussion of those aspects of virtual storage which concern sharing can best be understood in the context of a discussion of all aspects of sharing within TSS/360 and will be described in depth in a later section, Sharing, and only briefly discussed here.

Virtual storage is allocated when a task is created and whenever a task wishes to dynamically increase the amount of virtual storage in which the task is operating. For example, the Dynamic Loader frequently requests virtual storage allocation when mapping a user's program module into virtual storage. System service routines require virtual storage allocation to obtain working space for tables.

When a task asks to have its virtual storage extended it really requires two things:

- The virtual storage address at which the allocation begins.

- The modification of the tables describing the task's virtual storage.

Because each task obtains and releases virtual storage in an essentially unpredictable fashion, the determining of where a virtual storage allocation should be assigned within a task's virtual storage is considered to be a task oriented function and, as such, is performed by a system service routine. The actual modification of a task's page tables is performed by the appropriate supervisor routines.

The storage allocated to a task's virtual machine is described by tables residing in the variable portion of the task's Extended Task Status Index (XTSI). These tables consist of:

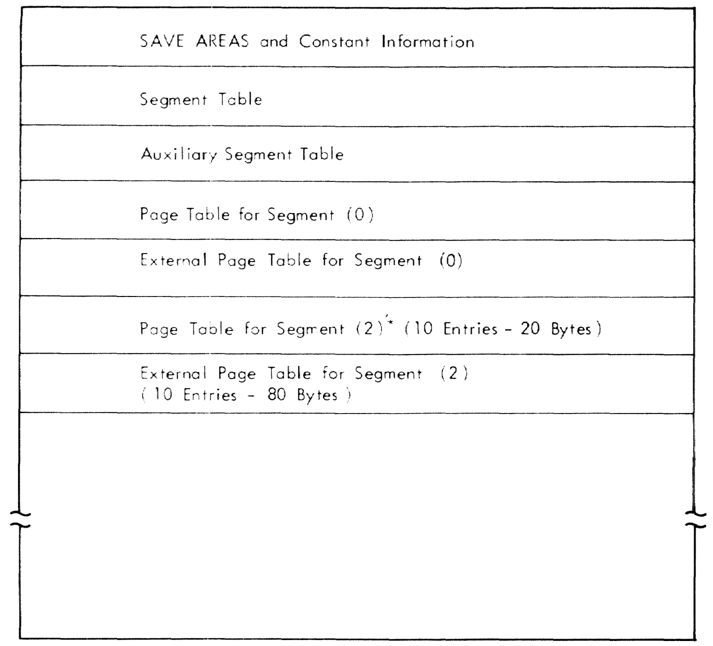
Segment Table
Auxiliary Segment Table
Page Tables
External Page Tables

Figure 57 schematically shows these tables in a sample XTSI.

The Segment Table appears first in the XTSI. Each segment table entry is one word in length. The first byte of the entry contains the length of the page table for that segment. The remainder of the entry, except for the last bit, contains the origin of that page table in real storage. The last bit in the entry is an "availability" bit which indicates whether the page table is in real storage. Immediately following the Segment Table is the Auxiliary Segment Table. For each entry in the Segment Table, there is a double word Auxiliary Segment Table entry in the same relative position within its table as the corresponding Segment Table entry is within the Segment Table. Each Auxiliary Segment Table entry contains the auxiliary storage location of the corresponding Page Table if this segment is allocated, and some flags concerning the status of this segment.

Immediately following the last Auxiliary Segment Table entry are the individual Page Tables and External Page Tables. The Page Table entries are half-word entries and the External Page Table entries are double word entries on word boundaries.

Each Page Table entry contains the storage block (i.e., page) address of this page if it is in core, and an availability bit. Each External Page Table Entry contains the auxiliary or external storage address for this page, if applicable, and various flags concerning the status of this page (e.g., unprocessed by the Dynamic Loader, storage protection class).


SAVE AREAS and Constant Information
Segment Table
Auxiliary Segment Table
Page Table for Segment (0)
External Page Table for Segment (0)
Page Table for Segment (2)* (10 Entries - 20 Bytes)
External Page Table for Segment (2)
(10 Entries - 80 Bytes)

*The Page Table and External Page Table for Segment One is a Shared Page Table and does not appear in the XTSI.

Figure 57. Sample XTSI With Virtual Storage Allocation Tables

In order to keep the External Page Table entries on word boundaries, a dummy Page Table entry and External Page Table entry is created whenever a segment contains an odd number of pages. This dummy entry is not reflected in the length of the page-table-field of the corresponding Segment Table entry but is reflected in an XTSI field which specifies the number of bytes available in the bottom of the first XTSI page. The External Page Table for any particular segment immediately follows the Page Table for the segment. However, these tables are not necessarily in order (i.e., the Page Table and External-Page-Table for segment 3 may appear in the XTSI before the Page Table and External-Page-Table for segment 2). The primary reason for placing the External-Page-Table for a segment immediately following the Page Table for that segment is so that the pair of tables can be located using one pointer.

The virtual storage allocation operations concerned with modifying the tables describing a task's non-shared virtual storage are performed by the following SVCs and their corresponding Supervisor SVC Processors:

ADDPG -	Assign additional contiguous pages to a task's virtual storage and create any necessary page table and External Page Table entries.
DELPG -	Release a contiguous set of pages from a task's virtual storage and delete the associated page table entries, when possible.
MOVXP -	Move the contents of a contiguous set of page table entries and external page table entries to a new location in the task's Page Tables and External Page Tables.

Briefly, the addition of pages to a segment in a user's virtual storage may involve either the addition of a page to a segment that already has pages assigned to it or the addition of pages to an unassigned segment.

When a page is being added to a segment which already has pages assigned, the addition may be made to the end of the segment, thus expanding the corresponding Page Table and External Page Table of the segment, or the addition may be made within the segment. An addition within a segment is an insertion in unassigned table entries (i.e., PT/XPT) which have been released by the execution of a Delete-Page (DELPG) SVC but which could not be physically deleted because they were not at the end of a page table.

If no space is available in an existing segment, a new Page Table will be created. In the 32-bit system, the Segment Table is expanded in blocks of 16 entries.

The management of page and segment tables requires some algorithm to prevent the continual obtaining and releasing of virtual memory from resulting in excessive fragmentation within XTSI pages. Excessive fragmentation could lead to a large number of sparsely utilized XTSI pages.

Whenever the size of relocation tables within the first XTSI page is changed, the XTSI page will be repacked so that empty bytes are located at the end of the page.

If there is not sufficient space in the first XTSI page to expand a relocation table, the expanding Page Table and External Page Table will be moved to another XTSI page where space is available and the first XTSI page will be repacked, if necessary.

Space with XTSI pages beyond the task's first page, however, is not reclaimed until the entire page becomes free or until a limit on the number of XTSI pages the task may have is reached. The logic here is that the algorithm for Virtual Memory allocation makes it more likely that only the last Page Table/External Page Table pair in the first XTSI page is likely to grow and shrink; whereas the pattern is not likely to be so neat when a large number of segments is used.

The assignment of virtual storage is performed by the Virtual Memory Allocation system service routine.

The entry points of this module which are concerned with page allocation are as follows:

Non-Shared Virtual Storage Functions

GETMAIN	- get private virtual storage by pages.
FREEMAIN	- free virtual storage by pages.
EXPAND	- expand an existing block of contiguous virtual storage.

Shared Virtual Storage Functions

GETSMAIN	- get shared virtual storage.
CONNECT	- connect a Segment Table entry to a particular Shared Page Table.
DISCONNECT	- DISCONNECT A Segment Table entry from directly pointing to a Shared Page Table.

The GETMAIN and FREEMAIN entry points correspond to user macro instructions of the same name.

GETMAIN obtains virtual storage for a user's program or a system service routine. The ADDPG supervisor call is used to perform the actual allocation.

FREEMAIN releases virtual storage previously allocated by GETMAIN; the supervisor call DELPG is used to release the area.

A nonprivileged routine uses GETMAIN to acquire contiguous pages of virtual storage. If additional contiguous pages are required, a second use of GETMAIN will not in general provide pages contiguous with pages obtained from the first use of GETMAIN. Thus, a nonprivileged routine must either request sufficient contiguous storage through GETMAIN at the outset, or else chain together groups of contiguous pages obtained through successive uses of GETMAIN. A privileged routine, on the other hand, may acquire additional pages contiguous with an existing group by using EXPAND. For example, VAM may request additional virtual storage in order to enlarge a RESTBL. EXPAND is used for this purpose. EXPAND normally allocates pages for the expanded RESTBL in a manner similar to GETMAIN. EXPAND then uses the Move External Page (MOVXP) supervisor call to move the contents of the associated Page and External Page Table entries to their new positions. This gives the RESTBL a new location in virtual storage. The Delete Page supervisor call is then used to release the old group of virtual storage pages from the task's virtual storage and if possible to physically delete the associated table entries.

There are two versions of the Model 67 dynamic address translation. One version allows the generation of 24-bit virtual storage addresses, while the second allows the generation of 32-bit virtual storage addresses. The difference between the two versions, as supported in TSS/360, is that the 24-bit version permits utilization of up to 16 segments, while the 32-bit version permits the utilization of 4096 segments.

In TSS/360, the smallest quantity of virtual storage that can be shared is a segment. The reason for this is that the manner in which virtual memory is shared is for each task to share a page table which maps into main storage the program or data being shared. That is, each task which shares pages within that segment has one of its Segment Table entries pointing to the

Page Table for the segment and thus has access to all pages allocated for the segment. A Page Table thus shared is called a shared page table. This is shown in Figure 58.

In the 32-bit system, it is possible to place each data element to be shared all alone in a separate segment. In this fashion, User A and User B can share Program Y, and User B and User C can share Program X, but User A does not have access to Program X.

On the other hand, this is not possible in the 24-bit system since there are only 16 segments.

Thus, the greater number of segments in the 32-bit system permits a greater number of categories of sharing, for those installations that have a need for this extended capability.

A basic design goal of TSS/360 is to isolate in one module the need to be aware whether the hardware is 24 or 32 bit; and to provide a mechanism which satisfies the desire for segmentation on the part of those installations employing the 32-bit system, while allowing for more dense packing of virtual storage on the 24-bit system. This is accomplished by presenting to the system, during System Generation, a series of parameters which are interrogated by the Virtual Memory Allocation module in the course of responding to a request for virtual storage allocation.

The major virtual storage allocation parameters are:

- SYSTEM PACKING INDICATOR - If this is on, all requests for private virtual storage are to be page packed into available segments.

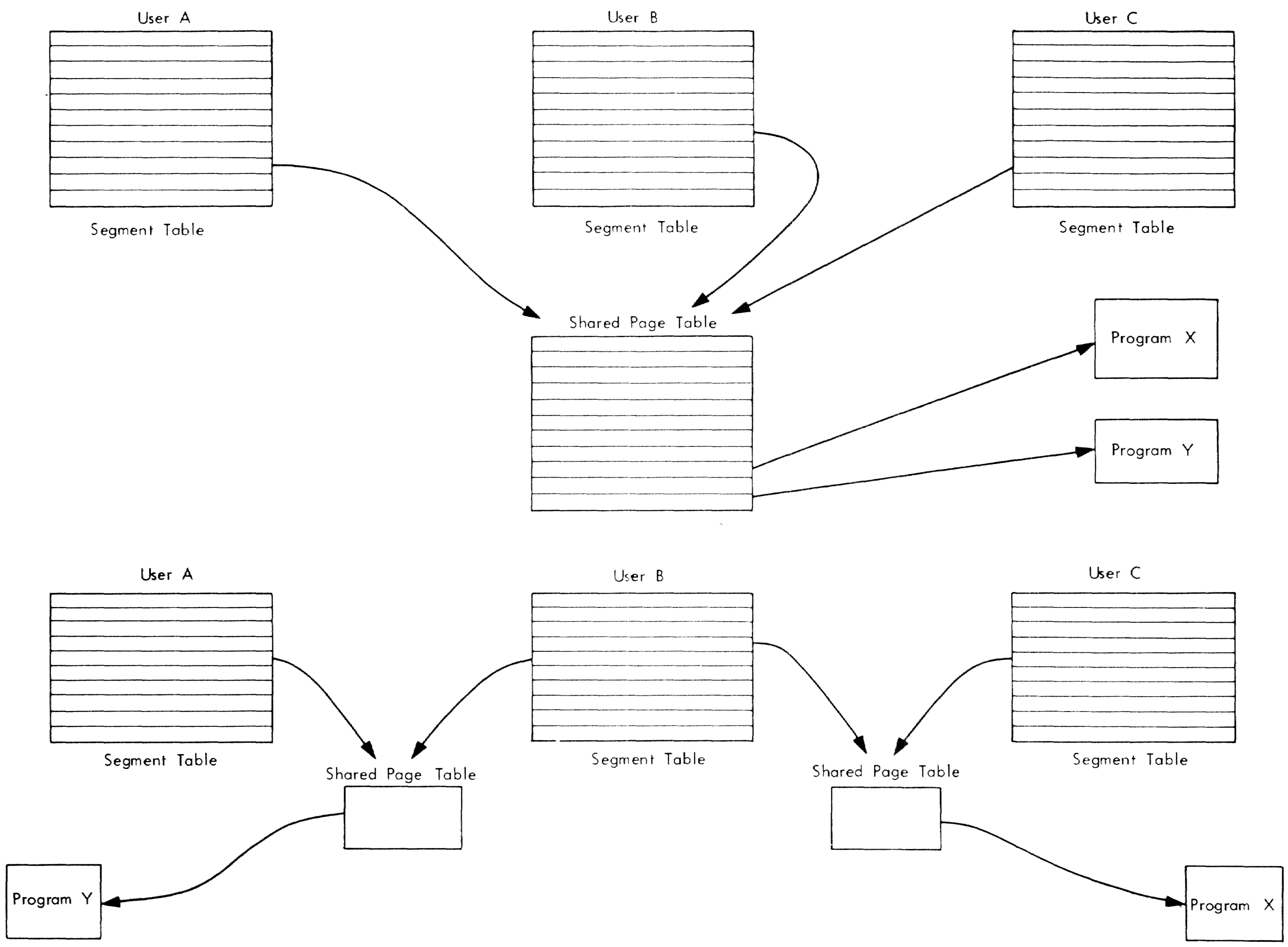


Figure 58. Sharing of Segments in the 24-Bit Versions of the Model 67

- PUBLIC SEGMENT INDICATOR - If this is on, sharable control sections are to be packed into a shared segment known as a Public Segment. If that segment becomes full, additional segments are allocated, as necessary.

When there are several public segments a user does not automatically have more than one public segment allocated to his virtual storage. Only if the user's task links to a program in a public segment will that public segment be allocated to his virtual storage. Thus, even a 24 bit system allows a more flexible handling of shared virtual storage than is possible in an unsegmented system. The first public segment is automatically referenced during task initialization (see "Initial Virtual Storage").

Packing refers to the assignment of successive pages of a segment until all 256 pages have been allocated, not to the condensing of information.

Normally, a 24-bit installation chooses system packing and Public Segments. A 32-bit installation may choose to have each request for virtual storage assigned to a separate segment. Figure 59 shows how virtual storage is allocated under such conditions.

There is, however, no firm division between the two systems in regard to virtual storage allocation. For instance, private PSECTs are always packed regardless of system type.

The selection of these system parameters is at the discretion of the installation. It is dependent on the hardware (24- or 32-bit system) only in a logical sense. A public segment can exist in either a 24- or 32-bit system; a 24-bit system does not have to pack control section, etc; therefore the combination of parameters which are logically best suited to the installation's needs can be selected.

Furthermore, Virtual Memory Allocation supports the ability to alter the basic assignment method to meet different requirements. The GETMAIN macro may specify parameters which override, for that allocation, those which System Generation presented to the system. For example, in a system where packing has been indicated, the GETMAIN issued by Data Management to obtain space for the RESTBL associated with a private data set might contain a parameter which instructs VMA to override the system packing parameter and allocate virtual storage on a segment boundary.

TYPE OF DATA	TYPE OF SEGMENT	
	24-bit System	32-bit System
Each group of Private Control Sections within one Program Module that have identical control section attributes	Packed Private Segment	Private Segment
*Each group of Public Control Sections within one program module that have identical control section attributes	Packed Public Segment	Shared Segment
Private Prototype Control Sections PSECTS	Packed Private Segment	Packed Private Segment
Private VAM Data Set (RESTBL) and Requests for Private work areas and buffers	Packed Private Segment	Private Segment
Public VAM Data Set RESTBL	Packed Public Segment	Shared Segment

DEFINITIONS:

Packed -	Each Request is allocated on a page boundary in a segment that may already have pages allocated to a previous request.
Private -	Pages Tables are in the tasks XTSI and are therefore available only to this task.
Public -	Page Table is in supervisor storage and is pointed to by a segment table entry for all tasks wishing to reference it. Packed by page boundary.
Shared -	Page Table exists in main storage and is pointed to by a segment table entry for all tasks who wish to reference it.

*If a Public control section is loaded by a User with an O or P Authority Code or from a private library, the Public attribute is erased by the Dynamic Loader, and VMA will treat the control section as Private.

Figure 59. Possible Scheme of Virtual Memory Allocation

If a GETMAIN request specifies the variable length parameter, the amount of virtual storage to be allocated is determined by the "variable allocation" system parameter which is specified during System Generation. The amount of virtual storage allocated is either a segment or an integral number of pages (less than 256) in addition to the specific amount of virtual storage requested by the GETMAIN macro. In conventional systems, the reservation of address space is tightly bound with the reservation of main storage. With variable-length allocation of virtual storage, a TSS/360 user can reserve a large address space in which a dynamic table can grow without having to make a "worst case" claim upon main storage.

Once Virtual Memory Allocation has determined whether a particular request is to be packed or not, it must then determine where to place the allocation.

If the request is to be packed and will fit in the available space remaining in the current segment, it is placed in that segment. No attempt is made to fill in holes within a segment.

If the request is to be packed, but can't fit in the current segment, or if the request is to be allocated on a segment boundary, the lowest numbered available segment is generally used.

It is possible, on the 24-bit system, that no free segment exists. In this case, Virtual Memory Allocation consults a map which specifies the availability or non-availability of each of the task's 4096 pages, to find a group of contiguous pages large enough to satisfy the request.

If a request can not be satisfied, the user has the option of regaining control or of having his task abnormally terminated (ABEND).

SMALL VIRTUAL MEMORY ALLOCATION

Through use of the "R" option of the GETMAIN and FREEMAIN macros, virtual storage may be assigned and freed in multiples of bytes (as compared to pages).

Small Virtual Memory Allocation (SVMA) can be used by all system service routines and users for dynamic virtual storage allocation. For instance, most PSECTs are considerably less than a page in length. TSS/360 conserves main storage and auxiliary storage by packing PSECTs on double-word boundaries, whenever possible, through the services of SVMA.

Operationally, SVMA may be thought of as a subset of large VMA, in that storage is first allocated by GETMAIN (pages) and then sub-allocated as appropriate by SVMA. SVMA maintains control information about each page in page headers which are placed in either of two types of allocation chains depending on the type of user: privileged or nonprivileged. This ensures that the virtual storage of the system service routines is not available for use by user programs, nor can it be freed by them.

Each page header contains all the information about its corresponding page of virtual storage, including a unit table, consisting of one bit for each doubleword (unit) of the defined page to indicate which units of the page have been allocated.

Requests for virtual storage are classified by SVMA into those for less than one page and those of one page or more. Requests for a page or more are always page aligned, while those for less than one page are aligned on a doubleword boundary. Allocations of the latter type are assigned areas within the same page if possible. The requestor can never assume that two requests for virtual storage back to back will necessarily result in a contiguous allocation of both requests. The only way to ensure a contiguous allocation of N bytes is through one GETMAIN of N bytes. This is because, transparent to the user, a task interruption may have occurred between the two apparently successive uses of GETMAIN. If such an interruption did occur, a privileged routine may have invoked GETMAIN between the user's first and second invocations of GETMAIN.

For operating within the boundaries of a page, SVMA maintains a next available unit (NAU) address, which refers to the next available doubleword on the page. The NAU is advanced by the amount of each GETMAIN (bytes) request; when combined with the page address it forms the address returned to the user.

For FREEMAIN requests, the NAU is compared with the maximum address to be freed. Except for requests for more than one page to be released, the maximum address to be freed must not exceed the NAU address, or an error condition results.

FREEMAIN frees the requested units by updating the corresponding bits in the unit table. The portions of the page released will not be reassigned until all units of the page which were allocated through SVMA are freed. When all units are free for a page, as indicated in the unit table, the page is freed via FREEMAIN (pages) before returning to the calling program.

Because virtual storage is not actually released until it is freed by FREEMAIN (pages), the user may be able to retrieve data from an area allocated using SVMA after he has given a FREEMAIN (bytes) request for the area. Likewise, since SVMA initially allocates virtual storage by pages, and subsequently operates on each page in byte requests, the user can potentially reference more virtual storage than he requests through Small Virtual Memory Allocation.

INITIAL VIRTUAL MEMORY

There are certain system service routines which must reside in a task's virtual storage when the task is initiated in order to prevent recursion; e.g., the Dynamic Loader calling the Dynamic Loader to load the Dynamic Loader. There are a large number of other system routines for which it is desirable to perform a full linkage at Startup to reduce the overhead involved in linking between routines during system operation. By tightly binding most system programs once at system startup, the overhead in conventional systems associated with library searches, binding, and unbinding can be significantly decreased. The trade-off here is time versus the auxiliary storage space necessary to hold the IVM.

The routines making up IVM consist of control sections with various attributes. All the tables and private control sections are packed segment (segment 0). All the Public control sections are mapped into shared segments 1 and 2. These segments become the initial Public Segments if the "Public Segment" Indicator is on. Unused space in either segment 0 or 2 is eligible for allocation after Startup. Virtual Memory Allocation is notified of the virtual storage allocated by Startup through information left in the ISA.

Initial Virtual Memory is defined as the total collection of privileged service routines and their tables including fence sitters and those programs which perform the functions indicated by the various TSS/360 commands. In addition, Initial Virtual Memory contains such non-privileged system programs as the Language Processor Control module, the TSS/360 FORTRAN IV Compiler, and the TSS/360 Assembler. The exact content of Initial Virtual Memory is specified at system startup (see "System Generation")

Additional significant characteristics of IVM are:

- The routines comprising IVM are mapped into identical virtual storage locations within each task. This is because Startup constructs a skeletal XTSI which reflects the virtual storage allocated to IVM pages. This skeletal XTSI is assigned to each task as it is created (see "Examples of System Operation - Creation of a Conversational Task").
- The only read-only control sections that will be accessed from the primary paging drum are those included in IVM. This is because unchanged pages are only paged into the system; they are never paged out. The pages containing the IVM read-only control sections are placed on the drum during system startup.
- The user cannot unload from his virtual storage routines contained within the task's IVM (see "Dynamic Loader").
- Because IVM routines are bound only once, it is reasonable to pack them so that the total number of pages occupied is minimized (see System Startup).
- The order in which system programs are allocated virtual memory address space affects system performance. This is because paging can be decreased by placing groups of control sections that tend to be referenced together, into a minimum number of pages. In TSS/360, this ordering is based upon a control section name list which can be easily altered (see System Startup).

The following programming conventions are observed by most programs operating as part of Initial Virtual Memory:

1. All linkages among programs contained in Initial Virtual Memory should be implicit (i.e., fully resolved by STARTUP time).
2. All linkages from programs contained in Initial Virtual Memory to programs not in Initial Virtual Memory must be explicit.

There are a few routines which do not fully observe the first convention. Explicit linkage may not be used by VAM, GATE, TAM or Virtual Memory Allocation because of recursion considerations. Explicit and Implicit linkages are described in "Dynamic Loader."

AUXILIARY STORAGE ALLOCATION

Auxiliary storage plays an important role within a time sharing system. For instance, if the effective access speed of the primary auxiliary storage device is too slow relative to the speed of main storage, the CPU cannot be used effectively. The effective access speed of a device is, in part, dependent upon the way in which the device is formatted and the fashion in which the I/O queue for the device is processed. In addition, the total capacity of auxiliary storage is directly related to the amount of virtual storage that the system as a whole can allocate. For instance, in TSS/360, much of Initial Virtual Memory and all of a task's changed pages are contained within the system's auxiliary storage. If more than one class of device is utilized for auxiliary storage, consideration must be given to the distribution of pages among the various devices. In TSS/360 auxiliary storage is confined to drum whenever possible. This is done because the drum is the fastest of all auxiliary devices on the system. When auxiliary storage is allocated to a task, the count of available storage space remaining is updated and checked. If the space falls below a system minimum, a task is selected for page migration. This process removes some of the task's pages from the drum and moves them to auxiliary disk thereby freeing drum space. Pages referenced during the last time slice are allowed to remain on the drum up to a maximum of the task's fair share. The task selected for this migration is the task on the inactive list with the most pages on the drum in excess of its fair share. If no task is found on the inactive list, the active list is searched.

As mentioned previously, auxiliary paging devices are specified at System Generation and may consist of drums and/or disks.

STARTUP creates an Auxiliary Storage Allocation Table (ASAT) which contains a bit directory, for each auxiliary device be it drum or disk, on the system. Each bit represents one page. The directories for the drums and disks are chained separately within the ASAT.

Each paging drum is formatted so that it logically appears to contain 100 tracks of 9 pages in its bit directory.

The bit directory of each drum has the pages allocated for the following purposes set as unavailable by STARTUP:

- Drum bit map
- Initial Virtual Memory (if this drum is the Primary Paging Volume)
- SERR/RECONFIGURATION Modules
- Standard surface test track (pages 882-890)
- Skeletal XTSI
- Any defective pages encountered in writing out IVM and SERR/ RECONFIGURATION

The bit directory of each disk has the following pages set as unavailable:

- PAGE Assignment Table page or pages
- Initial Virtual Memory (IVM) and any bad pages found while writing out the IVM pages, if the disk is the primary Paging Volume
- IPL Volume data sets if the disk is also the IPL volume
- Standard surface test cylinder

For a 2311, there are 203 cylinders formatted for 8 pages/cylinder. For a 2314, there are 203 cylinders with 32 pages/ cylinder.

After Startup, Auxiliary Storage allocation is performed by the Resident Supervisor Auxiliary Storage Allocation Queue Processor, and the Auxiliary Storage Allocation Release and the Suppress Auxiliary Storage Allocation subroutines.

When a task enters the system, its auxiliary storage requirements are compared with the available auxiliary storage count. The task is not allowed on the system if there is not sufficient auxiliary storage available. If the task should exceed its limit of auxiliary storage during execution, and available auxiliary storage is less than the installation minimum, it is first warned, if conversational, and then terminated. Under these conditions, nonconversational tasks are terminated at once.

Auxiliary storage is obtained whenever it is necessary to write a page out to auxiliary drum or disk. Auxiliary storage is released, as appropriate, whenever FREEMAIN issues the Delete Pages (DELPG) SVC, whenever a Task Status Index (TSI) is to be deleted (DELTSI SVC), for example, at LOGOFF time, and whenever a changed page which already has an old copy on auxiliary storage is ready to be written back out to auxiliary storage (see "Paging").

A request for auxiliary storage allocation is represented by one or more Page-Control Blocks pointed to by a GQE.

Requests for auxiliary storage allocation are assigned from a drum except when:

- The request for auxiliary storage has specified a "drum preference" and no drum storage is available.

- The request for storage has not specified a "drum preference" and available drum space has reached a minimal level. This minimal drum space threshold is a system parameter specified at System Generation and contained in the System Table.

Currently, the page containing the task's Extended Task Status Index, the page in which a task was executing just before it became dormant (PSW page), and the ISA page are represented by a drum preference request.

If there is no auxiliary storage of any sort available, the Auxiliary Allocation Queue Processor declares a major SYSER. If there is more than one drum on the system, each page is allocated space on a different drum, in rotating fashion. Disk pages are allocated, when possible, from the same cylinder.

In performing the Time-Slice-End processing associated with TWAIT, an algorithm is applied to determine if a task has utilized more than its "fair share" of high-speed drum space. If the task has not exceeded its fair share of drum space, normal time-slice-end processing is followed and all of the task's changed pages are written out to the drum. If the task has exceeded its fair share of drum space, all of the drum pages belonging to the task are migrated to an auxiliary disk. The intent is to prevent any task from building up too large a page image on the highest speed auxiliary device.

The method of allocating and accessing pages on drum and disk is further described in "Paging."

EXTERNAL STORAGE ALLOCATION

External storage allocation (ESA) consists of those service routines that allocate storage on direct access volumes used for external storage. ESA routines are privileged and can only be called by other privileged routines. In general, ESA is responsible for initial and secondary allocation of external storage to a data set and partial and total return of external storage allocated to data sets. Additionally, ESA routines provide the maintainance of those control blocks which describe the availability of external storage pages on a volume. For SAM data sets these control blocks are the Volume Table of Contents (VTOC) and the format 1, 3, 4, and 5 Data Set Control Blocks (DSCBs); for VAM data sets a Page Assignment Table (PAT) and format E and F DSCBs are used. The routines which perform these functions are:

Routine	Function
ALLOCATE	Initial Allocation for SAM Data Sets
EXTEND	Secondary Allocation for SAM Data Sets
FINDEXPG	Primary and Secondary Allocation for VAM Data Sets
GIVBKS	Partial Return for SAM Data Sets
RELEXPG	Partial or total Return for VAM Data Sets
SCRATCH	Total Return for SAM Data Sets
DELVAM	Total Return for VAM Data Sets via RELEXPG
SAM SEARCH	Scan DADSM DSCBs for space to allocate
All SAM Routines	Maintain VTOC on SAM Volume

DSCB Handling

OBTAIN, RETAIN, and RENAME routines handle Data Set Control Blocks for external storage allocation.

OBTAIN performs the following functions:

- Retrieves data set control blocks (DSCBs) from the VTOC of a direct access device and places them into a designated virtual storage location.
- Reads the volume label of a direct access device, or a data set label, and places it in a specified virtual storage location.

A DCB and DECB are constructed to maintain compatible linkage with common system hardware error routines and the Task Monitor. The locations of the symbolic device address and the VTOC DSCB are computed and placed in an IORCB. A channel program is then developed within the IORCB to perform the request specified. When the IROCB is complete, an IOCAL SVC is executed, followed by the OBTAIN CHECK routine and an AWAIT. When the I/O operations are completed, the Task Monitor links to the OBTAIN posting routine, which checks for errors. When the posting operation is complete, control returns to OBTAIN. On exit, OBTAIN indicates the status of the operation in a return code.

The function of RETAIN is to write one or more DSCBs or data set labels to specified addresses. An end-of-file marker can also be written with this routine. RETAIN, like OBTAIN, must construct a DCB and DECB to maintain compatibility with the Task Monitor and system error routines. RETAIN places in an IORCB the appropriate symbolic device address and other necessary information. When the IORCB is complete, an IOCAL SVC is executed, followed by the RETAIN CHECK routine that branches to an AWAIT macro instruction if the I/O is not complete. When I/O operations are complete, the RETAIN posting routine is invoked by the Task Monitor to check for hardware errors. Then the Task Monitor returns control to RETAIN. (If posting had detected errors it links to common system error routines for retry.) Although the RETAIN macro instruction can specify any number of records, the RETAIN routine handles them one at a time to facilitate error recovery procedures.

RENAME changes the fully qualified name in the key field of a Type 1 or Type A DSCB for all volumes specified in the JFCB to the name specified by the calling program. The routine uses OBTAIN to search the VTOC to determine if there is already a DSCB with the new name and the retrieve the DSCB with the data set name to be changed. If any of the volumes on which the data set resides is not mounted, the BUMP routine is called to dismount a volume on which RENAME has already been performed and to mount the unmounted volume on that device. After changing the key (the data set name), the DSCB is rewritten by use of RETAIN.

The organization of direct access volumes differs for SAM data sets and VAM data sets. As a result of this the following description of the allocation of storage is divided into two parts: the allocation of storage on SAM volumes and the allocation of storage on VAM volumes.

SAM Volumes

Each Direct Access SAM volume has a Volume Label in a standard location which points to the Volume Table of Contents (VTOC). The VTOC size is specified when the device is formatted during direct access storage device initialization. The VTOC contains Data Set Control Blocks (DSCBs) of which there are four types as follows:

Type	Use
1	Defines SAM data sets within a volume.
3	Extends Type 1 when additional space is needed.
4	First record in VTOC. Describes the VTOC.
5	Second record in a SAM volume. Direct Access Device Space Management (DADSM) DSCB.

Thus, the general arrangement of a VTOC consists of a DSCB describing the VTOC, one or more DADSM DSCBs, and a list of all SAM DSCBs. DADSM DSCBs describe the unused space on the volume that is available for allocation.

Figure 60 shows a diagram of the general arrangement of a VTOC.

The DDEF command processor (discussed under "Command System") calls ALLOCATE when a new data set is to be created. ALLOCATE inspects the Job File Control Block (JFCB) supplied by DDEF to determine the size of the initial allocation, and makes the total allocation on one volume. For SAM data sets, ALLOCATE calls SEARCH (described below), to assign the space to the data set. After updating appropriate tables, ALLOCATE returns to the caller.

EXTEND is called when a SAM access method requires additional space for a data set on direct access volumes. EXTEND obtains the relevant information (space required, volume available, etc.) from the input parameter list, makes the allocation (from one volume only), and updates the VTOC, Job File Control Block (JFCB), and Symbolic Device Allocation Table (SDAT).

The Symbolic Device Allocation Table (SDAT) contains an entry which, when initialized, specifies the gross space available for each volume. This field is initialized during Startup for Public volumes and by the Device Management routine PAUSE when a private volume is mounted. This gross space field indicates the total number of available cylinders and tracks on SAM formatted volumes.

There is another SDAT entry that specifies the number of available (unused) DSCBs left in the VTOC.

Figure 61 describes the algorithm for determining on which volume the external storage will be allocated.

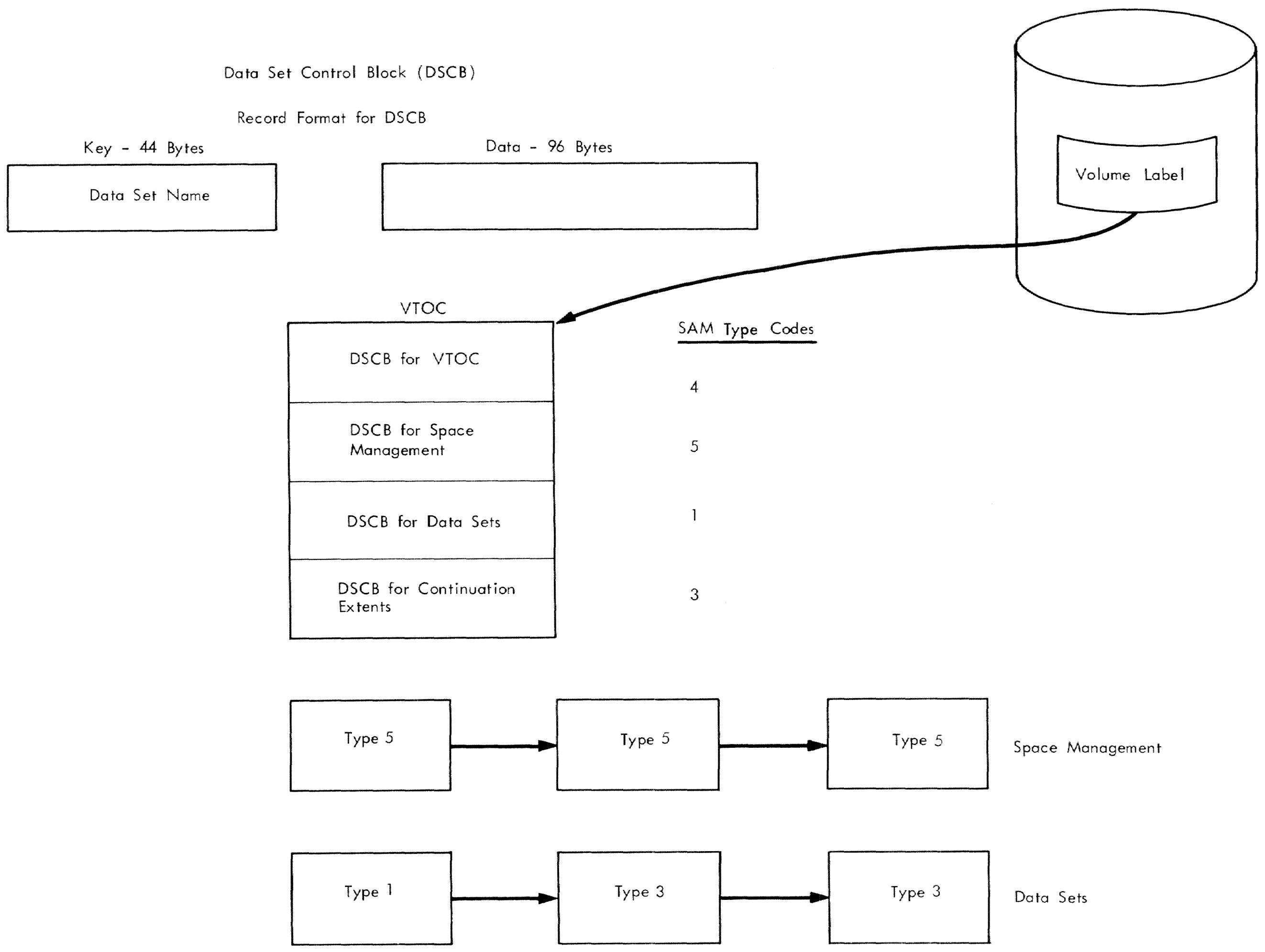


Figure 60. Direct Access Device Volume Table of Contents (VTOC) Format for SAM Volumes

	Request for SAM Private
ALLOCATE	JFCB will indicate one mounted volume. Space can be allocated only from that volume.
EXTEND	JFCB will indicate one mounted volume. Space can be allocated only from that volume. If there is not sufficient space available, Extend will link to the calling program (EOV) for label processing and the mounting of a new volume. EOV will return to Extend for the allocation from the new volume.

Figure 61. Allocation of External Storage to a SAM Volume

Two areas in this algorithm deserve further explanation.

The space to be allocated to a VTOC is specified when the volume is formatted. Any given VTOC size can contain only a fixed maximum of DSCBs.

If External Storage Allocation requires the creation of additional DSCBs, but there is no VTOC space left in which to create the DSCB (even though there may be plenty of data set space on the volume), the allocation cannot be made on that volume. If this is the only volume on which the allocation can be made, the task is abnormally terminated.

For a SAM data set, space can be allocated only on the volume in which the last allocation for that data set was made, because SAM volumes are processed serially and any other external storage allocation arrangement would result in parts of a SAM data set being processed out of sequence.

SAM SEARCH is used by the other External Storage Allocation routines to search the DADSM DSCBs on the particular volume selected for extents to be allocated.

SAM SEARCH, during its scan, creates a list of the five largest extents. If an extent equal to or larger than the request is not found, space is allocated from the extents in this list. If sufficient space still cannot be found, a return is made with a "No Space" indication.

When Standard User Labeling (SUL) is requested and the allocation request is for cylinders, an extra track is allocated anywhere in the volume to be used for labels. When the request is for track allocation, the SUL Track is assigned to the data set allocation.

All SAM routines use OBTAIN and RETAIN (catalog service routines) to read and write the DADSM DSCBs.

GIVBKS rewrites DSCBs for SAM data sets and, if necessary, returns unused space on volumes.

MERGE SAM adds returned extents (i.e., unused space) to the list of available extents in the DADSM DSCBs, maintaining the order of available extents by volume location and combining extents whenever possible.

SCRATCH is called to remove from a volume a data set the user wishes to erase. SCRATCH is also called when a generation data set overflow is found by the catalog services ADDCAT routine. SCRATCH merges the just released extents back into the DSCBs for direct access device space management (DADSM) located in the VTOC. SCRATCH also zeroes the just erased data set's DSCBs.

TSS/360 External Storage Allocation does not support the following OS/360 DEFINE DATA parameters for SAM formatted data sets: Absolute Requests, contiguous, suballocation, split cylinders, MIXIG, ALX.

VAM Volumes

The format of VAM volumes and the manner in which data is transmitted give the virtual access methods and the external storage allocation routines a great deal of flexibility in obtaining and allocating external storage for VAM data sets. VAM data sets are organized in physical units of 4096 bytes called pages. Consequently, each VAM volume is also divided into pages. A 2311 disk pack contains 1620 allocatable pages and a 2314 disk pack 6492 allocatable pages. Each volume contains a Page Assignment Table (PAT) which is pointed to by the volume label and which describes the status of each allocatable page on the volume. A page may be available for assignment, assigned as a DSCB page, assigned as a data set page, or in error and unassignable. At the end of the PAT, error page control and relocation entries are maintained. When a page is found to be in error, it is so marked and a replacement page is selected from the available pages on the volume. In order for the access method routines to locate the new entry when they discover the old page in error, a relocation entry is created at the end of the PAT. This entry contains the address of the original page and the address of the new page. Up to 96 such entries can be contained in the PAT.

A DSCB page can contain 16 DSCBs. Only DSCBs are written on DSCB pages and they may describe more than one data set. That is, a DSCB for data set A and a DSCB for data set B may be found on the same DSCB page. Since DSCBs may be placed on any page on the volume, the condition will never arise in which allocatable pages are available but no DSCB space remains as in the case of SAM data sets.

Two types of DSCB exist for VAM data sets:

Type	Use
E	First DSCB for the data set
F	Continuation DSCBs for the data set

In general, the format E DSCB contains the data set name and properties. For multivolume private data sets, the format E DSCB also contains a list of the volumes which contain the data set. For single volume private data sets, the volume ID is contained in the Data Set Descriptor. For all public data sets, a list of public volumes called the Public Volume Table (PVT) is maintained. This table contains the volume IDs of all volumes dedicated to public storage.

A format E DSCB can contain a list of up to 25 private volumes. If more space is required, a format F DSCB is used. Immediately following the volume list is a list of the external page entries for the data set. For single volume private data sets and for public data sets in which there is no volume list, the external page entries begin in the format E DSCB and, if needed, continue in the format F DSCBs. The page entries for a multivolume private data set may begin in the format E or a subsequent format F DSCB depending on the number of volumes the data set occupies.

External page entries for all data sets consist of a relative volume number and a relative page number. The relative volume number is an index into the PVT for public volumes.

Note: For private volumes an analogous list is maintained even though the volume IDs are contained in the DSD or the DSCBs. This is done to provide similarity of processing for all data sets.

The initial allocation of external storage for new VAM data sets is done when the data set is first opened. Space is assigned by the service routine FINDEXPG. Secondary allocation of space is also performed by FINDEXPG. ADDDSCB allocates space for DSCBs when necessary. The partial return of pages is accomplished by the service routine RELEXPG and total return by DELVAM which deletes a data set catalog entry and returns all space by means of RELEXPG. All modules manipulating the DSCB or PAT call READWRIT to read or write a DSCB, or to write a PAT to external storage. ESA LOCK effects the handling of interlocks on the SDAT table.

DEVICE ALLOCATION

There are three characteristics of the TSS/360 resource sharing environment that are especially significant in relation to device allocation:

- A device may have several hardware addresses.
- In a time sharing environment, quick access to data structures is highly desirable.
- A conversational session should not require the user to extensively preplan the demands he will place on the system's resources.

A device may have several hardware addresses because, for one thing, control units may be attached to more than one channel. In order to allow a user's program to be highly device independent and to allow the Resident Supervisor to remain relatively insensitive to dynamic changes in system configuration, the TSS/360 access methods employ a symbolic device address to designate each device, and TSS/360 users employ device class codes that can be used to describe a device as a member of a class of like devices. The one exception to this: a Command System Privilege Class E user may specify devices by their symbolic device address.

The Resident Supervisor recognizes these symbolic device addresses and uses a group of tables called Pathfinding tables to translate a symbolic device address into a specific hardware address that specifies a path through a Channel Control Unit, Channel, and Device Control Unit to the device. In a multiple-access environment, where there are a large number of concurrent users, device allocation must be carefully controlled to prevent a situation in which an installation might be required to maintain a large number of tape units or establish elaborate queuing conventions in order to ensure that each concurrent user can have access to his own private storage on short notice.

In TSS/360, all devices fall into one of two major categories:

- Reserved for system use
- Available to system users

Devices reserved for system use, such as auxiliary storage devices and the IPL volume, are specified during System Generation and are assumed to be on-line at all times and unavailable for allocation to system users (with the exception of the system programmer who has joined with a Command Language Privilege Class of E).

Devices available for allocation to system users can be usefully separated into two classes:

- Terminals
- Media-oriented devices

Strictly speaking, communications lines, not terminals, are allocated in TSS/360. This is because a complete TSS/360 I/O address consists of 13 bits: a 2-bit Channel Controller address, channel address (3 bits), control unit address (4 bits) and, finally, the device address (4 bits). This addressing structure only allows specification down to the line adaptor on a 2702 Transmission Control Unit.

Furthermore, terminals are not allocated in TSS/360 because, in general, terminals may be dial-up. A dial-up terminal may be connected to any system and cannot be uniquely identified with a particular system.

TSS/360 only allocates terminals as SYSIN/SYSOUT devices and does not support more than one SYSIN/SYSOUT terminal per task. However, a privileged user may, through the facilities of several privileged SVCs and by programming using the privileged facilities of the Terminal Access Method, allocate slave terminals to his task provided they can be reached either via a dedicated line or via an autocall unit.

Media-oriented devices are separated into two major classes:

- Public
- Private

Public devices are, in general, direct access devices which are defined as such during system generation. However, if a volume is mounted on a device at Startup, the device is classified as Public or Private according to a Public/Private field in the Volume Label. Each public device on the system is made available to each user's task when his task is initialized. Public devices are used to mount Public volumes.

Public volumes are meant to contain that part of a computation center's on-line storage which is to be shared among the users of the system, as well as containing cataloged private data sets. After Startup, public volumes are assumed to be permanently mounted, and the specification of any particular public volume from which to allocate a request for external storage is controlled by the TSS/360 External Storage Allocation routines. This is the means TSS/360 provides to ensure a user rapid access to his data sets.

Private devices are used to mount private volumes. A private volume may only be used by one task at a time. However, it is expected that in the typical time sharing installation, a large proportion of the total users will not use private volumes or devices.

The major virtual storage tables responsible for maintaining the description and status of the system's devices are the Symbolic Device Allocation Table and the Available Device Table. The relationship of these tables is shown in Figure 62.

Both these tables are created from the symbolic device addresses and device names (e.g., 2311, 2314, 2403, etc.) supplied in the Device Group macro instruction during System Generation. Each device has only one symbolic device address, although it may possess several physical addresses. The assignment of symbolic device addresses is not arbitrary in TSS/360, inasmuch as the symbolic address of a device specifies the relative position of the device's queue entry on the Resident Supervisor's Queue Scanner Table (SCANT). This arrangement provides a straightforward way of determining the priority for servicing a device, and also permits a change in system configuration without a corresponding change in the SCANT. Device independence for the user is obtained primarily through the DDEF command and, for the system, device independence may be obtained by suitably modifying the Pathfinding tables of the Resident Supervisor.

The Symbolic Device Allocation Table (SDAT) is located in shared virtual storage and has entries, sorted by symbolic device address (low to high), for each device on the system. For a Symbolic Device Address for which a device does not exist, a dummy entry is included. The SDAT entries for Public devices are linked together in a chain to facilitate searching these entries in connection with external storage allocation.

Each entry is divided into a fixed format area and a device dependent area. The fixed area specifies information such as device status, maximum number of concurrent I/O requests allowed for this device, and a count of the number of concurrent users of the device.

The device dependent area specifies information such as the gross space on this volume available for allocation, the

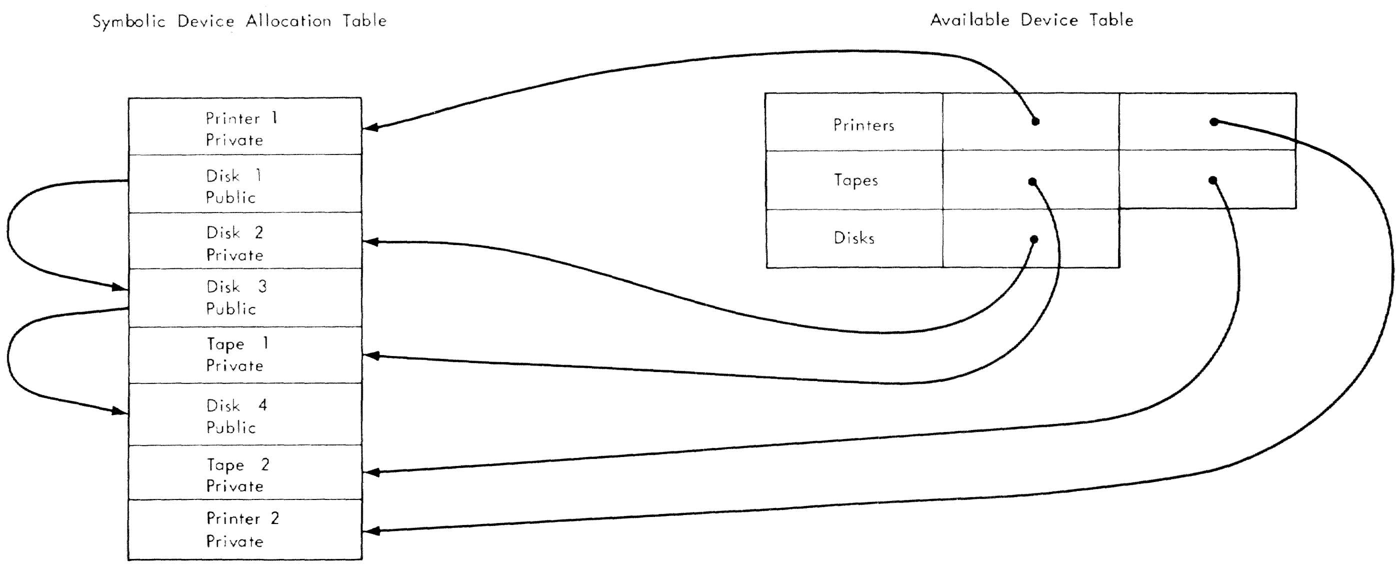


Figure 62. Major Virtual Storage Symbolic Device Status Tables

address of the volume's VTOC, and the number of logical cylinders per volume.

The Available Device Table (ADT) has an entry for every device that was not declared during System Generation to be a system, auxiliary storage or public device. These entries are grouped by device class and each entry points to its corresponding SDAT entry. In addition, a count is constantly maintained of the number of devices within each group that are tentatively available for allocation to users requesting private devices. Each major device class recognized by the DDEF command and each class of unit record device corresponds to an Available Device Table group entry.

Startup marks each SDAT entry as one of the following:

- Partitioned off from the system
- Nonexistent
- Unavailable (device is malfunctioning)
- Available

Device status is thereafter maintained by the HOLD and DROP command routines and the Private Device Management routines.

Startup also initializes the device dependent area of SDAT entries for Public devices and Auxiliary paging disks. The Private Device Management routine named PAUSE initializes this area for Private devices when they are mounted. The Command System routine GATE performs an equivalent service for terminals during task initialization.

The major functions performed in managing private devices can be categorized as follows:

- Ascertaining the availability of a device by device class or symbolic device address.
- Obtaining the device.
- Mounting a specific volume or printer form.
- Verifying that the correct volume has been mounted.

The handling of requests for private devices from a conversational task is functionally different from the handling of requests for private devices from a nonconversational task.

The conversational user requests private storage by means of a DDEF command. The DDEF command may be issued at any time during the terminal session before the device or data set is to be used (i.e., before OPEN). This allows the conversational user flexibility in structuring his terminal session. Device and data set information is obtained from the Catalog for a cataloged data set and from the DDEF command if the data set is not cataloged.

If a SAM organized data set is specified, only the first volume of a multivolume data set is mounted. If the command specified a VAM data set, all volumes are mounted. If the user did not specify data set organization, an installation assigned default value for data set organization is assumed (see "Data Management").

Devices are allocated and mounted one at a time, as they become available.

The DDEF routine invokes the Device Management routine named MOUNT REQUEST for both Public and Private devices. For Public devices, MOUNT REQUEST verifies that the volume is on-line and that the allocation of the device will not exceed the user's limit for that type of device. It then returns to the DDEF routine. For private devices, MOUNT REQUEST is invoked to determine if an appropriate drive is available and, if it is, to reserve it. If the specific volume the user requests is already mounted, MOUNT REQUEST reserves that device provided it is available. The Device Management routine PAUSE is then invoked, if necessary.

PAUSE sends a message to the Main Operator Task to instruct the operator to mount the user's medium - a volume or printer form (see "Communication"). The task is then placed in delay status (by issuing the AWAIT macro). If the user specified the SCRATCH keyword parameter in his DDEF command, the operator will select an appropriate scratch volume to mount. When the medium is mounted, the Main Operator Task sends a message back to the task and the task proceeds.

If the device or devices were not available, or the requested volume is being used by another task, the user's request is separated into device classes, sorted into appropriate subqueues within the Request Queue Table, and the task is placed in delay status.

The Request Queue table is located in shared virtual storage and contains a subqueue for each device class - much as the Available Device Table (ADT) does.

Entries within each subqueue are ordered by three categories of priority:

- Requests for a specific device or for a specific volume that is mounted but being used by another task.
- Requests by device class from conversational tasks.
- Requests by device class from nonconversational tasks.

Within each of these categories, entries are ordered on a first-in first-out basis.

As devices become available, either through tasks logging off or issuing RELEASE commands, the Device Management RELEAS routine searches the appropriate Request Queue table subqueues to find the highest priority request for each device or volume being released.

If RELEAS finds an outstanding request, the waiting task is informed that a device the task wants is available. (See the description of the Command System External Interrupt Processor.)

Volumes are not demounted when the device is released.

The Mount Request routine in the reactivated task proceeds to reserve the device and, if necessary, requests the Device Management PAUSE routine to issue messages to have the volume mounted.

PAUSE sends messages to the operator requesting him to mount volumes or ready unit record devices.

After PAUSE receives the operator's reply, PAUSE verifies the volume labels of direct access volumes and labeled tapes and verifies that unlabeled tapes are indeed unlabeled. A tape that is specified and found to be unlabeled will be accepted.

When all the devices requested by the DDEF command have been reserved and mounted, control is returned to the user.

At any time while the task is in delay status, the user may cancel his request by pressing the attention button on his terminal. This will result in control returning to the DDEF routine.

The nonconversational user presents special problems for private Device Management because of the system's inability to converse with the user.

Whenever devices are allocated dynamically, a possibility always exists that an interlock may develop between contending users. The following situation is an example. Two users are each using one of the two available tape units on a system. Both now require another drive to continue. The only way that either task can proceed is for one task to rescind its request. If either or both of the users are conversational, this is a readily available alternative. If a similar situation develops between nonconversational tasks, it could result in an endless wait in the queue. Indeed, because nonconversational tasks frequently have low dispatching priority within TSS/360, any wait for devices can result in a low priority task tying up other scarce system resources such as auxiliary storage for a prolonged period of time.

TSS/360 has attempted to resolve this problem by establishing the rule that a nonconversational user must state his total private device requirements through a SECURE command issued immediately after the LOGON command.

It might be possible to scan the input to the Batch Monitor to determine what devices a task will use. However, certain situations cannot easily be determined by examining the input. Examples are commands in the form of macros and the requirement for additional external storage extents. If all the devices requested in a SECURE command are not immediately available, the request is sorted into the Request Queue and the task is placed in delay status until all the requested devices have been reserved.

When the nonconversational user later issues DDEF commands, his private device requests will be allocated from his pool of already reserved devices (Reserve List) and his volumes or forms will be mounted as with conversational tasks.

If for any reason a nonconversational task requires more devices than it has reserved, the task will be abnormally terminated (ABEND).

Unit record devices are allocated in the same fashion as other private devices with a few exceptions.

First, any unit record devices specified in the SECURE command are not actively requested until all other needed devices have been reserved, making unit record devices the last devices a task obtains.

Second, only Command System Privilege Class E users or the BULKIO task may request unit record devices.

Bulk data transfers (such as printing data sets produced by language processors) are requested in TSS/360 through BULKIO issued by system users, language processors, etc.

These commands result in a work request being queued in a data set called the Batch Work Queue.

This queue is administered by the Batch Monitor task which creates nonconversational tasks to handle some entries and assigns those entries requiring public unit record devices to the BULKIO II task (task 002).

If the Batch Monitor creates a private background task (i.e., Print, Punch, RT, WT), this background task must issue its LOGON and DDEF commands. However, as with all nonconversational tasks, if the input or output is on private volumes, the private background task will be placed in delay status until all required devices are obtained.

The private Device Management routine BUMP is used by the BSAM end-of-volume routines, or when BSAM invokes the External Storage Allocation EXTEND routine. BUMP may also be required by the External Storage Allocation SCRATCH routine.

BUMP is used to demount a specific private volume and mount another volume on the same device. (BUMP can also be used merely to reverify the label of a mounted tape volume.) If a demount-and-mount operation is required, BUMP verifies its input parameters and instructs the system operator, via PAUSE, to mount the second volume. A user may concurrently process more than one data set on a private volume.This also applies to multi-volume VAM formatted data sets because all VAM volumes are mounted before processing begins.

On the other hand, SAM formatted volumes are mounted one at a time. Therefore, BUMP will abnormally terminate a task if the volume to be demounted contains an active data set other than the multi-volume data set on whose behalf the BUMP operation is being performed. In this case, the phrase "active data set" means a data set for which a DDEF command has been issued, but which has not been released through use of the RELEASE command.

There are two major Resident Supervisor tables concerned with device allocation:

- Task Symbolic Device List
- Pathfinding Tables

The Task Symbolic Device List (TSDL) contains an entry for each symbolic device assigned to a task. All I/O requests are checked against the TSDL. If the referenced device has no TSDL entry for this task, the Resident Supervisor rejects the request.

During virtual storage task initialization, the Add Device (ADDEV) SVC is used to place an entry for each Public device into the task's TSDI.

Whenever the allocation and mounting of a private device is completed, MOUNT REQUEST issues an Add Device SVC for the device.

Whenever the Resident Supervisor receives an initial attention interruption from a terminal, the Resident Supervisor creates a task for that terminal. In so doing, the Resident Supervisor places an entry in the task's TSDL and a pointer to the new task's TSI in the Pathfinding Device Group Table entry for the terminal. Thereafter, unsolicited interrupts from the terminal will be properly routed to the task.

During the final stages of LOGOFF processing, the TAM CLOSE routine will issue a Set Asynchronous Entry (SETAE) SVC to remove the terminal from the task's TSDL and to remove the TSI pointer from the terminal's Pathfinding entry.

PATHFINDING

The status of the actual devices, control units and channels of the system is described and maintained by the Pathfinding tables.

The Pathfinding tables are constructed during System Generation from information supplied in the Configuration macro instructions. These tables are initialized by the Startup program to reflect system components which are partitioned, nonexistent or malfunctioning.

The status of these system components is thereafter maintained by the Pathfinding subroutine. This subroutine is primarily used by the Device Queue and Channel Interrupt processors of the Resident Supervisor and by the Configuration Control Subsystem in response to HOLD or DROP commands. The Configuration Control subsystem invokes the Pathfinding subroutine through the Set Path (SPATH) SVC.

The Pathfinding subroutine performs four major functions:

- Finds and reserves an available path to a device, normally for the duration of one I/O operation. This is called Pathfinding.
- Releases all or a part of a path when an I/O operation is completed. This is called Reverse Pathfinding.
- Sets all or part of a path to partitioned or malfunctioning status. This is called Set Path processing.
- Determines if a terminal is attached to a task and obtains a symbolic device address from a hardware device address. This is called Translate-only Reverse Pathfinding.

The Pathfinding group of tables consists of:

- A Symbolic-to-Actual-Address Conversion table
- One Device Group table for each device control unit or switch unit
- A Channel Table
- One Control-Unit-Assigned-to-Channel table for each channel
- A Control Unit Table

These tables can best be understood if a walk-through of the operations for locating an available path to a device is performed. This operation is schematically depicted in Figure 63.

TSS/360 uses the program maintained Pathfinding tables to determine if a device is busy instead of merely attempting to physically address a device. This is because it is expected that in a typical environment there will exist multiple paths to most devices. In such a situation, the efficiency of I/O processing can be increased by reducing the number of "busy" or "unavailable" conditions encountered during an attempt to initiate an I/O operation. In addition, the use of common pathfinding tables assists in synchronizing I/O processing in a multiprocessing environment. This is because an I/O interruption may be accepted by an available CPU, not just the CPU that initiated the I/O.

The TSS/360 access methods deal only with symbolic device addresses. Therefore, the first function of the Pathfinding subroutine is to convert this symbolic address to a hardware device address.

The Symbolic-to-Actual Address Conversion table is used for this purpose. This table contains fixed length entries and there is an entry for each symbolic device address. The purpose of this arrangement is to enable symbolic device addresses to be converted to actual addresses by a direct look-up scheme. Each entry contains an actual device address corresponding to the symbolic device address and a pointer to the Device Group Table associated with the device.

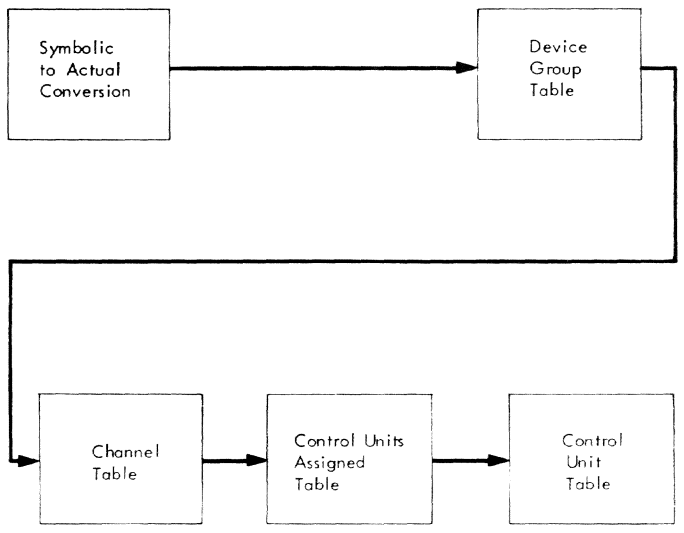


Figure 63. Pathfinding

A Device Group table contains entries for each device that is addressable through a particular control or switch unit (e.g. 2816 Tape Switch Unit) and a listing of one or more 9 bit paths to that control unit.

A complete physical address can be separated into one or more 9-bit paths to a control unit or switch unit and a 4-bit device address. The reason for organizing device status information in such a fashion is merely to save having to repeat the 9-bit path information in each device entry.

The maximum number of device entries in a Device Group table is 16. Few control units can handle 16 I/O units. However, the IBM 2702 Transmission Control Unit is an example of such a control unit. Most control units can access up to eight devices or access arms.

If the control unit for the device or devices is equipped with a two-channel switch, then all the devices on that control unit can be reached via their control unit from either of two channels. That is, there are two paths to the devices.

The maximum number of paths that can be specified in a Device Group table is eight. Eight paths to a device could arise in the case where an IBM 2316 Tape Switch is connected to four tape control units, each of which has the two-channel switch feature. (See Figure 64).

The device entry in a Device Group table indicates whether the device is parti-

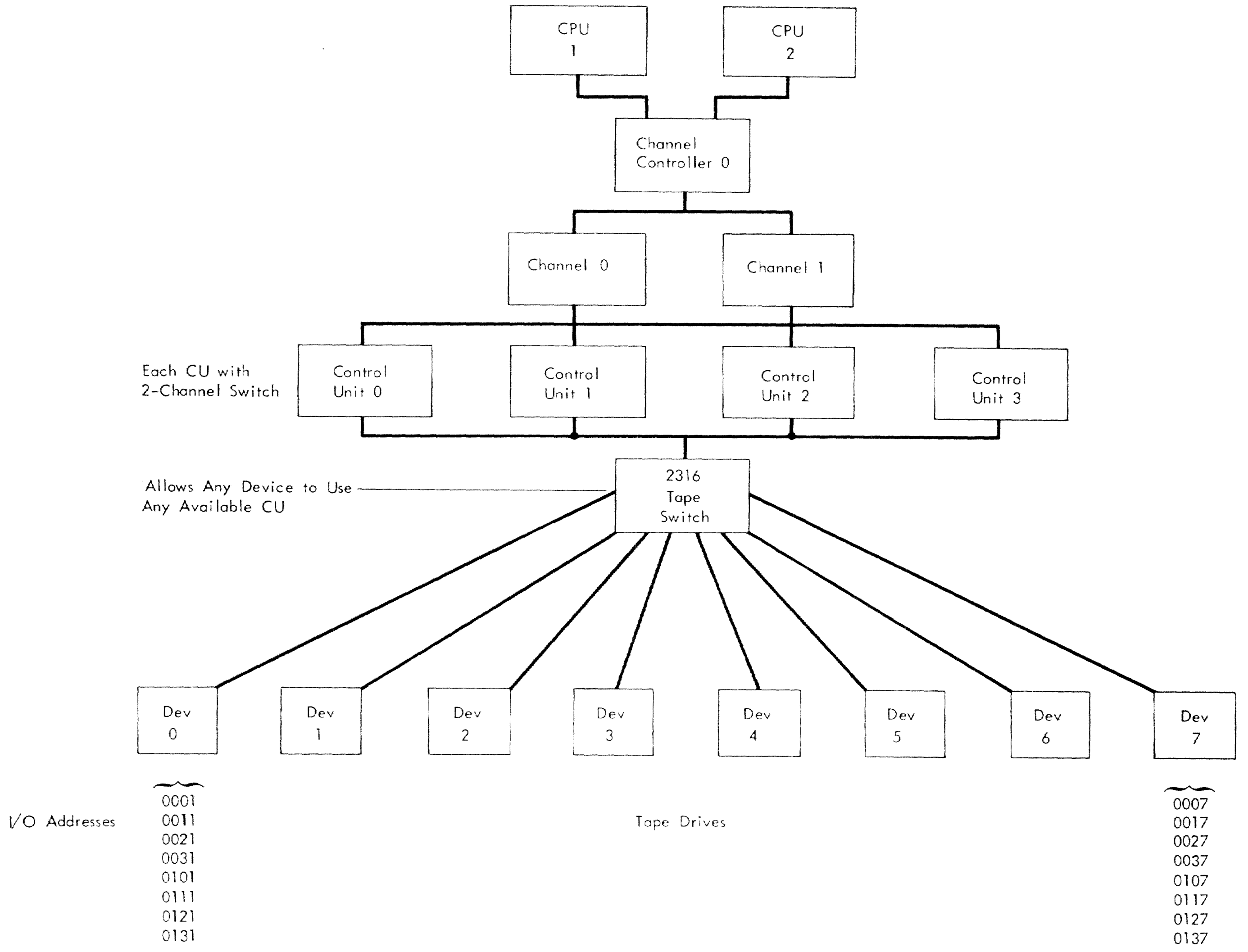


Figure 64. Tape Switch Connected to Four Tape Control Units with Two-Channel Switch Feature

tioned, nonexistent, malfunctioning or busy. If the device is available, it is marked busy and a 9-bit path to the device's control unit is selected.

The first five bits of a 13-bit address are used to locate the appropriate channel entry in the Channel Table. The availability of the IBM 2846 Channel Control Unit (CCU) is not specifically checked, because if a channel is obtained for use, the Channel Control Unit to which the channel is attached is also available.

There is one entry in the Channel Table for each channel on the system. If the channel is not available, the search will return to the appropriate Device Group table and select another path, if possible.

If the channel is available, it is marked busy (unless it is a multiplexor channel) and a pointer to the Control Units Assigned to Channel table associated with the channel is picked up.

A Control Units Assigned to Channel table has entries for each device control unit on the channel (as represented by the low-order four bits of the 9-bit path obtained from the Device Group table).

A device control unit equipped with a two-channel switch may be logically disconnected from one of the channels to which it is attached through a flag in the appropriate Control Units Assigned to Channel table. This facility to logically disconnect a device control unit from a channel without physical partitioning is a useful feature when error control routines are attempting to pinpoint the source of an I/O error.

Thus, the primary function of Control Units Assigned to Channel table is to maintain the status of the connection between a particular multiplexor or selector channel on the one hand, and each control unit attached to the channel, on the other hand.

The Control Units Assigned to Channel table for a multiplexor channel contains, in addition, an entry from which the cumulative total of the data rates for each subchannel currently in use can be inferred. The "weight" that each newly activated device will add to the multiplexor channel is obtained from the Device Group table entry for that device. With this information, it is possible to keep from overloading a multiplexor channel.

If the control unit is logically connected to the channel, a pointer to a control unit entry in the Control Unit Table is picked up. The information contained in each Control Units Assigned to Channel table could have been placed in the Control Unit Table. It was organized separately because space is saved by not having to maintain two entries for control units which are permanently attached to only one channel.

The Control Unit Table has an entry for each control unit in the system. Each Control Unit Table entry contains status information and a pointer back to the Device Group Table associated with the control unit. The Device Group Table pointer will be used to mark the device not busy after the I/O operation is completed. Because device status information is organized around Device Group Tables, only one pointer is needed to locate the status entry for any device on the control unit.

Under certain conditions, the device address stored upon a Control Unit End interruption can not be relied upon. Therefore, the symbolic device address associated with an I/O operation is temporarily placed in a field within the appropriate control unit entry. This symbolic device address will be retrieved during reverse pathfinding.

One aspect of the format of the control Unit Table is connected with the fact that 2702 Transmission Control Unit may have up to 31 lines on it. Since the 13-bit physical I/O address only allows four bits for a device address such control units are given two addresses in order to provide a unique address for each device attached to the control unit.

Maintaining the status of one control unit in two Control Unit Table entries is awkward. So, one of the entries (called the "Child" entry) merely points to the other entry - called the "Parent" entry, which contains the status information for the control unit.

If the control unit is not available, we return to the Device Group table and select another path, if possible.

If the control unit is available, it is marked busy and pathfinding is complete.

After an I/O operation is complete, it is desirable to free the units that were flagged as "busy." This process is called Reverse Pathfinding and is depicted in Figure 65. Reverse Pathfinding, as its name implies, is just the reverse of the processing performed during Pathfinding. If the I/O operation encountered an error, the system may wish to obtain more information about the error. In such a case, only parts of the path will be immediately released.

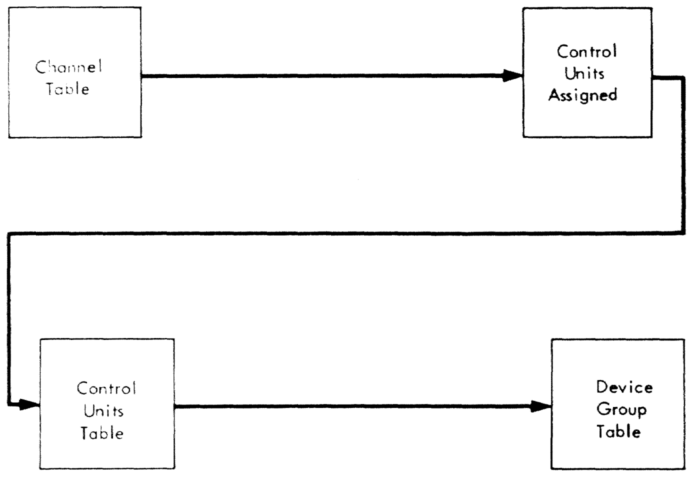


Figure 65. Reverse Pathfinding Set Path

TIMER SERVICES ALLOCATION

There is a requirement within a time sharing system for the maintainance of various forms of elapsed time. The user requires the ability to set a timer which will measure the time of his task's execution or the elapsed calendar time. The system requires the measurement of time slices and of an interval of time that a CPU may be allowed to remain in wait state before going to the Queue Scanner.

There is one hardware timer for each CPU in the system. TSS/360 uses the services of those timers in maintaining four classes of time:

- CPU interval time
- CPU elapsed time
- Task interval time
- Task elapsed time

CPU Interval Time

Intervals of time are needed by the system to measure time-slices for tasks, to measure user time intervals, and to limit the time that a CPU may remain in wait state. These intervals are measured by setting the timer to the required value and letting an interruption signal the end of the time interval. If a task is running and is interrupted, the value in the timer is saved in the current timer cell in the task's XTSI for use the next time the task gains control of a CPU. However, if the interruption signals time slice end or the end of a user specified time interval, the current timer must be re-initialized.

When the Dispatcher determines that it has no work for a CPU, the timer is set to a predetermined value and the CPU is put in the wait state. This allows an outstanding event or the timer to cause an interrupt. Thus, for a multi-CPU system, an idle CPU may be re-activated by an interrupt so that any GQEs generated by other CPUs can be processed.

CPU Elapsed Time

The Model 67 timer is an interval timer. Therefore, it is necessary that the elapsed time since system startup be updated each time the timer is reset with a new interval. This resetting happens whenever an interrupt occurs, whenever a task is put in execution, and whenever the CPU enters wait state. For any interrupt the timer value is saved and a new timer base value is stored in the timer. Then the saved value of the timer is subtracted from the timer's old base value (the value last set before interruption). This result is then added to the elapsed time as measured by the CPU which was interrupted. The above procedure is performed by the Interrupt Stacker.

Any time that the Interrupt Stacker gains control, the maximum value is set in the timer. This is done to prevent a timer interrupt from occurring while the Resident Supervisor is processing work. The Dispatcher will place a new base value in the timer and update the CPU elapsed time before giving the CPU to a task or placing the CPU in the Wait state. Whenever a timer interrupt occurs, a test is made of the Old External PSW to determine if the CPU was in the Supervisor state. If the CPU was in Supervisor state, an error exists unless the CPU was also in the wait state. If the CPU was in both Supervisor and wait states, the increment of elapsed time is added to a cell in the PSA in order to record the time spent in wait state. Then control passes to the Queue Scanner to search for outstanding work.

Task Interval Time

The Task Monitor Timer Interrupt Processor provides the processing of the two types of task timer interruptions "real time" and "task time." Real time refers to an actual time of day value. Task time refers to the accumulated task time, a value maintained in the XTSI of each task by the Resident Supervisor.

The task can have 32 timer interrupts queued up for processing, 16 real time and 16 task time types. However, to eliminate the unwarranted amount of overhead that could exist if the Resident Supervisor had to maintain 32 timer values for each task, the Task Monitor Interrupt Processor maintains the 32 clock values (timer interrupt intervals desired) in its PSECT and presents them to the Resident Supervisor one at a time. That is, a second timer interrupt request is presented to the Resident Supervisor only after the first timer interrupt is received by the Task Monitor. This scheduling is possible because of the sequential nature of timer interrupts.

A privileged program may request a future real time or task time interruption through use of the Specify Timer Entry Condition (STEC), Specify Interrupt Routine (SIR) and Interrupt Inquiry (INTINQ) macro instructions.

If a nonprivileged routine uses these macros an ENTER SVC is generated when each macro is executed. The amount of time required to process all these supervisor calls might be sufficient to distort a request for an interrupt after a small time interval. Therefore, nonprivileged routines use the Set Timer (STIMER) or Test Timer (TTIMER) macro instructions. Both macros expand into an ENTER SVC linkage to the STIMER-TTIMER system service routine which issues the STEC, SIR or INTINQ macros on behalf of the nonprivileged routine.

In order that the user may specify at any time a new time interval value representing the time that the task will run, the TIME command may be used. Upon elapse of the specified time interval, this routine will ABEND a task in background mode or notify the user in conversational mode (see "Command Routines").

For tasks to be able to set their own CPU interval, a cell in each task's XTSI is defined which contains the time remaining in a task's specified interval. This interval is set by means of a Set User Interval Timer (SETTU) SVC issued from the Task Monitor.

When such an interval is defined, the length of time for the task's execution is taken as the smaller of the system time slice value (in the System Table) and the time remaining in an interval specified by the task. Thus a task may actually get less than the CPU interval specified.

The Timer Interrupt Processor recognizes a user timer interruption and causes a Task Timer Interrupt GQE to be enqueued on the task's TSI. The Timer Interrupt Processor must also adjust the user's last time slice value to indicate the amount of time remaining in his time slice and update the user's accumulated CPU time. Both of these values are kept in the task's XTSI.

Task Elapsed Time

A request for a Real Time task timer interruption will cause the Set Real Time Interval (SETTR) SVC to be executed by the Task Monitor. An entry in a list in supervisor storage will be constructed which will identify the task which issued the SVC and the time at which a task timer interrupt should be constructed. This list is called the real time interrupt pending queue. The queue entries are arranged in order of increasing real time. The entry whose time value is closest to the present time is first in the queue. The Dispatcher must compare the first queue entry with the system elapsed time and link to its Create Real Time Interrupt Subroutine if the creation of a task interruption is indicated. This subroutine will remove the first entry in the queue, adjust the list, and create a task timer interruption for the proper TSI.

INTRODUCTION

This section is presented in three parts. Part one presents an overview of the Dynamic Loader, part two an overview of the Task Dictionary Table, and part three describes the Dynamic Loader processing in more depth.

The traditional function of a loader has been to cause object program modules to be read from external storage into main storage before program execution begins. This involves the allocation of main storage to the program. The loading process usually resulted in the object program being loaded into a location other than that specified by the language processor. This process is called relocation. In IBM System/360 systems, relocation involves only the adjustment of values contained in address constants.

Thus, the loading process can usefully be characterized in terms of three functions:

- When loading is performed.
- How allocation is performed.
- How relocation is performed.

In TSS/360 the loading function may be performed dynamically. That is, during execution one program may reference another program not previously processed by the Dynamic Loader. This is another of the means by which a TSS/360 user is given flexibility in conducting a terminal session.

The TSS/360 Dynamic Loader resides in virtual storage. The basic function of the loader is to load programs into virtual, not main, storage and to relocate only those address constants contained in pages of text actually referenced during execution of the program.

The process of loading a program into virtual storage does not involve the movement of any text and is performed by the Allocation phase of the Dynamic Loader.

Loading a program into virtual storage consists, in large part, of establishing the addressability of the program within virtual storage.

This is done by calculating the virtual storage addresses (i.e., the V and R values) by which the program is to be referenced; by modifying the task's relocation tables so that they map the external storage locations of the object module text into virtual storage; and by calculating the virtual storage address used in adjusting address constants.

When the Dynamic Loader (Allocation phase) is invoked, it utilizes the services of the Virtual Access Method to locate the partitioned data set containing the wanted object program module and to cause the module's Program Module Dictionary (PMD) to be mapped into virtual storage. The contents of this PMD are placed in a private task table called Task Dictionary.

Utilizing information from the PMD, the loader invokes the services of the Virtual Memory Allocation to allocate virtual storage for the object module text. The Dynamic Loader then utilizes the services of the Virtual Access Method MOVEPAGE Routine to place the external storage addresses of the module's text pages into the appropriate External Page Table entries. For each text page that contains address constants, an "unprocessed by loader" flag will be set on in the page's External Page Table entry.

During this Allocation phase, the Dynamic Loader, among other functions, examines all external references of the module and obtains and processes the PMDs for any additional object program modules required to satisfy these external references.

This process results in the Dynamic Loader recursively invoking itself if additional PMDs must be obtained. This recursive processing will continue as long as the resolution of external references requires loading the PMDs for additional object modules.

When the allocation phase is complete, the Dynamic Loader exits, supplying the V and R values which point to entry points in the loaded program. Further references using these values are direct. That is, they do not require the services of the Dynamic Loader or the use of indirect addressing.

The actual relocation of address constants is performed on one page at a time when that page is referenced and consequently paged into main storage during program execution. The process of relocating address constants involves applying the values calculated during the allocation phase of the loader. This second phase of the Dynamic Loader is called the Relocation phase.

As pages of the loaded program are referenced for the first time, relocation exception interruptions will occur, because the referenced pages are still on external storage.

After the Resident Supervisor has caused a referenced page to be brought into main storage, a flag in the External Page Table entry for this page is inspected.

The MOVEPAGE routine sets this flag via the Set External Page Table Entry (SETXP) supervisor call during the Dynamic Loader's allocation phase if the PMD indicated that the page contains one or more address constants.

If the flag is off, no special processing is performed by the loader and this relocation exception is treated as any other (see "Paging").

If the flag is on, this indicates that the page contains address constants. In this case, the Resident Supervisor will create a task interruption that will result in the invocation of the Dynamic Loader.

Because this page has never been in main storage until now, the address constants contain whatever values were placed in them by the language processor.

In order to place the proper values into each address constant, the Dynamic Loader is invoked at its Relocation phase entry point.

After all address constants on the page have been relocated, control is returned to the point in virtual storage where the page was referenced.

In summary, the primary functions of the Dynamic Loader include allocating virtual storage for a task's programs and relocating address constants for only those pages of text actually referenced during execution. The Dynamic Loader never loads program text into main storage. Text is brought into main storage by paging operations.

A secondary function of the Dynamic Loader is to enforce certain TSS/360 protection rules concerning the loading and referencing of program modules. This is discussed in the following sections.

As mentioned above, the Dynamic Loader loads and processes all needed PMDs during the Allocation phase. An alternative scheme would be to load and process all PMDs after the first one only during the Relocation phase, and then only if the PMDs are actually needed to satisfy an address constant located in the page being processed. This scheme was rejected in favor of the current design for the following reasons:

- Most system service routines used by the Dynamic Loader also frequently reference user pages. Because any user page could contain unprocessed address constants, these system routines would have to be recursively written.
- Paging would be increased if the Task Dictionary had to be searched by both Dynamic Loader phases.

TASK DICTIONARY

In performing its functions, the Dynamic Loader uses a table called a Task Dictionary (TDY).

The TDY contains the information needed to load (and unload) modules for a particular task. The organization of a TDY is shown schematically in Figure 66.

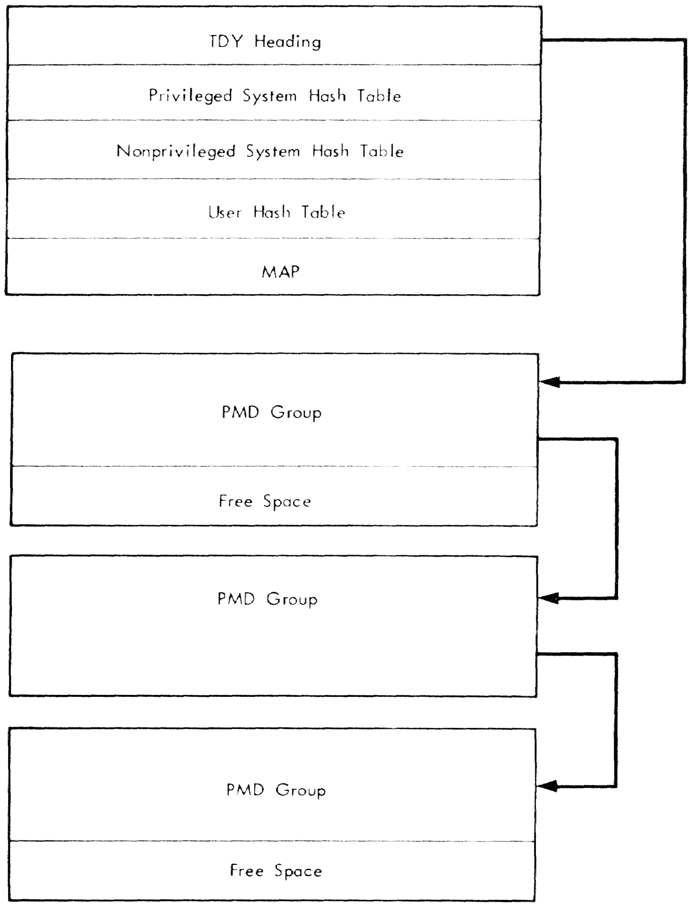


Figure 66. Task Dictionary Organization

A TDY consists of a heading, three hash tables, a storage Map table (MAP), and one program Module Dictionary for each module loaded for this task.

The PMDs are placed in the TDY in the same order that the modules are loaded by the loader (and STARTUP). Because of the variable length of each PMD, they are chained together.

An extensive TDY, built by STARTUP, describes a task's Initial Virtual Memory (IVM). This TDY is stored on auxiliary storage by STARTUP and its location is described in an External Page Table contained in the skeletal XTSI assigned to a task when it is created. The pages occupied by this initial TDY are not normally referenced after system startup.

PMD space is allocated within the TDY on a group basis. The primary purpose of grouping PMDs is to conserve storage. An integral number of pages of virtual storage is obtained for the first PMD of a group. However, this PMD may not occupy all of the space in the last (or only) page so obtained. If the next PMD is less than a page in size, it may fit into the space between the end of the preceding PMD and the end of the page. Otherwise, GETMAIN is called to obtain storage for this PMD which becomes the first member of a new PMD group.

The TDY heading defines the beginning of the TDY and contains pointers to the User, and System, hash tables; and to the Storage Map Table.

HASH TABLES

In order to link programs dynamically, the Dynamic Loader must be able to look up all external symbol definitions.

To make the process of looking up external symbol definitions more efficient, hash tables are used. A hash table consists of a header and a number of hash chains.

A hash table header consists of a number of word length entries which either contain zeros or a pointer to an external symbol definition entry in a PMD.

Whenever a new PMD is to be placed into a task's TDY, an arithmetic or logical operation ("hashing") is performed on the alphanumeric name of each external symbol definition. This operation is such that the result will be a whole number that is not larger than the number of word length entries in the hash table header. This number is then used to form an index to inspect one of the hash table header entries. If the entry is zero, a pointer is placed in this hash table header entry. This pointer will identify the location of the external symbol definition entry associated with this symbol. (See Figure 67.)

If the entry is not zero, this means that the hashing algorithm has already produced the same whole number for some previously processed external symbol definition.

In this case, the PMD entry for each such external symbol definition (called "synonym") will be chained to the previous entry. In this fashion, the TDY can be treated as a large number of small tables and the entry for any external symbol definition can be found by inspecting only one of these small tables.

To make this process even more efficient and to prevent a nonprivileged user from accidentally linking to a system routine, or a system routine from erroneously link-

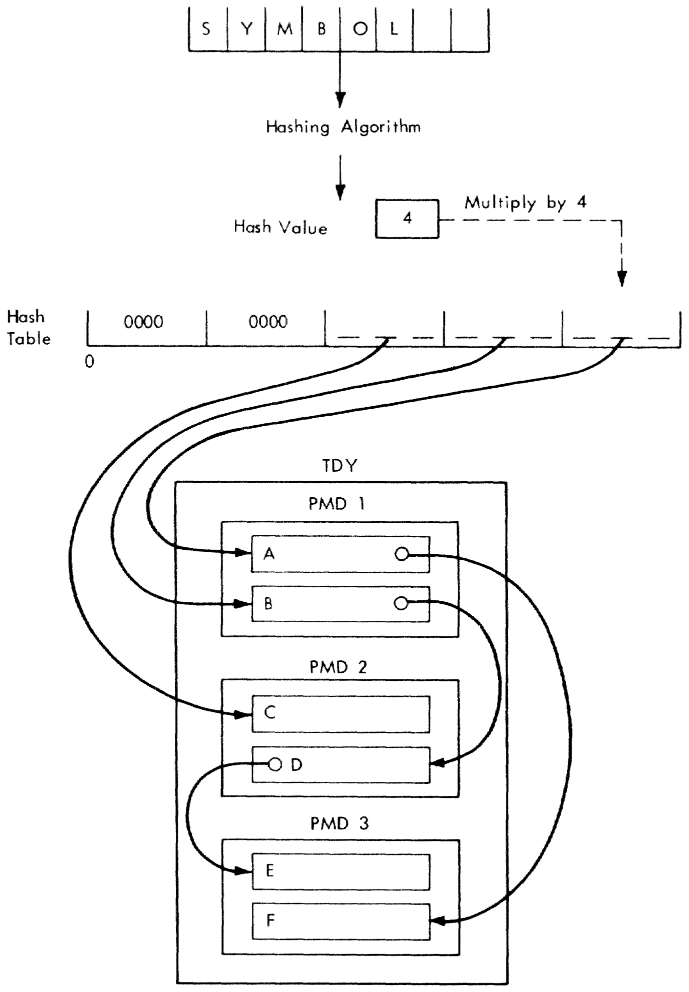


Figure 67. Hash Table Processing

ing to a nonprivileged user routine, three hash table headers are defined: Privileged System, Nonprivileged System, and User hash tables.

Two of these tables are used for system symbols (i.e., external symbols found in routines extracted from SYSLIB or SYSIVM whose control sections are marked "System" or "Privileged"). External symbols defined in control sections with the privileged attribute must begin with the letters CZ or CHB and will be processed in the Privileged System hash table (SYSHASHP). All privileged system routines provided by IBM are contained in the Initial Virtual Memory dataset (SYSIVM) which is link-loaded at system startup.

The Nonprivileged System hash table (SYSHASHNP) will contain nonprivileged system symbols. A convention has been adopted that the initial entry points of those nonprivileged routines that are to be directly invoked by a nonprivileged user, must begin with the letters SYS. An example of such a nonprivileged system routine is the Language Processor Control module, which contains such initial entry points as SYSASM and SYSFTN used for invoking the TSS/360 Assembler and Fortran language processors.

System efficiency is enhanced by providing two system hash tables - Privileged and Nonprivileged - instead of just one system hash table. As a result, the Dynamic Loader does not have to search through a hash chain containing a large number of privileged symbols defined in Privileged Initial Virtual Memory routines when it is attempting to resolve references to nonprivileged system symbols.

In addition to the system tables, a third hash table is constructed for the normal user (authority U). The use of this additional table gives rise to the term "split hash." The purpose of this hash table is primarily one of protection. It is employed to provide close control over the interface between the normal user and system routines. This control is effected by separating the normal user's symbols from system symbols. The normal user's external symbol definitions may begin with any characters (including CZ and CHB) except SYS and are processed in the User hash table (USERHASH).

The general rules for processing external definitions and references are as follows.

The normal user's external symbol definitions are placed in the User hash table and the external symbol definitions from system control sections are placed in the appropriate system hash table.

System symbol definitions can only be inserted in the System hash tables if the program module with the "system" attribute was loaded from the System Library or the System IVM datasets.

Only a privileged system programmer is permitted to stow an object program module in these data sets. This means that a non privileged user may not directly substitute his own copy of a system routine.

External symbol references (except those beginning with the letters SYS) are satisfied from the hash table with the same name as the privilege attribute of the control section containing the reference.

Thus, a privileged routine can not directly link to a user routine, but must use the Type III (Leave Privilege) linkage mechanism or the Load Virtual PSW supervisor call. There is a special provision that allows the privileged Command System LOAD routine to load a nonprivileged user routine on behalf of the user.

A user program can not directly link to a privileged routine, but must use the Type II (Enter) linkage mechanism.

A nonprivileged system program, such as the TSS/360 Assembler, can only directly link to another nonprivileged system routine. A nonprivileged system routine can not link to a user routine and can only link to a privileged routine through a Type II linkage.

User routines and privileged routines can only link to nonprivileged system routines through external symbol references beginning with the letters SYS.

For instance, whenever a nonprivileged program makes an external reference to a system symbol beginning with the letters SYS, the Nonprivileged System hash table (not the User hash table) will be used in attempting to locate the external symbol definition entry. If the symbol is not contained in the hash table, the Dynamic Loader knows from the letters SYS that only the System library need be searched, not the entire heirachy of open libraries.

In addition to the protection function, the presence of a nonprivileged System hash table and the restriction on defining SYS symbols relieves the user of the necessity of knowing what external symbols are used by modules contained in the system library. That is, the user can employ an external symbol with the same name as an external symbol definition in the System library without fear of confusion.

For a privileged user, (i.e., "O" or "P" authority codes) only the two System hash tables are constructed. The User hash table pointer in the TDY heading is set to point to the System Hash tables. In this fashion, protection is relaxed for the privileged user and all external symbol references are resolved in the two system hash tables. The primary purpose for relaxing protection is to allow system programmers greater latitude in testing system programs. These extended capabilities apply only to system routines loaded by the Dynamic Loader. All Initial Virtual Memory modules are loaded by the Startup program and are insensitive to authority codes.

The rules for posting external symbol definitions in hash tables are summarized in Figure 68. The rules for searching these hash tables in order to resolve external symbol references are described in Figure 69. Those terms and relationships which have not yet been defined are discussed in the following sections. A few additional considerations, involving internal labels used in expansion of macro instructions, are contained in the appendix section of <u>Assembler Programmer's Guide</u>.

STORAGE MAP TABLE

The Storage Map Table is an ordered table which contains the virtual storage address of each control section which has been loaded into the user's virtual storage and the virtual storage address of the Control Section Dictionary for that control section.

The Storage Map Table is maintained in ascending order of virtual storage addresses, thus facilitating a binary search for lookup purposes. For example, this table is used during the relocation phase of the Dynamic Loader to find the Control Section Dictionary associated with

1	2	3	4		5	6
if	and	then	and if		then	and
authority class is:	control section that contains DEF came from:	control section attributes may be altered:	control section that contains DEF has attributes:		DEFs may begin with only the symbols:	all legal symbols from control section are posted in:
U	SYSLIB	If control section is PRVLGD, loader sets SYSTEM attribute; hence, NON-SYSTEM <u>and</u> PRVLGD is impossible.	SYSTEM	PRVLGD	CZ, CHB	SYSHASHP
				NONPRVLGD	CZ, CHB	SYSHASHNP
					All others	SYSHASHNP
			NONSYSTEM		Any but SYS	USERHASH
	USERLIB or JOBLIB	PRVLGD and SYSTEM erased.	NA*			
P	SYSLIB	PUBLIC and READONLY erased.	NA*		Any	SYSHASHP or SYSHASP
	USERLIB or JOBLIB	PUBLIC, READONLY, PRVLGD, SYSTEM erased.			Any but CZ, CHB	SYSHASHNP
O	NA*	PUBLIC and READONLY erased.	NA*		Any	SYSHASHP or SYSHASHNP
*NA - not applicable, in the sense that the condition is not tested by the loader.						

Figure 68. Symbolic Posting Rules for Inserting DEFs Into the Task Dictionary Hash Chains

1	2	3	4	5	
IF	AND	AND	AND	THEN	
Authority class is	Loader is resolving symbol from	High-order bit of C1 or C3 byte of adcon group is	Control section containing adcon group or REF is	Look up symbols in which hash table (or)	Search which libraries if symbol not found in hash table.
"U"	Explicit LOAD/ CALL or DELETE adcon group.	0	SYSTEM	SYSHASHP or SYSHASHNP	SYSLIB
			NON-SYSTEM	USERHASH***	ALL**
		1	NON-SYSTEM	SYSHASHP or SYSHASHNP	SYSLIB
			SYSTEM	USERHASH***	ALL**
	External REF	NA*	SYSTEM	SYSHASHP or SYSHASHNP	SYSLIB
			NON-SYSTEM	USERHASH***	ALL**
"P" or "O"	NA*	NA*	NA*	SYSHASHP or SYSHASHNP	ALL**

Notes: *NA means not applicable in the sense that the condition is not tested by the loader.

**ALL means the entire hierarchy of open libraries beginning at the last defined JOBLIB and ending with SYSLIB (or with that library yielding a valid definition).

***If the symbol to be resolved begins with the letters "SYS," the loader will always look only in SYSHASHP or SYSHASHNP (and SYSLIB if not found in SYSHASHP or SYSHASHNP).

Figure 69. Dynamic Loader Symbol Lookup Rules for Resolving Symbols in Either Explicit CALL/LOAD or DELETE Adcon Groups or in External REFs

the virtual storage address at which a relocation exception interruption occurred. A Control Section Dictionary will contain the Relocation Dictionaries needed to properly relocate the address constants contained in the text associated with the control section. The Storage Map Table may also be used by the Program Control System.

Control Section Dictionary (CSD)

Contained within each PMD are one or more Control Section Dictionaries (CSD).

Each CSD is made up of the following components:

- CSD Heading
- Definition Tables
- Reference Table
- Relocation Dictionaries
- Virtual Memory Page Table (VMPT)

CSD Heading

The CSD heading contains general information such as the attributes possessed by the control section and the length of the control section.

Definition and Reference Tables

When a TSS/360 user issues a LOAD or RUN command, he specifies a symbol naming an entry point in the module to be loaded. This symbol may be the module name, a control section name, or an ENTRY symbol.

When a TSS/360 user compiles a program he submits a list of parameters for the language processor. One of these parameters is a name for the object module. This module name is automatically treated by the language processor as an external symbol. The V value associated with the module name is called the standard entry point of the module.

The name of each control section defined in a module is also automatically treated by the TSS/360 Language Processors as an external symbol.

The TSS/360 Assembler user may explicitly request that a symbol defined in that assembly be treated as an external symbol by naming the symbol in the operand of an ENTRY statement.

For each external symbol defined in a module, the Language Processors create an external symbol definition entry (DEF). Thus, DEFs generally represent points within a module at which data is to be accessed or to which control is to be transfered.

There are three types of DEF: Absolute, Relocatable, and Complex.

An absolute DEF is an external symbol whose value is not dependent upon the virtual storage location of the module. An external symbol defined by an assembly language Equate (EQU) statement whose operand is an absolute expression would cause the creation of an absolute DEF. For an absolute DEF, the language processor places the value of the symbol in the DEF. The Dynamic Loader does not modify this value in any way.

Relocatable DEFs are those whose values are virtual storage location-dependent. For example, a relocatable DEF will be created for a control section name.

For a relocatable DEF, the language processor places a value representing the symbol's displacement from the origin of its control section into the value field of the DEF. The R value field will be zero.

After the Dynamic Loader has established the virtual storage address of the control section, the V and R value fields of the relocatable DEF are processed by adding the base address of the control section to the values assigned by the language processor.

A complex DEF is one which requires information concerning the relocation of one or more control sections other than the one in whose CSD the DEF resides.

Before further describing a complex DEF, it is necessary to discuss external symbol reference entries (REFs).

A language processor will create an external symbol reference entry (REF) for each symbol named in the operand field of an EXTERN statement or in V and R type address constants. Such REFs are called external REFs and represent symbols referrred to within that program module but defined in another. (i.e., separately assembled) program module.

A language processor may also create a REF for a control section name within the module in connection with a complex DEF or for an internal symbol defined in one control section but used in an address constant in another control section within the same module. Such a REF is called an internal REF.

The Dynamic Loader tries to satisfy each REF by obtaining the appropriate V and R values from a corresponding DEF. The DEF which satisfies a REF may be located in the same or another PMD. The V and R value fields of REFs are generally used for relocating address constants.

DEFs are contained in DEF tables and REFs are contained in REF tables within the appropriate control section dictionaries, with one exception. One complex DEF and an associated REF are placed in the PMD heading. This is the complex DEF for the standard entry point associated with the module name.

A complex DEF is also constructed for a symbol defined by an Equate (EQU) statement whose operand field contains one or more external symbols. However, of more interest is the fact that a complex DEF is created for a relocatable DEF whose associated ENTRY statement appears in a control section other than the one in which the symbol itself is defined. This is the means by which a user causes association of a particular Prototype control section (PSECT) with an entry point name.

For example, consider the following assembler language statements:

```
PSECTAA   PSECT
          ENTRY JOE
          .
          .
          .
CSECTBB   CSECT
          .
          .
          .
JOE       LR 3,4
          .
          .
          .
```

A complex DEF will be created for JOE and placed in the CSD of PSECTAA. Note that the DEF is placed in the CSD of the control section is which the entry statement appears, not the CSD in which JOE is defined. The language processor will place a value representing the displacement of JOE from the beginning of CSECTBB into the V-value field of the complex DEF. The R-value field will be zero.

In addition, the language processor will construct a REF for the control section name CSECTBB which will also be placed in the CSD of PSECTAA.

The language processor automatically constructs relocatable DEFs for the two control section names and places them in the CSDs for their respective control section.

During the loading process, the Dynamic Loader will first process relocatable DEFs. As previously described, this involves adding the base address of each control section to the relocatable DEFs in that control section's CSD. At this time, the loader will also partially process complex DEFs by adding the base for the control section to the R-value entries only of the complex DEFs in the CSD. (The R-value for the module name is obtained somewhat differently and is not discussed here).

Thus, the R-value field for the complex DEF created for JOE will contain the base address of control section PSECTAA.

The Dynamic Loader next processes REFs and then completes the processing of complex DEFs. The REF for CSECTBB will have its V and R-value fields filled in from the DEF for CSECTBB.

Lastly, the V-value of the complex DEF will be adjusted by adding the V-value from the REF for CSECTBB to the V-value field in the complex DEF.

The Dynamic Loader knows that the V-value of this particular REF is to be applied to this particular field in the complex DEF in the same way that it knows what values to use in adjusting address constants. In this sense, the V-value field of a complex DEF can be considered to be an address constant.

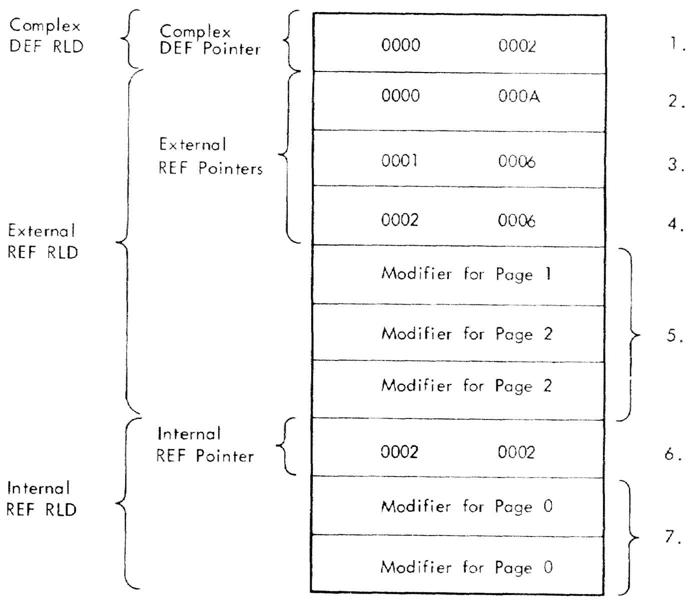
Relocation Dictionaries

Relocation Dictionaries are used by the Dynamic Loader's Allocation phase to modify complex DEFs and by the loader's Relocation phase to relocate address constants.

Three Relocation Dictionaries (RLDs) appear in each control section dictionary. The three RLDs are, in order, for REFs associated with complex DEFs, for external REFs, and for internal REFs.

Figure 70 depicts sample RLDs for a control section. Each RLD consists of a header and modifiers. The header consists of modifier pointers which describe which modifiers are associated with which page of the PMD or text. This is necessary because address constants are modified on a page only as it is referenced.

The first word of each modifier pointer describes the number of modifiers for that page. The second word points to the first modifier for that page. If there are no modifiers for that page, the second word points to the end of the header. This second word pointer represents the number of bytes from the beginning of the second word.



1. No complex DEFs
2. No External Modifiers for Page 0
3. 1 External Modifier for Page 1
4. 2 External Modifiers for Page 2
5. External Modifiers
6. 2 Internal Modifiers for Page 0
7. Internal Modifiers

Figure 70. RLDs for a Sample Module

Modifiers are used to resolve address constants on pages of text, or on pages of the PMD itself (in the case of a complex DEF).

These modifiers are constructed and filled by the language processors for use by the Dynamic Loader. The Dynamic Loader uses these modifiers after all REF's have been resolved.

Each page of the CSD, if it contains a complex DEF and each page of the text for that control section, if it contains address constants, will have associated with it one or more modifiers. There will be at least one modifier for each address constant. The modifier to be used in resolving the complex DEF for the standard entry point is located in the PMD Heading.

The modifier will contain the following information:

1. The length of the adcon to be resolved.

2. The identification number of the REF whose value is to be applied to resolve the adcon.

3. The action of the application (addition, subtraction, or substitution of R value).

4. The byte location on the page of the adcon or complex DEF to be resolved.

The Virtual Memory Page Table (VMPT) is the means by which the external storage pages containing the text of a control section are related to the virtual storage assigned to the control section. This is necessitated by the fact that the TSS/360 language processors do not generate a blank page of text when a large ORG or DS statement results in one or more completely skipped pages.

The reason that blank pages are handled in this fashion is that the language processors build the object program in storage obtained by use of the GETMAIN macro. Not creating blank pages thus conserves main and external storage.

DYNAMIC LOADER PROCESSING

Now that the construction of the Task Dictionary has been described, it is possible to describe the loading process in a little more detail. The loading process is composed of a number of steps:

- INVOCATION
  Implicit linkage
  Explicit linkage

- ALLOCATION

  Determining if the program module containing the referenced symbol is already mapped into the task's virtual memory.

  Initiating a library search if the symbol cannot be found in the task's Task Dictionary (TDY).

  Determining whether or not a found member of a library is a valid module, and if not, rejecting the member as invalid.

  If a module is found that defines the symbol, its PMD is placed into the task's TDY.

  Determining if the Dynamic Loader should reject any control sections described in the module's PMD.

  Mapping the control sections composing the program module into the user's virtual storage.

  Processing the external symbol definitions (DEFs) and references (REFs) described in the PMD.

  Repeating all the above steps (except for the first) for any external symbol references that cannot be found in the TDY.

- RELOCATION

  Computing the correct value for each address constant contained in the page.

INVOCATION

The user may invoke the loading process through the LOAD and RUN commands, through the LOAD and CALL macro instruction, or thorugh in-line statements in assembler language.

IMPLICIT LINKAGE

If an external symbol is referenced by including V and R-type address constants in a program during language processing (for instance, through usage of the implicit form of the CALL macro instruction) the reference constitutes a request for implicit linkage. Program modules containing external symbol definitions satisfying these address constants are implicitly loaded whenever the module containing the address constants is loaded. Program modules that are implicitly loaded cannot be explicitly unloaded from a task's virtual storage.

EXPLICIT LINKAGE

Within a given program there may be references to a number of different subprograms. However, the situation could arise that a given execution of the program requires the presence of only a few of the subprograms. Since dependence on normal implicit linkage would require the presence of adcons in the calling program for all the subprograms, some unnecessary overhead would be experienced in preparing the unused subprograms for linking.

It is also possible that the external name of the module or the entry point which is to be explicitly linked is developed during program execution. In this case, it may not be possible to specify the modules to be linked at assembly time.

To allow for these situations, two explicit functions are provided which cause the desired subprogram to be retrieved at object time: (1) The LOAD macro instruction loads the desired program. (2) The explicit CALL macro instruction causes the program to be loaded and the necessary linkage to it established.

The Command Analyzer and Executor uses the LOAD and CALL statements in processing the LOAD and RUN terminal commands.

The CALL and LOAD macros are generally expanded in line as follows:

```
CALL

DS     0H

L      15,CHD&SYSNDX+12

L      14,CHD&SYSNDX+16

ST     14,72(0,13)

BASR   14,15

LOAD

LA     15,CHD&SYSNDX

EX     0,0(0,15)
```

As a result of these macros an adcon group is generated in the user's first declared PSECT.

```
            ADCON

            CNOP 0,4

CHD&SYSNDX  SVC  127
                 DLINK SVC for explicit
                 loading

            DC   H'C1C2
                 Option codes

            DC   CL8'name'
                 Module name1 (or alias)
                 of module to be loaded

            DC   A(*-12)
                 V-value of "name"
                 filled in here by
                 loader

            DC   F
                 R-value of "name"
                 filled in here by
                 loader
```

When the DLINK SVC is executed, the Resident Supervisor will pass the interruption to the Task Monitor as a program interrupt. The Task Monitor SVC Interrupt Processor will immediately dispatch the Dynamic Loader. It will appear to the Dynamic Loader that it has been called by a Type I linkage. The virtual storage address of the adcon group is placed in a general purpose register by the Task Monitor. The loader then allocates virtual storage for the named module and places the V and R value in the adcon group, loading any and all modules required to resolve external symbols. The DLINK SVC is changed to a no-operation instruction (for LOAD) or a Branch and Store Register instruction (for CALL) by the Dynamic Loader so that it will not be invoked unnecessarily should the LOAD or CALL macro be executed again. This is called disarming an adcon group.

When the loader has completed the resolution of the adcon group, control returns to the Task Monitor. In the case of error-free processing, the Task Monitor then determines whether an explicit CALL or LOAD was executed. (The Loader's return code indicates both error condition and type of adcon group, i.e., CALL or LOAD.) In the case of a LOAD the Task Monitor merely returns control to that point in virtual storage immediately following the EX instruction that caused the DLINK SVC to be executed. In the case of a CALL, the Task Monitor picks up the resolved R-con and places it in the 19th word of the calling program's save area, i.e., in the 19th word following the address contained in register 13. The Task Monitor then effects linkage by placing the resolved V-value into the instruction counter field of the user's old SVC Virtual PSW, such that when control is returned to the interrupted program (via the Load Virtual PSW SVC), the called routine is entered at the V-value location.

At the point of entry, register 13 will be pointing to the caller's save area, the 19th word of which will contain the R-value, by convention the PSECT origin, of the called routine. Register 15 will also contain the V-value. This makes the DLINK SVC invisible to the calling routine. The linkage will appear to be a simple Type-I linkage.

In the case of an error return from the loader while in the conversational mode, the Task Monitor will effect linkage to the Command Analyzer and Executor to prompt the user. The Task Monitor disregards error codes in the nonconversational mode.

The option codes, C1 and C2, are interpreted by the loader as follows:

C1 Code

If the low-order bit is a 1, this indicates a CALL.

If the low-order bit is a 0, this indicates a LOAD.

The high-order bit of C1 may be set to alter the normal symbol lookup algorithm in the loader. If this bit is set, the loader will resolve Adcon groups located in SYSTEM CSECTs in the user hash table, rather than in the system hash table which is the usual case. This feature is implemented so that the Dynamic Loader will properly load user routines when invoked by the Command System LOAD command routine, which is a system routine. The LOAD command routine is invoked by the LOAD command.

C2 Code

The C2 code governs the loader's actions in the case of serious load errors encountered during the response to a LOAD macro. For example, if the loader is unable to locate the module named in a LOAD macro instruction.)

Control over this additional load error indication is in the hands of the user in the case of the LOAD macro. The C2 code within the LOAD adcon group may be set by the user. In the normal case, this code is set to zero. Should a serious error occur, the user is given the appropriate diagnostic by the loader, after which control is returned with an error indication to the task Monitor. If the task is conversational, the user is then queried by the Command System Director and given the option of entering new command statements that might correct the error situation (such as a new DDEF statement to define a library that contains a symbol previously undefinable by the loader). In the nonconversational environment, such error conditions are ignored.

If the user should set the C2 code to one, prior to executing a LOAD or CALL macro, and the loader should detect a serious error, a diagnostic is issued and the C2 code set to seven as the error indication to the calling program (which may initiate program checks for such condition). In this case, the loader will return to the Task Monitor without error indication and the user will not be prompted by the Command Analyzer and Executor.

LOADING PROCESS

The Allocation phase begins with the looking up of the symbol to be loaded. (See Figure 71 which provides an overview of the Allocation phase of the Dynamic Loader.) The appropriate hash chain in the TDY is searched first. The module defining the symbol is already a part of the task. In this case, the loader merely fills in

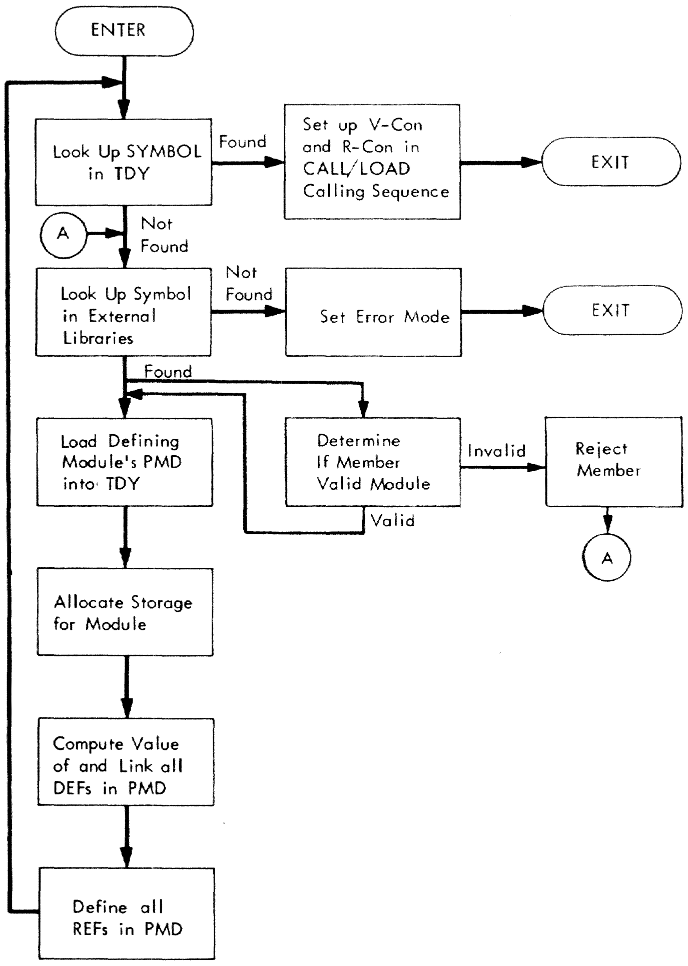

ENTER
Look Up SYMBOL in TDY
Found
Set up V-Con and R-Con in CALL/LOAD Calling Sequence
EXIT
A
Not Found
Look Up Symbol in External Libraries
Not Found
Set Error Mode
EXIT
Found
Load Defining Module's PMD into TDY
Determine If Member Valid Module
Invalid
Reject Member
Valid
A
Allocate Storage for Module
Compute Value of and Link all DEFs in PMD
Define all REFs in PMD

Figure 71. Functional Dynamic Loader Allocation Phase

the V-value and R-value in the adcon group associated with the calling sequence, and returns to the user via the Task Monitor.

If the symbol cannot be found in the TDY, a library search is initiated.

If the symbol cannot be found in this instance, an error condition exists, i.e., the symbol is undefinable for this task.

Realizing that partitioned data sets may contain other than object modules, the Dynamic Loader ascertains, during the loading process, whether or not a found member is actually a module. The relative page position of the module's PMD, text, and ISD, as well as the respective length in bytes of these items, are used in this verification process. If the member retrieved cannot be verified as a module, the member is rejected as being invalid.

It is possible that a symbol may be defined in the TDY and yet be unavailable to a task. For example, suppose that a nonprivileged user loads a module from SYSLIB and the control section in which an external symbol 'CZEYK' is defined has the privileged attribute. The dynamic loader will put CZEYK in the privileged system hash table. Now suppose that the user references CZEYK from another module. The dynamic loader will search the user hash table because the user is nonprivileged and the symbol does not begin with SYS. CZEYK will not be found in the user hash table and will be declared unavailable.

If a library is found that defines "name," the defining module's PMD is now transferred from external storage into the TDY, which is maintained in each task's virtual storage.

The rules governing which hash tables and which libraries are searched by the Dynamic Loader are described in Figure 67.

CONTROL SECTION REJECTION

After the PMD is loaded into the TDY, each control section name within the module is checked. Those control sections whose names either duplicate entry point names already within the TDY or whose names are determined to be illegal are rejected. When a control section is rejected none of the entry points defined by the control section are entered into a TDY hash table. References to these entry points must be satisfied elsewhere or not at all. Control section rejection finds its primary application in the treatment of COMMON control sections. The loader will accept the first COMMON control sections of a given name (or blank), reject all subsequent COMMON control sections of the same name (or blank), and tie all common references to the loaded common control sections.

The treatment of unnamed control sections deserves some special comment here. Unnamed common control sections are assigned a name of eight alphameric blanks. After the first common control section is loaded, subsequent unnamed common control sections will be rejected as discussed above.

Unnamed uncommon CSECTs are assigned a name of 16 hexadecimal zeros by the assembler. In order to render such names unique to the module in which they were declared, the loader places a module sequence number in the two low-order bytes of the first word of the name part of the DEF entry and all REFs of the same "zero" name within the module. This technique eliminates control section rejection for unnamed CSECTs since unnamed CSECTs from different modules will be distinguishable one from another.

CONTROL SECTION STORAGE KEY ASSIGNMENT

Virtual storage and storage protection keys are assigned for each nonrejected control section.

Fixed length control sections with identical attributes are allocated storage as a group. This grouping of control sections is done to reduce the number of calls on Virtual Memory Allocation. Variable length control sections are individually allocated virtual storage by using the "variable" parameter when invoking GETMAIN. Common control sections are systematically assigned the variable length control section attribute by the TSS/360 language processors.

Storage protection key codes are assigned to each control section group at the time storage is requested for that group. The Resident Supervisor will mark each External Page Table entry with a code representing the appropriate storage key. Whenever the page corresponding to an entry is brought into main storage, the appropriate hardware storage key is set up for that page. Read-only sections are assigned a storage key that will not allow the user to store in the storage assigned. Privileged control sections are assigned a storage key that will not allow the user to store into or to read the assigned storage. Privileged control sections will only be found in certain system service routines. All other control sections are assigned a storage key that allows unlimited user reading and writing of the assigned storage.

Public control sections in modules loaded from shared data sets are assigned shared storage so as to make such control sections potentially available to other tasks. If some particular public control section has not previously been loaded by some other task, the loader will assign shared virtual storage such that this task's copy of the CSECT will be loaded into shared virtual storage. If some public control section has already been allocated shared storage by another task, then the current task is merely "connected" to such shared storage, i.e., all references to such public storage will be tied to the already loaded control section (see "Sharing").

Page table entries are set up for each of the nonrejected CSECT's text pages. The external library storage address is associated with each page table entry and each page is marked "unavailable." Any user reference to any byte on the page will cause interruption. This interruption will cause the Resident Supervisor's paging routines to transfer the page from external storage into main storage.

At the time the page tables are set up, the loader checks each page for the presence of adcons. Those pages containing adcons are marked "unprocessed by loader" in addition to "unavailable." The referencing of pages marked "unprocessed by loader" will cause the Supervisor to effect a call via the Task Monitor to the page relocation entrance of the loader. This action is described more fully under "Relocation."

Now the value of all DEFs in the nonrejected control sections of the module are computed except those DEFs whose names duplicate previously loaded DEFs or whose names are judged illegal.

Duplicate or illegal DEFs are rejected with diagnostics. Relocatable DEFs are computed by adding to the DEF value the base address allocated by the loader to the containing control section. Absolute DEFs require no computation. Complex DEFs are computed last. Recall that complex DEFs have associated with them REFs to other control sections. If the external name to which such a REF refers is not found in the TDY, the entire loading process is initiated again in order to load a PMD that will so define the REF. After the complex DEFs are computed, all of the remaining REFs in the module are satisfied. Some or all of the remaining REFs may effect the loading of additional modules' PMDs. Loading of additional PMDs will repeat until all REFs in all modules have either been satisfied or have been marked undefinable.

It is quite possible for the loader to satisfy some REF by locating an entry point in some external library only to have that entry point lost in the allocation process due to control section rejection. For example, some module has a REF to symbol X which is found in CSECT C in module A in some library. During allocation CSECT C is rejected by the prior occurrence of some other CSECT C, such that when allocation for module A is completed, symbol X is still unsatisfied. The loader checks for this condition and accommodates it by initiating the symbol search (and allocation cycle) once again, this time in the next library in the hierarchy. A symbol is determined to be undefined when all libraries from the hierarchy starting point up to and including SYSLIB have been searched, yielding no definition.

RELOCATION

Whenever a "page unavailable" interruption occurs, the Resident Supervisor Page Posting routine checks the "unprocessed by loader" bit in the page table. If this bit is not set, no loader action is required. If the bit is set, the Resident Supervisor enqueues a program code 17 GQE on the task's TSI. When the Task Monitor Program Interrupt processor receives control it will immediately dispatch the Dynamic Loader at its page relocation entry. The loader's action in this event is merely to compute the correct value of each adcon on the page triggering the interruption. The processing of adcons will always involve the application of some REF value to that portion of the text occupied by the adcon. There are three possible applications:

1. Add the V-value of the REF to the text value.

2. Subtract the V-value of the REF from the text value.

3. Store the R-value of the REF into the text.

At the time this relocation occurs, all REF values will have been satisfied (during the allocation phase of the loader). Once all adcons have been relocated, the loader returns to the Task Monitor and eventually to the point in virtual storage where the relocated page was originally referenced.

There is a special exception in processing a page relocation program interrupt, and, though it is a rare occurrence, the Task Monitor must deal with it. The Dynamic Loader page relocation processing it is not recursive, i.e., it cannot be entered to process a second interrupt until it has returned from its first dispatch. However, the Relocation Phase of the Dynamic Loader itself can cause a software relocation exception. This occurs when two contiguous pages are both "unprocessed by loader" because an adcon group crosses the page boundary between them. That is, the word in which the R-value is to be placed falls at the beginning of the next page. However, the second page relocation interrupt does not have to be processed in order for the Dynamic Loader to complete the processing of the first page. When this second interrupt occurs, the Task Monitor's program Interrupt Processor saves the exception address and then restores control to the loader. When the Dynamic Loader returns the first time, it is reinvoked with the second exception address, and normal processing continues. The Task Monitor's Program Interrupt Processor maintains flags in the ISA to indicate these circumstances.

DELETING PROGRAM MODULES

The Explicit Unlinking entrance to the Dynamic Loader is called whenever a DELETE macro is executed either as a result of the command UNLOAD or as inline code. The major argument is either a module name or alias (control section name or other entry point name) whose containing module is to be unlinked from all other programs and deleted from the task.

Modules included in a task's Initial Virtual Memory are not loaded by the Dynamic Loader and hence can not be deleted or unlinked by the Dynamic Loader.

Unloading is, in general, infrequently used in TSS/360 because of the large virtual storage environment. However, if a user compiles the same module a second time during a terminal session, he must unload the first copy of the module before attempting to run his revised version. Otherwise, the Dynamic Loader will transfer control to the first version of his module because it is still loaded in virtual memory.

Module deletion, or unlinking, involves several processes:

1. Locating all explicit references to the module to be deleted and "rearming" them.
2. Tracing explicit references from this module to identify subordinate modules that may be deleted as well.
3. Tracing implicit references from this module for the same purpose as (2).
4. Deleting all control sections and freeing allocated storage.
5. Deleting all deletable modules' PMDs from the TDY. This processing includes removing all DEFs for these modules from the TDY.

A deletion candidate, then, is either a module whose name (or alias) appears in the DELETE statement (primary candidate) or some other module (secondary candidate) that is referenced by the primary or by a secondary candidate. There are two ways in which a module may reference another module. An explicit reference is effected by a module's executing a LOAD or CALL macro naming an external symbol defined as another module. An implicit reference is effected by a module's having a REF entry that is satisfied by a DEF entry in another module. The allocation phase of the loading process sets up appropriate explicit and implicit chains linking referenced with referencing PMDs. These chains are contained in a Module Usage Table (MUT). Each task has its own MUT. A MUT entry is linked into two chains which have their origin in two different PMDs. A MUT entry serves to tie together a called module and its explicit caller. When a MUT entry is created, it is linked into the calling PMDs PAPA chain and into the called PMDs BABY chain. Thus, if A calls B and C, two MUT entries and three chains are created. (See Figure 72.)

Secondary deletion candidates are located during the unloading process by tracing these chains and placing every

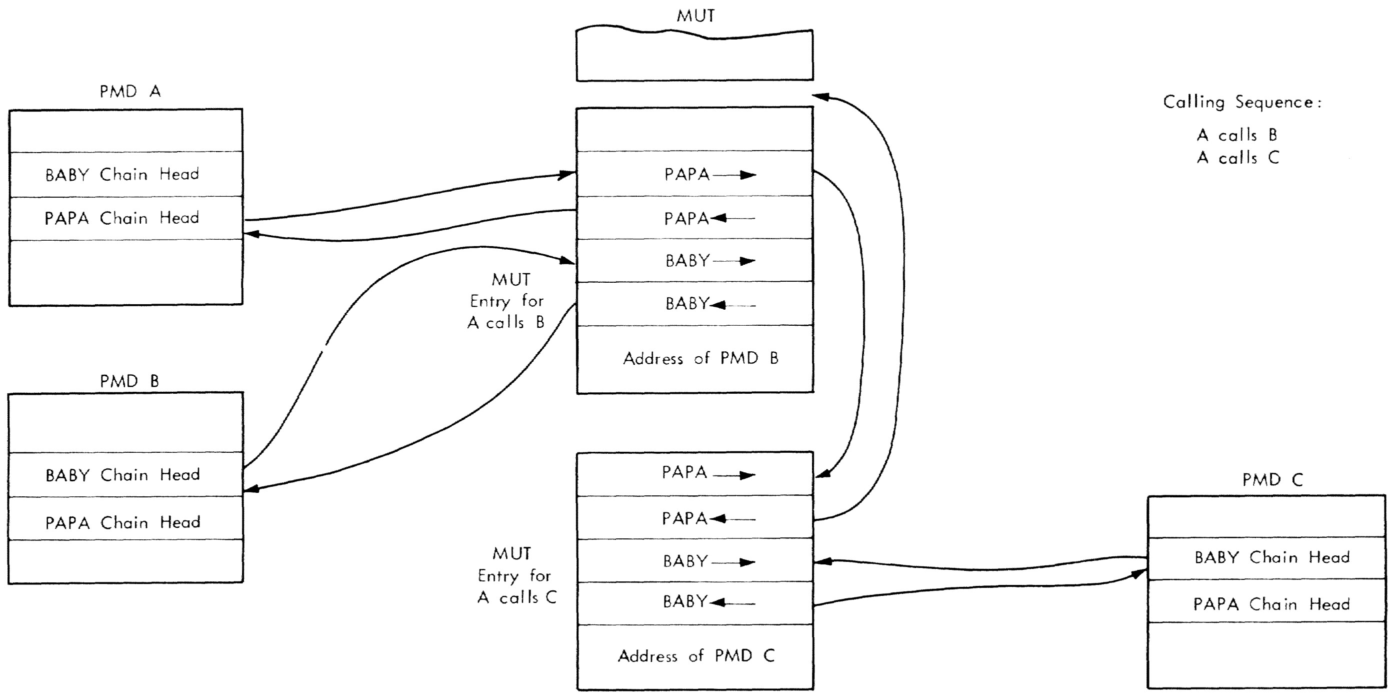


Figure 72. Diagram of Sample Module Usage Table

referenced module on a candidate list. This tracing process repeats until all modules referenced by the primary and secondary deletion candidates have themselves become deletion candidates.

Only those deletion candidates that have no outstanding explicit or implicit references to them are retained on the candidate list. The removal of any candidate on the list may result in the removal of a previous candidate from the list. Now this process is reiterated until a stable candidate list results and all those modules remaining on the list may be deleted from the task.

There is one exception to the foregoing algorithm. The primary deletion candidate is deleted so long as there are no outstanding implicit references to it. Explicit references to the primary candidate are traced to their source (CALL or LOAD adcon group) and the original SVC is "rearmed" such that subsequent execution thereof will cause reloading of the deleted module. At this point storage is released for all nonrejected control sections of all modules to be deleted. The DEFs in each control section are removed from the TDY DEF chains, and finally the PMD itself is deleted from the TDY. This process is repeated for each module to be deleted. Unloading is complete when the last module on the deletion list has been removed from the task.

The user may specify in a DELETE macro or UNLOAD command that only the module named is to be unloaded. In this case no attempt will be made to delete modules referenced by this "named" module -- only the primary deletion candidate is ever entered on the candidate list. The tracing of implicit and explicit links is eliminated.

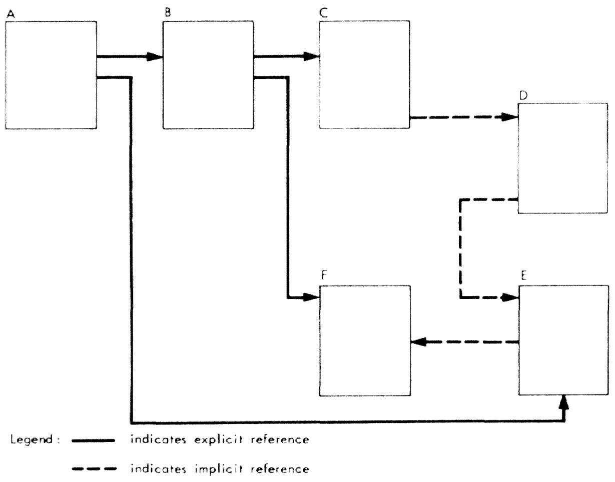

A
B
C
D
E
F
Legend : ——— indicates explicit reference
- - - indicates implicit reference

Figure 73. Unloading Example -- Before Unloading

UNLOADING EXAMPLE

Figure 73 shows the allocation for six modules. Module A has explicit links to B and E. Module B has explicit links to C and F. Note that module C implicitly links to D, which implicitly links to E, which implicitly links to F. If the statement DELETE B is executed from within module A, the following unloading action will occur:

B is placed on the candidate list. B's references are now traced; this results in C and F being added to the candidate list. C's references are now traced; this results in D being added to the list. F has no references, so it causes no new secondary candidates to be added. Now D's references are traced, resulting in E being added to the list. E references only F, which is already on the list.

Now all modules are checked for outstanding references. B has one outstanding reference from A; but since B is the primary deletion candidate, this explicit linkage in A is rearmed such that B remains on the list. Modules C and D have no outstanding references, so they also remain. However, E has an explicit link from A which is outstanding. Thus, E is removed from the list, reestablishing the implicit link between E and F. Now F is examined, and it is discovered that F has an outstanding implicit reference (just reestablished from E). Thus, F is removed from the list.

At this point, modules B, C, and D are deletion candidates, and none has any outstanding references. Unloading proceeds, then, with the removing of modules B, C, and D from the task. This results in the allocation diagrammed in Figure 74.

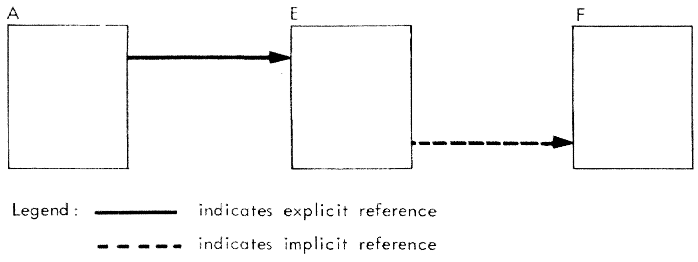

A
E
F
Legend : ——— indicates explicit reference
- - - - - indicates implicit reference

Figure 74. Unloading Example -- After Unloading

CATALOG SERVICE ROUTINES

Time Sharing System/360 contains catalog service routines designed to allow the user to update, add to, and delete from his private catalog. Catalog service routines are directly entered only from a privileged program. An example of the use of various catalog service routines is provided in the section "External Sharing."

Catalog service routines are divided into those that are invoked by the user's program, and those that are called by other catalog service routines. Those service routines invoked by the user are: ADDCAT, DELCAT, SHARE, UNSHARE, SHAREUP, CATFLUSH, DSCB/CATALOG Recovery, and USERCAT SCAN. SEARCHSBLOCK, GETSBLOCK, and INDEX provide services for other catalog routines. A brief description of each of the catalog service routines follows.

INDEX Routine

This routine constructs chained index levels in the catalog and creates new members within the catalog data set. The names of the indexes are defined by the fully qualified name supplied by the user. INDEX also checks for sharer updating privileges, upon locating some, but not all, levels of the fully qualified name.

The fully qualified name is inspected to determine if a new use is being created, or if a new index level is to be added to the user. When a new user is to be added, the catalog is opened in the update mode and INDEX uses LOCATE to determine that the ID is unique. When index levels are to be chained for the user, the lowest level found is searched for an empty pointer, and a pointer is constructed to the first SBLOCK of the level being created.

ADDCAT Routine

This routine performs the following functions:

- Creates a data set descriptor in the user's catalog which points to the format-E DSCB and gives volume information.
- Creates any index levels defined by the fully qualified name, which must precede the data set descriptors and do not presently exist.
- Allows updating of a data set descriptor.
- Controls the number of generations allowed under a generation index by performing deletion of out-moded generations, as required.

DELCAT Routine

This routine performs the following functions:

- Deletes index levels from the catalog structure.
- Recatalogs index levels under a different fully qualified name.

DELCAT calls LOCATE to get the specified index level, and then determines if an owner's catalog is referred to by checking the first byte of the 45-byte buffer used as an entry parameter to LOCATE. If the flag is set, the sharer disposition flag in the 64-byte SBLOCK retrieval buffer is checked.

SHARE Routine

This routine adds sharing privileges to a catalog level.

An unshared level can be set to sharable, or a shared level can have its sharing access modified. Sharing can be universal (meaning that any user may share), or selective (meaning that only those users whose user-IDs are included in the input parameter lists are allowed to share). LOCATE is called to retrieve the proper level for the fully qualified name supplied. For selective sharing, a sharing list is created or updated, depending on the type of request.

UNSHARE Routine

This routine removes sharing privileges from a catalog level.

First the proper level is located and checks are made to see that the sharing mode of the level is compatible with the type requested. If the sharing mode is universal and the request is to delete all sharers, the sharing flag is set to private, and the new index level is updated in the catalog. If the sharing mode is selective and the request is to delete all sharers, the additional operation of deleting the sharer's list is performed. When the sharing mode is selective and the request is not to delete all members, the sharer's list is searched and only the members passed in the parameter list are deleted.

SHAREUP Routine

This routine links one user's private catalog to a level in another user's sharable catalog, by constructing a sharing descriptor in the sharer's catalog that points to a level in a user's catalog that was previously designated as sharable. The shared index level is retrieved to determine if the calling program is allowed to share. If the calling program or user is allowed to share and his fully qualified name is unique, a sharing descriptor is constructed by the INDEX routine.

CATFLUSH Routine

This routine copies:

- Specified members to USERCAT without deleting them from SYSCAT.
- Inactive members to USERCAT, deleting them from SYSCAT.
- All members to USERCAT, erasing SYSCAT.
- SYSOPER0 if SYSCAT exists at startup.

Prior to any copy, SYSSVCT is tested to determine whether the user catalog and the system scratch catalog (SYSCAT) are already identical. If so, the copy is not made.

DSCB/CATALOG Recovery Routine

This routine resynchronizes a user catalog if the current member in SYSCAT cannot be used. It also rebuilds a user catalog from public DSCBs should no member exist in SYSCAT and the user catalog becomes unusable. In this instance, all sharing information is lost. This routine assumes that the USERCAT member is locked in SYSCAT, preventing multiple users from attempting recovery of a user catalog.

USERCAT SCAN

This routine rebuilds the SYSSVCT. Such an operation becomes necessary if the Auxiliary Volume is restored, or if SYSSVCT is corrupted in some manner. The new SYSSVCT is built from the User Table and a scan of public storage for user catalogs.

LOCATE Routine

This routine determines the location of SBLOCKs within the catalog, either by name or relative address, and retrieves them. LOCATE then attempts to find the first SBLOCK for the last level pointed to by the fully qualified name.

When the SBLOCK has been located, it is moved into the calling routine's buffer area (given to LOCATE as an entry parameter).

SEARCH SBLOCK Routine

This routine acquires and chains an empty SBLOCK as either an extended SBLOCK of a cataloged entity or the first SBLOCK of a cataloged entity.

The count of SBLOCKs in each page is checked until an available SBLOCK is found. The relative address of the SBLOCK within the page is then located by searching for a block containing all zeros. The new SBLOCK is retrieved by GET SBLOCK, and its virtual storage address is returned to the user. The new SBLOCK is linked to the parent SBLOCK before returning control to the user.

GET SBLOCK Routine

This routine locates a specific SBLOCK from a pointer address and calculates the virtual storage address of the block for the using program.

If the requested SBLOCK is already in the page buffer, the virtual memory address is calculated and returned to the user. If the requested SBLOCK is not in the page buffer, the GET macro instruction is executed to read the proper page into the page buffer. The virtual storage address of the SBLOCK is then calculated and returned to the user.

Sharing is defined as the permission for more than one user to access a program or data set. There are four major types of sharing:

1. External - there exists one copy of the data set externally, and each user receives his own copy internally. Examples: SAM data sets, object program modules with only "private" control sections.
2. External with internal control - one copy of a VAM data set exists externally and each user receives his own copy internally. However, as opposed to EXTERNAL above, all concurrent users do share a RESTBL, and control of the external copy is via this shared table.
3. Internal and external - part of a data set is shared internally by all users, and each user gets his own copy of another part. Example: program modules with public CSECTs and prototype CSECTs.
4. Internal - only one copy exists internally for all users. Example: Shared Page Tables, Shared Data Set Tables.

EXTERNAL SHARING

EXTERNAL SHARING OF DATA SETS

External Sharing of Data Sets is implemented in the catalog. A user can allow any portion of his catalog to be shared. He can specify a particular data set to be shared, or he can specify an index level (part of a fully qualified name) to be shared. In the latter case, all index levels below the shared index are sharable.

The user who authorizes the sharing of a portion of his catalog is referred to as the owner; those authorized by the owner to share are sharers.

The owner has the option of specifying the accessing privilege of those who are allowed to share a data set or index level. The classes of accessing privilege are:

- A Sharer has unlimited access to shared data sets and to the shared portion of the catalog.
- A Sharer can read and write data sets cataloged in shared portions of catalog, but cannot make changes to the owner's catalog entry. He cannot ERASE the data set.
- A Sharer can read only the data sets cataloged in the shared portion of the catalog, but cannot make changes to the catalog.

The owner can specify a data set or index level as universally sharable, or he can specify explicitly those users who are allowed to share. The list of permissable sharers is in the Catalog. If the owner specifies universal sharing, any user is allowed to share provided he completes the required linkage (by using the SHARE command).

The owner permits sharing through the PERMIT command, but the sharer must provide for the linkage between his catalog and the owner's catalog. The sharer does this through the SHARE command, providing to the system:

- The owner's identity.
- The fully qualified name assigned by the owner to the data set or index level.
- The fully qualified name assigned by the sharer.

To indicate how the sharing linkage is accomplished, consider the following example. The owner, User 1, whose identity is USER1, specifies that he will allow other users to share index level A. User 2 wishes to access User 1's data set A.B and call it X.Y. User 2 must then provide USER1. A.B and X.Y as parameters for the SHARE command.

A subsequent access to the data set X.Y would cause a catalog search through index levels X.Y.USER1.A.B to reach a catalog entry in User 1's catalog (see Figure 75).

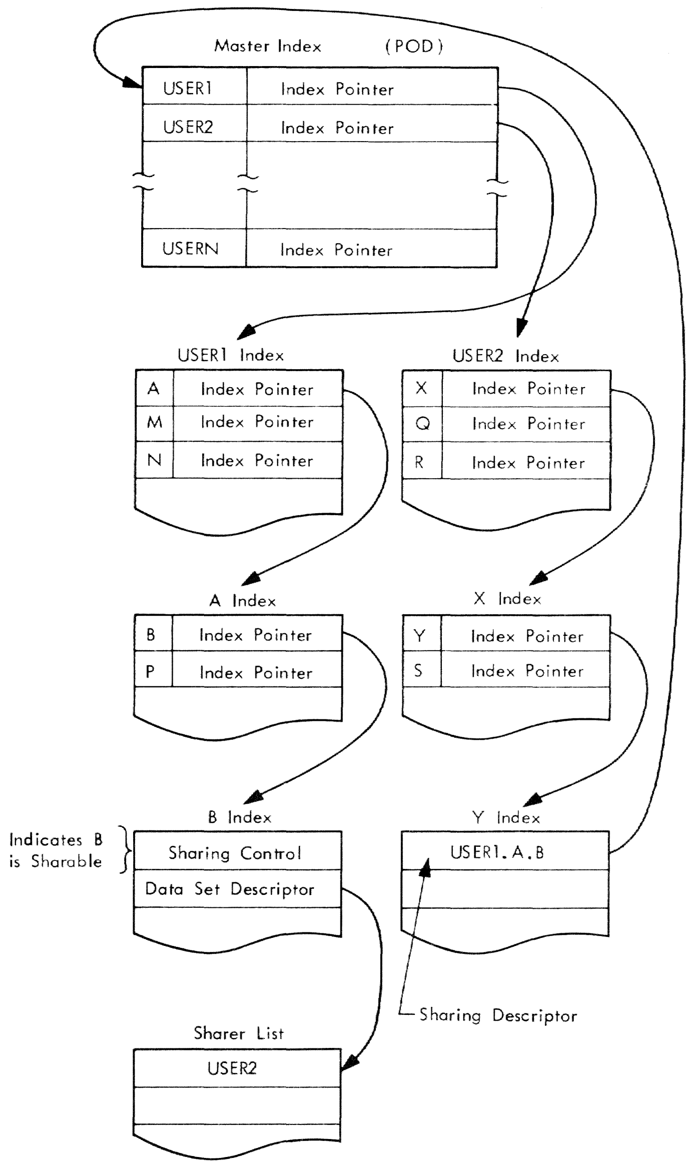


Figure 75. Example of External Sharing

That is how the catalog is searched. Let us now see how it is built.

1. User 1 creates his data set.

   ADDCAT is called to add a new data set to the catalog. Its input is the fully qualified name A.B. ADDCAT calls INDEX which in turn calls LOCATE to make sure the data set is not already in the catalog. If it is not, INDEX will look for the lowest level of fully qualified name already in the catalog (in this case there are no previous entries). INDEX will call SEARCH SBLOCK to retrieve unused SBLOCKs (one for each index level) and chain them together. When INDEX returns to ADDCAT, ADDCAT will take the lowest level SBLOCK and create a Data Set Descriptor which points to the volume on which the data set resides.

2. User 1 issues a PERMIT command:

   SHARE will call LOCATE to retrieve the lowest level of SBLOCK (the Data Set Descriptor). SHARE will insert in the sharing flag the indication that there is a sharing list.SEARCH SBLOCK will be called to obtain an unused SBLOCK (for the sharing list) and chain it to the Data Set Descriptor. Finally SHARE will insert the sharer's userid and the sharing disposition (read only, read/write, or unlimited) in the sharing list.

3. User 2 issues a SHARE command.

   SHAREUP is called to link the sharer's catalog to the owner's catalog. SHAREUP then calls INDEX to create the index levels for the sharer. SHAREUP also calls LOCATE again to make sure the sharer's fully qualified name is unique (i.e., he doesn't already have a data set called X.Y. Finally SHAREUP creates the sharing descriptor from the lowest level of SBLOCK set up by INDEX.

4. User 1 later decides to delete User 2 from the sharing list.

   UNSHARE calls LOCATE to retrieve the lowest level of SBLOCK in the fully qualified name (A.B.). UNSHARE calls GETSBL to get the sharing list and removes User 2 from the list. Since User 2 was the only userid in the sharing list, this SBLOCK is now freed and designated as unused.

   Finally DELCAT will be called to free the sharing descriptor in the sharer's (User 2's) catalog. (DELCAT will call LOCATE to retrieve this sharing descriptor.) DELCAT will also remove the pointers X. & Y. from the higher levels of indexing.

External Sharing of Programs

Sharing of access to program modules is facilitated primarily through the System Library (SYSLIB). SYSLIB is automatically opened for all users during Virtual Memory Task Initiation and will contain, in addition to privileged system modules, the commonly used problem programs which the installation wishes to make available to all users, such as language processors, mathematical subroutines, etc.

Two considerations exist in sharing programs through the system library.

1. Only the system programmer with the User ID of TSS***** (C Authority Class) may stow into SYSLIB.

2. Any module placed in SYSLIB is available to all users.

A user may allow restricted sharing by placing a module in SYSULIB or in a JOBLIB and issuing a "permit" on that data set to anyone he chooses. This makes all members of that data set (i.e., library) available to the sharer, providing he issues a DDEF command defining that data set as a JOBLIB so that the LIBMAINT module will create and open a DCB for him which is linked to the library search chain.

If an owner wishes to share only one program he must create a JOBLIB in which that program is the only member.

A program (object module) is a member of a particular VPAM data set which may have been defined as a library. Since entries are made in the catalog only for data sets, no entries for program modules (or their components) are found in the catalog. These entries are found in the Partitioned Organization Directory (POD) associated with that particular data set.

Concurrent External Access

For any data set to be shared concurrently among users (in the sense that User B can access the data set while User A is still processing it) the data set must be contained in the system's public storage.

The Sequential Access Method (SAM) is supported under TSS/360 in order to provide a limited data set compatibility with Operating System/360 and is, consequently, not designed to take full advantage of the time sharing environment. Because of this, TSS/360 permits concurrent external access to SAM organized data sets, but does not provide any interlocks beyond the access privilege class specified in the catalog and checked when the OPEN macro is issued. The Virtual Access Methods, on the other hand, are designed to facilitate concurrent sharing among users.

EXTERNAL SHARING WITH INTERNAL CONTROL

This type of sharing is unique to the processing of Virtual Access Method (VAM) formatted data sets. Internal control of shared VAM data sets functions primarily through two types of interlocks: Read Interlocks and Write Interlocks.

A Read Interlock (RI) is imposed to prevent another user from writing into a data set or page of a data set. A Read Interlock cannot be set if a Write Interlock has already been set.

A Write Interlock (WI) inhibits any user, other than the user who set the Write Interlock, from reading or writing into a data set or page. Only a single Write Interlock can be set at one time which implies that once set, neither Read nor Write Interlocks can be applied thereafter until the existing Write Interlock is reset.

There are three levels at which Read or Write Interlocks can be set:

- Data set level
- Partitioned data set member level
- Data set page level

DATA SET INTERLOCKS: A Data Set Interlock is set according to OPEN options (INPUT, OUTPUT, INOUT, OUTIN, UPDATE) and restricts the kind of OPEN options that will be accepted from future concurrent users according to the following rules:

INPUT

A Read Interlock is set for VSAM and VISAM data sets. A Read Interlock will be set for a VPAM member when the FIND macro instruction is issued.

OUTPUT

A Write Interlock is set for VSAM and VISAM data sets. A Write Interlock will be set for a VPAM member when the FIND macro instruction is used.

INOUT, OUTIN, UPDATE

A Write Interlock is set for VSAM data sets. A Read Interlock is set for VISAM data sets. VPAM members are Read or Write interlocked according to their status upon issuance of the FIND macro.

Data set interlocks are reset when the CLOSE macro is issued.

MEMBER INTERLOCKS: A VPAM data set is not interlocked at the data set level as is the case with VSAM and VISAM data sets. VPAM data sets are interlocked at the member level upon issuance of the FIND macro. The Partitioned Organization Directory (POD) is interlocked only during the execution of the FIND and STOW macros. Member Interlocks are set within the POD and are imposed by the FIND and released by the STOW or CLOSE macros.

Interlocks are placed at the data set or data set member level when the INPUT or OUTPUT options are used and, for VSAM, whenever INOUT, OUTIN or UPDATE are used because the access method interprets each PUT macro instruction as an end-of-data set indicator. This means that a PUT issued within a data set truncates the data set and, in effect, deletes all the following records. This is a useful feature in updating data sets.

However, if a Read or Write Interlock were not imposed during sharing, one user could attempt to read or write to a portion of a data set that was just deleted by another user.

Shared VISAM data sets opened for UPDATE are forbidden to use the PUT macro instruction and, in return, are assigned only a Read Interlock at the data set level.

PAGE INTERLOCK: When shared VISAM data sets are opened with either UPDATE, INOUT, or OUTIN options, only a data set or member level Read Interlock is set. Therefore it is necessary to provide additional Read and Write Interlocks at the page level. Although the macros which set these interlocks reference individual records, the interlocks set as a result of executing these macros apply to all records contained within the referenced page. Page level Read and Write Interlocks for a VISAM data set are set as follows:

- Page level Read Interlocks are imposed on the page referenced by execution of the GET or the Read-according-to-specified-key (type KY) macro. Neither SETL nor OPEN impose any page level Read Interlocks.

- Page level Write Interlocks are imposed only by a READ-exclusive (type KX) macro. Neither SETL nor OPEN impose any page level Write Interlocks.

- Page level Read Interlocks are released by a READ (type KX), WRITE, ESETL, DELREC or RELEX if issued against the DCB which caused these interlocks to be set. They are also released by execution of a CLOSE or any other macro which references a page other than the page referenced by the macro instruction which caused the interlock to be set.

- Page level Write Interlocks are released by the GET, READ (type KY), RELEX, WRITE, DELREC, or CLOSE macros or by execution of any other macro which references a page other than the page referenced by the macro instruction which caused the interlock to be set.

Read or Write Interlocks on the data set or page levels may only be released by using the appropriate DCB within the task which originally set the interlock. Only a single Read or Write Interlock may be imposed by a given DCB.

There is a trade-off between sharing a common control mechanism (i.e., RESTBL) and giving each user a private image of the physical page as is done in TSS/360 and sharing the buffer contents themselves. In the former case, control of access is simple and efficient, but cannot be safely brought down to the record level. The latter case implies a more complex control mechanism and still does not guarantee a significant improvement in the accessability of shared data. This is because there is little probability that two or more users will wish to refer to the same page of a large data set at the same time.

USER CONSIDERATIONS:

1. The only way a user can gain exclusive control of a shared VISAM data set is to use the OUTPUT keyword when the data set is opened. It should be noted that although the data set is opened for output, a user can, in fact, read the data set.

2. If updating of a VISAM data set is to be effected, the record to be updated should have been obtained by a READ-exclusive (type KX) macro. If users of a shared data set do not employ this procedure, two tasks may concurrently reference the same page, using either the GET or READ by key (type KY) macros and decide that a record within the page should be updated. Since both tasks use WRITEs to the same page, the task which last issues the WRITE to the page in question will cancel the effects of the previously executed WRITE.

3. A READ (type KZ) by retrieval address should not be employed by users of VISAM shared data sets since the desired record can be shared by another task.

4. Coding sequences within a task may produce an intra-task interlock that cannot be detected by the access method. For example, the sequence

```
READ          DECB,KX,(1),(0),(2)

GET           (1)
```

where the READ and GET macros reference different DCBs within the same task will produce a task loop, since the GET will cause the task to become dormant while waiting for the WI set by the previous READ to be reset. The WI will not be reset since the same task is now dormant awaiting resetting of the WI. The user should pay close attention to the rules of interlock set/reset when dealing with multiple opened DCBs within a given task.

The system's use of interlocks for sharing requires careful control. For instance, system operation can be affected if one task sets an interlock in a system table and then becomes inactive for a long time. Furthermore, substantial system overhead is incurred if those tasks waiting for an interlock to be re-set are continually being dispatched only to find that the interlock is still set.

PROCESSING

When the DDEF routine creates the Job File Control Block (JFCB), the following two indicators are set that reflect the sharing access privilege class described in the catalog:

- A flag that indicates whether the data set may be shared.
- A flag that indicates whether a data set is to be write protected.

Whenever an OPEN macro is issued on a VAM DCB, COMMON OPEN will invoke OPEN VAM. OPEN VAM first checks the JFCB to determine if the data set is a sharable one. If the data set is sharable, the user's OPEN option is checked against the security indication in the JFCB to see if the user is authorized to perform the functions (read or write) that he is requesting. If he is not authorized, control is returned to the COMMON OPEN routine and a diagnostic is issued.

System data sets (i.e., those data sets with the User ID of TSS*****) are universally sharable.

There is a special entry point in the Define Data routine that allows a privileged system routine to specify a User ID other than the User ID for the task. For example, this allows system routines to create JFCBs for system data sets without having to first issue a SHARE command. However, OPEN VAM applies protection for these data sets. The Catalog and User Table data sets are considered privileged data sets. Only a user with an "O" Authority Code and certain system routines, such as Virtual Memory Task Initialization, are allowed to access these data sets. A non-privileged user can read all other system data sets, while privileged system routines and users with "O" or "P" Authority Codes are allowed read/write access to such data sets.

If the user is authorized to perform the requested operation on a shared data set, the Search the Shared Data Set Table (SRCHSDST) routine is entered to determine if the shared data set is already open.

When users share a program or a data set control table, they share a common page table. However, the virtual storage service routines cannot directly address shared page tables. This means that the Resident Supervisor must provide a method of symbolically associating the shared item with the page table that maps it.

The Shared Data Set Table (SDST) is the means by which Shared Page Table (SPT) entries mapping the control tables for shared data sets and data set members are located. The SDST also keeps a count of the concurrent users of shared data sets and is the repository for data set interlocks. A skeleton SDST is created by Startup. The SDST is processed by the Search Shared Data Set Table (SRCHSDST) routine.

If the SRCHSDST routine does not find an entry for the data set in the SDST, the routine assumes that the sharable data set has not yet been opened.

If the data set is not yet open, the following operations are performed:

- All Data Set Control Blocks (DSCBs) describing the extent of this data set are read from external storage.

- The amount of virtual storage required by the RESTBL and, if required, by the POD and Index Sequential Directory is determined and GETSMAIN is used to obtain shared virtual storage for that number of shared pages.

- For a sharable data set, the skeletal SDST entry created by the SRCHSDST routine is completed by filling in a Shared Page Table (SPT) identification number, the number of pages obtained through GETSMAIN, and the relative position within the Shared Page Table where the virtual storage allocated by GETSMAIN begins.

- The RESTBL is built.

RESTBL: While the sharing of VAM data sets is controlled principally through the Shared Data Set Table, the RESTBL (Relative Page External Storage Correspondence Table) also plays a major role in internal sharing.

Each RESTBL is contained in an area of virtual storage protected from the user. This virtual storage area has the read-only protection key. For a sharable data set, the RESTBL is contained in pages which are shared among the tasks using it. The RESTBL and the SDST are the only data set information mapped into Shared Page Tables. The sharers of a data set do not share JFCBs, DCBs or buffers.

The page entries within the RESTBL are ordered by page number relative to the data set. For a sharable data set, there are two four-byte words per entry as follows:

WORD 0	INTERLOCK CONTROL WORD
WORD 1	EXTERNAL PAGE ADDRESS WORD

The format of the interlock control word is as follows:

W	R	N	I

W - Write Interlock indicator
R - Read Interlock indicator
N - Read Interlock counter. The number of times the Read Interlock is imposed.
I - Read Interlock control flag. If on, a Read Interlock is in the process of being imposed or released, and a subsequent request to impose or release an interlock must wait.

If the data set is a shared one and has already been opened, the following steps are performed during OPEN processing rather than those mentioned previously:

- A check is made to see if the data set is open in this task. If it is, the following step is not performed.
- CONNECT is used to allocate virtual storage space for the shared RESTBL and, if required, for the POD and Index Sequential Directory, by connecting one of this task's segments to the existing Shared Page Table(s) which already describe the RESTBL, POD, and Index Sequential directory. Once these steps have been performed, VAM OPEN processing continues essentially as it does for non shared VAM data sets.

INTERNAL AND EXTERNAL SHARING

This class of sharing is implemented by the Dynamic Loader. The Dynamic Loader places a control section containing the Public attribute in shared virtual storage, if the control section fulfills all the following conditions:

- The Public control section must, in general be named.
- The Public control section comes from a shared library.
- The user's Authority Code is U.
- If a control section contains the Public attribute, it must not contain address constants (ADCONs).

If a control section with a "public" attribute fails to fulfill any of these requirements (except the last) it is placed in private virtual storage. The Dynamic Loader will not load a Public control section containing address constants.

Because shared pages remain in main storage for relatively long periods of time, internal sharing through the use of the Public control section attribute only conserves main storage and paging overhead if the Public part of the program module is concurrently referenced with high frequency. An example of such a routine might be the virtual memory allocation module. Auxiliary storage space is conserved by virtue of a control section being read-only; not by virtue of its "public" attribute.

The main storage residence of shared pages is monitored in the following fashion:

Once every n time slices, the storage protect key reference bits for shared pages are inspected and re-set. The value n is a system parameter specified during system generation. If a shared page has not been referenced since the previous inspection, it is placed on the User Core Allocation pending list. If the page has not been referenced during the preceeding n time-slices, but has been changed since it was brought into main storage, it is paged out to auxiliary storage.

The control flow involved in loading a program from a shared dataset is depicted in Figure 76. When a user requests linkage to a routine, the Dynamic Loader searches the user's Task Dictionary to see if the module has already been loaded by the user. If the module has been loaded, the appropriate V type and R type address constants have already been defined, and the loader simply returns to the Task Monitor with a pointer to these values.

If the module has not been loaded, the Dynamic Loader calls its LIBE SEARCH to search the PODs of open libraries until the module is found. When the module or its alias is found in a POD, the Dynamic Loader takes the Program Module Dictionary address from the member entry and issues a VSAM SETL and a VSAM GET to bring the PMD into main storage. The Dynamic Loader then

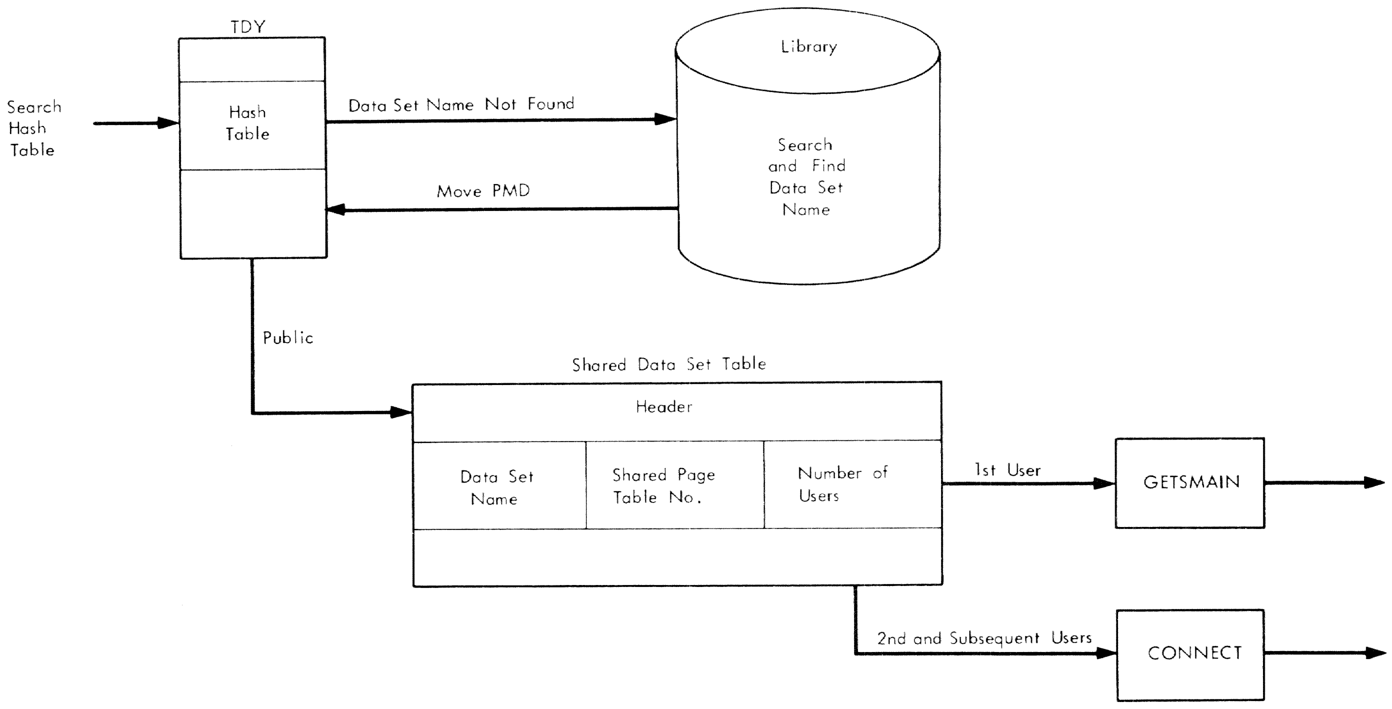


Figure 76. Control Flow for Shared Data Set Program Loading

links to its Allocate routine which, together with its subroutines, adjusts control section attributes based on the user's authority code and whether the library is shared, and verifies the acceptability of the control section name. The Public attribute is erased if the user authority code is P or O in order to allow a system programmer to receive private copies of any (virtual storage) system routines not included in Initial Virtual Memory. If the Public control section was obtained from a private library, the Public attribute will be erased so that shared virtual storage is not needlessly allocated.

The Dynamic Loader then proceeds to allocate shared or private virtual storage on the basis of groups of control sections with similar attributes. The control sections are grouped in order to minimize the number of calls that must be made on Virtual Memory Allocation.

The Dynamic Loader calls its Get Storage routine to select the proper entry point to the Virtual Memory Allocation. If Get Storage discovers that the public bit is onfor the group of control sections being allocated, Get Storage calls SRCHSDST to see if an entry exists in the Shared Data Set Table for this member. If an entry does exist, Get Storage calls Virtual Memory Allocation at the entry point CONNECT. The function of CONNECT is to assign a segment to the shared page table which maps the control sections to be shared. If there is no entry corresponding to this member, SRCHSDST creates a skeletal entry and notifies Get Storage. Get Storage then calls Virtual Memory Allocation at the GETSMAIN entry point. The function of GETSMAIN is to allocate shared virtual storage. GETSMAIN returns to Get Storage with the Shared Page Table identification number and the relative location of the allocation within the shared segment. Get Storage uses this information to complete the Shared Data Set Table member entry.

Get Storage goes through this procedure for each group of public CSECTS to be allocated. For the first group, the module name is used as a search argument. For each subsequent group, the search argument is the name of the first control section within the group.

Consider, for example, a module, M, of the following control section structure:

CSECT	A	(READ ONLY, PUBLIC)
CSECT	(unnamed)	(PUBLIC)
CSECT	B	(READ ONLY)
CSECT	C	(READ ONLY, PUBLIC)
CSECT	D	(PUBLIC)
PSECT	E	(PROTOTYPE)
CSECT	F	(PUBLIC, VARIABLE)
CSECT	G	(PUBLIC, VARIABLE)

These SDST entries would appear as a result of public storage allocation for M:

- An entry named M describing the control section group composed of A and C.
- An entry named D describing the control section group composed of the unnamed CSECT and D.
- An entry named F describing the variable length CSECT F.
- An entry named G describing the variable length CSECT G.

Note that the module name M is used to identify the first control section group, while the second group is identified by the first named CSECT in the group, D. If the unnamed CSECT had been the only one in the module with just the Public attribute, the CSECT would not have been allocated to public storage. Variable length control sections F and G are allocated storage individually, hence the unique SDST entries for them.

INTERNAL SHARING

Certain tables exist in Initial Virtual Memory which are available to all tasks. An example of such a table is the Shared Data Set Table (SDST). There is only one copy of the Shared Data Set Table and all tasks reference this copy. Internal sharing is also used for the RESTBIs of shared data sets and for Public control sections. Internal sharing is implemented in virtual storage through various facilities of the Virtual Memory Allocation module and by several Resident Supervisor modules and tables. Before discussing the virtual storage interface, an examination of the Resident Supervisor sharing mechanism is necessary.

The key to understanding sharing at the supervisor level is to know the functions and relationships of the:

Segment Table (ST)
Auxiliary Segment Table (AST)
Shared Page Tables (SPT)
External Shared Page Tables (ESPT)
Resident Shared Page Index (RSPI)

The general relationship of these tables is shown in Figure 77.

Private page tables are kept in the task's XTSI. Since a shared segment is, by definition, used by more than one task, keeping a copy of the Shared Page Table in every sharing task's XTSI would involve excessive overhead when modifying the segment's contents. Therefore, Shared Page Tables are contained in Supervisor storage.

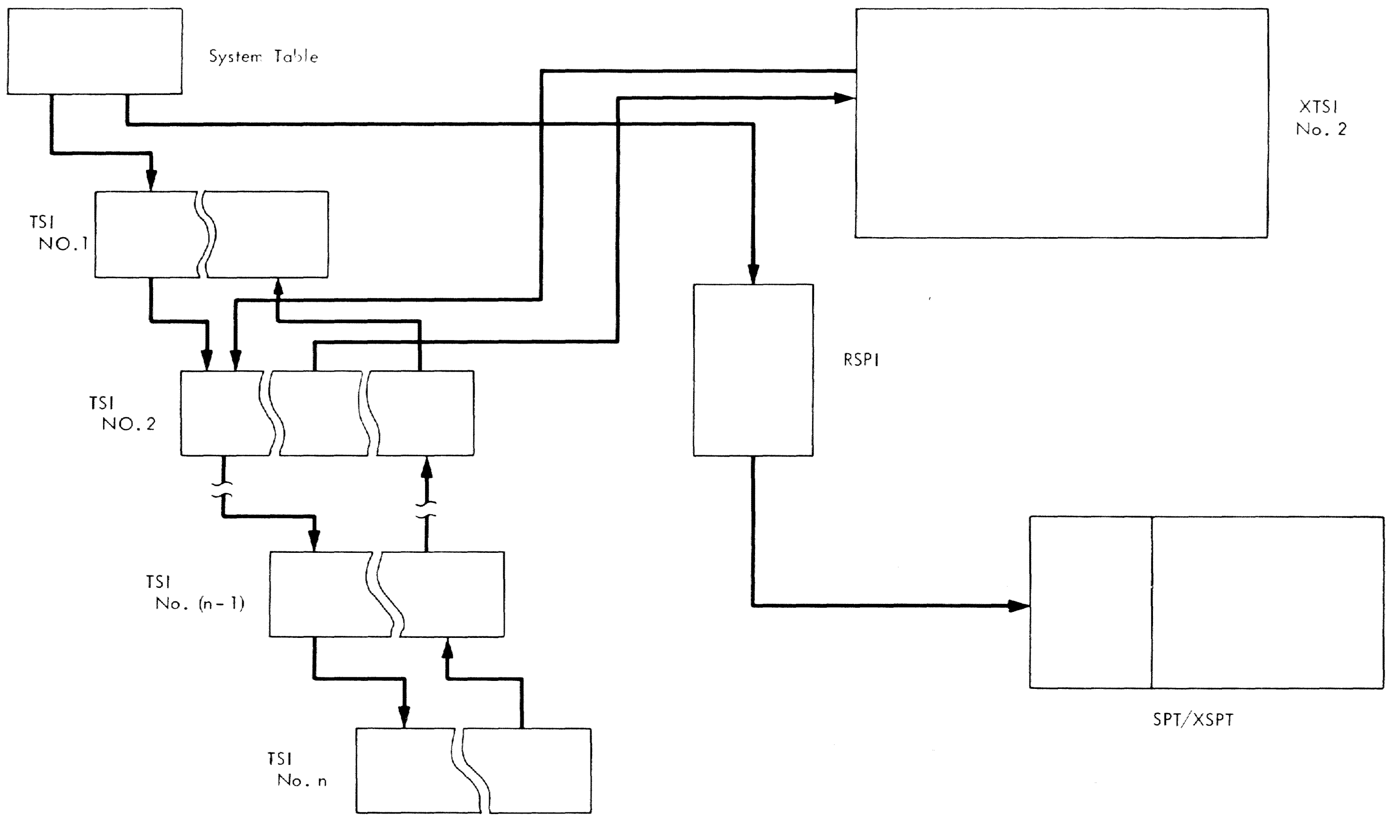


Figure 77. Relationship of Tables Involved in Internal Sharing

Segment Table entries are identical whether the segment is shared or private. A segment Table entry either contains the Shared Page Table address or indicates that the address is unavailable.

The Auxiliary Segment Table is examined only if the Segment Table indicates that the associated page table is unavailable. In this case a program interrupt of code 16 is caused. The Auxiliary Segment Table plays an important role in sharing. First of all, it indicates whether or not the segment is assigned and if the segment is shared. If it is shared, the identification number of the Shared Page Table (SPT#) which describes the segment will be in the AST entry. The SPT# is used by the Supervisor as a search argument to locate an entry in the Resident Shared Page Index (RSPI). The Resident Shared Page Index is located through a pointer in the System Table and is used to update Segment Table entries when they have been marked "page table unavailable."

The RSPI has an entry for each Shared Page Table in the system. Each entry indicates the length, main storage address and SPT#. The RSPI is essential to maintain a continuous description of each Shared Page Table. This is because a Shared Page Table occupies an integral number of contiguous 64 byte blocks obtained from Supervisor Core Allocation. If it is necessary to expand a Shared Page Table (for instance, to load a new module into the Public Segment), it is frequently necessary to obtain a new, larger Supervisor storage allocation and to move the to-be-expanded Shared Page Table to this new location. There must then be some method to associate a segment with its Shared Page Table that is more permanent than the Shared Page Table's address in main storage. This is accomplished through the unique identification number (SPT#) assigned to each Shared Page Table. The Shared Page Table is identical to a private Page Table. It either indicates the main storage address for a page or else that the address is not available. The External Shared Page Table (XSPT) is identical to the private External Page Table (XPT) except the former has an additional word for each entry. This is used, when needed, to point to a chain of GQEs. It is used to enqueue a GQE for any user who wants to access a shared page that is not in main storage but which is already being brought in by another user. While the page is in transit each user's task that tries to reference the page is placed in the page wait status. When the page is finally in main storage the GQEs are again used to locate the waiting task's Task Status Index in order to remove each task from the page wait status.

The relationship of these tables is shown in Figure 78. Control Register 0 points to the beginning of the Segment Table. After the proper Segment Table entry is located, a hardware check is made to determine if the main storage address of the page table is available. (All this is the Dynamic Address Translation (DAT) unit's function.)

If the Shared Page Table is not available, a code 16 program interrupt occurs. To handle the interrupt, the Auxiliary Segment Table entry is examined to determine the proper identification number of the Shared Page Table that describes this segment. The RSPI is then searched on SPT# to find the length and location of the Shared Page Table. This information is then placed in the Segment Table entry.

Whenever a task is in time slice end processing, the Segment Table entry for each shared segment is set to "unavailable" since the Shared Page Table address and length could be changed before this task is given another time slice.

If the Shared Page Table is available, it is inspected by the DAT unit to determine if the shared page itself is in main storage. If the page is available, then the address translation is complete. If the page is not available, a code 17 program interrupt occurs. To handle this interrupt, a page-in procedure is started if it has not already been initiated. The page can be located on auxiliary or external storage by the address stored in the associated External Shared Page Table entry.

The Virtual Memory Allocation routines that are concerned with shared virtual storage are GETSMAIN, CONNECT, and DISCONNECT.

GETSMAIN is the routine used by VAM and the Dynamic Loader to obtain shared virtual storage. GETSMAIN and the Add Shared Pages SVC [ADSPG] that GETSMAIN invokes are used to create a Shared Page Table when one does not exist or to obtain additional space in a segment whose Shared Page Table has already been created. The input parameters to GETSMAIN consist of the number of pages, the type, protection class, Shared Page Table number (SPT#), and a variable allocation indicator. The number of pages is the amount of shared virtual storage requested. The type is either data set, CSECT, or PSECT. The protection class is either Read/Write, Read Only, or Read Protected. The shared page table number is either an existing SPT# or an SPT# of zero.

An existing SPT# is used in cases where data sets are being packed into a shared segment and the SPT# is known from the first GETSMAIN.

An SPT# of zero is used when data is to be placed in a new segment. If a Public segment exists and the type is CSECT, GETSMAIN locates and uses the SPT# of the most recently allocated Public segment.

When the request is for a new segment, GETSMAIN checks to see whether a freed segment exists. If so, it will be used. If not, the next available segment is used.

GETSMAIN does not keep track of where a new request in a packed or Public segment is to be allocated. This is handled by the Resident Supervisor Add Shared Pages (ADSPG) SVC processor. If the request does not fit in at the end of the current segment (no attempt is made to fill in holes within a segment), ADSPG creates a new Shared Page Table and External Shared Page Table, links the next available segment to the new Shared Page Table, and informs GETSMAIN of this fact.

CONNECT is used to allocate shared virtual storage when a Shared Page Table already exists for the data object. The input parameters to CONNECT consist of a shared page table number and a relative page location. CONNECT first checks for a deleted segment and, if found, uses this as the segment number for input to the Connect Segment (CNSEG) SVC. If not, CONNECT uses the next available segment. The shared page table number and relative page location are also used as input to CNSEG.

If a segment is already connected to the Shared Page Table, the corresponding segment number is noted. If not, CNSEG creates a segment table and auxiliary segment table entry and connects these to the Shared Page Table. In either case, the segment address used is passed to CONNECT as output by CNSEG.

DISCONNECT is used to unlink a task's segment table entry from an existing Shared Page Table.

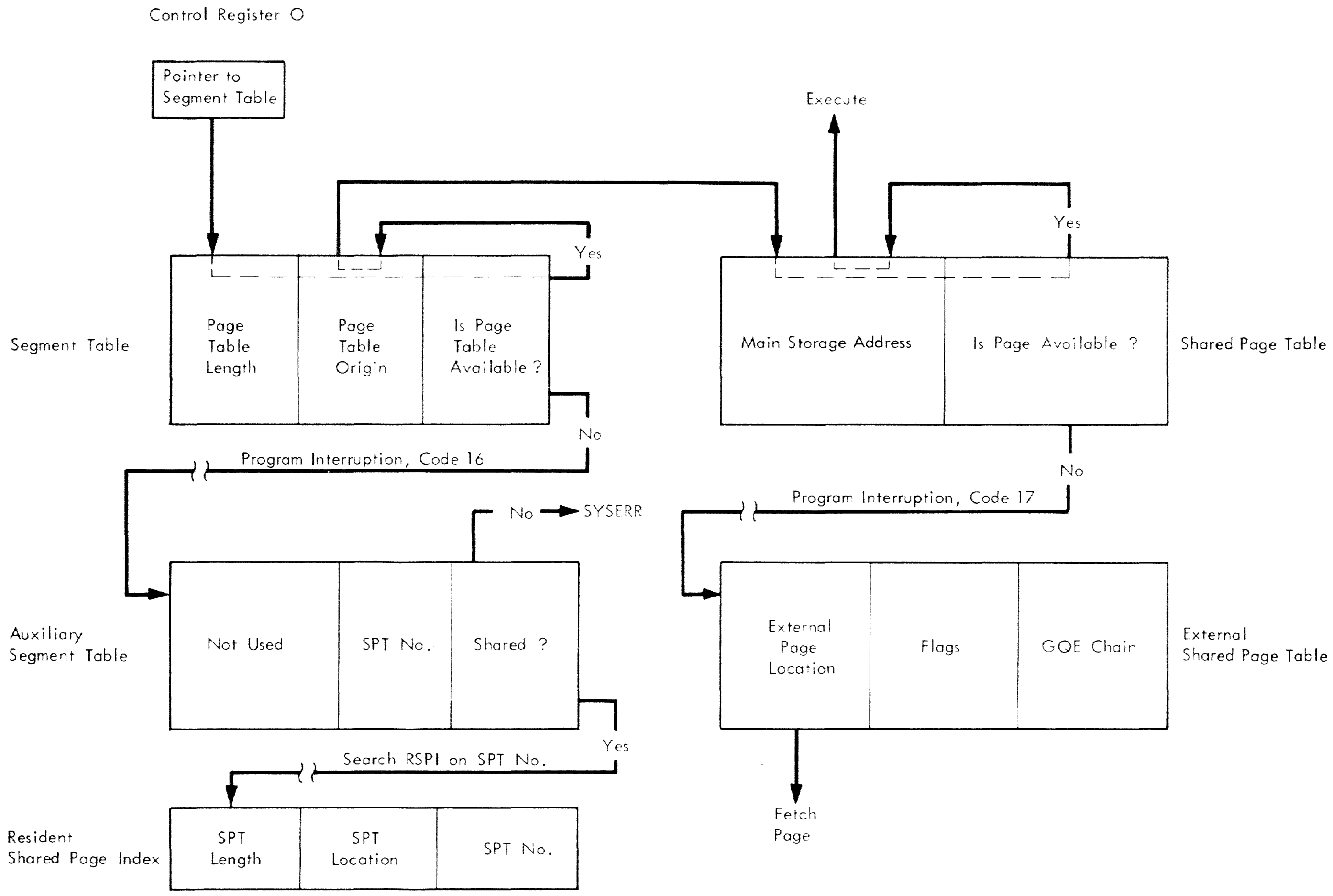


Figure 78. Relationship Between Relocation Tables and Resident Shared Page Index

Paging can be initiated in TSS/360 by the occurrence of any of the following events:

- A Page Relocation exception interruption caused when a task attempts to reference a page not currently in main storage (see "Paging Relocation Exception Example").

- The Internal Scheduler discovers that the first XTSI page, which contains the PSW and relocation tables for a task to be dispatched, is not in main storage.

- After the first XTSI page has been brought into main storage, the Page Posting routine will initiate a paging operation to bring in any additional XTSI pages which exist, and the ISA page.

- The Page Out (PGOUT) SVC is invoked to write one or more data set pages out to external storage (see "Disk Paging" and "Example of Virtual Sequential Processing").

- The Page Out (PGOUT) SVC processor discovers that one or more pages destined for external storage are currently residing on auxiliary storage. These pages must be read into main storage before being written out to external storage.

- The I/O Call (IOCAL) SVC processor discovers that one or more buffer pages for an I/O operation are on auxiliary storage. These pages must be brought into main storage for the duration of the I/O operation.

- The Timer Interrupt Processor receives a time slice end interruption (see "Time Slice End Processing Example").

- The Timer Interrupt Processor is invoked to process a forced time slice end or TWAIT SVC. In these cases a modified time slice end processing will be performed.

- The Write-Shared-Pages subroutine is invoked by User Core Allocation to page-out all changed shared pages that have not been referenced since the last time the subroutine was invoked.

Each of these events, briefly described elsewhere in this manual, has much processing that is unique to the event. However, all have in common the fact that they may initiate paging operations, and this is the aspect with which this section is concerned

The basic control block representing a unit of work within the TSS/360 Resident Supervisor is the GQE. Whenever the work associated with a GQE is a paging operation, one or more additional control blocks, called Paging Control Blocks (PCBs) are constructed and linked to the parent GQE.

Each PCB can contain up to three Page Control Block entries (PCBEs). Each PCBE represents a request to move one page.

The types of paging operations that may be represented by a PCBE are disk paging operations and drum paging operations.

Each of these types of operations can be further classified according to whether the request is for paging into main storage or for paging from main storage to auxiliary or external storage.

To present the overall logic of paging operations, three examples are presented:

1. Page relocation exception (Page-in, drum).

2. Time slice end interruption (Page-out, drum).

3. PGOUT SVC processing (Page-out, disk).

DRUM PAGING

Before discussing the two drum paging examples, the strategy and tables used in TSS/360 to maximize the rate at which pages are moved to or from the IBM 2301 drums used for auxiliary storage must be examined.

The basic strategy used to maximize drum throughput is called slot sorting and this depends on the following organization of the drum. Each pair of tracks is formatted to contain nine page-size records (4-1/2 pages on each track and the track overflow feature utilized between tracks of a pair). Dummy records are written between the page records to allow time for track-to-track electronic head switching. These dummy records are used in recording errors. (See "Error Handling"). Each of the nine pages may be called a "position." Thus, each pair of tracks has nine positions. Since the drum contains 200 tracks or 100 pairs, there are altogether 900 positions on the drum. Corresponding positions may be grouped together. A group of corresponding positions is called a slot. Thus, there are 100 positions in a slot and 9 slots on a drum.

The purpose of this slot organization is to have pages transferred at the maximum transfer rate of the drum (1.2 million bytes per second). A maximum of nine positions may be read from the drum in two drum revolutions. A maximum of nine pages may be written to the drum in two drum revolutions if each of nine slots has an available position. Figure 79 shows the relationship between slot number, and hardware drum addresses. The physical address of a record on a direct access device is composed of a bin, cylinder, head, and record number (BBCCHHR). The bin number is not used on direct access devices supported by TSS/360.

To conserve space, TSS/360 routines and tables (such as the RESTBL), use a relative page number and a symbolic device address to represent the location of a page on a direct access device. The relative page number can be translated into the physical page address when the physical address is needed in a channel program (e.g., in a SEEK or SEARCH command).

When the PCB is associated with a drum, the relative page number is replaced by a one-byte head number and a one-byte slot number. For IVM pages, Startup places a one-byte slot number and a one-byte head number into the two-byte field of the External Page Table or External Shared Page Table entry normally reserved for the relative page number.

Slot sorting may be described as follows: Assume there are a number of drum access requests pending and that a channel program must be constructed to service the requests. For a write request, auxiliary storage must be allocated. Auxiliary storage is allocated in such a way that pages are assigned by slot number in cyclic order. A drum access request indicates whether the operation is read or write and gives a slot number for the page to be accessed. The channel program is constructed in such a way that requests are selected by slot number in cyclic order from the queue of drum paging requests. This process of arranging the requests is called slot sorting.

The two queue processors of the Resident Supervisor that are exclusively concerned with drum paging operations are the Paging Drum Queue Processor and the Paging Drum Interrupt Processor.

		Hardware Address			
Head No.	Slot No.	Bin	Cyl.	Track	Record*
0	1	0	0	0	1
0	2	0	0	0	3
0	3	0	0	0	5
0	4	0	0	0	7
0	5	0	0	0	9
1		0	0	1	1
1	6	0	0	1	3
1	7	0	0	1	5
1	8	0	0	1	7
1	9	0	0	1	9
2	1	0	0	2	1
2	2	0	0	2	3
2	3	0	0	2	5
2	4	0	0	2	7
2	5	0	0	2	9
3		0	0	3	1
3	6	0	0	3	3
3	7	0	0	3	5
3	8	0	0	3	7
3	9	0	0	3	9
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
199	9	0	0	199	9

* The even numbered records are not shown. They are dummy records used for head switching and error recording.

Figure 79. IBM 2301 Drum Symbolic Address Relationships

The primary function of the Paging Drum Queue Processor is to Perform slot sorting and construct channel programs.

Each drum has associated with it two chains of nine channel programs each (one channel program for each slot on the drum), anchored in the System Table.

Each System Table anchor is a work area that contains information describing the status of the drum as well as pointers to the PCBEs, IORCBs, and associated GQEs. These System Table anchors are called Drum Interface Control Blocks and are used by both the Paging Drum Queue and the Paging Drum Interrupt processors.

The System Table DICB contains status indicators plus 18 two-doubleword fields where each field contains the following:

- first doubleword -- complete 7-byte seek argument (BBCCHHR).
- second doubleword -- PCBE/IORCB and GQE address.

An overview of this relationship between control blocks is shown in Figure 80.

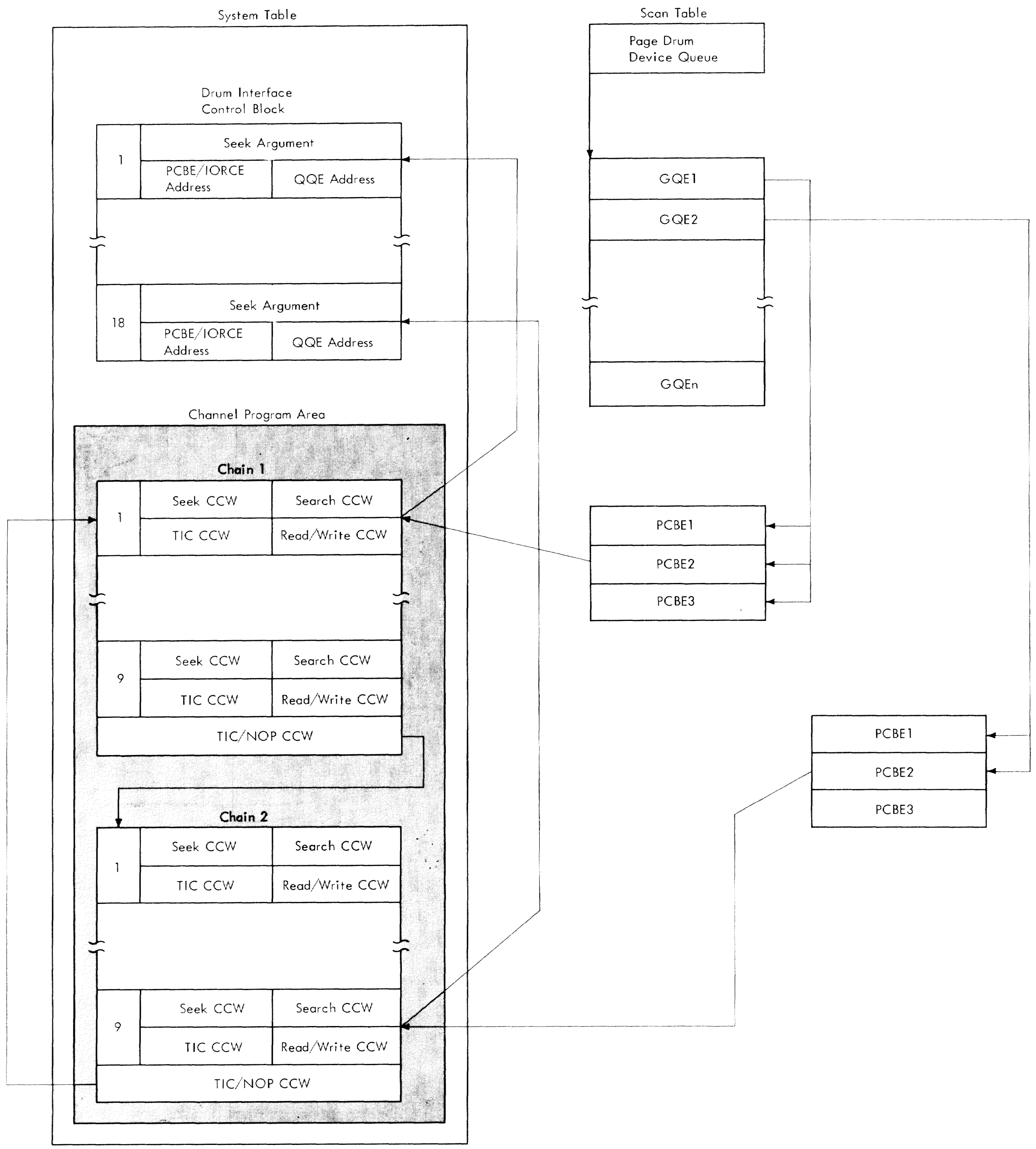


Figure 80. Relationship Among Control Blocks Associated With Drum Paging

The primary function of the Paging Drum Interrupt Queue processor is to process interruptions from the one or more paging drums on the system.

These interruptions can generally indicate one of three conditions:

- An error condition
- A channel program controlled interruption
- The drum has reached the end of its channel program

An overview of error handling for paging operations is presented in "Error Control" and will not be discussed here.

The drum channel program is constructed in such a way that each set of Channel Command Words (CCWs) is command chained to the next. This is done in an effort to keep the drum continually running.

However, when the paging queue processors cannot keep ahead of the drum, it is necessary to invoke the Start I/O subroutine to restart the drum.

Because the intent is to keep the drum continuously running, there must be some way to notify the Paging Drum Interrupt Processor whenever a maximum number of pages has been transferred. Among other reasons, this is done because a task cannot make use of a page that has been brought into main storage until the page's new location has been posted in its page tables. Similarly , the main storage occupied by a page written out to a drum cannot be released until the results of the operation have been posted.

Therefore, the drum channel programs are constructed in such a way that a channel program controlled interrupt (PCI) flag is used to make nine pages the maximum that may be posted for any given interrupt.

If the Paging Drum Queue Processor determines from the System Table that a GQE representing a program controlled interruption has been stacked on a paging drum interrupt queue, the processor will exit to the Queue Scanner to give the Paging Drum Interrupt Processor a chance to post the completed paging operations as quickly as possible.

When the Paging Drum Interrupt Processor has completed its duties, it exits to the Queue Scanner.

The Paging Drum Queue processor performs slot sorting and channel program construction.

Upon entry, the Page Drum Queue Processor sets the DICB slot mask to all ones. The GQE-PCBs are then scanned in a sequential manner starting with the first GQE. The first time each GQE is processed, the PCB count will be copied to the PCB unprocessed and unposted fields. When a PCBE is found for which a channel program can be built, (i.e., slot is available) the channel program will be built in the DICB area immediately, using the slot number as a pointer to the proper program.

Each channel program is built by merely setting the proper opcode and address in the Read/Write commands. If the channel program involves a write check operation, the nontransmit bit must be set. If the channel program is for a DRAM operation, the suppress incorrect length bit must be set on. As each Read or Write without validity check PCBE is processed, the GQE and DICB slot masks will be updated by turning the corresponding slot bit off, reducing the PCBE unprocessed count by one, and setting the PCBE processed bit on. As each Write with validity check PCBE is processed, the DICB slot mask is updated, and the PCBE write check flag is turned on. As each channel program is generated, the DICB "HIGHEST SLOT ASSIGNED" field for the command chain being built must be maintained.

When all available slots are filled (i.e., the DICB slot mask goes to zero), the scan must continue to the end of the GQE currently being processed, in order to maintain the GQE slot mask properly. For each PCBE which is passed over (e.g., nonprocessed because the corresponding slot has already been taken), the program must set on the corresponding slot bit in the GQE slot mask.

Any PCBEs encountered which have the bypass flag on are ignored, with the exception that the PCBE unprocessed and unposted counts are reduced by one, and the PCBE processed flag is turned on.

When all PCBEs for a GQE have been processed (i.e., the PCBE unprocessed count is zero), the GQE will be dequeued from the Page Drum Queue.

The four CCWs required for any drum paging are a Seek (this command switches the heads to the proper track), a Search on ID equal (looking for the correct record on a track), a TIC (to branch back for an unsuccessful search), and a Read or Write, depending on the particular PCBE being processed. The CCWs are all command chained.

A command chain is completed in the following cases:

<u>All slots have been filled</u>. If the command chain just built is the first of the two,

the last CCW generated (Read/Write) will have its command chain bit turned off. In addition, if the drum is running (i.e., the second chain is being processed), the first CCW following the channel program of the last slot used in the second chain must be set to a TIC pointing to the first CCW of the channel program for the first slot used in the first chain.

If the command chain just processed was the second of the two, the ending CCW (TIC/NOP) will be set to a NOP. In addition, if the drum is running (i.e., the first chain is being processed), the last generated CCW of the first chain will have its command chain bit turned on, and if it is not the last CCW of the chain, the next CCW will generate a TIC command pointing to the first CCW of the channel program for the first slot used in the second chain.

No more work to be done, or more work to be done but not for any available slot. In addition to the processing described above, a check will be made to ensure that a drum revolution will not be wasted. The DICB "HIGHEST SLOT USED" fields for chains I and II, and the DICB slot mask may be used to determine the first and last slot used of the chain just built, and the last slot used in the previous chain, if any. This information can be used to connect the two chains, if possible, or to set the CAW if an SIO is necessary.

If the drum is not running (i.e., the "drum is currently running" flag is not on), the Pathfinding subroutine is called and the symbolic device address of the drum is passed to it. Pathfinding responds by returning the address of a physical device path. The Paging Drum Queue processor then initiates the channel program execution by calling the Start I/O subroutine, and processing continues as previously described.

Figure 81 depicts a drum program that might be generated.

PAGING RELOCATION EXCEPTION EXAMPLE

The flow through the system for this example is shown in Figure 82 which is keyed to the following description.

1. This example describes a typical paging-in operation resulting from a relocation exception, and drum paging is assumed. This particular situation might arise for a conversational task on its first entry into the Task Monitor. Previous page-in situations for the task may have already occurred (e.g., the Dispatcher bringing in the XTSI page and Page Posting causing the

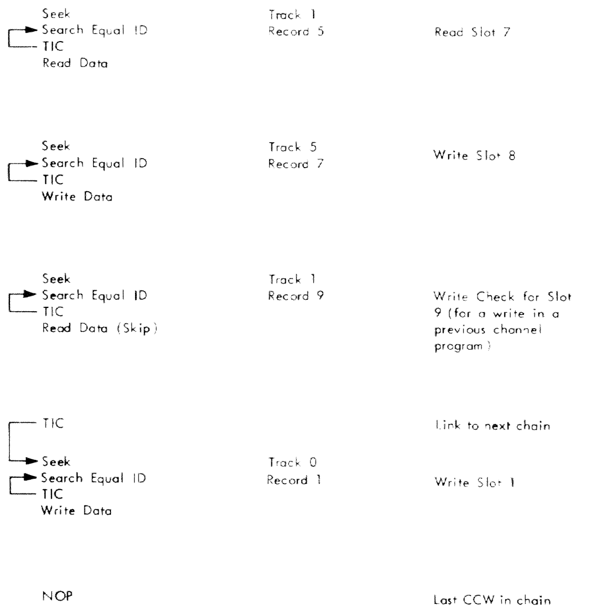


Figure 81. A Possible Channel Program Generated by the Page Drum Queue Processor

ISA to be read in). However, these paging operations do not involve a page relocation exception interruption (code 17).

Moreover, we assume for this example that both chains of channel programs are running, and that both chains are busy performing Write operations, due to Time Slice End paging operations.

2. Entry to the Resident Supervisor is at the Program Interrupt routine of the Interrupt Stacker.

3. Supervisor Core Allocation is invoked to get main storage for the GQE.

4. The Interrupt Stacker determines that the interruption code is greater than 15 and so directly links to the Program Interrupt Processor. The Program Interrupt Processor is never suppressed and is re-enterable. By linking directly, faster service can be provided for this type of interruption which occurs frequently. (Interruptions with codes 1-15 are sent back to the task as software interruptions without being touched by a Supervisor Queue Processor).

5. Having determined that this is a code 17 interruption, the Program Interrupt processor (PIP) picks up the virtual storage address that caused the interruption (from Control Register 2) and calls FINDPAGE. Extended

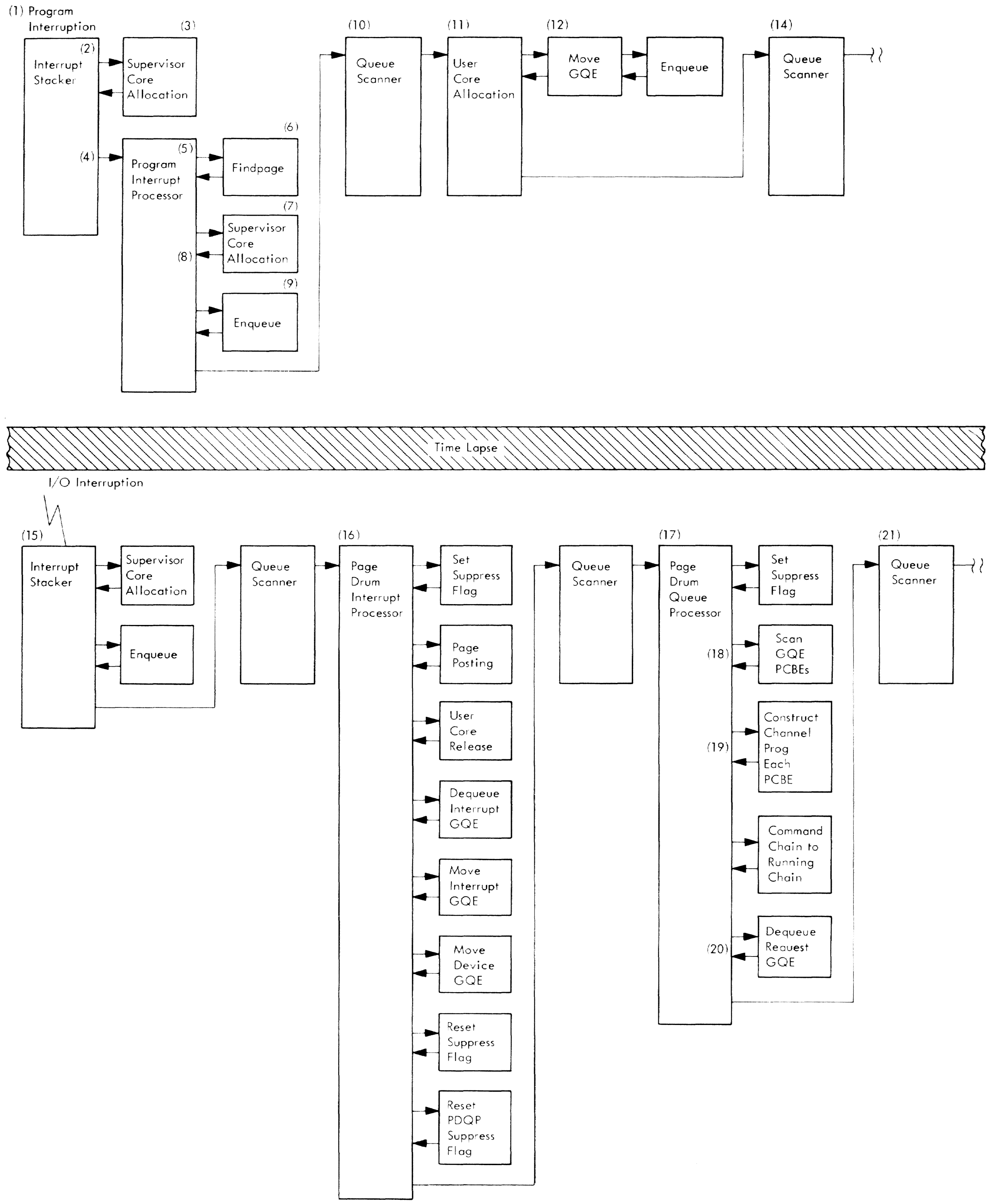


Figure 82. Resident Supervisor Paging Relocation Exception (Part 1 of 2)

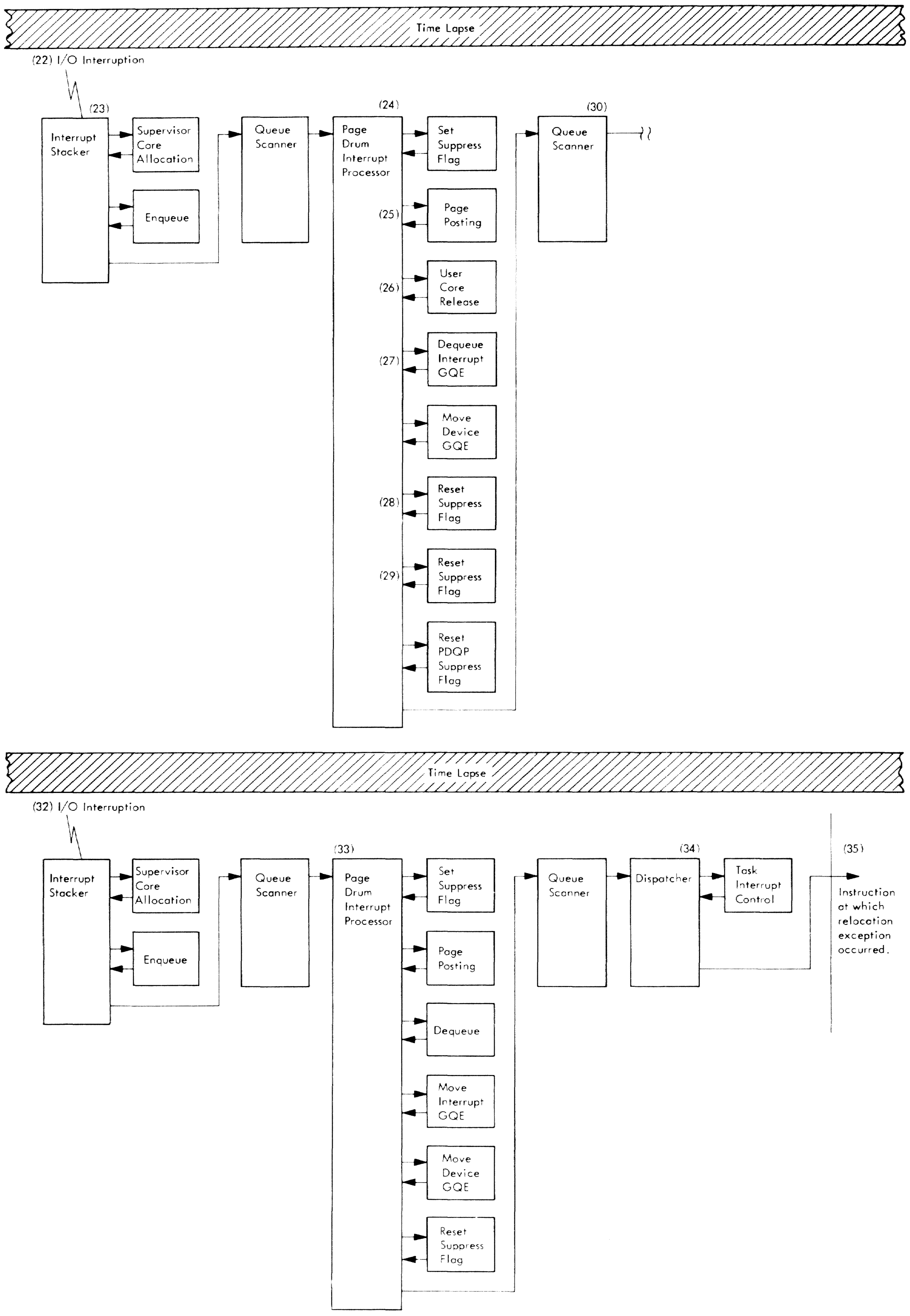


Figure 82. Resident Supervisor Paging Relocation Exception (Part 2 of 2)

6. FINDPAGE returns the main storage location of the Page Table entry and External Page Table (XPT) entry for the virtual storage address furnished to it.

7. PIP checks the Page Table entry and finds the page still not available. PIP backs up the instruction counter in the PSW stored in the XTSI such that the instruction counter now points to the first byte of the instruction that incurred the relocation exception, rather than the first byte of the following instruction. PIP then goes to Supervisor Core Allocation to get 64 bytes in which to build a Page Control Block (PCB).

8. The PCB is linked to the GQE and PCB fields are filled. (External location from the XPT entry and virtual storage address from Control Register 2.) PIP places in the first GQE routing field the queue number of the User Core Allocation Queue, and, in the second routing field, the number of the Drum Device Queue.

9. ENQUEUE places the GQE on the User Core Allocation Queue to obtain a page of main storage into which the page may be read.

10. PIP changes the task status in the task's TSI to page wait, and then exits to the Queue Scanner.

11. User Core Allocation searches the Core Block Table, finds an available block, and puts its address in the Internal Address field of the PCB. This is the location into which the page will be read.

12. MOVE GQE shifts the routing fields in the GQE and finds the drum queue number now in the first GQE routing field and calls ENQUEUE to put the GQE on that queue.

13. MOVE GQE then returns to User Core Allocation which, in turn, exits to the Queue Scanner.

14. The Queue Scanner will not invoke the Page Drum Queue Processor because the suppress flag is on since both chains are busy. As soon as one chain completes, the suppress flag will be reset by the Page Drum Interrupt Processor.

15. At some later point, a PCI interruption, designating completion of one of the chains invokes the Page Drum Interrupt Processor (PDIP).

16. PDIP calls Page Posting to post completion of the Writes from that chain. In addition, PDIP resets the Page Drum Queue suppress flag, and exits to the Queue Scanner.

17. The Queue Scanner invokes the Page Drum Queue Processor (PDQP). PDQP first invokes the Set Suppress Flag routine to prevent entry into the queue by another CPU. The Set Suppress Flag routine sets the appropriate flag in the Scan Table entry for this queue.

18. The Page Drum Queue Processor scans the GQE-PCBs in a sequential manner in order to locate a PCBE for which a channel program can be built.

19. When an appropriate PCBE is found, the channel program is built in the DICB area immediately, using the slot number as a pointer to the proper program. Associated with each slot number is a two-doubleword field containing the complete 7-byte seek argument (BBCCHHR), the PCBE or IORCB address, and the GQE address.

20. When all PCBEs for a GQE have been processed, the GQE is dequeued from the Page Drum Queue.

21. Since this paging operation will be chained to the chain currently running, i.e., both chains will be currently running, the suppress flag is not reset. The Page Drum Queue Processor then exits to the Queue Scanner.

    Ultimately, when all processable GQEs have been handled, the Dispatcher will be invoked to select a task for execution. However, since the task which just experienced the page relocation exception is in Page Wait status, the task is not eligible for consideration. Some other ready task will be chosen.

22. At some later point the channel will reach the CCW containing the PCI flag. This will generate an I/O interruption, taking the CPU away from whichever task was operating in problem state.

23. Standard Interrupt Stacker processing is performed for an I/O interruption. However, since the Interrupt Stacker determines in this situation that the interruption came from a paging drum, the GQE is sent to the appropriate Page Drum Interrupt Queue. (There is one such queue for each drum.)

24. The Page Drum Interrupt processor (PDIP) interlocks its queue.

25. Assuming that the pages whose I/O completion was signalled by the PCI were being written out, Page Posting will merely decrement the Page I/O Count in the TSIs for the tasks to which the pages belong. No further processing will be done until the count reaches zero.

26. Since pages were just written out, the main storage blocks they occupied can be released. A call to User Core Release accomplishes this.

27. The interruption GQE is removed from the Page Drum Interrupt Queue (PDIQ) and erased via MOVE GQE.

28. The device GQE and associated PCB are removed.

29. This PDIQ is now unlocked since the GQE that had been enqueued there has been processed.

30. The chain is now free, so PDIP resets the suppress flag for the Page Drum Queue Processor, and exits to the Queue Scanner.

31. The Dispatcher is invoked to select a task for execution, and, as the task is still in Page Wait status, some other ready task is selected. One more pass through the series of steps will be made before the page required is posted as being in main storage, and the task changed from Page Wait to Ready status.

32. Sometime later the Interrupt Stacker will receive the I/O interruption indicating channel program termination. The Interrupt Stacker will send the interruption GQE to the Page Drum Interrupt Queue.

33. The Page Drum Interrupt Processor interlocks its queue and calls page posting to post completion of the Read. Since the count has reached zero, Page Posting will mark the task Ready. PDIP then releases the interruption GQE and the device GQE and its associated PCBs, unlocks the Page Drum Interrupt Queue, and exits to the Queue Scanner.

34. When the task is finally selected for execution again, TIC will find no task interruptions pending.

35. Consequently, the task will be started once again and will attempt to perform the instruction which originally caused the relocation exception (since PIP had moved the instruction counter back from the succeeding instruction).

TIME SLICE END PROCESSING EXAMPLE

The flow through the system for this example is shown in Figure 83 which is keyed to the following description.

1. The situation described herein occurs at the end of a time slice for a task. It is triggered by the timer value decrementing below zero, generating an external interruption.

2. The Interrupt Stacker is entered at the External Interrupt Processing Routine.

3. A GQE is created and placed on the Timer Interrupt Queue.

4. The Queue Scanner finds work to do in the Timer Interrupt Queue and invokes the Timer Interrupt Processor (TIP).

5. The Timer Interrupt Processor determines if the task should be removed from the Dispatchable list. If the task's scheduling parameters indicate that it would be immediately returned to the Dispatchable list, the TIP saves time by omitting its paging and time slice end processing and simply releases the space occupied by the GQE and returns to the Queue Scanner.

   If TIP determines that the task will not be returned to the Dispatchable list right away, it calls Rescheduling which recomputes the task's scheduled start time and places the task on the Eligible list.

6. The task status in the task's TSI is changed to time slice end. Supervisor Core Allocation [SCA] is called to get space for the PCB(s) needed to control the paging-out process.

7. TIP scans all main storage pages of the task. When it finds a nonshared page that has not been changed since it was brought in, the main storage is returned to the appropriate core block pending queue via User Core Release (since a good copy is still available on auxiliary or external storage). TIP decrements a system count of timer interruptions. If it becomes zero, it is reset and the Write Shared Pages subroutine is called to initiate page-out operations on all shared pages that have not been referenced since the last time the count was reset.

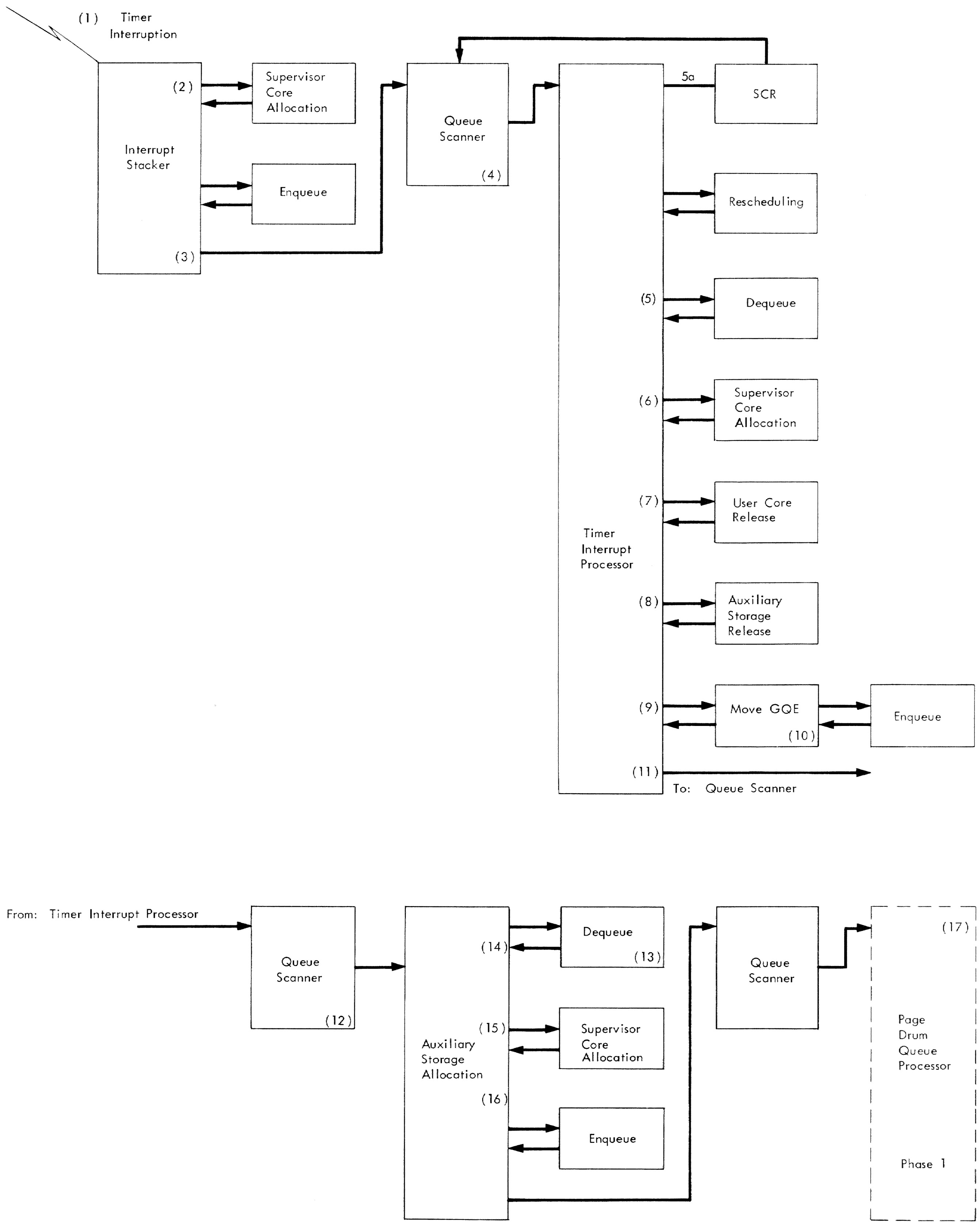


Figure 83. Resident Supervisor Time Slice End

8. When a changed page that came in from auxiliary storage is found, a call is made to Auxiliary Storage Release to make available the auxiliary storage location which now contains an obsolete version of the page.

9. TIP constructs a PCB entry for each changed page. TIP fills in the Internal Address field with the current main storage location of the page. The PCB(s) are then linked to the GQE. TIP places the symbolic queue number of the Auxiliary Storage Allocation(ASA) Queue in the second field of the GQE routing area.

10. MOVE GQE shifts the routing fields and consequently calls ENQUEUE to place the GQE on the ASA Queue.

11. TIP has now set up the paging operations and so exits to the Queue Scanner.

12. The Queue Scanner, finding a GQE on the ASA queue, invokes the Auxiliary Storage Allocation Queue Processor to handle the GQE on its queue.

13. Auxiliary Storage Allocation calls DEQUEUE to remove the GQE from its queue for processing.

14. The ASA Queue Processor assigns a drum location for each page and fills the PCBE External Location field with the address chosen. The ASA queue processor also posts this value to the appropriate XPT entry, the address of which has been furnished to the queue processor by TIP.

15. When more than one paging drum is available, the Time Slice End Processor, CEAKT, will set a pointer into the ASA Table to the drum with the greater available page count. ASA will use this pointer and allocate all storage requests to this drum.

16. ASA then calls ENQUEUE to place the GQE on the appropriate Page Drum Device Queue.

17. From this point the drum I/O operation is identical with the paging-in process described in the section: Paging Relocation Exception Example.

DISK PAGING

This section describes how disk paging differs from drum paging. It also describes the specific manner in which disk paging is performed and presents an example of disk paging flow.

The three instances in which disk paging may be necessary are the occurrence of a Program Interruption Code 17 (Relocation Exception), page migration and VAM Page Out (PGOUT). The example which is presented here is for the last case, a VAM PGOUT.

In drum paging, an extensive effort is made to maximize the rate at which pages are moved to or from the drum by slot sorting. The resulting channel program which performs the paging, transfers up to nine pages with a minimum amount of rotational delay.

In disk paging, there is no process analogous to slot sorting. Disk paging is performed for one GQE at a time in order that the transfer of the page can be posted as soon as possible. Furthermore, not only are disk paging request GQEs handled serially (one at a time and in order), but so are the PCBEs associated with the GQE being processed. For example, the first PCBE may specify a read from cylinder 5 of the device, the second PCBE a read from cylinder 6, and the third PCBE another read from cylinder 5. That is precisely the order in which the channel program would read the pages into main storage. No attempt would be made to sort together the two reads from cylinder 5 to lower the seek time.

If the paging request concerns writing to an auxiliary storage disk (as in TWAIT processing), the PCBEs associated with the paging request GQE will contain contiguous disk storage addresses if such storage was available. This will result in a fairly efficient channel program with a minimum of seek time (cylinder-to-cylinder arm movement). The reason for this efficiency is that Auxiliary Storage Allocation attempts to allocate auxiliary disk storage from one cylinder, if possible.

Disk paging requires the services of the Resident Supervisor to build and initiate the execution of a channel program to perform the I/O to or from a non-drum direct access device (e.g., 2311, 2314). The two Resident Supervisor queue processors which are primarily concerned with processing non-paging I/O request (see "Example of BSAM Processing") are also the queue processors that coordinate disk paging operations. These are the Device Queue processor and the Channel Interrupt processor. While it is true that these two queue processors are called to process disk paging request and interruption GQEs, they perform only minimal functions for disk paging, and rely heavily on sub-processors called the Page Direct Access Queue subroutine (PDAQS) and the Page Direct Access Interrupt subroutine (PDAIS).

The Device Queue Processor (DQP), when given control by the Queue Scanner, immediately checks to see if the first GQE in the queue has an I/O Request Control Block (IORCB) associated with it. The absence of an IORCB for a GQE is taken by the DQP to mean that the GQE is a request for disk paging. In this case the DQP calls the Pathfinding routine to obtain an available channel and control unit to the device specified by the GQE. Once a path is found, the DQP passes control to the Page Direct Access Queue Subroutine.

The main functions of the Page Direct Access Queue Subroutine are to:

- Build a Direct Access Interface Block.
- Call the Start I/O subroutine to initiate the paging I/O.

The Direct Access Interface Block (DAIB) built for the GQE contains four subareas:

1. Header (pointers to the other parts of the DAIB and a save area)
2. Seek and Search Argument Table (used to obtain parameters for building the CCWs.)
3. Channel Program Area
4. Entry Area (contains one-for-one entries each pointing to the PCBE for each single paging operation in the channel program; used for posting to the page tables.)

The DAIB format is shown schematically in Figure 84.

The PDAQS builds a skeleton header for the DAIB then calls the External Page Location Address Translator (EPLAT), passing it a pointer to the PCB for the GQE; the pointer is in the GQE itself. The EPLAT routine converts the symbolic address in the PCBE (symbolic device address,relative page number) into a physical address: i.e., bin-cylinder-head-record number (BBCCHHR). Storage areas of direct access devices are organized into records, tracks and cylinders. Based on the type of device on which the paging is to be done, the EPLAT routine calculates a unique record, head and cylinder number for the relative page number it has been given. The bin number is not used for direct access devices supported by TSS/360. The BBCCHHR for each PCBE constitutes,when returned to

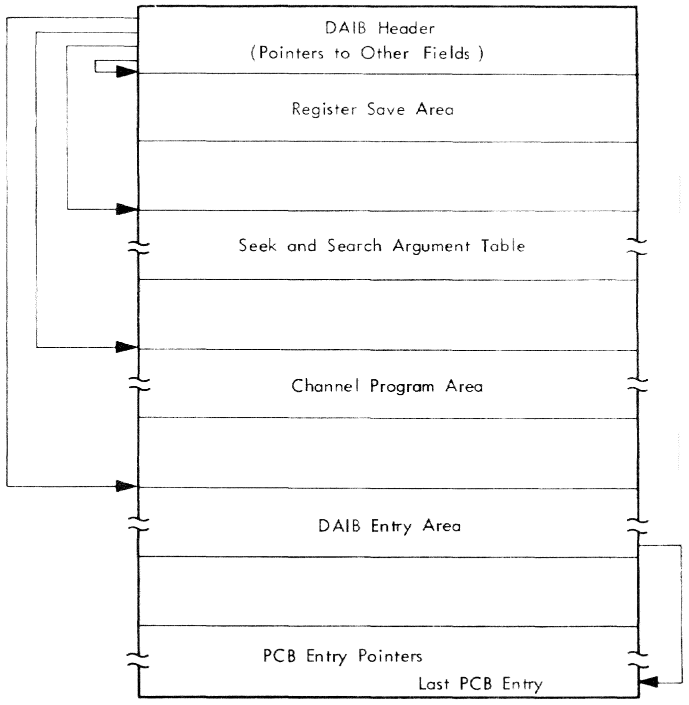


Figure 84. General Format of DIAB

PDAQS, the entry for that PCBE in the DAIB Seek and Search Argument Table. When all of the PCBEs of the GQE have been processed through EPLAT, PDAQS builds the channel program, using the information in the Seek and Search Argument Table to construct the CCWs. For each channel program built to transfer a page to or from the address specified in an entry in the Seek and Search Argument Table (SSAT), PDAQS places a pointer to the pertinent PCBE (that PCBE which contains the symbolic address from which the SSAT entry being processed was translated) into the next consecutive position in the DAIB Entry Area. The entries in the Entry Area are therefore in the same order as the channel programs. To complete the bookkeeping, a pointer to the "last PCB entry posted" is maintained by the Page Direct Access Interrupt subroutine so that, as interruptions occur for the device, the correct page table can be posted. The usefulness of the Entry Area can be seen by looking at a disk paging channel program for two pages as constructed by PDAQS:

Seek (not command chained - I/O interruption occurs)

Seek

Search on ID equal

Transfer in Channel (TIC)

Read or Write Data (first page transferred)

Seek (not command chained - I/O interruption occurs)

Seek

Search on ID equal

TIC

Read or Write Data (second page transferred)

The significant detail is the use of two seeks that are not command chained to each other. This is done so that the channel will not be tied up waiting for an access arm to find the correct cylinder. After the device end I/O interruption occurs signifying the completion of the seek operation, the Page Direct Access Interrupt Subroutine, will eventually receive control and will know by inspecting the DAIB that it is to execute Start I/O on the channel program at the point of the second seek. The device is still marked busy, so no other I/O operation can cause the access arm to move. Therefore, the second seek does not cause the access arm to move and is necessary only to set up the internal register of the device control unit since it is entirely possible that another device has used the control unit since the first seek.

The same technique is used for each paging operation in the channel program. Thus, an I/O interruption occurs after each page is transferred. This allows page tables to be updated as soon as possible. Page Posting is called for this purpose. Page Posting is supplied with a pointer to the Entry Area so that Page Posting can find the correct PCBE (the one which is associated with the just completed paging operation). After each page posting,the Page Direct Access Interrupt subroutine checks the DAIB for remaining channel programs. If there are any remaining, they will be initiated by calling Start I/O. If there are no more pages to be transferred for the DAIB, PDAIS exits to Queue Scanner.

In discussing the PDAQS and DAIB, the Page Direct Access Interrupt Subroutine (PDAIS) was described indirectly. To reiterate, PDAIS:

- Calls Start I/O to initiate the next paging operation in the DAIB,if there is one.
- Calls Page Posting to update the Page Tables.
- Maintains the entry area pointer so that it always points to the PCBE associated with the next paging operation.

Example of Disk Paging

In the following example, the Page Out Routine (PGOUT), called by the SVC Queue Processor at the time of the PGOUT SVC, has just completed its function of validity checking the I/O Page Control Block passed to it by the VAM MOVEPAGE routine and possibly bringing in pages from auxiliary storage, if any were found to be on the drum and not in main storage. The PGOUT routine then enqueues GQEs upon the device queues of the device or devices to which the VAM pages are to be transferred. PGOUT obtains the symbolic addresses of the locations to which the paging is to take place from the entries in the IOPCB passed to it by the VAM.

The flow through the system for this example is shown in Figure 85 which is keyed to the following description. For the purposes of the example, it is assumed that there is only one active task in the system at the time of the PGOUT SVC.

1. The PGOUT Routine is called by the SVC Queue Processor to check the IOPCB built by the VAM routine Move Page for certain errors. Assuming the page to be paged out is in main storage at the time of the SVC, PGOUT will call the Enqueue routine to place the GQE on the queue of the device to which the page is to be written. The PGOUT routine then exits to the Queue Scanner.

2. The Queue Scanner invokes the Device Queue Processor. This GQE is a paging request.

3. The Device Queue Processor invokes the Pathfinding Subroutine to find and assign a real path (device, control unit, channel, channel controller) to the device.

   Upon return from Pathfinding, the Device Queue Processor checks the IORCB flag in the GQE to determine if this is a paging or nonpaging request (paging, in this example).

   The Device Queue Processor then checks the flag in GQE indicating whether the Channel Interrupt Processor has just finished processing a paging interruption on this device.

   At the present point such processing has not been finished.

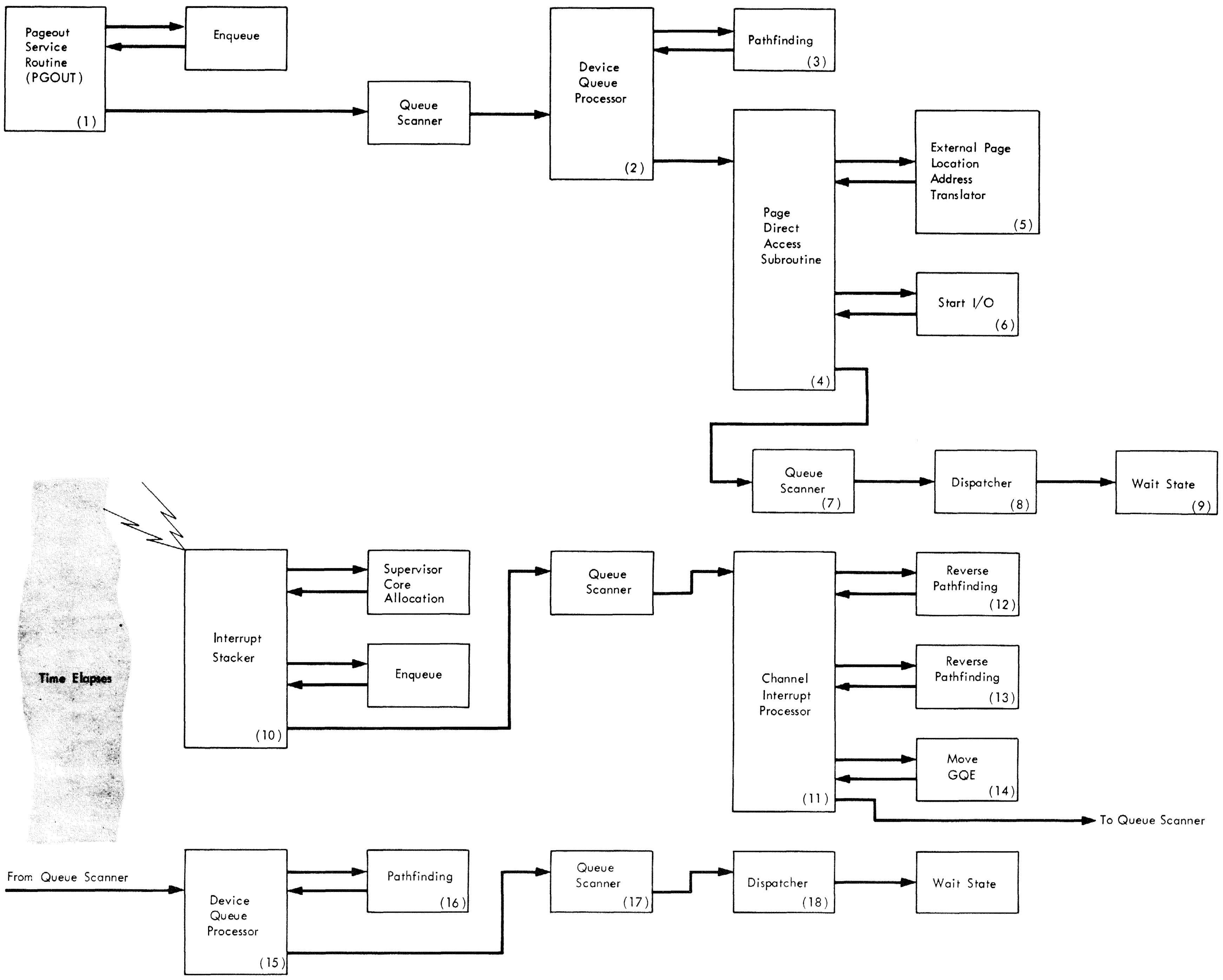


Figure 85. Disk Paging Example (Part 1 of 2)

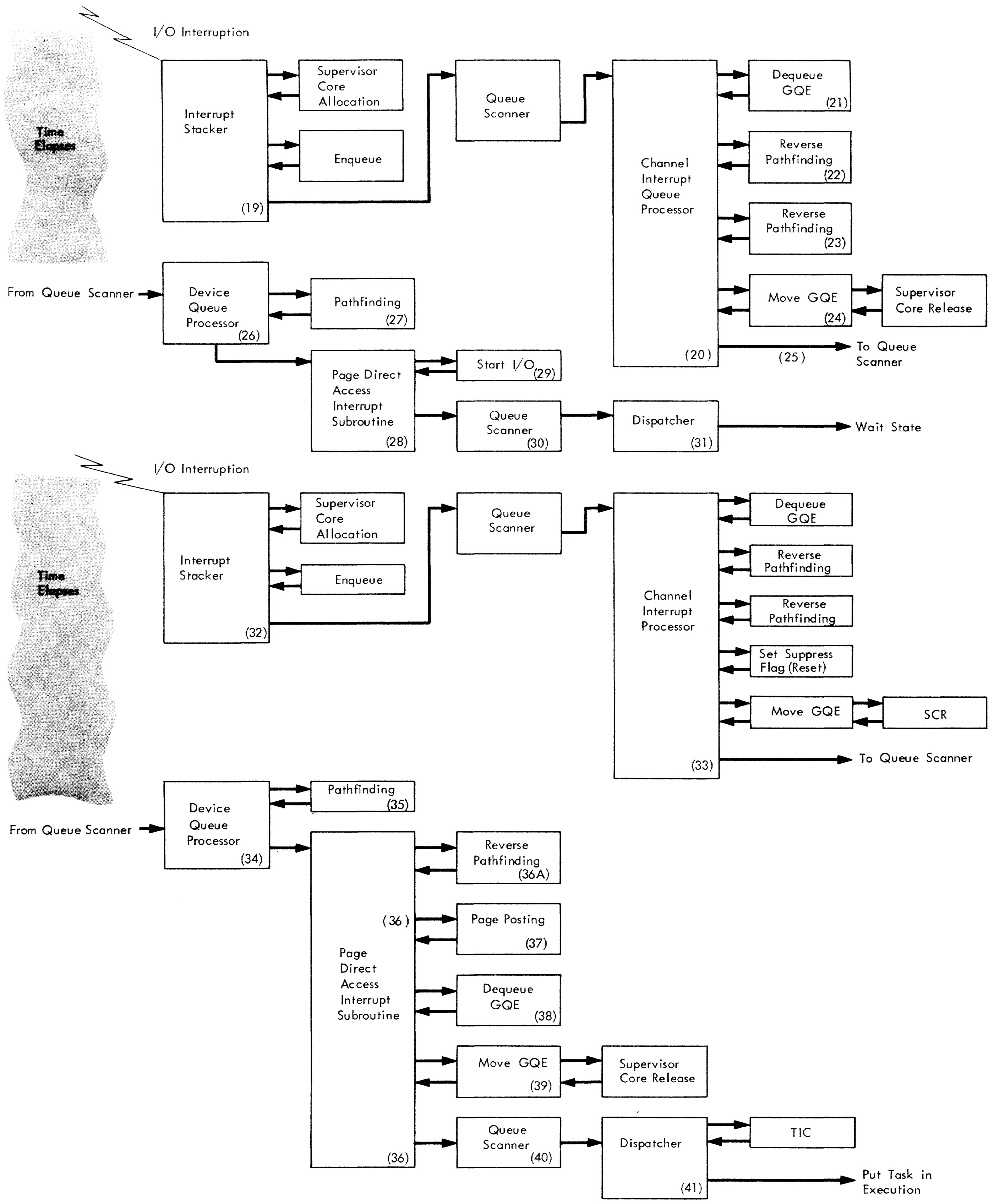


Figure 85. Disk Paging Example (Part 2 of 2)

4. The Device Queue Processor then links to the Page Direct Access Queue Subroutine (PDAQS). It is the responsibility of this routine to build the channel command sequences and issue a call to START I/O. The PDAQS builds a Direct Access Interface Block (DAIB) for each PCB. (The DAIB holds the command sequence.) The DAIB is functionally quite similar to the Drum Interface Block.

5. Using the relative page number and device type, the External Page Location Address Translator calculates the proper ~~drum~~ address (cylinder,head, record) of the page.

disc

The Page Direct Access Queue subroutine then completes the command sequence. The resulting channel program for a one page I/O transfer would look like this:

Seek ____________ not command chained

Seek

Search on ID equal

Transfer in Channel (TIC)

Read or Write Data

Notice that omitting the command chain flag between the two seeks prevents the channel from being tied up while the access arm is moving to the desired track. The second seek is necessary only to set up the internal address registers of the device control unit, since the arm should be correctly positioned due to the first seek.

6. The Page Direct Access Queue Subroutine now sets an input switch and calls the Start I/O Routine to initiate the execution of the channel program in the DAIB.

7. The Page Direct Access Queue Subroutine then exits to the Queue Scanner. The PDAQS has done all possible work at this time for the paging GQE. Under the specific case where this is the only task in the system, there are no other GQEs to process so the Queue Scanner exits to the Dispatcher.

8. The Queue Scanner calls the Dispatcher, where, in this case, no other active tasks are found.

9. Since there is no other task to use the system, the Dispatcher places the system in wait state with all interruptions permitted.

10. The I/O interruption for the channel end occurs indicating that the seek is now under way. A GQE will be built and placed upon the Channel Interrupt queue since the interruption came from a disk.

11. The Channel Interrupt Processor (CIP) is invoked and it uses DEQUEUE GQE to remove the GQE from the Channel Interrupt Queue.

12. CIP then invokes Reverse Pathfinding to translate the hardware device address stored by the channel end interruption into the appropriate symbolic device number.

The Channel Interrupt Processor then determines if the I/O interruption is solicited (in this example it is). This is accomplished by checking the Device Queue for this symbolic device and finding the proper flag set indicating that an I/O operation has been started. Note: the first GQE on the Device Queue is the soliciting GQE.

13. Reverse Pathfinding is invoked to make the channel and the control unit free in the Pathfinding Table. As a result of the single seek of the first part of the channel program, a channel end is indicated in the channel status word stored as a result of the I/O interruption. The CIP sees the channel end indicator, and calls Reverse Pathfinding to free the channel and control unit. The device is still working at this time, and since no device end was received, the device will not be freed at this time. The CIP then continues processing. It uses the symbolic device address to determine the soliciting GQE. CIP resets a suppress flag in the Scan Table entry for the Device Queue to allow the Device Queue Processor to process the GQE on its queue a second time. CIP then moves the CSW information from the GQE which was attached to the Channel Interrupt Queue to the device GQE (attached to the Device Queue), and sets a flag in the device GQE to indicate a paging interruption has occurred.

14. CIP next invokes Move GQE to release the GQE by calling on Supervisor Core Release. Note: The GQE movement field should contain 'FF' as the next location for this GQE. The 'FF' causes Move GQE to release the GQE.

15. The CIP has completed its processing for now and exits to the Queue Scanner. The Queue Scanner eventually invokes the Device Queue Processor.

The Device Queue Processor notes that only channel end has occurred, not device end. This implies that the seek is under way.

16. The Device Queue Processor then calls Pathfinding to obtain a path once again to the paging device. The full path is not available because the device is still busy. The Device Queue Processor then sets a suppress flag in the Scan Table entry to indicate I/O is under way for this device.

17. Exits to the Queue Scanner, since, until an I/O interruption occurs signalling device end, execution of the DAIB cannot continue because the device is still busy (seek operation going on).

18. In this example there is nothing left to do but wait. The TSI is in page wait and the only queue with a GQE is locked so there is nothing for the system to do.

19. An I/O interruption will occur with device end indicating that the seek is complete. It will be necessary to issue another Start I/O to begin the actual paging operation. A GQE is created and stacked on the Channel Interrupt Queue.

20. The Channel Interrupt Processor processes the device end interruption.

21. The Channel Interrupt Processor links to Dequeue GQE to remove the GQE from the Channel Interrupt Queue.

22. Then, CIP links to Reverse Pathfinding to translate the hardware device address into a symbolic address. Using the Symbolic Device Address as a queue number, CIP determines if this is a solicited I/O interruption.

23. CIP links to Reverse Pathfinding again, this time to free the device. Having determined the soliciting GQE the CIP resets a flag in the Scan Table to allow processing by the Device Queue Processor. CIP then moves the CSW information from the GQE, formerly on the CIP queue, to the soliciting GQE on the Device Queue. Next, CIP sets a flag in the soliciting GQE to indicate that a paging interruption has occurred.

24. CIP finally links to Move GQE to release the interruption GQE through Supervisor Core Release.

25. Exit to the Queue Scanner.

26. The Device Queue Processor is invoked. It recognizes the completion of the seek and wishes to initiate a Start I/O sequence that will perform the data transfer.

27. Pathfinding is invoked to find and assign a real path to the device. (Device , control unit, channel, channel controller.) Now the entire path is available, since the first seek is complete. The IORCB flag in the GQE is checked to determine if this is a paging or nonpaging request. It is a paging request. The flag is checked to determine if a paging interruption has occurred. In this example it has.

IOPCB

28. The Device Queue Processor then links to the Page Direct Access Interrupt Subroutine (PDAIS) to test to see if an additional channel program in the DAIB should be initiated (it should).

29. Start I/O is invoked to start the I/O to execute the channel program for the actual paging.

30. The PDAIS is finished for now and exits to the Queue Scanner.

31. Again, because there is only one task in the system and because the task is in page wait and the queue with the GQE is locked, the wait state is entered.

32. Channel end and device end are received indicating the completion of the paging operation. A GQE is created and stacked on the Channel Interrupt Queue.

33. The Interrupt Stacker exits to the Queue Scanner. The Channel Interrupt Processor is invoked. Note that the sequence here is the same as previously occurred involving reverse pathfinding and then the releasing of the GQE. (Steps 19 to 25.)

34. CIP exits to the Queue Scanner. The Queue Scanner invokes the Device Queue Processor. At this point it is recognized that the paging I/O has been completed.

35. Pathfinding is invoked to obtain a path. The path will not be used since there are no more paging operations to be executed in the DAIB. The PDAIS will free the path later through Reverse Pathfinding. The Device Queue Processor makes several checks: paging or non-paging check (in this example, paging). Is there a flag indicating paging interruption? (Yes).

36. The Page Direct Access Interrupt Subroutine is entered to determine if the paging operation is complete. It is.

37. Page Posting is called to post the page (mark it available) in the proper location in the XTSI.

38. Dequeue GQE is called to remove the GQE from the chain of GQEs attached to the device queue of the direct access device to which paging has just been completed.

39. Move GQE is called to remove the device GQE and associated PCBs through Supervisor Core Release.

40. Since the Page Direct Access Queue Subroutine has completed its processing, the Queue Scanner is called to look for other work to perform. For this example there are no other GQEs to process and Queue Scanner exits to the Dispatcher.

41. This time the Dispatcher will find the task ready to use a CPU and will dispatch the task, thus ending this disk paging example.

ERROR RECOVERY AND RECORDING

TSS/360 provides the means for attempting recovery from several types of errors and for the recording of a variety of data pertaining to certain error conditions. Provision has also been made for the retrieval of this error data for analysis by responsible system personnel.

ERROR DEFINITION

TSS/360 defines errors according to four basic types. Each type is handled individually and in a unique manner. The four classifications of error are:

1. Machine errors
2. Paging I/O errors
3. Task I/O errors
4. System errors

Machine errors are caused by hardware malfunctions in central processing units, storage elements, or channel control units and are detected by the checking circuits of the unit involved. These errors are indicated by the generation of a machine check interruption.

Hardware malfunctions are further divided into internal and external machine check errors. The former type occurs in a CPU or a storage element while the latter occurs in a channel control unit.

Task and paging I/O errors are the result of hardware failures in devices, device control units, and channels and are detected by checking for abnormal end indicators in the Channel Status Word (CSW).

I/O errors are called task I/O errors when they are associated with a channel program constructed in a task's virtual storage and passed to the Resident Supervisor through the IOCAL supervisor call. I/O errors are called paging I/O errors whenever they are associated with paging channel programs.

This distinction is made because the bulk of error processing logic is performed by virtual storage routines in the case of task I/O errors and by Resident Supervisor routines in the case of paging I/O errors.

A system error occurs when some system function in TSS/360 cannot operate as expected. Errors of this type are detected by the Resident Supervisor, which reports the error by means of the ERROR macro instruction, or by a privileged virtual storage program, which reports the error by means of the SYSER macro instruction. The SVC interruption which results from the execution of one of these SVC instructions will cause control to be transferred to the system error processor with the contents of the CPU registers preserved.

Included in these macro instructions is an indication of the nature of the error. Three classifications are used:

Major
: an error detected by programming or signalled by a program interruption, the effect of which is assumed to be global or is unknown.

Minor
: an error detected by programming or signalled by a program interruption, the effect of which is assumed to be localized to a task.

Hardware
: an error detected by programming, which is assumed to be the result of a machine error (CPU or main storage) that was not detected by the machine. (System routines do not normally assume that an error condition is the result of an undetected hardware failure.)

ERROR RECORDING

Error handling mechanisms are implemented in order to minimize the impact of errors on the overall operating environment. The various mechanisms, while differing in detail, all adhere to the same general procedure. The immediate reaction to an error signal is the analysis of the error condition. If retry of the failing operation is possible, it is attempted.

In all cases the error environment is recorded in some manner. This information is then made available to qualified system personnel. Some I/O errors are not important enough to be recorded permanently unless they occur an excessive number of times on a single device. Taking this into account, certain errors, if successfully retried, are duly noted internally. Only when the same (successfully retried) error occurs 16 times on one device (over a period of time) is it permanently recorded. This type of error is called intermittent.

Other errors are of such a nature that they cannot be retried. These errors and errors which are unsuccessfully retried are called solid and are permanently recorded whenever they occur.

Error recording is performed differently according to whether the error is paging I/O, task I/O, system, or machine.

The separation of task and paging I/O errors is continued in the error recording portion of the system. Those errors pertaining to paging operations are recorded by the Real Core Statistical Data Recording (RCSDR) routine and the Real Core Error Recording (RCER) routine. Task I/O errors are recorded by the Virtual Memory Statistical Data Recording (VMSDR) routine and the Virtual Memory Error Recording (VMER) routine). The logic of these error recording procedures is similar. Figure 86 depicts the error data recording interface.

PAGING I/O ERROR RECORDING

The occurrence of an I/O error on an auxiliary paging device or a VAM formatted external storage device may result in an entry to RCER. Before this entry is made,

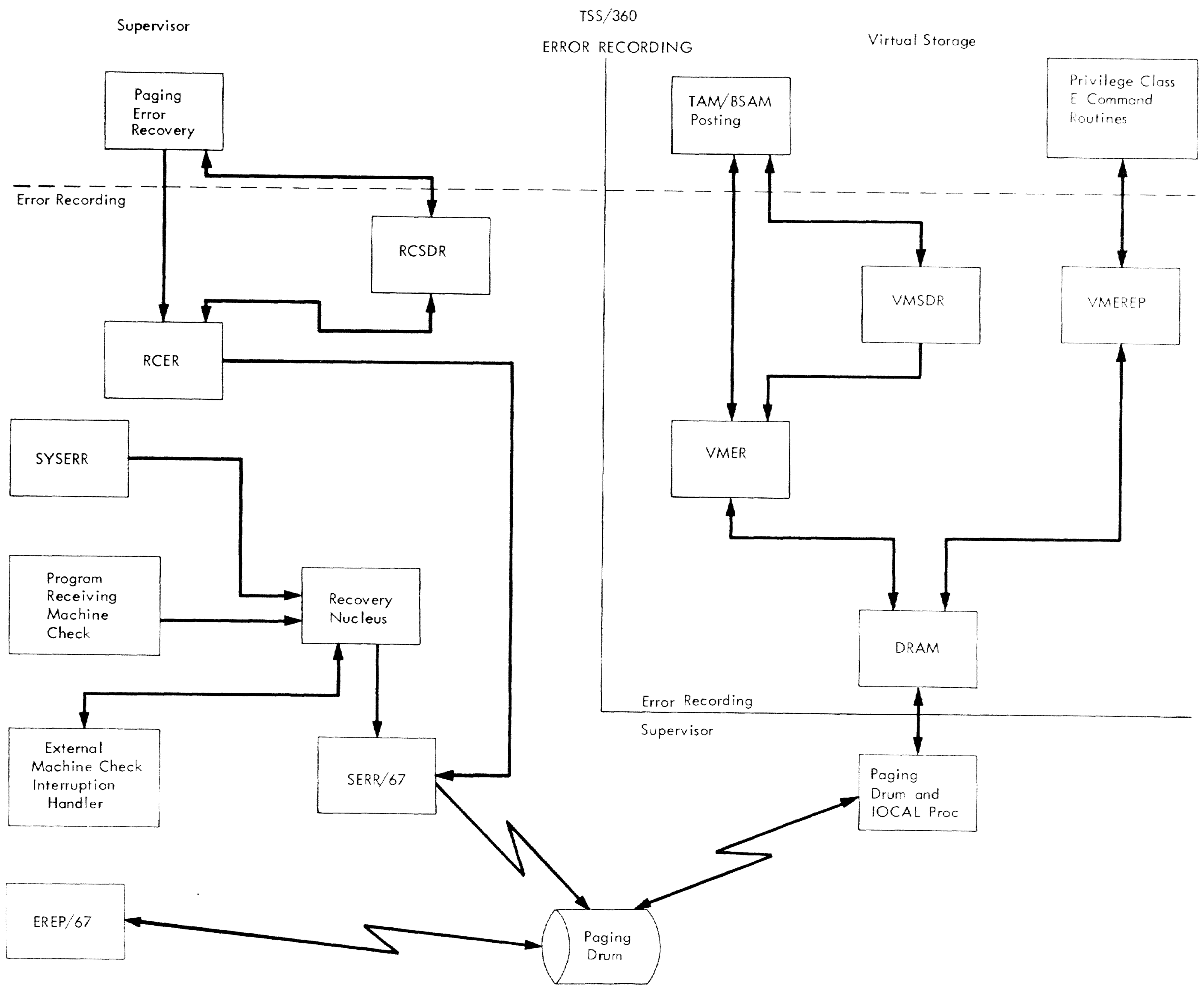


Figure 86. Error Data Recording Interface

the error will have been analyzed and possibly retried.

If the error was not successfully retried or could not be retried at all, control is directly passed to the Real Core Error Recording routine. RCER formats an error record containing a full description of error.

RCER then invokes the SERR Bootstrap module, which pages-in Environment Recording to place the error record in those areas of the paging drum not required for paging (i.e., the inter-page gaps).

If an error has been successfully retried, its occurrence will be noted in the Paging Direct Access Statistical Data Recording Table. Each device has its own entry in this table and each type of error is assigned a four bit entry or "bucket". One of the functions of the Real Core Statistical Data Recording Routine is to increment the error count in the appropriate bucket as each error occurs. If the bucket overflows (i.e., the error recorded is the sixteenth of its type to occur on the device), a call is made to the Real Core Error Recording Routine for record formatting and for permanent recording.

TASK I/O ERROR RECORDING

The logic of task I/O error recording is very similar to that of paging I/O error recording. I/O errors occurring on task I/O devices are passed by the appropriate posting routine to the Virtual Memory Statistical Data Recording Routine. This routine is analogous to the Real Core Statistical Data Recording Routine. VMSDR accumulates statistics on intermittent task I/O errors and calls the Virtual Memory Error Recording Routine on bucket overflow or solid failures.

Virtual Memory Error Recording formats the error record and writes it to the drum via DRAM (Drum Access Method).

SYSTEM ERROR RECORDING

The detection of a programming or a suspected hardware error by the Resident Supervisor or by a privileged virtual storage program will eventually result in an error indication message to the operator's terminal and, possibly the generation of a dump to tape.

The data supplied to the operator's terminal includes a code representing the source, nature, and severity of the error; and the ID of the user whose task was in execution when the error occurred.

An error processed by the system error processor may be one of five types:

- Real core minor software
- Real core major software
- Virtual memory minor software
- Virtual memory major software
- Hardware

For each type, a switch may be set indicating that control be given to the Time Sharing Support System. In this case, system error analysis can be conducted and a dump taken.

If the switch is not set, processing continues by:

- Resuming at the point of interruption in the case of real core minor software errors.
- System restart via the Recovery Nucleus in the case of real core major software errors or hardware errors.
- A return to the caller in the case of virtual memory minor software errors.
- Abnormal task termination of the offending task in the case of virtual memory major software errors.

MACHINE ERROR RECORDING

Machine check errors are recorded by the Environment Recording Program of the System Environment Recording and Retry (SERR) group. Such information as channel logout data, machine register status and user identification is recorded on the interpage gaps of the paging drum or drums.

An immediate print program in the SERR group records a limited amount of information on the operator's terminal when a machine check error occurs. The single line message this program delivers includes such information as the ID of the failing CPU or storage element and the type of error.

ERROR RECORD RETRIEVAL

Two means are provided for retrieving task and paging I/O errors and machine check error records from the drum. Virtual Memory Error Recording Edit and Print (VMEREP) is a virtual storage program which runs under TSS/360. It can selectively retrieve records from the drum and print them and also release the drum space for further recording. VMEREP accesses the drum via DRAM.

The Error Recording Edit and Print (EREP/67) program provides the same services as VMEREP but is a stand-alone program and is not run under TSS/360.

In addition to these, a stand-alone program is provided which is capable of dumping the machine check environment in edited format. This program is known as System Environment Recording, Edit, and Print (SEREP). This program may be invoked when the system is unable to perform any error recording function.

ERROR RECOVERY AND RETRY PROCEDURES

TSS/360 error recovery and retry procedures are designed to dynamically correct errors or to minimize the effect of errors on the system as a whole. Although the specific recovery procedures differ for each type of error, the general approach to recovery is the same. Failing operations are retried where possible, failing hardware devices are checked and intermittent failures retried. System errors are analyzed to determine their impact. Where an operation cannot be retried at all or is retried without success, and when a hardware element cannot be made to perform correctly, the failing element or device is removed from the system in an orderly manner so as not to disrupt system operation. It is only as a last resort, when recovery is not possible and when removal of the failing component would render the system inoperative, that the system is shut down.

In the case of errors resulting from I/O operations, if the retry fails, even after repetitive attempts, or if a necessary retry operation exceeds the capabilities of the analysis program, a "hard" failure is recognized and fault localization, to the component level, is invoked. The malfunctioning component is set unavailable and an environment record is generated for later analysis by Customer Engineering personnel.

CPU, storage element or channel control unit failures are indicated by a machine check or malfunction alert. The occurrence of a machine check in duplex mode will result in the failing CPU being placed in the wait state and a malfunction alert being sent to the other CPU. The occurrence of a machine check in simplex mode or the receipt of a malfunction alert in duplex mode results in an entry to the Recovery Nucleus.

The error environment is recorded by SERR which also analyzes retry possibility. (Note: A maximum of two simultaneous machine check interruptions will be accepted in an attempt at recovery. The occurrence of a machine error during SERR operation will result in retry being considered not possible.)

If SERR successfully retries the instruction on which the machine check occurred, the affected task or the Resident Supervisor is resumed at the point of interruption. If retry is not possible or the error is solid, the offending CPU, storage element or channel control unit will be set unavailable. The system, in its reduced configuration, may be restarted or system operation may resume depending on the circumstances at the time the machine check error occurred.

The term I/O retry, when used in this document in connection with I/O errors, is defined by the following rule: I/O retry shall proceed along the control path which initially produced the error incident in an attempt to hold the error environment constant. The retry procedure shall continue until: (a) the I/O sequence is executed without error, or (b) a maximum number of retry operations have been attempted, whichever occurs first. The maximum number of retries is a device dependent number obtained from a system table.

I/O fault localization is performed along an alternate data path or paths, by executing the failing I/O sequence (CCW list) as though it were the original attempt.

All components of the Resident Supervisor associated with error control are resident in main storage with the exception of the SERR and Reconfiguration modules which normally reside on all the IBM 2301 paging drums in the system and which operate in an overlay area assigned at system startup. There exists only one copy of the Recovery Nucleus and SERR Bootstrap modules in TSS/360. However, each PSA (one exists per CPU) contains an Error Recovery Control Table which provides the CPU with unique work and control areas for these modules. Thus, the integrity of multi-CPU operations is preserved. The Recovery Nucleus identifies the various types of machine check interruptions which may occur and directs processing of the recovery procedures. The SERR Bootstrap program is charged with the responsibility of calling the Reconfiguration or other SERR modules for the purpose of error recording, instruction analysis and retry, CPU/storage checkout, and logical partitioning of CPUs, storage elements, and channel control units.

Figure 87 represents the overall relationship between the Supervisor Error Control modules. Four entries are shown, each

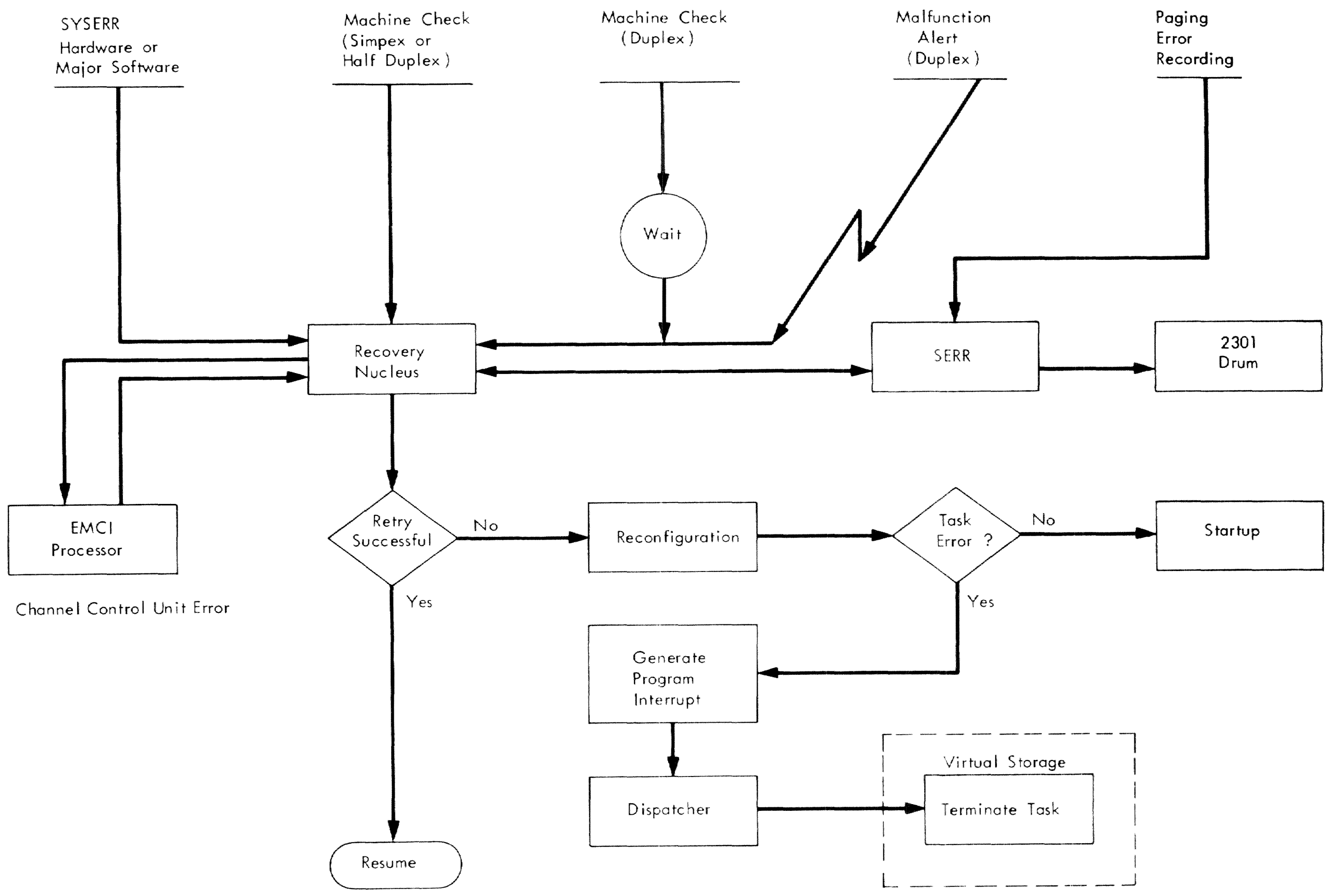


Figure 87. Resident Supervisor Error-Handling Overview

of which reflects the occurrence of a specific interruption. Three exits are possible: either system operation resumes, an automatic system restart (simulated IPL sequence) is initiated, or the affected task is abnormally terminated and system operation continues.

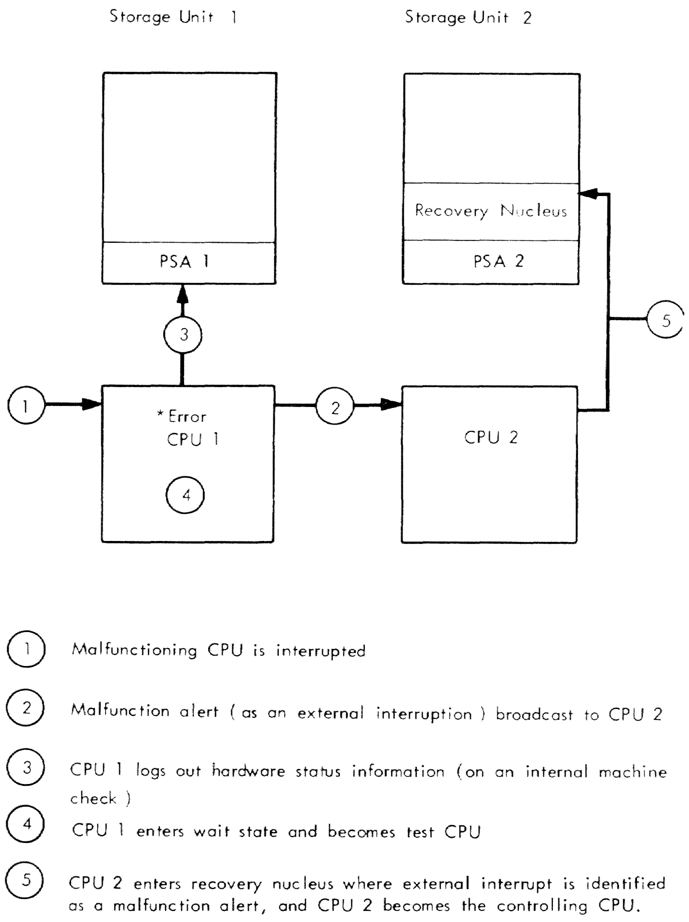
MACHINE ERRORS

When a machine error is detected, certain actions are automatically carried out by the hardware. First, the CPU which is involved is interrupted by a machine check interruption and hardware status information is logged out into the prefixed storage area (PSA) for that CPU. If there is another CPU in the system, a malfunction alert is sent to this CPU as an external interruption. The sending CPU then enters the wait state and becomes the test CPU. The receiving CPU enters its own Recovery Nucleus routine which resides in its PSA and becomes the controlling CPU. This processing is described schematically in Figure 88.

If there is only one CPU available in the system, a machine check interruption will cause CPU control to pass directly to the Recovery Nucleus. The Machine Check New PSW for the single CPU situation would have been initialized to effect this transfer, whereas in the two CPU case, the Machine Check New PSW for each CPU would have had the proper bit in the Machine Check New PSW set in order to cause the affected CPU to enter the wait state. (The sending of a malfunction alert and the logging out function are both automatically generated by the hardware at the time of the machine check.)

Once the CPU is in the Recovery Nucleus, the interruption code associated with the Old (Machine Check) PSW in the failing CPU is examined to determine whether the interruption was caused by an internal machine check or by an external machine check.

If the interruption was caused by an external machine check, control is transferred to the External Machine Check Interrupt processor which attempts to pinpoint



Storage Unit 1
Storage Unit 2
Recovery Nucleus
PSA 1
PSA 2
* Error
CPU 1
CPU 2
1
2
3
4
5
Malfunctioning CPU is interrupted
Malfunction alert (as an external interruption) broadcast to CPU 2
CPU 1 logs out hardware status information (on an internal machine check)
CPU 1 enters wait state and becomes test CPU
CPU 2 enters recovery nucleus where external interrupt is identified as a malfunction alert, and CPU 2 becomes the controlling CPU.

Figure 88. Initial Machine Actions on the Detection of a Machine Error in a Duplex CPU System

the error and to take appropriate corrective action which will cause the error and its consequences to be invisible to the system. If this cannot be done, it may be necessary to create GQEs to simulate I/O interruptions for all I/O being processed by the Channel Control Unit and then reset the CCU. Additionally, it may be necessary to issue Prepare commands for 2702 terminal lines controlled through the CCU. This will result in what will appear to the routines involved as a channel error. The External Machine Check Interrupt processor then invokes the error recording module of SERR (System Environment Recording and Retry) to record information on the error. After recording, system operation is resumed. If the error is a solid error the CCU will be marked unavailable and the system will be re-started.

If the interruption was an internal machine check, SERR is called via the SERR Bootstrap module for recording and retry analysis.

SERR is a collection of routines that assess the error and record the error environment. Each of the SERR routines operate in the same overlay area and are brought into main storage by the Bootstrap module.

The following is a simplified example of SERR operations. The first routine invoked is the Checker routine. The Checker routine will save key portions of the error environment. Then, the CPU Memory Checkout routines will be invoked. The first is an instruction checkout routine which uses all the instructions of the System/360 instruction set and compares functional answers. The second routine checks out the CPU's read-only store (i.e., ROS) and local store control circuitry. The next routine to be invoked is the Pointer routine whose function is to locate the failing instruction. This routine is necessary, because the Instruction Counter may not be valid. Next, the Restore and Validate routine will analyze whether the instruction can possibly be retried. For instance, an ROS parity error can usually be retried. On the other hand, a bad parity indication in a critical register will prevent the instruction from being retried because there is no way to reconstruct the correct data. The Memory Checkout routine would next be invoked to determine whether a particular path to a memory from a CPU is faulty or whether there are any memory bytes with bad parity. Following the memory checkout, the Instruction-Execution-Retry routine is invoked. This routine forms a damage report which will be used by the Recovery Nucleus to determine what further action should be taken. The Environment Recording routine next preserves the appropriate error information, as discussed in "Error Recording." Following the environment recording, the Instruction-Execution-Retry routine is invoked a second time to determine whether the elapsed time between the current error and the last previous error is short enough to indicate a solid error condition. Lastly, the Immediate Print routine sends a message to the system operator and the Checker routine restores the saved system environment and restores control to the Recovery Nucleus.

If the instruction is retryable, the Recovery Nucleus attempts to continue system operation from the instruction at which the machine check occurred. If the instruction is not retryable, or the failure is solid, the Reconfiguration routine is invoked. The Reconfiguration routine determines, mostly on the basis of the SERR analysis, whether system operation can resume or whether the system must be restarted. If (as in a simplex system) an automatic restart is impossible, Reconfiguration places the system in the Wait state and awaits operator intervention. If a solid error occurred while the system was in the Supervisor state, the system must be restarted. If the problem state, system operations can generally continue.

Reconfiguration sets any solidly failing hardware device logically unavailable. Reconfiguration then sends a message describing the extent of the damage to the system operator. If the system is to be restarted, Startup Prelude is read in and the system is restarted. However, if system operation can be resumed without logical partitioning, STARTUP is not called and system operations resume. For instance, if less than four User Core pages are faulty, they can be set unavailable in the Core Block table and partitioning is not necessary.

If the failure occurred in the problem state, Reconfiguration will set up the necessary control blocks and information for notifying the affected task or tasks and then will cause system operations to resume. As part of this procedure, Reconfiguration will queue an ABEND request in the SERR Auxiliary Queue for the affected task. After control has been returned to the system, the Interrupt Stacker will remove the request and build a GQE on the SVC Queue Processor requesting that the affected task be abnormally terminated.

PAGING I/O ERRORS

Paging I/O errors can occur either during the process of starting a paging operation (i.e., when Start I/O is issued) or during the operation itself. The former are called immediate failures and the latter are called delayed failures. An overview of paging error handling is presented in Figure 89. In either event, a module called Paging I/O Error Recovery Control is entered. The purpose of this module is to identify the component involved, record the error, and initiate appropriate recovery procedures. If the failure is immediate it may be either an inboard or an outboard failure. An inboard failure is a channel failure. An outboard failure is a failure associated with a control unit or device. If the Start I/O operation has been retried a prescribed number of times without success, Paging I/O ERROR Recovery Control calls the Alternate Path Retry routine which will make part of the first path unavailable and retry the operation along an alternate path to the component if one can be found.

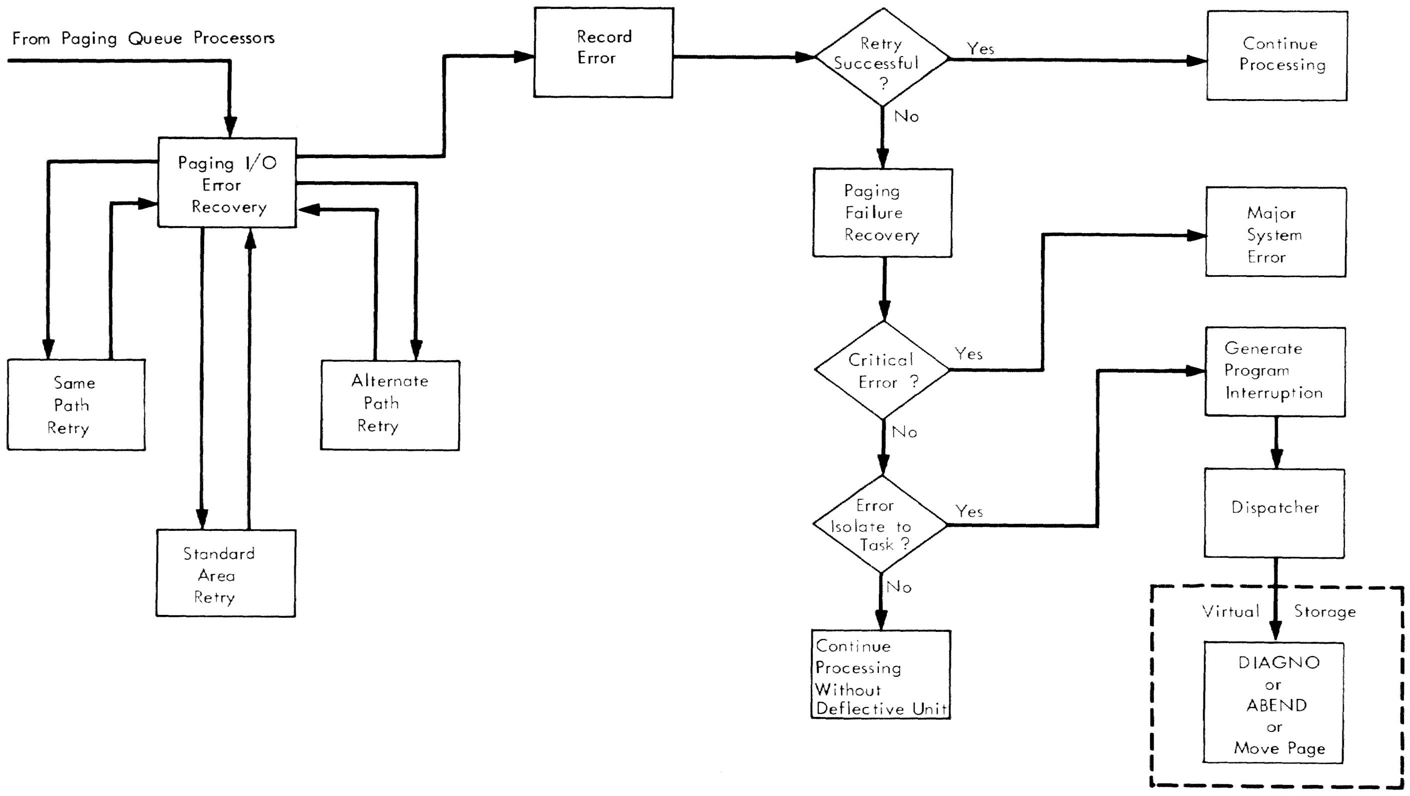


Figure 89. Flow of Control During Paging Error Handling

A delayed paging failure is detected by checking the CSW stored upon an I/O interruption. A delayed paging failure may be either an inboard or an outboard failure. Paging I/O Error Recovery Control will call the Same Path Retry routine to perform error recovery procedures. These procedures involve retrying the paging operation along the same path a (device-dependent) number of times.

For some errors, including channel-check conditions, if retry is unsuccessful, and an alternate path is available, retries are made along the alternate path until the threshold value is reached. If the error cannot be attributed to a defective device or recording medium, the failing path is marked defective.

The Paging Failure Recovery module is invoked whenever retry is unsuccessful.

When it is not clear whether the cause of the paging failure is due to a defective device, a defective volume, or a bad page, the Standard Area Retry routine is invoked.

The Standard Area Retry routine retries the paging operation along the same path on the standard area of the recording medium of the device. If this procedure is successful, a defective page is implied.

If the standard area retry is not successful, the cause of the paging failure can be distinguished as a defective device.

If the cause of the paging failure is determined to be a defective channel or control unit, the system will attempt to continue without the path or paths that have been set unavailable.

If no good path to the device can be obtained, the Paging Failure Recovery module will perform processing similar to that for a defective device and a defective page.

If the cause of the paging failure is determined to be a defective device, the device is set unavailable and processing similar to that for a defective page is performed. If the defective device is the primary paging device, Paging Failure Recovery issues a major system error SVC.

If the cause of the paging failure is a defective page, the operation could have been either a page-in or page-out.

If the operation was a page-out to a system or auxiliary storage device, another page will be allocated and the paging operation retried. If necessary, all further auxiliary storage allocation on the device will be suppressed.

If the paging operation was either a page-in or page-out associated with a Virtual Access Method PGOUT request, a return code is stored in the appropriate task's XTSI. When the Dispatcher eventually transfers control back to the VAM MOVEPAGE routine, the return code will be passed in a register.

For a page-out operation, MOVEPAGE will select a new page and retry the operation (see "Example of Virtual Sequential Processing"). For a page-in, MOVEPAGE will invoke ABEND to abnormally terminate the task.

If the operation was a page-in operation, it may be associated with a relocation interruption, an IOCAL SVC, a TWAIT operation or a Dispatcher or Page Posting read operation.

For TWAIT, the page is left dormant until it is required, since it is entirely possible that the task can continue processing without reading the page.

Should the page-in be as a result of a relocation interruption, a task program interruption is generally generated. It will be processed by DIAGNO as a program error, unless the error affects a privileged routine. If the error effects a privileged routine, the task is abnormally terminated. However, if the page-in error is for a shared page, Paging Failure Recovery will issue a major system error SVC.

If the operation brings in a buffer page required as the result of an IOCAL, the instruction counter will have been backed up so that the IOCAL will again be initiated and further processing as specified above for a relocation interrupt follows.

The last and most catastrophic case is when the paging was initiated by the Dis patcher or Page Posting routine. In either case, the task generally cannot accept a programmed interruption. The only choice is to deactivate the task, mask off task interrupts and inform the operator. He caninitiate proper action (perhaps including cancellation of the task). If it is the operator task which is so affected, a major system error is issued.

When a return code or program interruption code is passed to virtual storage, the code specifies whether the error is associated with a defective device, volume or page. If the error is associated with a volume, the code indicates whether or not the volume is demountable.

Whenever a solid I/O error occurs or a defective channel, control unit, or device is involved, the Resident Supervisor Task Communication Control subroutine is invoked to enqueue a Message Control Block (MCB) on the Main Operator Task TSI. This will result in a message to the operator's console informing him of the situation. If the cause of the error was a defective demountable volume, the operator may be able to take corrective action.

TASK I/O ERRORS

Task I/O errors, like paging errors, can be either immediate or delayed failures. The processing of task I/O errors within the Resident Supervisor is supervised by the Device Queue Processor and the Channel Interrupt Processor. An overview of task I/O error handling is presented in Figure 90.

For an immediate failure, the Device Queue processor will retry the Start I/O operation a specified number of times. This failing, the processor will set a flag in the GQE and initiate a sense operation. The Channel Interrupt processor will receive an interruption when this operation is completed. After inspecting the flag set by the Device Queue processor, the Channel Interrupt processor invokes the Dequeue I/O Requests (DQIOR) routine to dequeue and release all other requests from this task for this particular device. An interlock will be set to prevent any new requests from being accepted from this task, the path will be released, and all the infomation the access method needs will be placed in the IORCB for this operation. The IORCB will be passed back to the task and the appropriate access method will attempt error recovery.

The processing for delayed failures is essentially the same as that for immediate failures except that the Channel Interrupt Processor first becomes aware of the error condition. The Channel Interrupt Processor will ask to have a sense operation initiated by setting a flag in the device GQE (not the interrupt GQE). When the Device Queue Processor is next invoked by

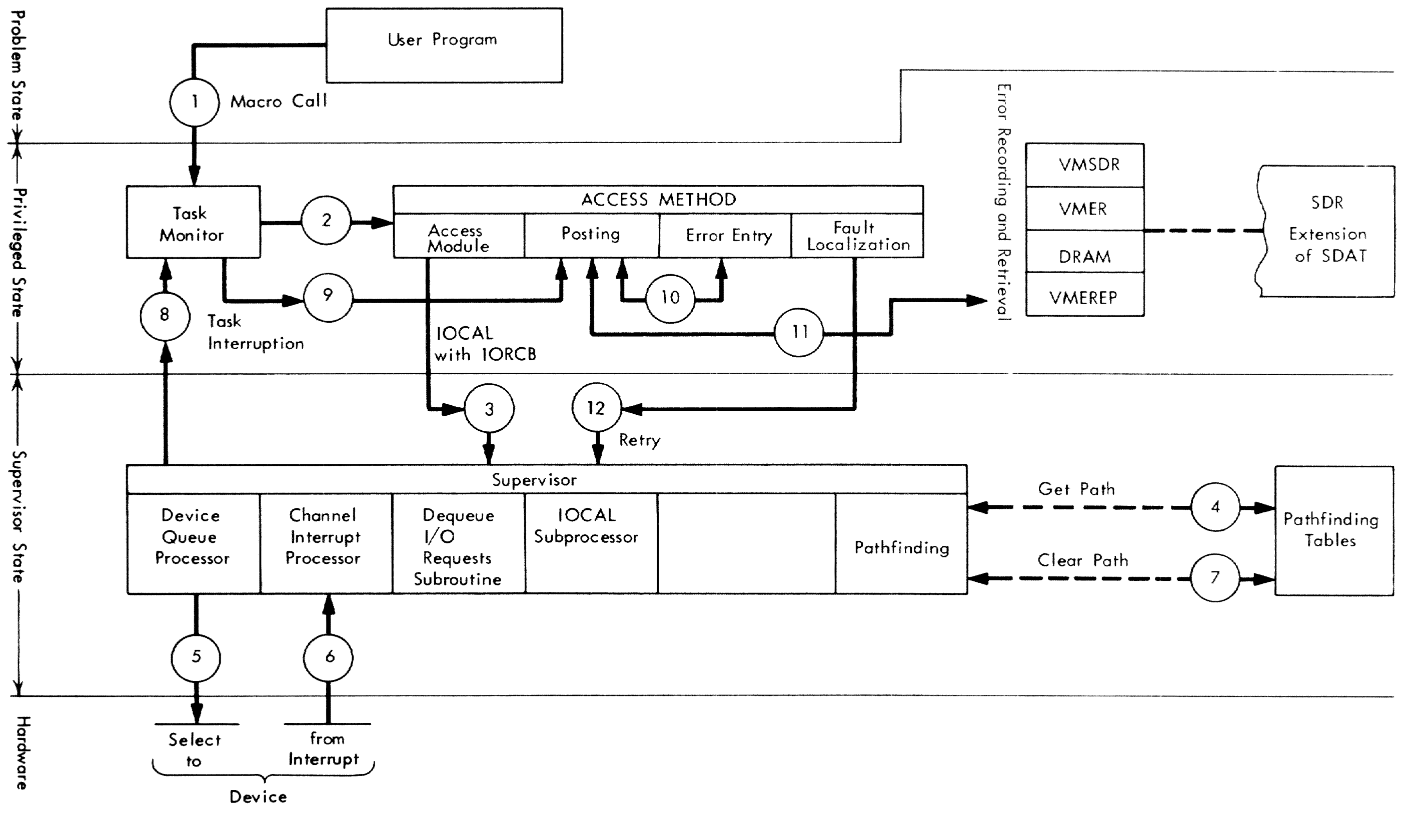


Figure 90. Task I/O Error Handling

the Queue Scanner, it will initiate a sense operation and further processing will proceed as in the case of an immediate error.

The access methods each differ in their recovery procedures, but the procedures will generally involve several retries. Depending upon the type of error, there may be no retry (e.g., equipment check from a tape unit); there may be retries only along the original path to the device (e.g., seek check on a direct access operation); or there may follow retries along the original and alternate paths to the device (e.g., channel data check on a direct access operation). Further reads or writes issued by the user will be queued on the Data Extent Block until the error is resolved. (This is true for all access methods except IOREQ, and OLTAM).

An IOCAL supervisor call is most frequently associated with MSAM, BSAM, and TAM services. When a synchronous I/O interruption is passed to the Task Monitor, the appropriate posting routine is immediately dispatched by the Task Monitor. The IORCB associated with the operation is moved from the ISA and examined. Successful operations are posted as complete in the Data Event Control Block (DECB). If an error can not be retried at all or if it is unsuccessfully retried, the user's Synchronous Error routine (i.e., SYNAD) will be entered when the CHECK macro is executed, or if appropriate, the task will be abnormally terminated. This procedure is based on the assumption that SAM and IOREQ are never used for system functions. MSAM has its own posting routine but performs functions analogous to SAM.

Synchronous interruptions, resulting from SAM or OBTAIN/RETAIN operations on a direct access device, are processed by the Direct Access Error Retry routine. This routine makes appropriate modifications in channel programs and rebuilds the IORCB so that the I/O request can be issued (via IOCAL).

TAM posting processes all I/O interruptions resulting from the termination of a TAM generated channel program. TAM Posting consists of two main sections: a common section, which processes the CCW list and posts data, and an exception and error section, which provides error analysis and recovery actions. The failure of an IOCAL operation issued by TAM, which indicates the unavailability of the SYSIN/SYSOUT device, is treated as an ABEND.

Input errors occurring on VAM data sets result in an entry to the Virtual Memory Input Error Recovery (VMIER) routine via the Task Monitor. VMIER marks the appropriate page in error on the volume and either abnormally terminates the task or performs recovery. Recovery is only possible if the data set is duplexed. Duplexing is an option, available to users, which maintains two identical copies of the data set. When an error occurs on the primary page of this type of data set, the secondary or backup copy can be used to recover.

The error page in the primary copy is marked in error on the volume as in other cases. Then the secondary copy is read and a replacement page is assigned to the primary copy of the data set. A relocation entry is made in the PAT to enable the access method to locate the replacement page and the secondary copy is written to the new primary page. At this point both copies are correct and the user is free to continue.

Recovery is also attempted when processing error conditions resulting from VAM data set manipulation. This type of error may not be serious enough to warrant the destruction of the user's task. The VAM Data Management Error Processor (VDMEP) routine, if possible to do so, closes out the data set in question, generates appropriate diagnostics, and returns control to the user. The user may then attempt error recovery using available system facilities.

SYSTEM ERROR PROCESSOR (SYSERR)

The System Error Processor (SYSERR) can be invoked by either Resident Supervisor routines or privileged virtual storage service routines. The purpose of SYSERR is to provide a system dump facility after the detection of a system error, and to initiate a recovery procedure appropriate to the error classification. In calling SYSERR, the following parameters are specified:

- classification of the error
- dumping option
- identification of the calling module
- an error code to identify the error condition

For minor errors that may be pinpointed to a task, SYSERR returns control to the calling program. If the calling program is a privileged virtual storage service routine, it can then invoke the ABEND routine to dispose of the task in a manner consistent with a completion code which is specified as a parameter to ABEND.

For major errors (such as a program interruption occuring while a CPU was operating in the Resident Supervisor), the Recovery Nucleus and the Reconfiguration routine is invoked. A message is sent to the system operator and Startup Prelude is invoked to restart the system.

For a hardware suspected system error, the Recovery Nucleus and SERR are invoked. SERR will check out the system's CPUs and storage elements. If a CPU or storage element error is discovered, the unit is set logically unavailable and the system is restarted. If no error is found, the system is restarted.

ABNORMAL TASK TERMINATION (ABEND)

The ABEND routine is invoked for task error situations such as invalid submission of parameters or commands. ABEND consists of several privileged service routines. These routines are a part of initial virtual storage and operate as privileged routines. Service routines calling ABEND may use completion codes 1, 2, or 3.

The completion code values may be generally characterized as follows:

Completion code 1
: the logical task sequence is terminated at the point the ABEND condition occurs and control is returned to the user. Completion code 1 is declared for those conditions caused by the user programs on the premise that he should either remove the cause of the error and/or direct the task termination.

Completion Code 2
: the logical task sequence is terminated at the point the ABEND condition occurs and the system attempts to terminate the task in an orderly fashion. Recurrent ABEND conditions cause the system to abandon the associated phase of the termination (e.g., the closing of a data set) and go onto the next phase. Completion code 2 is declared either for localized conditions caused by a system error or recurring completion code 1 conditions.

Completion Code 3
: processed the same as completion code 2 except no new task is logged on.

The actions associated with these completion codes are as follows:

Completion Code 1:

Conversational Task

- Task I/O is terminated
- System interlocks set by the task being abnormally terminated are released. (Each component which may have system interlocks set at the time ABEND is called must provide a routine to reset the interlocks. It may also be necessary to initialize data fields associated with the interlocks).
- A message indicating abnormal termination is issued to the user and an EXIT is used to return control to the user program.

Nonconversational Task:

- The procedures are the same as for the conversational task up to the EXIT point. At this point ABEND locks for a new SYSIN data set with ddname SYSABEND. Subsequent commands are then taken from SYSABEND if it exists. If it does not exist, then LOGOFF is called to terminate the task.

Completion Code 2: (conversational or nonconversational)

- The procedures are the same as for completion code 1 up to the point where control is turned over to the user program (conversational) or SYSABEND (nonconversational).
- Close data sets.
- Erase uncataloged data sets and release private devices.
- Unlink shared modules.
- Call LOGOFF.
- Logon a new task for the user.

The Command System represents the principle interface between a time-sharing system and its users. Because of this, the Command System achieves a position of special importance in any attempt to make the facilities of a general-purpose operating system like TSS/360 easy to use.

In designing a command system, there are two fundamentally different approaches that can be taken. The first is to use a small basic vocabulary, which tends to produce long expressions. The second is to enlarge the basic vocabulary to obtain shorter expressions. The latter approach tends to increase the number of commands that must be learned and remembered in order to use the system but simplifies its use as each command is less complex.

In TSS/360, the approach taken is that of constructing a Command System which is delivered with a large set of simple commands that can either be employed as is or completely replaced or expanded in a straightforward fashion. Such an approach allows each installation and each user the flexibility needed to customize the system-user interface. The key external characteristics of this Command System are:

- Regularized syntax
- Default-override facility
- System-supplied "null" commands
- Personalized user profile
- Macro-command definitional capability
- Facility to easily define new command primitives
- Dynamic facility to tailor system-supplied messages
- Flexible attention-handling and command push-down procedure

The TSS/360 Command System syntax is simple and natural. Each individual command consists of an operation name followed by operands; a command statement comprises one or more commands. The delimiting character for the operation name is a blank; the delimiter between operands is a comma; the delimiter between commands in a statement is either a semicolon or the end of a line of input; and the line continuation flag is a hyphen entered as the last text character of a line.

The operation name can be either the name of a command processing routine or the name of a stored command procedure. If the operation is the name of a routine, the Command System basically processes the input string into a parameter list of operands for the invoked routine. If the operation is the name of a stored procedure, the command system processes the input string in conjunction with the stored procedure to produce new command statements for processing.

Each user can establish his own spellings, abbreviations, or operation names for commands through a SYNONYM facility. Use of this facility sets up one or more equivalances for the original name but does not destroy it. Interestingly enough, the inclusion of the SYNONYM facility has simplified implementation of the Command System because it is only necessary for a command routine to recognize a single name -- the string substitution is done before the command routine receives control.

Any command operand is represented in two ways: by position <u>and</u> by keyword. Keywords may appear in any order and have the general form: KEYWORD = value, where KEYWORD is the name of the operand and "value" is the actual value of the operand.

In contrast to many other systems, almost every command operand has a default value. Furthermore, in TSS/360, the user does not have to accept rigid default values for operands, for he can override the system-supplied default values if he so wishes. Every user has the ability to establish the values to be used when he chooses not to enter those values explicitly. The SYNONYM facility, available for command operation names, is also available for keywords. Thus, the use of keywords achieves more flexibility than the use of reserved words as proposed in other procedural syntaxes. In TSS/360, a keyword may be thought of as a global, reserved word. The principle reason for this design is that a keyword is associated with a value to be passed, not with the command invoked.

The sequence of events that the system goes through to resolve operands is as follows:

1. The list of synonym values is searched for equivalent terms.
2. If synonym values exist, they are inserted; if not -

3. The list of user-specified default values is searched.

4. If default values exist, they are inserted; if not -

5. User-supplied default values are inserted whenever they exist. If any are not supplied

6. The operand is not filled in; it is given a null string.

Tailoring the Command System

The capabilities available to a user or his installation for tailoring the Command System will be partially covered in the following paragraphs and will be more fully described later in this chapter.

TSS/360 maintains a special prototype data set that is copied into the User Library for an individual when he is initially joined to the system. This data set, called a User Profile, contains three tables. The first specifies the initial default values for command operands. The second, called the Character and Switch Table, contains the defined character translation list (to allow redefinition of printing characters, and control characters, such as the New Line control character). The third, called the Primary Dictionary, contains command operation names and equivalances.

Whenever a user initiates a terminal session or a nonconversational task, a copy of the User Profile data set from his user library is placed into the Combined Dictionary used by the Command System for processing his system input stream. During a session, an individual can change and delete entries in the Combined Dictionary. These changes exist only for the duration of the terminal session, unless the user wishes to make them a part of his permanent User Profile.

The power of the Command System is further enhanced by a command procedure facility. This permits the user to create and store a procedure, named by use of the PROCDEF command, which consists of commands and of logical (e.g., IF) statements which control the flow of command execution. The PROCDEF command invokes the Text Editor, enabling the user to utilize any of the text editing commands while he is defining a procedure. After a procedure has been written, it may be edited or shared, etc., as with any other data set.

The invocation of a command procedure is identical to the means of invoking a system-supplied command. The command statement consists of the procedure name followed by a series of parameters. The parameter values are inserted at the proper points in the procedure. The Command System then interprets the resultant statements as though they had originated in the input stream. For maximum power, TSS/360 allows procedures to be nested and/or recursive.

The BUILTIN command and an assembly language macro instruction (BPKD) make the full power of the Command System available to any user-written routine. They specify to the Command System the positional and keyword parameters that are to be associated with input parameters. Performance is enhanced using this BUILTIN facility as the Command System can directly bypass much of the interpretive processing required in the exfpansion for normal command procedures.

A user may augment the system message-handling facility by use of the Command System in several ways:

- He can request explanations of system messages or key words within a system message. Word explanations may continue to a large number of levels.
- He can dynamically specify the particular classification of messages he wishes to receive. This filtering or masking capability recognizes five levels of severity of messages and three different lengths of messages.
- He can use the facilities of the Text Editor to construct a personalized file of messages that will be issued in lieu of any system messages with the same message identification codes.

There is a flexible system for handling attention interruptions. For instance, suppose a user has forgotten to identify a library containing a particular subroutine required by his main line program. When the user receives the system diagnostic message at his terminal, he can use the attention button to re-enter the command mode, define the library, and then continue processing from the point at which the diagnostic message was issued.

The routines that constitute the Command System are:

- Command Controller
- Program Control System
- Interruption Processors
- Command Routines
- Batch Monitor
- System Operator and Administrator Routines
- Accounting Services
- Language Processor Control

Figure 91 shows in tabular form the major components of the Command System and some of the Command System routines. The Command System is structured so that the user can initiate commands either conversationally or nonconversationally. A user may also initiate a number of commands through macro instructions contained in an object program. The expansion of these macro instructions results in object code that will link to the proper Command System routine at execution time.

COMMAND CONTROLLER

The Command Controller serves as a central link between the user and the system. It provides an interface between the remainder of the Command System and the SYSIN and SYSOUT data streams. The Command Controller consists of the Command Analyzer and Executor (CA&E) and the following system support routines:

• Source List Handler	Used to service the source list.
• Dictionary Handler	Used to access the combined dictionary.
• GATE	Used to access SYSIN and SYSOUT for both conversational and nonconversational tasks.
• SCAN	Used by command routines to fetch and validate command parameters.
• User Prompter	Used to communicate with external users.
• Virtual Memory Task Initialization (VMTI)	Used to perform functions required to initiate a task.
• Attention Handler	Used to accept and to interpret attention interruptions from the Task Monitor.

The Command Analyzer and Executor obtains the next command to be processed, and uses the verb scanner to determine the verb type from the control dictionaries located in the combined dictionary. The verb type dictates the remaining scan to be used for each verb. When each command has been completed or terminated, control returns to the Command Analyzer and Executor. The support routines are employed as needed.

The usual flow of control within a task in the command mode is from the Task Monitor Scanner-Dispatcher to the Command Analyzer and Executor (CA&E), which calls the Source List Handler to check the source list for a command. The Source List Handler calls GATE to obtain a command from the user's SYSIN by issuing an underscore and backspace.

Once the command has been obtained from SYSIN, GATE puts it in the source list via the Source List Handler, and control is returned through the Source List Handler to CA&E. The Command Analyzer and Executor then isolates the command verb by calling the Verb Scanner and gives control to the appropriate command routine. Before processing the command, each command routine checks the authority of the user to issue the command.

The command routine uses the SCAN routines to validate command parameters as

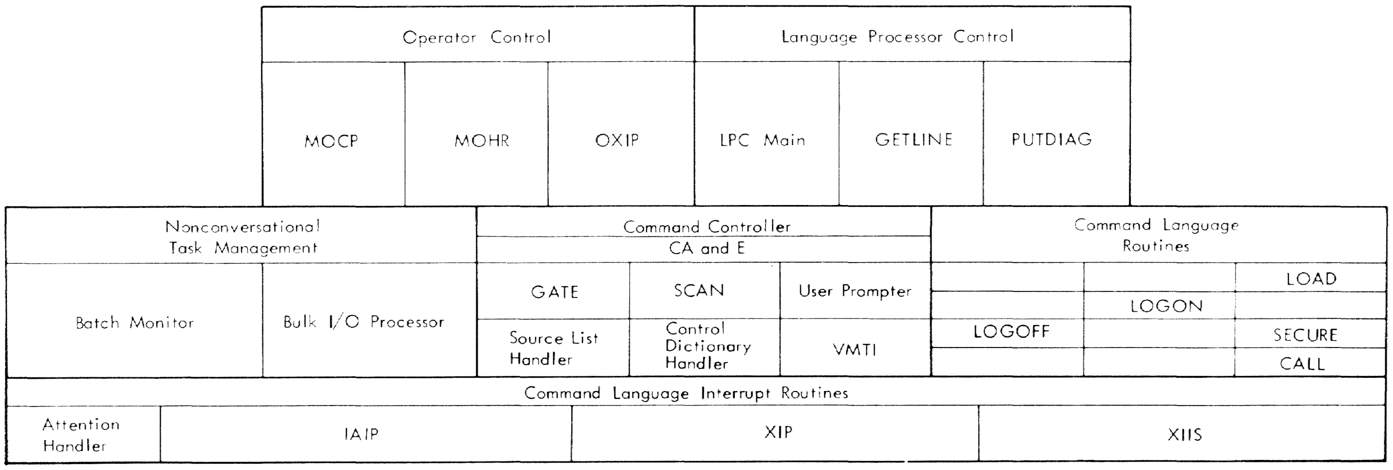


Figure 91. Command System

well as to fetch those parameters. The command routine then performs the requested actions which may necessitate using other Command System and system service routines. The command routine then returns control back to the Command Analyzer and the Task Monitor.

If errors are discovered during the processing or if the user is to be prompted for parameters, the User Prompter routine will be used to obtain a message from the System Message data set and to invoke GATE to issue the message to the user's SYSOUT. In conversational mode, the user may make corrections when responding to system messages or may reenter the command. Since the user is not present for a nonconversational task, any message that requires a user response results in termination of the nonconversational task.

The PUNCH, RT, WT, and PRINT commands are an important exception to the usual pattern of performing command functions within the invoking task. When the Command Analyzer and Executor encounters one of these commands, it calls the Bulk I/O preprocessor command routine. The Preprocessor then validates the command operands and, if they are acceptable, sends a message requesting task initiation to the Batch Monitor. Acting upon this request, the Batch Monitor will enter this request into a data set called the Batch Work Queue. If the data set involved resides on a private volume and system resources permit, the Batch Monitor initiates the PUNCH, PRINT, WT, or RT command routine as a new nonconversational task, independent from the task that originally issued the command. If the data set involves a public device, the request is assigned to the BULKIO II task (task 002). In this way PUNCH, RT, WT, and PRINT functions -- which depend upon the availability of input/output devices -- can be performed at the system's convenience, and the user's task does not have to wait.

An overview of the Command Controller is presented in Figure 92.

Source List Handler

As commands are entered into the system, they are recorded in the source list. The Source List Handler processes all entries and deletions to the source list, which operates through a pushdown, popup structure, containing a sequenced list of events which are to happen in the future. The user may change the sequence of events in the source list by invoking an OBEY macro instruction or an expanded procedure.

Whenever a procedural verb is encountered, the commands to be inserted in its place are placed in a sublist, and a pointer to the sublist is placed in the original

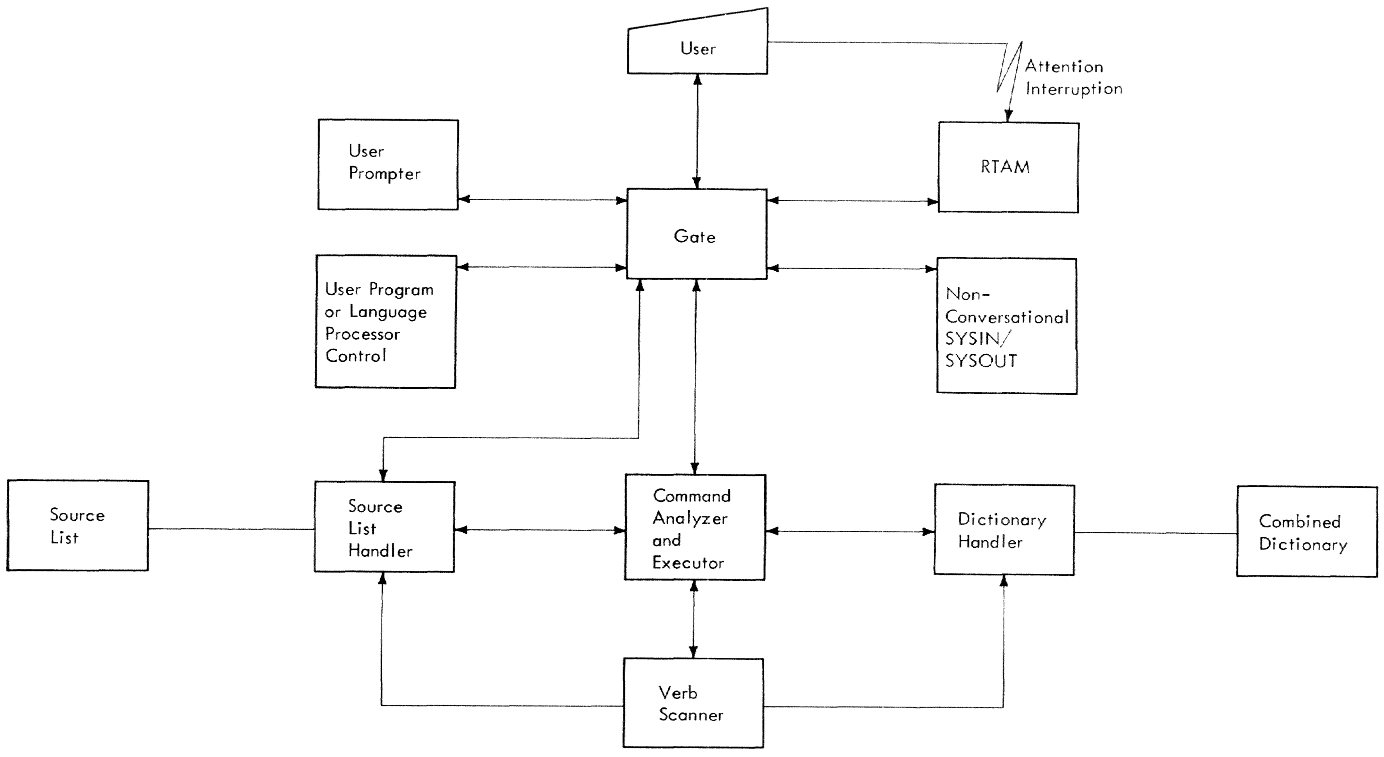


Figure 92. An Overview of the Command Controller

procedural verb position. These sublists can be generated almost indefinitely.

The Command Analyzer and Executor always checks the source list for the next command. When the source list is empty, the Source List Handler calls GATE to prompt the user for additional input.

Dictionary Handler

The Dictionary Handler contains routines for processing entries for the combined dictionary, which is the source of the names of all procedures, synonyms, and defaults. The combined dictionary contains the following control dictionaries: synonym dictionary, default dictionary, builtin procedure dictionary, textual procedure dictionary, and command module dictionary.

The system library provides a source of information for the combined dictionary. A dictionary of system procedures and system builtin procedures is established at the same time the system library is established, and it becomes part of the combined dictionary every time LOGON occurs. If the user adds any dictionary entries in the form of procedures or builtins to his user library, these entries will become part of the combined dictionary the next time this user logs on. The dictionary search logic allows a search of either the system library version or the user library version; the user may designate the user library in preference to the system library.

If the user has created his own profile, his user library will contain a copy of the system prototype file (SYSPRX). This data set, along with the synonyms, defaults, and command variables are added to the combined dictionary. If the user does not have a profile, his user library will contain no SYSPRX member.

GATE Routine

The GATE routine is used to access SYSIN and SYSOUT for both conversational and nonconversational tasks. For a conversational task, SYSIN and SYSOUT are the same terminal device. For a nonconversational task, each is a Virtual Access Method format data set.

GATE is invoked by Command System routines, by the Language Processor Control (LPC) routines, and by User Prompter and is invoked indirectly by user programs to access the task's SYSIN/SYSOUT. GATE is normally invoked through one of the GATE macro instructions.

The GATE macro instructions are:

- GATE Read (GATRD)
- GATE Write (GATWR)
- GATE Write with carriage control (GTWRC)
- GATE Write with available response (GTWAR)
- GATE Write with spontaneous response (GTWSR)

For privileged and nonprivileged routines, GATE receives control at its proper entry point in the GATE supervisor, which analyzes the parameter list. If a valid call requires output, GATE formats the output, if required, and writes each line segment if SYSOUT is a VAM data set. If a valid call requires only reading, the Terminal Communications Subprocessor (TCS) is invoked.

The Format and VAM Output Routine sequence replaces all internal line divide (breakpoint) characters with space characters. For conversational tasks it replaces terminal breakpoint characters with new line characters; for nonconversational tasks, it discards terminal breakpoint characters. If the requested output message is larger than the maximum SYSOUT line for the task, the message is broken into device-sized lines or into line segments, and, for nonconversational tasks, VAM is used to write each line segment. For conversational tasks, GATE issues the ATCS SVC to accomplish read/write operations.

If an attention interruption is pending, the Conversational Write and Read sequence first determines if the caller's program is privileged or nonprivileged. For a privileged call, the user's input length is set to zero for GATRD, GTWAR, or GTWSR calls, the attention return code is set, and return is made to the caller's program.

If an attention interruption is not pending, the SYSIN device is set to the keyword for a GTWSR call if the value of the RSVP parameter in the Profile is not set. Otherwise the SYSIN device is set according to the keyboard/card reader switch of the Profile. TCS then builds and dispatches the proper CCWs.

Finally, GATE processes the end conditions accompanying completion of the transmission.

Scan Routine

SCAN consists of a set of subroutines used by other command routines to isolate and validate input parameters.

The six major subroutines of the set are

Routine Name	Function
NEXTPAR	To locate the next parameter field and inspect it for invalid characters.
CHEKDS	To check a data set name for valid form and characters.
ALFNUM	To check for a valid symbol.
NUMSTG	To check for a string of valid numeric characters.
CHKNUM	To check a string of numeric characters and convert them to a binary value.
ALFBET	To check for a string of valid alphabetic characters.

User Prompter Routine

The User Prompter is a centralized message locator, display, explanation, and response handling facility. It uses a user-defined message file and/or the system supplied message file to perform its functions.

Upon receiving a PRMPT macro call, the User Prompter locates the message in the message file, and inserts parameters as required, after checking with the filter in the User Profile. If a response is required the prompter displays the message and reads the response via GTWSR. This initial response is checked to see if further explanation is required. If so, the type of explanation is determined, and the explanatory message required is retrieved, using GTWSR until the user is satisfied. If no explanation is required, the response is checked against valid responses, or it is stored if the unpredictable response option has been used.

Command Analyzer and Executor

Command analysis and execution is generally done in the following manner: the verb is recognized, the verb operands are translated, the verb is executed. The Command Analyzer and Executor (CA&E) is invoked upon initial task entry. In addition, it may be invoked recursively by a user, upon intervention of an object program, or during control of language processors. Verb resolution is performed by the Command Controller after calling the Verb Scanner.

A user or system procedure verb calls the Procedure Parameter List Analyzer and the Procedure Expander. The output from the procedure expansion is added to the source lists.

An IF verb invokes the PCS Phase 1 IF routine to expand the condition, and calls the Object Executor to determine the value of the IF condition. If the condition was not true, a special scanner is invoked to find the end of the sentence. Otherwise, the next clause is begun by CA&E. An AT verb invokes the AT Analyzer and enables the dynamic statement switch so that the remaining statements in the line are analyzed but not executed until the AT is satisfied. Statements which are not executable dynamically are recognized and appropriate diagnostics are produced.

If the verb is DISPLAY, DUMP, SET, or a program call, the Program Parameter List Analyzer (PCS FORMLIST and PCS Phase 2 parameter processing) is called to analyze the parameters; the commands are then invoked by the appropriate routine (DISPLAY/DUMP, SET or LINK). A DEFAULT or SYN verb calls the DEFAULT/SYN processor to make the appropriate changes to the combined dictionary.

If the verb is GO, then the last interrupted program will be resumed. Any commands still not executed in the source list at the time of intervention become eligible for execution upon completion of the intervening routine. Any commands in the source list following GO will be discarded.

When a program issues an ENTER LPCINIT, the Edit Conroller is initialized, and the calling program is treated as a Language Processor Controller. Subsequent ENTER LPCEDIT statements invoke the Edit Controller, which may in turn transfer control to the Command Analyzer and Executor. Any edit commands recognized by CA&E cause it to recall the Edit Controller. The Edit Controller in turn calls the appropriate edit routines, and exits to the Language Processor Controller upon completion.

An overview of the Command Analyzer and Executor is presented in Figure 93.

Virtual Memory Task Initiation Routine (VMTI)

VMTI is a Command System service routine that performs many of the first initialization functions required to prepare a task for LOGON processing.

For conversational tasks, VMTI is entered from IAIP when the initial attention interruption is received by the task (see "Creation of a Conversational Task").

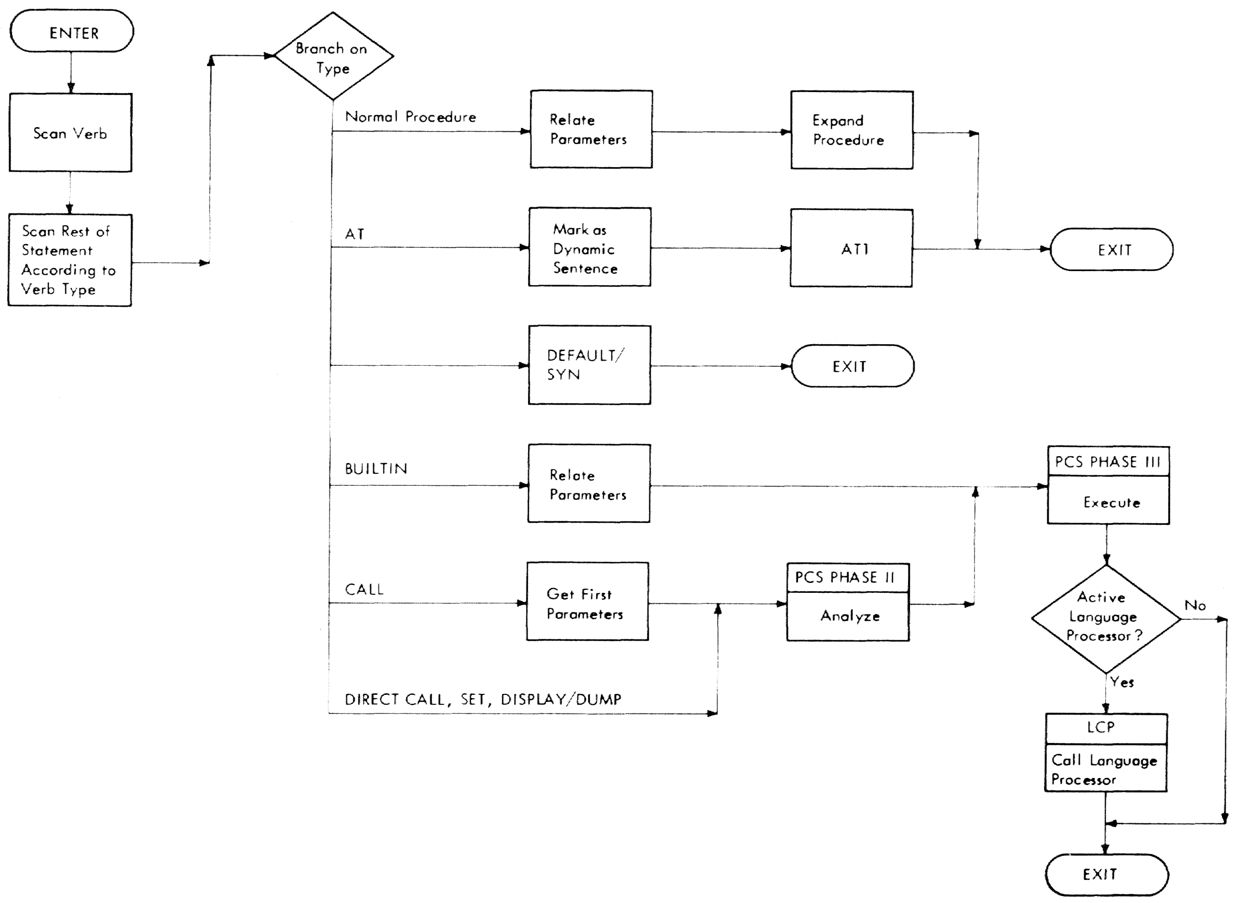


Figure 93. Command Analyzer and Executor -- Operational Flow

For nonconversational tasks, VMTI is invoked as a result of an external task interruption (see "Nonconversational Processing").

This external task interruption is associated with a task initialization message sent via the VSEND supervisor call (see "Communication"). This message originated with either the Main Operator Task or the Batch Monitor task.

The (conversational) Main Operator task is created by Startup. The Main Operator Task, in turn, issues a Create TSI supervisor call to create the BULKIO task and sends an initialization message to the Batch Monitor task.

In the course of its processing, BULKIO issues the Create TSI supervisor call and task initialization messages for Bulk I/O and other nonconversational tasks.

In response to an external task interruption, the Task Monitor External Interrupt processor calls the Command System External Interrupt processor (XIP).

XIP inspects the Task Identification code (TID) and message code, and, from this, determines to invoke VMTI.

VMTI performs the following functions that are required to initiate a task:

- Adds to the Task Symbolic Device List all those devices which contain public volumes (see "Device Allocation").
- Issues an OPEN macro to open the catalog in UPDAT mode to allow the updating of the data set from virtual storage and to prevent two tasks from accessing the same user catalog simultaneously. The user's catalog is obtained by means of the VPAM FIND routine which is supplied with the user's identification as a parameter. The user's identification is always the first component of a fully qualified name (see "Catalog").
- Makes successive calls to DDEF to create and fill in the JFCBs for the System Message Data Set, the System Library, the User Table, the System Accounting Table, the system macro library, and the macro library index.

INTERRUPTION PROCESSING

Interruptions that the Task Monitor (or one of its routines) determines must be brought to the attention of a task are generally processed by the following Command System routines:

- Attention Handler Command routine.
- Initial Attention Interrupt Processor (IAIP).
- External Interrupt Processor (XIP/XIIS).
- Program Interrupt Processor (DIAGNO).

ATTENTION HANDLER

Normal asynchronous interrupts are handled by the Attention Handler command routine. This routine allows user routines, specified in an AETD macro instruction, to process attentions; or it obtains a command from the terminal and processes the command so that the Command Analyzer and Executor will execute it; or it obtains and acts upon a terminal direction to ignore the attention, or to invoke the ABEND procedure.

For actual attentions, this routine first determines if the user wishes the attention suppressed. If so, an attention SVC is armed. When executed it will provide a simulated attention. For either real or simulated attentions, the Attention Handler determines whether the current attention count has a corresponding, valid entry in a connected Attention Entry Table (AET). If it does, the user's attention routine is executed, with help from the User Controller. If no routine is indicated in a connected AET for the attention, the Attention Handler tests the attention count to direct its continued action.

The attention count is decremented on entry to the Attention Handler for a real attention. If it tests out other than zero at this point, the Attention Handler returns to the Task Monitor, from which it will be redispatched until the count is zero.

When the count is zero, the Attention Handler uses GTWSR to print the attention character on the terminal. Further action by the Attention Handler depends upon the response detected. Another attention causes immediate exit to the Task Monitor. A continuation response causes the Attention Handler to issue a GATRD until it detects another response. If input is null, the Attention Handler also exits. The other possible responses involve the "attention response" commands -- STRING, RTRN, EXIT, STACK, and PUSH. If the response is STRING, the Attention Handler displays the source list. If it is one of the others, the Attention Handler uses User Control to take the appropriate action. (For a complete description of the attention response commands, see the Command System User's Guide, GC28-2001.)

INITIAL ATTENTION INTERRUPT PROCESSOR

The Initial Attention Interrupt Processor (IAIP) performs certain initialization functions for conversational tasks.

Initially IAIP uses the Find JFCB routine to locate the skeletal SYSIN/SYSOUT Job File Control Blocks that are automatically included in the skeletal Task Data Definition Table (TDT) created at Startup and included in each task's Initial Virtual Memory. IAIP places a pointer to the SDAT entry for this terminal in each JFCB, thus completing the JFCB.

Then IAIP sets a flag in Task Common to indicate that the new task is conversational. Lastly, IAIP calls VMTI which provides further initialization functions for the task.

EXTERNAL INTERRUPT PROCESSOR

The External Interrupt Processor consists of two principal routines: the External Interrupt Processor (XIP) proper, and the External Interrupt Initialization Subprocessor (XIIS).

XIP receives intertask messages, checks their validity, copies them from the ISA, and enqueues a linkage request to the appropriate Command System routine or the appropriate XIP subprocessor. If the message is an awaited reply, XIP moves the message to the designated reply area, and returns control to the Task Monitor. If the message is not a reply, XIP tests for a code indicating that the Device Management RELEAS routine has sent a message both to take the task out of delay status and to inform the task that a device the task needs has just been released (see "Device Allocation"). This special function has been placed in XIP so that Device Management does not have to maintain the coding to handle this sort of message. If the message is from RELEAS, control is returned to the Task Monitor Scanner-Dispatcher. Eventually, the Device Management MOUNT REQUEST routine will be dispatched.

For all other messages, XIP calls GETMAIN to allocate storage into which XIP can move the message control block from the ISA. The code in the message is used as an index to locate an entry in a table which describes the routines to be invoked for those message codes that are predefined within TSS/360.

There are 256 possible message codes. Message codes with a value less than 128 are processed by XIP. Of these, approximately 26 message codes are defined in this table. The message codes from 128 to 255 are available for dynamic definition by TSS/360 users.

Among the messages processed by XIP are the following: messages from the Main Operator's task specifying that the system is being shutdown, messages to the Main Operator's task, and messages to invoke initialization of nonconversational tasks.

If the appropriate routine to handle the message is already loaded, XIP invokes the Queue Linkage Entry (QLE) routine to enqueue a linkage request for the routine. XIP then exits to the Scanner-Dispatcher.

If the appropriate routine has not been loaded, XIP uses the QLE routine to enqueue a linkage request to a second XIP entry point. At this entry point, the required routine will be loaded. XIP uses this method of linking to its second entry point because XIP does not know when, if ever, control will return. By enqueuing a linkage request and returning to the Task Montor Scanner-Dispatcher, orderly save area management is preserved.

A message to invoke nonconversational task initialization results in the eventual invocation of the External Interrupt Initialization Subprocessor (XIIS).

XIIS performs initialization functions for nonconversational, Batch Monitor, and Bulk I/O tasks. For the Batch Monitor or a Bulk I/O task, XIIS loads the specified routine and places the Batch Sequence Number into Task Common. A linkage entry to the routine just loaded is enqueued and FREEMAIN is called to release the storage occupied by the message. Control is then returned to the Task Monitor.

For normal nonconversational tasks, XIIS creates a nonconversational SYSIN JFCB containing the SYSIN data set name from the task initialization Message Control Block.

Included in the Task Data Definition Table assigned to a task as part of its Initial Virtual Memory are prebuilt Job File Control Blocks (JFCB) for a conversational SYSIN and SYSOUT. Because this is a nonconversational task, XIIS uses the RELEASE macro to release the conversational SYSIN JFCB and then uses the DDEF macro to create a nonconversational SYSIN JFCB. GATE performs a similar function for the nonconversational task's SYSOUT.

XIIS then sets a flag in Task Common to indicate that the task is nonconversational. The User ID is also placed in Task Common.

Finally, XIIS calls the Attention Handler for further task initialization. XIIS then exits to the Scanner-Dispatcher.

PROGRAM INTERRUPT PROCESSOR

The Program Interrupt Processor (DIAGNO) informs the conversational user that a program interruption has occurred within his task. A message is sent to SYSOUT containing the interrupted instruction, its virtual storage location, and a specific description of the cause of the interruption.

Entry to this processor is made from the Task Monitor SVC or Program Interrupt Processor. The Dynamic Loader routine MAPSEARCH is called to search the Storage Map Table to get the name and base address of the CSECT in which the interruption occurred. This base address, together with the interruption address given in the VPSW, is used to develop the location of the interruption. Using User Prompter the routine now issues to SYSOUT the diagnostic message, the old VPSW, the CSECT name, and the displacement of the interruption. After processing the interruption, this processor exits to the Command Analyzer if the task is conversational. Otherwise, the Abnormal Task Termination (ABEND) Routine is invoked.

BATCH MONITOR

The Batch Monitor is a part of the Operator Task and is created at Startup time. The Batch Monitor runs with the same schedule table entry as the main operator task. In addition, three other parameters that govern the activities of the Batch Monitor are specified at each installation at system generation time:

- The maximum number of nonconversational (non-BULKIO) tasks that may be concurrently multiprogrammed.
- The maximum number of background tasks (i.e., tasks which have been placed in the nonconversational mode by means of the BACK command) that may be concurrently miltiprogrammed.
- An indication of whether BULKIO jobs or batch tasks should be given preference for selection from the Batch Work Queue by the Batch Monitor.

The principal functions of the Batch Monitor are:

- To initiate all nonconversational tasks except those started by the BACK command.
- To assign a batch sequence number (BSN) to each nonconversational task.
- To scan the Batch Work Queue (BWQ) for initiation requests from the BULKIO preprocessor and EXECUTE routines.

The Batch Monitor task is initialized in response to a call from the Main Operator Housekeeping Routine.

The Batch Monitor creates the Active Remote Task (ART) table. The ART table is initially empty and is used to maintain the status and number of active tasks which have been placed in the nonconversational mode through the BACK command.

After creating the ART table, the Batch Monitor invokes DDEF to define the Batch Work Queue (BWQ). The Batch Work Queue is a shared VISAM data set that contains entries representing requests for the creation of nonconversational tasks sent to the Batch Monitor by the BULKIO preprocessor, the EXECUTE command, and the BULKIO task.

The Batch Monitor then opens the BWQ, insures that it is shared, and examines each entry to determine the appropriate corrective action to take as a result of a previous system failure or normal shutdown.

If a task had not yet been created for the entry when the system stopped, the entry is retained in the BWQ. Thus, the request is preserved even though the system was restarted.

If a task had been created, the task will be restarted provided it is a PRINT, PUNCH, or WT task.

All other entries are removed from the BWQ. These entries generally represented input Bulk I/O operations that may be restarted at the discretion of the system operator, and nonconversational jobs in execution at the time of the system failure or shutdown.

Once this initial processing is complete, the Batch Monitor Initialization routine exits to its caller, the Main Operator Housekeeping routine (MOHR).

There are a number of entry points to the Batch Monitor. Most of them are associated with subprocessors of the Batch Monitor which handle the messages received by the Batch Monitor. For the most part, these are messages requesting the creation or cancelation of a background task.

At its normal entry point, the Batch Monitor processes requests entered in the BWQ. The presence of requests in the BWQ is indicated by an interruption flag set by one of the Batch Monitor subprocessors each time a subprocessor receives a request from a Command System routine to initiate a nonconversational task. The Batch Monitor checks for pending requests essentially each time the Operator Task is awakened by an interrupt and, when resources and system parameter limits permit, invokes either its Batch or BULKIO subroutine to complete the processing of all pending requests. When processing is completed, or, if there is no work to process, the Batch Monitor invokes the Task Monitor to enqueue an entry to the Batch Monitor.

The BULKIO subroutine selects entries from the BWQ and determines if the entry involves a public or private device. If the device is private, a new task is created and BULKIO initiates the task by means of a VSEND to its external interrupt processor. If the entry involves a public device, the subroutine selects an available device by searching the S entries in the BULKCOMM table. When an available device is found, the data from the BWQ entry is moved into the corresponding S entry and the BWQ entry is marked in service. The actual work is performed by the BULKIO task.

The Batch subroutine determines whether the system limit of nonconversational tasks has been reached, and issues a supervisor call to create a TSI for the new task. It then sends a task initialization message.

SYSTEM OPERATOR AND ADMINISTRATOR SERVICES

System operation involves the interaction between the system operator and the time sharing system. System administration involves the granting and withdrawal by a system administrator of permission to use the time sharing system and the establishment of accounting records covering use of the system. System operators and administrators have at their disposal commands which are not made available to other users.

The functions that support these services can be grouped as:

- System operation control
- System operation command processing
- System administration
- Accounting services

SYSTEM OPERATION CONTROL

The system operation control function is centered in the Main Operator Task. The Main Operator Task is a conversational task created during system startup. The STARTUP program simulates an initial attention interruption from the operator's terminal by creating and enqueuing a GQE on the interruption queue for the Channel Interrupt Processor. (See "Creation of a Conversational Task").

During the initialization of the Main Operator Task, the Command Controller invokes the Main Operator Housekeeping routine (MOHR) after LOGON is complete. MOHR performs functions such as opening the System Log data set and setting a flag which allows other tasks to be created. This flag prevents a user from logging onto the system before the Main Operator Task has completed initialization processing. After setting this flag, MOHR issues a supervisor call to create the Batch Monitor task and sends a task initialization message to the Batch Monitor task. The Main Operator Task is interruption driven. That is, it only receives CPU time when there is a task interruption to be processed.

The Main Operator Communications Program (MOCP) processes messages from other tasks to the operator, and the operator's replies to those messages. MOCP processes, in three separate routines, incoming messages, operator replies to messages, and overdue replies. MOCP is initially entered with the MCB for the message, and assigns a reply number for any message requiring a reply (WTOR messages). The message is then written to SYSLOG. If a reply is requested, MOCP queues the MCB on the Reply Queue.

Later, at MOCP's second entry, reply numbers and text are checked for validity. MOCP dequeues the MCB from the Reply Queue, writes the reply to SYSLOG data set, and sends the reply to the task. If, as is usual, the reply is to a task other than the Main Operator task, MOCP uses the VSEND SVC to send the reply. MOHR is called when answering the System Configuration Message at STARTUP time.

When a message is sent to the system operator, it may or may not require a reply. If the message requires a reply, the operator will answer by using the REPLY command. The REPLY command routine correlates the reply with a message that had been previously sent to the operator.

The reply to a message may be either specific or general. A specific reply consists of a combination of characters such a YES or NO that is specified as acceptable in a Reply Checking Table. The Reply Checking Table is created during system generation. An index contained in the original message specifies which entries in the Reply Checking Table constitute valid replies to the message. A general reply consists of any arbitrary combination of valid characters entered by the system operator. For instance, it may be the symbolic device address of a tape unit on which the operator has just mounted a tape volume.

If a message requires a specific reply, the REPLY command routine checks the reply and stores the reply in the original message entry in the Reply Queue. REPLY also sets a flag in the original message entry to indicate that a specific reply has been received.

If a general reply is required, the REPLY command sets a flag in the original message entry in the Reply Queue to indicate that a general reply has been received. The REPLY command routine then passes the reply on to the MOCP through the VSEND supervisor call. This method of communication is used as a convenient way of posting the reply even though the REPLY and MOCP routines both reside in the Main Operator Task.

SYSTEM OPERATION COMMAND PROCESSING

System operation commands are processed by routines that perform such functions as:

- Verifying the parameters contained in an operator's reply to a message (REPLY Command).
- Writing messages from the system operator to conversational tasks (MESSAGE and BCST). If the Message Broadcast routine is called via MESSAGE, the routine sends a message to a specified conversational user; if called via BCST, the routine sends a message to all currently active conversational users. The message is also written in the operator's log (SYSLOG).
- Terminating a user's conversational task (FORCE).
- Terminating a user's nonconversational task (CANCEL). The Batch Monitor is called as requested to cancel a nonconversational task that is executing or awaiting execution.

- Adding devices to, and removing devices from the system (DROP and HOLD). The HOLD routine removes a hardware element from the time sharing system configuration; the DROP routine cancels the effects of a previous HOLD.

- Terminating operation of the system (SHUTDOWN). The Batch Monitor is called to terminate all nonconversational tasks in the system.

- Reading batched data sets into the system from a card reader or unstacked data sets from magnetic tape (RT).

- Initializing the BULKIO task and adding or deleting devices assigned to that task by means of the ASNBD command.

SYSTEM ADMINISTRATION

The system administration function is performed by the JOIN and QUIT routines. The JOIN routine operates only in conversational mode to define a legitimate new user to the system, and to create a user library for this new user. The QUIT routine removes a specified user from the system, and erases, reassigns, or stores his data sets on a private volume.

ACCOUNTING SERVICES

Accounting services are performed by various system routines. At the end of a task, the accumulated CPU time the task has used from its initiation to its conclusion is obtained and stored in the User Table. Terminal time, storage time, and device usage is also recorded. The time is stored as an accumulated count in an entry keyed by the user's user identification and charge number in SYSUSE (see "Resource Allocation and Control").

COMMAND ROUTINES

The command routines that support commands available to the user (not including those reserved for system operators and administrators) can be divided into eight functional groups:

- Task management routines

- Data management routines

- Object module handling routines

- Information request routines

- Command Creation routines

- Language Processor Control routines

- Text Editor command routines

- Program Control System (PCS) routines

Task Management Routines

These routines allow the user to initiate, terminate, or change the system's operation in his task. Three of them (EXECUTE, BACKGROUND, and CANCEL) are used in conjunction with the Batch Monitor.

LOGON: identifies the user to the system for task initiation by validating his credentials and performing certain other task initialization functions.

LOGON uses the SCAN routines to fetch and validate each operand. As each operand is validated, the operand's contents are stored in Task Common.

Additional task initialization functions of LOGON are:

- Updating the User Table entry for this user to show the number of tasks currently in execution for this user and to indicate whether the new task is conversational. Only one conversational task may exist for each user.

- Completing fields in the Task Status Index by using the SETUP SVC to insert userid, external priority, and authorization values.

- Adding further information to Task Common. Values are set to show the user's privilege class, completion of logging on, and whether the task is a Bulk I/O task.

- Using the AVAUX and AUXSET macro instructions to test for the availability of auxiliary storage and to warn or terminate a task when it exceeds its limit.

- Using the SCHED macro instruction to calculate the schedule table entry assigned to the task.

- Creating the JFCB for the User Library via DDEF.

- Marking the date and time of LOGON on the task's SYSOUT via User Prompter.

- Indicating in Task Common and in the Dynamic Loader PSECT, if CSECT packing is requested.

Control then passes to the Command Controller.

LOGOFF: terminates a user's current task. That is, it disposes of any uncataloged data sets that were defined by the task, releases all devices allocated to the task, and releases the task's virtual storage and auxiliary storage. When LOGOFF is completed, the task no longer exists.

LOGOFF processing consists of two major sections. The first is always performed. The second is only performed for uncataloged data sets that have been defined for the task by DDEF.

LOGOFF functions in the first section include:

- Blocking any recursions of the Abnormal Task Termination (ABEND) routine.
- Informing the Batch Monitor when a nonconversational task is logging off so that the Batch Work Queue may be updated.
- Indicating logoff in the User Table entry.
- Closing the SYSIN and SYSOUT data sets via GATE. For a nonconversational task, LOGOFF will issue a PRINT command to request the listing of the SYSOUT data set.
- Using RELEASE to release all the task's data sets and private devices.
- Using the RCR macro instruction to compute the task's CPU time and charges.
- Using LOADER LOGOFF to release the task's virtual storage.
- Using the DLTSI SVC to eliminate the task entirely by releasing the Task Status Index.

The second LOGOFF section disposes of uncataloged, defined data sets by using ERASE for erasing them or CATALOG for cataloging them depending on such factors as data set type, task mode, whether the system is being shutdown, and/or user response.

SECURE: This routine is called once by a nonconversational task to reserve all devices that will be needed during execution of that task. The Device Management Mount Request (MTREQ) routine is invoked to allocate the requested device. If the requested device is not available, the request remains unqueued and the task is suspended until the device is available. The SECURE command is discussed in "Device Allocation."

EXECUTE: This routine sends a message to the Batch Monitor asking it to initiate a nonconversational task and telling it what the task's SYSIN will be. After the data set name has been fully validated, EXECUTE builds a request message and sends it, via VSEND, to the Batch Monitor which assigns the task a Batch Sequence Number (BSN) and adds an entry for the task in the Batch Work Queue, after writing the BSN assigned to SYSOUT. When this data has been passed to the Batch Monitor, EXECUTE returns to the DIRECTOR.

BACK: This routine changes a conversational task to nonconversational mode.

The BACK routine first invokes the Find Data Set (FINDDS) routine to ascertain that the new SYSIN data set name (supplied as an operand in the BACK command) is either cataloged or has already been defined in the task through a DDEF command. If the data set has not been defined during this task, FINDDS will create a job file control block (JFCB) for the data set, provided the SYSIN data set name has been cataloged. The RCR macro instruction is issued twice. The first time it is issued in UPDATE mode to update the User Table entry; the second time in RATION mode to determine if the user has exceeded his limit of nonconversational tasks. Next, the Batch Monitor is called to check that the system can accept a new background task at this time. The Batch Monitor will return a Batch Sequence Number if a new task is acceptable, and BACK will then write that number on the conversational user's terminal, confirming the acceptance.

The remaining operations switch the task's mode. The SYSIN and SYSOUT of the conversational task are closed and the terminal device allocated to them is released. The user's entry in the User Table is then updated to show that the user no longer has an active conversational task, and the mode indicator in the Task Status Index (TSI) is reset accordingly. The new nonconversational SYSIN and SYSOUT are set up for use, and opened. Finally, the routine issues the SCHED macro instruction, changing the task's schedule table entry to reflect the new mode. Further execution of the nonconversational task is controlled by the scheuling algorithm.

CANCEL: This routine terminates the specified task by creating a request message and passing it via VSEND to the Batch Monitor. The routine informs the Batch Monitor of the command issuer's privilege class to determine whether he has the right to cancel the specified task. If he has, the Batch Monitor cancels the task. If not, the command is rejected. At the end of its processing, CANCEL returns to the Command

Controller. Before sending the request message to the Batch Monitor, CANCEL determines if, during its processing, an attention interruption has been enqueued by the Task Monitor. If one occurred, CANCEL immediately returns control to the Attention Handler.

TIME: This routine enables a user to specify a time interval at any time for a task. The interval must not be less than zero nor more than 450 minutes. Upon normal entry, the time parameter is validated and converted from milliseconds to a binary number. Timer interrupt handling is established via DIR, SIR, and STEC macros. Upon entry from VMTI, timer clock #14 is initialized to a default value specified by the SCMTIM field of System Common (CHASCM). When the time interval expires, an interrupt occurs and control is passed to the interrupt routine, which issues another DIR, STEC, and SIR macro to reset clock #14 to one minute. It then issues a message declaring the time interval elapsed and ABENDs the task if it is in nonconversational mode. Finally it will link to the director via queued linkage entry, and return to the task monitor.

Data Management Routines

The data management command routines allow the user to operate on his data sets. Eleven of the routines are for general data management. The three remaining routines (PUNCH, PRINT, and WT) are for bulk output capabilities.

CATALOG: Using the Catalog Services routines, CATALOG enters or changes SAM data set names in a user's catalog. For VAM data sets, CATALOG may only be used to change a data set name in the catalog or specify a generation data group. CATALOG invokes the FINDDS routine to locate the Job File Control Block (JFCB) for the current data set name operand and checks to ensure that the state of the data set (cataloged or not cataloged) as specified by the operand agrees with the state shown in the JFCB. If the data set is to be added to the user's catalog under its current name (or an existing catalog entry with this name is to be updated without changing the data set name), the entry is created (or updated) by calling ADDCAT.

When a new data set name operand is entered in addition to the current name, the same check is made to determine if the current data set name is already cataloged. If the data set is cataloged and not currently in use, the current name in the catalog data set descriptor is replaced with the new name (via DELCAT). If the data set resides on direct access, the current name in the data set control block (DSCB) is changed by RENAME. Then ADDCAT is called to change fields within the catalog, and control returns to the caller.

When the catalog entry and the appropriate control blocks (JFCB and DSCB) have been updated, control returns to the caller.

DELETE: used to remove (via DELCAT) one or more data set names from the user's catalog without affecting the data set in storage.

DATA: enables a user to create a data set or a member of a partitioned data set by entering data from SYSIN. The records can be of variable length and the user can specify either a VISAM line data set or virtual sequential (VSAM) organization. If a line data set is being created, the user can modify, insert, or delete entries. This routine, after obtaining and validating input parameters, creates a JFCB and then opens a data set. The routine requests data one line at a time from SYSIN. In a conversational task, GATE is used to fetch the lines.

DATA checks for attention interruptions before and after each call to GATE. The path taken when an attention interruption occurs depends on the type of data set being built, as well as how far processing has advanced. If the data set has not yet been opened, DATA merely returns control, leaving the JFCB set up (if it was generated). If a VAM data set has been opened, the routine erases it, by a call to ERASE, before returning control. For a VISAM data set, the routine closes the data set in normal fashion and then returns control to its caller. The VISAM data set, once opened, thus exists and may be added to later, if desired, by means of the MODIFY command routine.

DDEF: Defines a data set (either existing or one being created) and describes its characteristics to the system by creating a JFCB entry in the Task Data Definition Table. In addition, the routine issues requests for device allocation. External storage is allocated for a new data set on direct access external storage. The DDEF command is described in "Data Management."

The operation of this routine may be divided into four steps:

1. Read the command (or macro) operands and move these operand values to the new JFCB. During this step, if confirmation has been requested the routine will prompt for every omitted operand.

2. Move default values (where indicated) into the new JFCB. If the data set being defined is cataloged, values

available in the catalog entry for that data set are also moved to the JFCB.

3. Distinguish between data sets on a private or public volume. A data set on a private volume results in a request for device and space allocation. A new data set on a public volume causes a request for space allocation and the creation of a catalog entry for the data set.

4. Link the newly created JFCB into the proper chain or chains in the Task Data Definition Table.

<u>CDD</u>: Processes prestored DDEF commands from a line data set, thus permitting the user to establish the DDEF commands once and reference them later either individually or as a set.

If the command operand field does not include ddnames, this routine will fetch all commands in the prestored data set and present them one by one to the appropriate routines for execution. When one or more ddnames are supplied as operands, the routine scans sequentially through the prestored data set looking for a match. As each match is found, the prestored DDEF command is readied and passed to the DDEF command routine for execution.

<u>CDS</u>: Copies logical records of a specified data set. The new data set can be created for a different device type if the organization of the original data set is permitted on that device. For VAM partitioned data sets, the CDS routine copies one member at a time into another partitioned data set. Members which are load modules must be copied by the Linkage Editor. To copy complete data sets, including those of partitioned organization, regardless of contents, the VV (VAM to VAM) command entry of the VAM Tape routine should be used.

The CDS routine obtains a JFCB for the original data set and one for the data set to be created and compares the two. Normally, both data sets must have the same kind of organization; i.e., VAM or SAM. Any combination of VAM data sets can be copied (e.g., VSAM to VISAM, VISAM to VPAM). The single requirement is that a new data set can be changed from VSAM to VISAM organization only if the key is defined to be a field already in the record. If this type of change is requested and the user has omitted any required key information, he is prompted to enter the necessary DCB fields. When the copy is complete, or when an attention interrupt is detected, the data sets are closed and control returns to the calling program.

<u>ERASE</u>: The operation of ERASE varies for SAM and VAM data sets. For SAM data sets, ERASE releases external storage occupied by a data set on direct access volumes and/or removes their entries from the catalog for a data set owner or user with the proper access privilege. If the data set is partitioned and followed by member names, ERASE opens the data set, deletes the member (by issuing a STOW macro instruction), and closes the data set. For data sets other than partitioned, ERASE obtains a list of data set names whose leftmost qualifiers match the input name and processes the names one at a time. The External Storage Allocation routine SCRATCH is then called to delete the Data Set Control Blocks (DSCBs) from the Volume Table of Contents. If the data set is cataloged, the Catalog Services routine, DELCAT is called to delete the various index levels from the catalog structure. If a data set has been opened by other sharers, the ERASE command is ignored.

For VAM data sets, ERASE only deletes a member of a partitioned data set. It does this in the same way as for SAM data sets above. The actual deletion of a VAM data set or member is done by DELVAM.

<u>DELVAM</u>: Deletes a VAM data set by calling DELCAT to delete its catalog and by calling RELEXPG to release its DSCB and data page entries on the volume.

<u>RET</u>: Alters the data set attributes specified in the DDEF command and contained in the catalog.

<u>MODIFY</u>: Inserts, deletes, presents, or replaces lines of a VISAM data set. Modifications are effected by employing the user supplied key (a line number if the data set is in line format) to point out the location of the specified record (line). Input records containing the user's modifications are obtained one at a time until the end-of-input record is reached. When this indication is reached, the data set is closed. For a partitioned data set, the POD is updated to reflect any alterations before the data set is closed.

<u>PERMIT</u>: Enables a catalog owner to authorize some or all other users to have access to some or all of his cataloged data sets, or to withdraw such authorization. The PERMIT routine sets up a parameter area for either the SHARE or UNSHARE routine of catalog services. If the sharing access qualifier is "R" (meaning that sharing privileges are being revoked) PERMIT calls UNSHARE rather than SHARE. If the access qualifier is defaulted, the read and write (RW) option is assumed. An example of the use of this command is presented in "External Sharing."

SHARE: Links a data set name in the sharer's catalog with an index level or data set descriptior in the owner's catalog. The routine obtains the input parameters, checks them for validity by calling SCAN, and then calls the Catalog Services SHARUP routine to make the necessary sharing descriptor entries in the catalog. An example of the use of this routine is presented in "External Sharing."

RELEASE: Deletes Job File Control Blocks (JFCBs) from the Task Definition Table. If the volume is private and the last of several users is releasing the volume, RELEASE will also free the space occupied by the Private Volume Table. RELEASE may be used to release the devices associated with a dataset, to deconcatenate one or all data sets of a given concatenation, and to remove a specified job library from the program library list maintained by LIBMAINT. When the data set (or job library) is located, the JFCB will be deleted. The device management RELEAS routine is called if the data set is on a private device. Devices for public, uncataloged data sets are not released unless ERASE/DELETE has made the request.

PUNCH: Punches a set of cards from a specified VSAM or VISAM data set. PUNCH differentiates between public and private data sets, and performs approximately the same steps as PRINT (described below).

PRINT: Prints a public or private data set in nonconversational mode on a high-speed printer. The operation is somewhat different for public and private data sets.

For a data set residing in private storage, PRINT operates as a separate task, independent of the task that issued the PRINT command. A new task is therefore initiated. PRINT calls Virtual Memory Task Initialization, opens the SYSOUT data set via GATE, issues DDEFs for the input data set and printer (devices reserved by the Batch Monitor), opens the DCBs, and obtains buffers.

For a data set residing in public storage, no new task is created. The BULKIO task is invoked to put the data set out to the printer.

For both private and public storage the input data set is read, one logical record at a time. After each read, the edit option is tested to see if the record has an ASA control character (or, in the case of PUNCH, if a specific stacker was requested). When all input data has been processed, PRINT frees buffers and closes the data set.

For private data sets, exit is to the LOGOFF command routine which terminates the task. For public data sets, exit is a VSEND to the LOGOFF subprocessor in the Batch Monitor.

WT (BULK I/O): Writes a data set onto magnetic tape in proper format for off-line printing. An independent task is created to process WT, as is done for a PRINT or PUNCH of a private data set. The routine then builds a blank print line to provide for initial page positioning. The routine tests if editing has been specified, converting each ASA code to machine code, and moving it to the output record for writing. The data in the output record is from the previously read input record. This order is necessary since machine code control characters cause a print then space, but ASA codes specify space then print. By stepping a record ahead, the routine achieves proper spacing.

VT (COPY VAM TO TAPE): Copies a VAM data set on tape as a physical sequential data set. This routine may also be initiated from privileged modules via the CALL macro, the operand field of which specifies the entry point and the input and output dsnames. The input dsname must be the name of a VAM data set. For the output data set, a JFCB must be found with the ddname of DDVTOUT. Specified within this JFCB must be physical sequential organization and a tape volume on a nine-track drive. The first record on the output tape will contain the input JFCB and the common portion of the input data set's Format-E DSCB. Data pages are located by indexing through the RESTBL for the input data set, and are written to tape as a 4096-byte record by BSAM WRITE. After the tape operation is completed, both data sets are closed, and the output data set is cataloged. Cataloging will not be performed for the output data set where the dsname was preceded by an asterisk (*).

TV (COPY TAPE TO VAM): Restores a physical sequential copy of a VAM data set from tape to direct access storage. Like the VT command, this routine may also be initiated from privileged modules via the CALL macro. The output name must be for a new VAM data set. An output JFCB is required only if the data set is to be restored to a private VAM volume. Records from tape are input byBSAM READ and output by VSAM PUT. After the tape to VAM operation is completed, both data sets are closed, the new data set's Format-E DSCB is updated from the DSCB data retained from the original data set, and the output data set is cataloged.

VV (COPY VAM TO VAM): Produces an identical copy of a VAM data set on direct access storage. Like VT and TV, this routine may also be initiated from privileged modules via the CALL macro. The input dsname must be the name of a VAM data set. The output dsname must be for a new VAM data set. An output JFCB is required only if the data set is to be copied to a private VAM volume. The common portion of the input data set's Format-E DSCB is retained to describe the new data set once the copy operation is completed. Records are written into the output data set by VSAM PUT. The output data set at this point is treated as a VAM sequential Format 'U' data set. When the copy operation is complete, both data sets are closed, the new data set's Format-E DSCB is updated from the DSCB data retained from the original data set, and the output data set is cataloged. Processing concludes by RELEASEing any JFCBs created by this routine.

Object Module Handling Routines

These routines allow the user to call and initiate execution of nonprivileged object modules stored within the system.

LOAD: Places a specified object module in the user's virtual storage thereby allowing its execution. A minimal save area is allocated. This save area contains the location of the PSECT (working storage address constants) required by the object module. Upon completion of a successful load, this routine issues a confirmation message and returns to the caller. This command is described in "Dynamic Loader".

RUN: Loads a problem program into the user's virtual storage, if it is not already there, and initiates execution of the program. Messages and diagnostics are issued to the user through GATE. When processing has been completed successfully, RUN transfers control to the Command Analyzer, which transfers control to the problem program via the Task Monitor. This command is discussed in "Nonconversational Processing."

UNLOAD: Removes a specified object module from the user's virtual storage and closes all user-defined data sets associated with the module. However, the data set remains open if it has a reserved ddname. This command is discussed in "Dynamic Loader."

The BRANCH command causes the VSPW, containing the location in the user's program from which execution will start or be resumed, to be altered to an operand specified in the command. The Command Analyzer and Executor is notified of an end of statement or end of sublist.

The CALL command initiates program execution. If the module has not been loaded, PCS will request the dynamic loader to load it. The VPSW is then modified to start execution at the module entry point, and the contents of the user's linkage registers are set so that type-I linkage can be achieved. Control is given to the user control routine for this.

The user also has the facility of directly calling an object program or a procedure. By entering the name of the module and the parameters it requires, the user can cause the system to load the module, together with any implicitly referenced modules, and to transfer control to the module. This facility is restricted in that it cannot be employed in a dynamic statement. If a module and a procedure have the same name, the procedure is called.

When a GO command is entered, PCS notifies the Command Analyzer and Executor of an end of line and end of sublist. Since the location of the VPSW was not altered, execution will start or resume at the current address.

Information Request Routines

These routines enable the user to access certain system information.

DSS?: Presents the fully qualified name and certain attributes of a cataloged data set (or sets), as specified by the user. Attributes that are presented are: (1) sharing status, (2) Access status, (3) device type and volume number, (4) creation and expiration dates, (5) data set organization, and (6) data set length (for VAM data sets only).

LINE?: Presents the contents of a line (or lines) of a line data set, as specified by the user. It writes the lines on SYSOUT, using GATE.

POD?: Presents the member names (and optionally, the aliases and other member-oriented data) of individual members of a cataloged VPAM data set.

Command Creation Routines

The Command Creation routines, PROCDEF and BUILTIN, provide the user with the ability to create new commands from a combination of system supplied commands and/or object coding, permitting him to define his own parameters and establish the desired defaults for these parameters.

PROCDEF: This command routine defines a command constituting a combination of other system-supplied commands. In issuing PROCDEF, the user must specify, as a parameter, the name to be assigned to the new user-defined command. The user may then call his procedure by issuing the procedure name, which will result in a two-stage process. In the first stage, dummy parameter replacement is done where specified; in the second stage, the lines of the procedure are scanned and executed in the same manner as a system-supplied command.

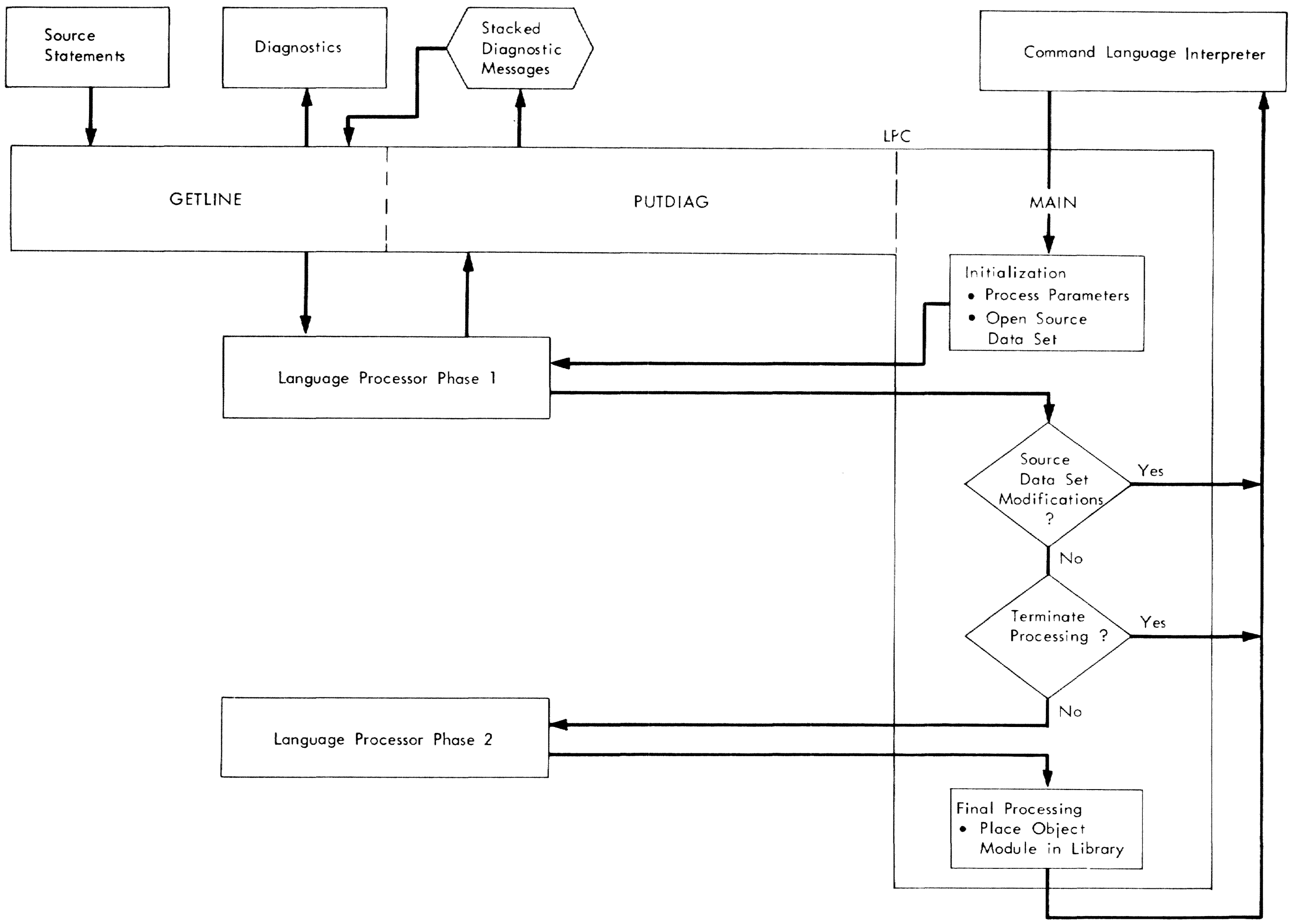
BUILTIN: This command routine creates a command procedure which will accomplish actions not achieved by any current system-supplied commands or combination of system-

supplied commands. The user creates an object program and defines the object code as a command by use of BUILTIN. A call on a BUILTIN procedure is just like that on a normal command. It differs from a normal object module call in that parameters may be supplied which follow normal command parameter rules, rather than normal program call rules.

LANGUAGE PROCESSOR CONTROL

The Language Processor Control (LPC) is the interface between the Command Controller and the TSS/360 language processors. LPC gathers input parameters from the RUN command for a language processor, loads the language processor, and passes parameters to it. The language processor calls upon LPC for source lines, and uses LPC to output diagnosics and obtain correction. LPC consists of three routines: LPC MAIN, GETLINE, and PUTDIAG. Figure 94 shows the flow of control. LPC MAIN operates at the start of language processing, at the end of the input scan phase, and at the end of processing. It is started by the RUN command, collects and verifies input parameters, issues DDEF (Define Data) instructions and opens the source data set. It then calls the language processor.

If input is from a terminal, each source line will be placed in this line data set and passed to the language processor. When the language processor has scanned the source data (for which it uses GETLINE) it returns control to LPC MAIN with a code indicating the next step, which may be to continue processing, terminate, or modify the source data set and restart processing. (In nonconversational mode, processing continues unless the language processor indicates that termination is necessary because of source errors.) When the language processor finishes, LPC MAIN stows the new module. Control returns to the caller.



Source Statements
Diagnostics
Stacked Diagnostic Messages
Command Language Interpreter
LPC
GETLINE
PUTDIAG
MAIN
Initialization
• Process Parameters
• Open Source Data Set
Language Processor Phase 1
Source Data Set Modifications ?
Yes
No
Terminate Processing ?
Yes
No
Language Processor Phase 2
Final Processing
• Place Object Module in Library

Figure 94. Language Processor Control Overview

GETLINE operates only when it is called by a language processor during source data set processing. It fetches input records either from SYSIN or from the source data set, depending on whether or not the data set is prestored. In conversational operation, GETLINE also issues to the terminal user any diagnostic messages stacked by PUTDIAG before getting the next input line.

PUTDIAG receives diagnostic messages from the language processor and either writes them via the GATE routine or stacks them for output by the GETLINE or LPC MAIN routine.

There is a trade-off involved between a system/language-processor interface that utilizes a limited interface such as that represented by LPC and one which allows a more general interface with system facilities such as the text editor. The latter requires more knowledge of the system to be built into the language processor. Thus, TSS/360 has chosen to utilize the type of interface represented by LPC.

Text Editor

The Text Editor provides a facility for editing lines of information in an existing VISAM data set, or as they are entered into the VISAM data set. It also provides the communications necessary to permit editing to be performed at the same time that a language processor is compiling or assembling from the data set.

The Edit Controller provides an interface with the other modules of the system. Its function is similar to the LPC function for assembler, FORTRAN, and linkage editor.

With the text editing commands, the user can create and edit data sets at the same time. He can correct, insert and delete lines; he can segment a data set; and he can transfer lines from one data set to another. The user can also display lines of data set at this terminal, and nullify previous changes that were made by the text editing commands.

CONTEXT: Replaces a specified string of characters within a line or range of lines with another specified character string.

CORRECT: Initializes the text editor to accept correction lines from SYSIN, and to make corrections on a line or range of lines from the object data set.

EDIT: Invokes the Edit Initialization routine to initialize the text editor and the transaction table for a new data set, and to locate and open the data set. This command must precede the other text editing commands.

EXCERPT: Incorporates a range of lines from another data set into the data set currently being edited. It uses, as entry parameters, the name of the data set to be sampled, and the numbers of the first and last lines to be included.

EXCISE: Deletes a range of lines from the object data set, using the first and last lines to be deleted as entry parameters.

INSERT: Prepares the text editor to accept data lines for insertion following a given line in the source data set.

LIST: Displays a range of lines from the object data set at the user's SYSOUT.

LOCATE: Searches a range of lines in the object data set for a given character string, using as entry parameters the first and last line numbers to be searched and the string to be located.

NUMBER: Renumbers a specified line or range of lines in the object data set.

REGION: Creates a subset of specified lines of data set to be located as an entity known as a region. The entry parameter is a character string that is saved in the transaction table as a region name, and is prefixed to all subsequent line numbers until another REGION command is encountered. If the entry parameter is defaulted, a null string prefixes to the subsequent line numbers, provided that the data set has regions.

REVISE: Replaces a specified line or group of lines with those entered following the command.

STET: Restores the object data set to its condition prior to the most recent set of unprocessed transactions. If the language processor is disabled, all the editing commands entered since it was last enabled will be reversed. If the language processor is enabled, only the last transaction is reversible.

UPDATE: Updates the current region with input lines read by SYSIN. The data set must have been opened prior to entry and the DCB address stored in the transaction table.

The commands END, ENABLE, and DISABLE do not have separate text editor routines, but have entry points in the User Control routine which provides the principal interface to the text editor.

END: Terminates processing by PROCDEF and/or the text editor.

DISABLE: Restores a data set to its original state if requested. Revisions to a data set are recorded in a table and are not made permanent until an ENABLE command preceded by a previous DISABLE is issued. DISABLE, ENABLE, and STET allow the user to enter all or part of his revisions before they are made permanent.

ENABLE: Reverses the effect of a previous DISABLE command.

PROGRAM CONTROL SYSTEM

The Program Control System (PCS) of Time Sharing System/360 permits the user to check the status of his program after it is loaded or at any stage of its execution, to modify the program during execution, and to pinpoint errors. These checking facilities eliminate the need for user-written debugging instructions that must be built into the user's programs and later removed when the program is debugged.

PCS commands can refer to data by either symbolic names or virtual storage addresses. The user can use PCS commands to:

- Request at any time during execution of a program, output of the contents of data fields, instruction locations, and registers.
- Modify the contents of the user's virtual storage.
- Specify locations within his program where execution is to be stopped or started. When execution has been stopped, the user can issue additional PCS commands before he resumes execution.
- Establish logical (i.e., true or false) conditions that allow or inhibit the execution of other PCS commands.

These control and checkout services are requested by means of the following PCS commands:

- AT
- DISPLAY, DUMP, and SET
- IF
- QUALIFY and REMOVE

The following restrictions apply to the above commands:

- The nonprivileged user may modify only virtual storage locations classified as read/write.
- An AT command may never reference public storage.
- The nonprivileged user may make symbolic references only to programs loaded from the User Library (SYSULIB) or a Job Library (JOBLIB).

Processing of PCS Statements and Commands

PCS statements may require immediate or deferred execution depending on the presence of an AT command. Those statements which do not contain an AT command are executed immediately. Those statements which do contain an AT command are executed upon arrival at the location specified in the AT command. This latter deferred type of execution is said to be dynamic and is accomplished by the insertion of a PCSVC supervisor call in the issuing task's program at the points specified in the AT command. In order to complete the processing specified by the other commands included in the statement with the AT command, the user must issue the RUN command.

PCS commands and statements may be issued in conversational or nonconversational mode. In the conversational mode, a syntax check is made, symbolic references are validated, and diagnostic messages are delivered to the user assuring a valid set of PCS commands. In the nonconversational mode, the same checks as in the conversational mode are made but no diagnostic prompting is possible. Diagnostics are delivered to the task's SYSOUT data set together with the PCS output, and incorrect commands are ignored. Output from the dump command is always written to the PCSOUT data set.

Processing of Immediate Statements

Phrases in immediate statements are processed individually, one at a time. Control is always returned to the Command Analyzer and Executor at the completion of processing for a phrase, unless otherwise noted.

One or more operands are allowed in both the DISPLAY and DUMP commands. The encoded information for each of the operands is placed in a phrase list. This information consists of the starting and ending virtual storage locations to be displayed, the type of symbol that the user referenced in the operand, and the location of the appropriate dictionary entry where the symbol was found. Any object code that was generated is located by linkages in the phrase list. The phrase list is then presented to the appropriate subroutines for displaying or dumping.

The SET command allows a data location to be set equal to an expression. The information pertaining to the data location to the left of the equal sign is tabularized into a phrase list, as in the DISPLAY command. Object code is compiled to compute the result of the expression to the right of the equal sign. The code is linked to via the phrase list. When the list is presented to the subroutine that performs the SET action, the object code is executed. The result of the evaluation is then returned from the generated code to the SET routine, which stores the result into the data location specified by the phrase list.

The presence of an IF expression in a statement makes its execution conditional. PCS compiles object code to evaluate the result of the expression, executes this code, and then notifies the Command Analyzer and Executor of the result of the logical evaluation. If the condition is false, the remainder of the statement is ignored; if true, the Command Analyzer and Executor calls the appropriate PCS routines to process the commands, one at a time.

The QUALIFY command allows the user to specify the name of the program module to which his internal symbols apply. Processing of the directive includes locating the internal symbol dictionary for the module, and storing the necessary information into its internal tables.

The REMOVE command permits the user to deactivate selected dynamic statements permanently. PCS processing consists of delinking and removing the appropriate table entries for the statements, and removing PCSVCs if required.

The STOP command in an immediate statement causes the user to be notified of the current status of his program. The Command Analyzer and Executor is notified that an end of line condition is met.

<u>Dynamic Statement Processing</u>

Dynamic statements are those whose execution has been deferred until a specified location in the user's program is reached during program execution. This is accomplished by means of the AT command.

PCS implements the AT command by inserting a PCSVC into the user's program at the location specified in the command operand. The user's instruction at that location is saved in internal tables. PCS then notifies the Command Analyzer and Executor that the statement is dynamic.

When the PCSVC is executed in the user's program, the Task Monitor recognizes it and enters PCS for processing. PCS searches its internal tables for the information pertaining to the SVC at that location. All the processing is then done for the actions requested at that event, as described in the paragraphs on processing of immediate statements, except as indicated below.

The condition imposed by an IF command is evaluated by executing the code generated during the immediate processing phase. If the result is false, the next statement effective at this location is processed. If the result is true, the remaining phrases in the current statement are performed.

When the STOP command is dynamically executed, the user is notified of the location of his program, and control is given to the Intervene system routine to halt the user's program and to place the task in the command mode.

Dynamic processing of the BRANCH command causes the VPSW in the ISA to be modified, and control is returned to the Task Monitor to resume program execution.

When the CALL command is dynamically processed, control is given to the user control system routine, which initiates execution of the called program. When control returns from the program, the remaining actions in the dynamic statement are performed.

<u>PCS Components</u>

PCS is divided into three major components:

- Input component - accepts and analyzes PCS statements.
- Output component - performs the indicated action and generates output either synchronously for immediate statements or asynchronously for deferred, dynamic statements.
- DISPLAY/DUMP component - called by the Output component for Display, Dump, and Set functions.

Figure 95 shows the relationships between these components and the logic of processing PCS statements.

<u>PCS INPUT COMPONENT</u>: The PCS input component consists of subroutines for the initial processing of all PCS statements. Two functional phases of this component are distinguished.

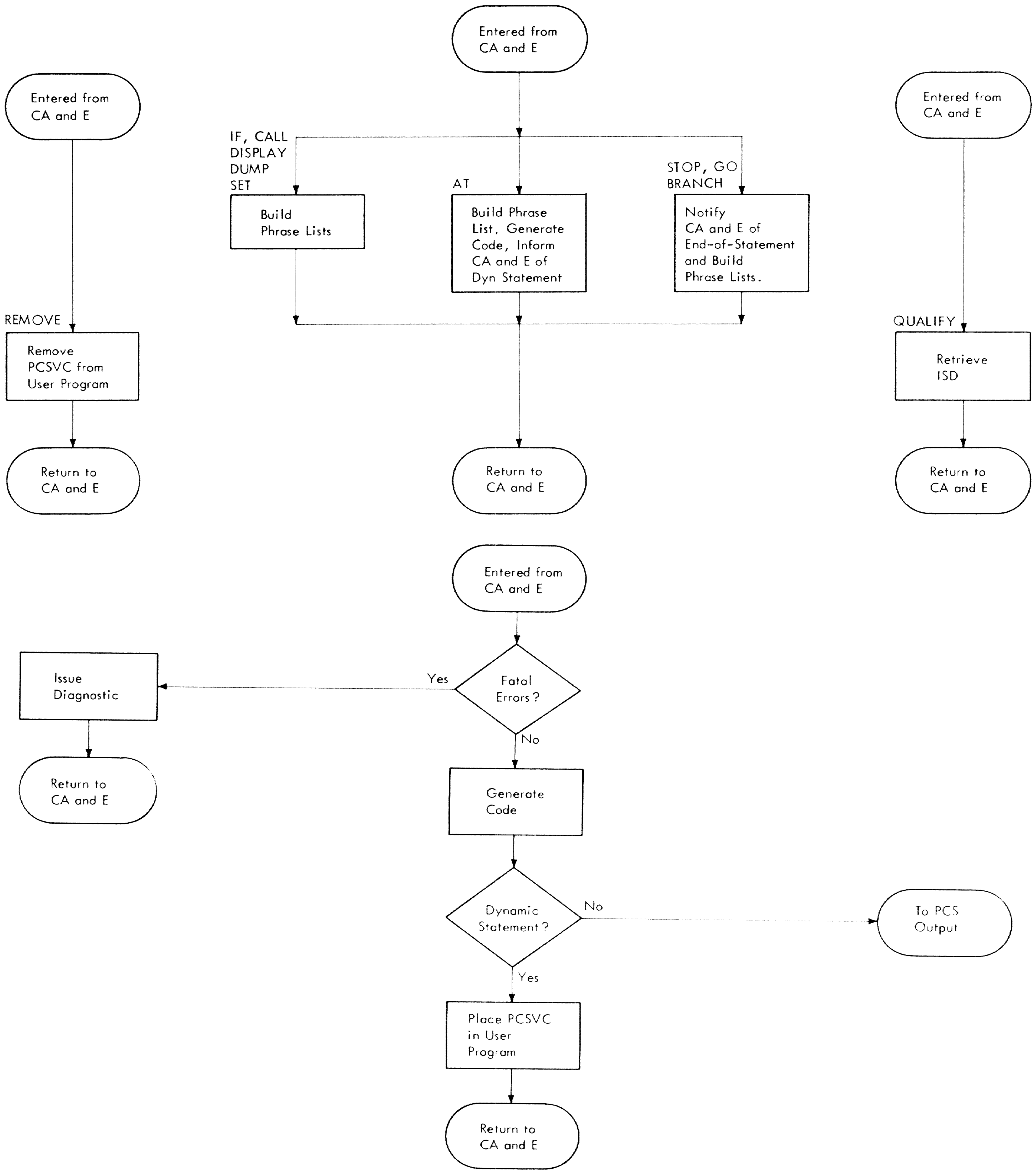


Figure 95. PCS Processing (Part 1 of 2)

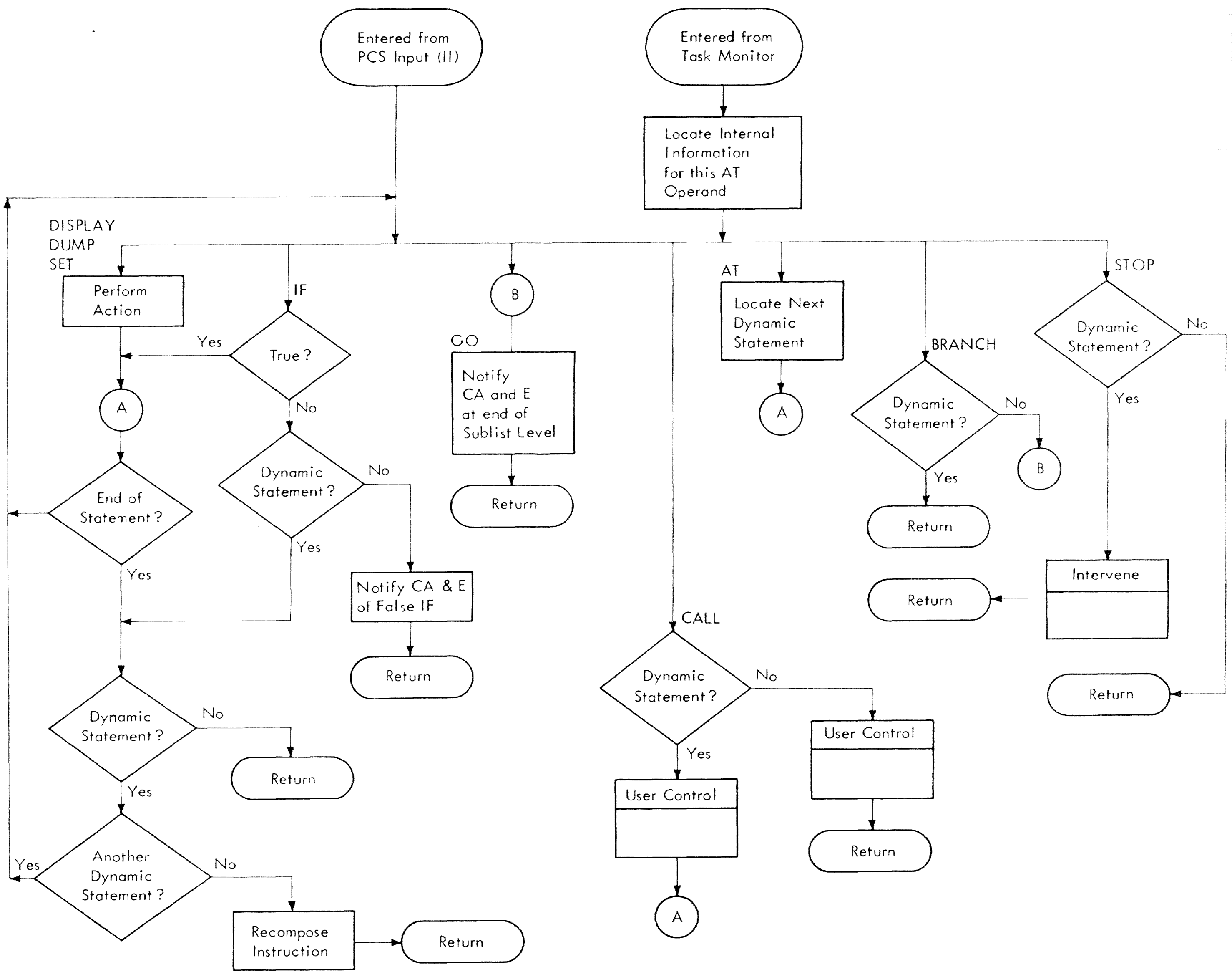


Figure 95. PCS Processing (Part 2 of 2)

Phase I accepts control from the Command Analyzer and Executor, evaluates operands into encoded forms, forms a phrase list, and makes necessary error checks. CA&E initializes a source list each time a user logs on. It obtains a user's command and places it in a level one sublist from which it then executes the command, calling the appropriate Phase I subroutine.

CA&E calls Phase II routines of the PCS input component to provide the final processing for immediate and dynamic statements. These routines scan the phrase lists generated during Phase I and generate all necessary code. If the statement is immediate, Phase II then calls the PCS output component to perform the actions indicated, and all storage is released. If the statement is dynamic, Phase II stores a PCSVC in the user's program, at the address specified by the AT command operand.

PCS OUTPUT COMPONENT: The PCS output component contains subroutines which perform the final processing for immediate and dynamic statements. PCS output is entered either from the Task Monitor, as the result of a PCSVC having been executed in the user's program, or, in the case of an immediate statement, directly from Phase II. After processing of immediate statements, control is returned to Phase II. After the processing of dynamic statements, before control is returned to the Task Monitor, the instruction in the user's program that was overlaid by the PCSVC is recomposed in working storage and followed by a return PCSVC. The VPSW is modified to point to the recomposed instruction. Control is then returned to the Task Monitor. When PCS output is again entered as the result of the return PCSVC, it clears the instruction, resets the VPSW to resume execution at the proper location in the user's program, and returns to the Task Monitor.

DISPLAY/DUMP COMPONENT: This component of PCS is called by the PCS output component whenever DISPLAY, DUMP, or SET functions are to be performed. The output component passes the address of the first phrase list as an argument. If DISPLAY is specified, the DISPLAY/DUMP component transmits values of items in a list to the user's SYSOUT. If DUMP is specified, it generates the same values onto a data set referenced as PCSOUT. This component also modifies and displays the contents of a data location referenced by a SET command.

The DISPLAY/DUMP component picks up successive item references from the phrase list, obtains the address and attributes of each item, converts the contents of each item according to its attributes, and places it in an area for output. Only one display list is processed for each entry into the DISPLAY/DUMP component. Each entry in the phrase list is processed and two display items are formed to define the attribute of the phrase list entry data. These attributes determine the output format.

PCS INTERFACES WITH SYSTEM MODULES AND TABLES

During the processing phase of its operation, PCS makes use of system modules and tables. The following paragraphs describe the interface of PCS with the system.

The Command Analyzer and Executor (CA&E)

The Command Analyzer and Executor (CA&E) serves as the primary link between PCS and the user's program. CA&E obtains the user's PCS statements from SYSIN in exactly the same way as for other commands. CA&E scans the statement and, detecting a PCS command, calls the appropriate PCS module to perform the required processing. The Command Analyzer and Executor also serves as a supervisor program for PCS, exercising control over the flow of work between PCS phases.

User Control Routine

When processing the CALL command, PCS gives control to the User Control service routine to initiate execution of the user's program.

Intervene Routine

During the processing of a dynamic statement, PCS determines if the task should be returned to the command mode. This could result from the processing of a dynamic STOP command or from an error condition recognized by PCS. Control is given to the Intervene system routine to halt the execution of the user program and give control to CA&E, thus placing the task in command mode.

Task Monitor

When the CALL command is issued, the task is put in execution via the Task Monitor. The Task Monitor also provides the processing required to call PCS when a PCSVC is encountered.

During its execution, PCS utilizes two Task Monitor subroutines. The Queue Linkage Entry (QLE) routine is called to place the task in command mode when a STOP command is dynamically executed or when certain error conditions are encountered by PCS. Queue Linkage Entry enqueues a Queue Entry on the Attention Handler Request Entry. When PCS processing is completed and control is returned to the Task Monitor, the Attention Handler will be called prior to task execution being resumed.

Data Management

PCS makes use of VPAM FIND to locate a program's Internal Symbol Dictionary (ISD) and VAM MOVE PAGE to read the ISD. PCS also uses VISAM for off-line output in response to the DUMP command.

Virtual Memory Allocation

The GETMAIN and FREEMAIN routines are used for the allocation and release of working storage. The CKCLS SVC processor is used to determine the storage protection and privilege associated with a virtual storage address.

Dynamic Loader

When a module must be loaded, PCS issues a DLINK SVC which results in an entry to the Dynamic Loader.

The MAPSEARCH subroutine of the Dynamic Loader is used when PCS is searching for a virtual storage address. PCS also uses the HASHSEARCH subroutine to resolve a module name or external reference in a PCS statement. When a referenced module is unloaded, the Dynamic Loader calls on PCS to remove from all modules which remain, all PCSVCs in the module being unloaded, and to clear the PCS tables. In this manner, remaining modules can be executed without portions of checkout statements remaining.

In addition to these modules and subroutines, PCS also accesses certain common data areas to extract data to be displayed or data required for execution of PCS commands. These data areas are:

<u>DYNAMIC LOADER'S TASK DICTIONARY (TDY)</u>: The TDY contains a task's virtual storage map and the Program Module Dictionaries. This information is required by PCS when it resolves external references into virtual storage locations.

<u>INTERNAL SYMBOL DICTIONARY (ISD)</u>: This is an optionally available table which the user must request at assembly or compile time if he intends to reference internal symbols in PCS commands. PCS locates internal references by use of this table.

<u>INTERRUPT STORAGE AREA (ISA)</u>: A user's general and floating point registers are saved in the ISA. When the contents of these registers are referenced for display or alteration, PCS finds the information in the ISA.

The ISA also contains the virtual program status word (VPSW) which PCS must modify in response to a RUN command.

PCS also performs some validity checks which depend on user type and authority codes. PCS finds this information in the ISA.

<u>TASK COMMON (TCM)</u>: PCS finds the task's operating environment described in TCM (i.e., conversational or nonconversational mode, confirmation or nonconfirmation mode).

CREATION OF A CONVERSATIONAL TASK

A conversational task is created when the user enters the LOGON command.

1. This example describes the creation and initialization of a conversational task from the point at which the Resident Supervisor receives the attention interruption, through task initialization and LOGON processing performed by the Task Monitor and Command System.

   The flow for this example is described in three charts which are keyed to the following description. The first two charts, Figures 96 and 97, cover the Resident Supervisor and Task Monitor processing up to the point at which the Command System is first invoked. The third chart, Figure 98, is presented in a style that more graphically depicts the levels of linkage involved in task initialization in virtual storage.

2. The Interrupt Stacker is the routine of the Resident Supervisor which receives control when any hardware interruption occurs. The Interrupt Stacker identifies the interruption type and creates a GQE. In this example the Interrupt Stacker is entered at its I/O interruption entry point. The entry point address is carried in the instruction counter field of the new PSW.

3. Supervisor Core Allocation is called to get 64 bytes in which to build the GQE for this interruption.

4. At this point the Interrupt Stacker proceeds to build the GQE. The Interrupt Stacker inserts the hardware device address automatically stored with the I/O interruption as the interruption code), the Channel Status Word, and the symbolic designation of the queue on which the GQE is to be placed for later processing. A test is made to see if the interruption came from a paging drum since such interruptions are handled differently from other I/O interruptions. Since this is not a paging drum interruption, the Interrupt Stacker designates that this GQE is to be placed on the Channel Interrupt Queue.

5. The Enqueue routine is called by the Interrupt Stacker to attach the GQE to the Channel Interrupt Queue as specified by the symbolic queue number inserted in the GQE by the Interrupt Stacker.

6. The Interrupt Stacker has completed processing this interruption and exits to the Queue Scanner.

7. The function of the Queue Scanner is to look for a GQE which can be processed. When one is found, the appropriate supervisor routine (Queue Processor) is invoked. In this instance a GQE is found on the Channel Interrupt Queue and the Channel Interrupt Processor (CIP) is called.

8. The CIP calls Set Suppress Flag to prevent a second CPU from being invoked to process the same queue.

9. The Channel Interrupt Processor determines that this is a Terminal I/O interrupt.

10. The Channel Interrupt Processor calls Supervisor Core Allocation to obtain a save area for use by the Terminal Control Subprocessor (TCS) and calls TCS.

11. TCS determines that this is the initial interrupt from the terminal (DEVTSI pointer = 0) and calls Supervisor Core Allocation for 64 bytes (TIOCB) and 256 bytes (Input Buffer) of core.

12. TCS generates a Read channel program for the terminal device, and calls Pathfinding to obtain the device path.

13. TCS returns to the CIP.

14. CIP calls Dequeue to remove the GQE from the queue of the CIP.

15. CIP calls Move-GQE to dispose of the GQE.

16. CIP calls Set Suppress Flag to reset the suppress flag.

17. CIP calls Supervisor Core Release to release the save area core, and CIP then exits to the Queue Scanner.

18. Queue Scanner finds no GQE in its Scan Table and exits to the Internal Scheduler.

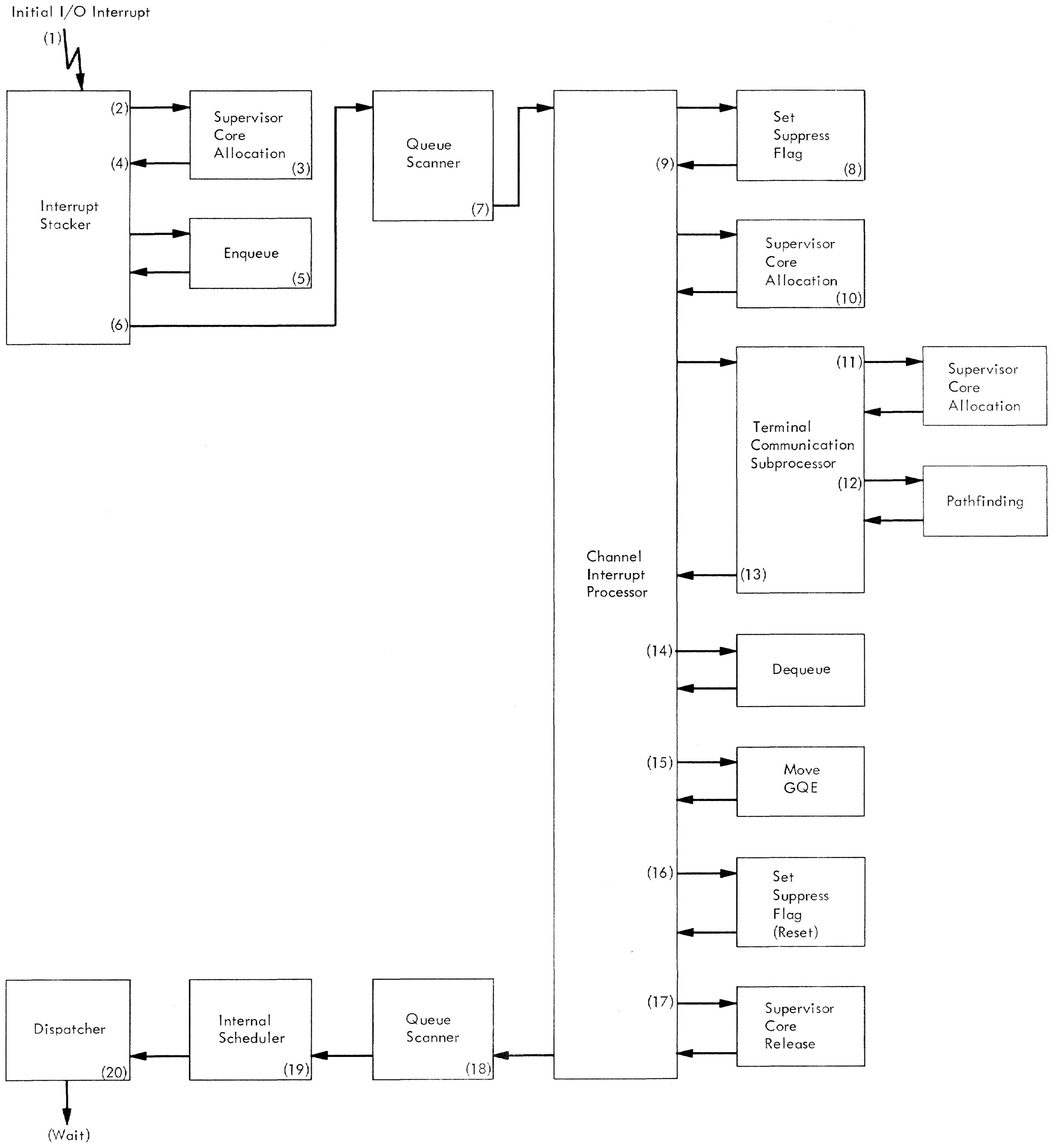


Figure 96. Resident Supervisor Task Initiation Flow (Initial Interrupt)

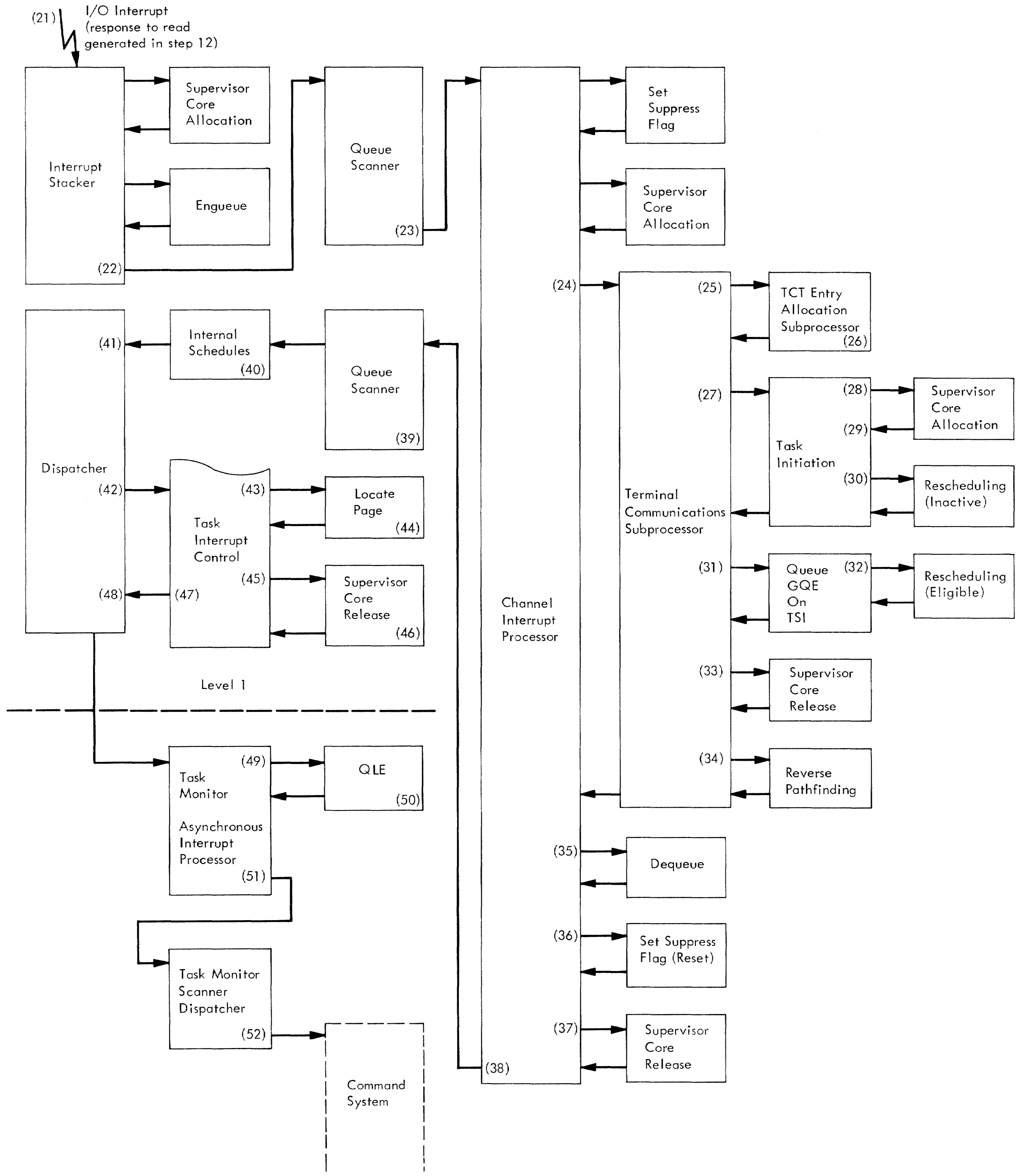


Figure 97. Resident Supervisor Task Initiation Flow (Read Response)

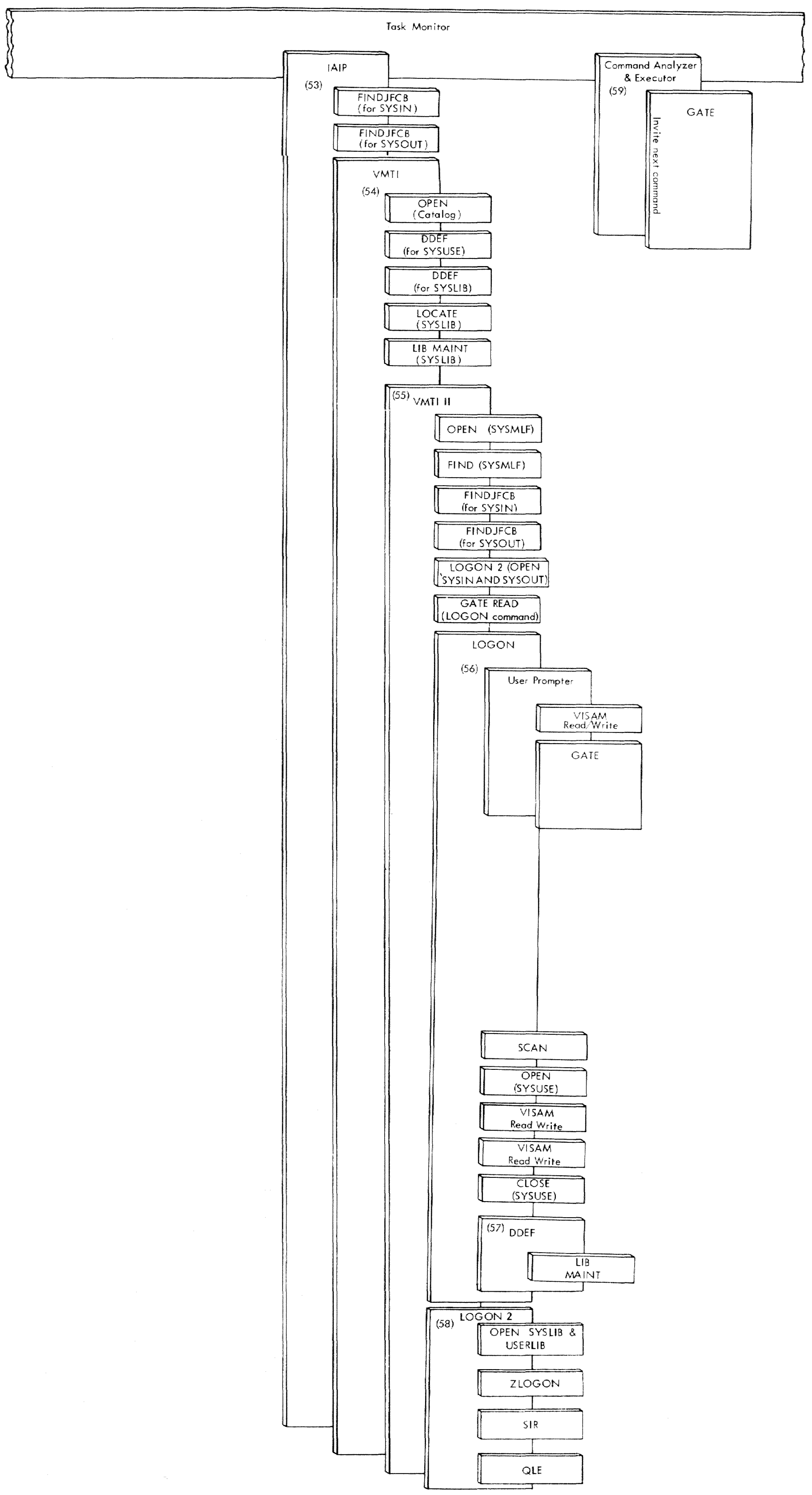


Figure 98. Conversational Task Example

19. Internal Scheduler finds no work and exits to the Dispatcher.

20. Dispatcher has nothing to dispatch and enters the Wait state.

21. Eventually the terminal responds to the Read on its line (generated in step 12) and another I/O interrupt occurs. (Assume this is LOGON)

22. Interrupt Stacker (as in steps 2-6) identifies the interrupt and creates the GQE after calling SCA for the space in which to build it. Enqueue attaches the GQE to the Channel Interrupt Queue, and the Interrupt Stacker exits to the Queue Scanner.

23. Queue Scanner finds the GQE on CIP's queue, and activates CIP.

24. CIP (as in steps 8-10) determines that this is a Terminal I/O interrupt, calls SSF to set the Suppress Flag, calls SCA to obtain save area core, and calls TCS.

25. TCS finds DEVTSI non-zero this time (points to TIOCB), finds the LOGON command in the input buffer obtained previously (step 11), and calls TCT Entry Allocation Subprocessor.

26. TCT Entry Allocation Subprocessor assigns a system TCT slot and a Buffer Page slot to the task.

27. TCS calls Task Initiation to create a new task.

28. Task Initiation calls Supervisor Core Allocation to get 128 bytes of core in which to build a Task Status Index (TSI).

29. Task Initiation then constructs the Task Interrupt Mask field in the TSI such that task interruptions are enabled, finds the address of the skeleton XTSI in the System Table and places this external address in the TSI, makes the task status "delay", assigns the task identification, assigns the schedule table entry of 20, and places the symbolic device address of the SYSIN/SYSOUT terminal in the TSI.

30. Task Initiation calls Rescheduling to place the TSI in the Inactive Lists in ready status, and returns through Task Initiation to TCS.

31. TCS calls Queue-GQE-on-TSI to place the asynchronous (this interrupt does not meet the Synchronous criteria) I/O interrupt on the tasks' TSI.

32. Queue GQE on TSI calls Rescheduling to move the TSI from the Inactive List to the Eligible List, and then returns to TCS.

33. TCS initializes the TCT and places the TCT pointer in the Device Group Table. TCS sets the RTAM flag, and calls Supervisor Core Release to release the buffer (obtained in step 11) core.

34. TCS calls Reverse Pathfinding to release the device path, then TCS exits to the CIP.

35. CIP calls Dequeue to logically disconnect the GQE from the channel interrupt queue.

36. CIP calls Set Suppress Flag to reset the flag originally set in step 24 and thus enables processing of the Channel Interrupt Queue when subsequent GQEs are placed on it.

37. CIP calls Supervisor Core Release to release the save area obtained when CIP called TCS.

38. CIP at this point has finished processing the attention interruption and exits to the Queue Scanner.

39. The Queue Scanner searches for any pending GQEs. For this example, it will be assumed that there is none. Finding no work, the Queue Scanner exits to the Internal Scheduler.

40. The Internal Scheduler finds the new task and submits it to the Entrance Criteria module to determine if the task can be brought into main storage. In this case we will assume that there is room and the task is moved to the Dispatchable list. In order for the task to execute, the Internal Scheduler must first cause the XTSI page to be brought in. For this newly created task this will be the skeleton XTSI. The paging activity of the supervisor is omitted here and is shown in another flow example. (See Paging.) The Internal Scheduler also reorders the Dispatchable list so the Dispatcher is presented with the proper task to place in execution.

41. When the Internal Scheduler is finished, it exits to the Dispatcher. The Dispatcher selects the first "ready" task from the Dispatchable list and proceeds to place it in execution.

42. The Dispatcher now wishes to determine if there is a task interruption pending for this task. Task Interrupt

Control (TIC) is called to make this determination.

43. TIC provides an important link between the Resident Supervisor and the Task Monitor. Since there is a pending task interrupt, the function of TIC here is to make sure that when the task is placed in execution, it is started at the appropriate interruption processing routine in the Task Monitor. In order to do this it must set up the correct data in the first double word (PSW location) in the XTSI. This data is obtained from the new Asynchronous I/O Virtual PSW in the ISA.

44. TIC invokes Locate Page to find the main storage address of the ISA.

45. TIC translates the VPSW information in the ISA into extended PSW format, places it in the XTSI, and sets the TSI interruption mask field. TIC also copies information from the GQE into the ISA.

46. Supervisor Core Release is then called to release the space used by the GQE.

47. Task Interrupt Control then returns to the Dispatcher.

48. The Dispatcher now sets the new task in execution by loading the PSW and registers from the XTSI. The instruction counter in the PSW points to the Task Monitor's Asynchronous I/O Interrupt entry point. (The page pointed to by the PSW will now be paged in if it is not already in main storage.)

49. The Task Monitor Asynchronous I/O Interrupt Processor discovers that it is processing an initial attention interruption by noting that the Task Initiation Complete Flag in the ISA is not on. It thus knows that it must set up a task initialization sequence.

50. Queue Linkage Entry (QLE) is the routine used by the Task Monitor to create an element of work (i.e., a Queue Entry) to be done later by the Task Monitor. QLE is analogous in function to the Resident Supervisor's Enqueue routine. The Queue Entry is created and an entry in the Task Monitor Interrupt Table, called a Request Entry, is activated. A Request Entry describes the routine that will process the work represented by the queue entry. The Request Entry in this case will contain a control block that identifies a Command System routine that will supervise task initiation.

51. Control is returned to the Task Monitor Asynchronous I/O Interrupt Processor, which in turn exits to the Scanner/Dispatcher.

52. The Task Monitor Scanner/Dispatcher is analogous to the Resident Supervisor's Queue Scanner. The Scanner/Dispatcher scans the Interrupt Table looking for a Request Entry with work that can be processed, and dispatches the appropriate routine when such a Request Entry is found. In this case it finds an active Request Entry for the Initial Attention Interrupt routine and calls that routine.

53. The Initial Attention Interrupt Processor performs initialization functions for conversational tasks. After setting a flag in Task Common to indicate that the task is conversational, IAIP makes an explicit CALL to load and transfer control to the Virtual Memory Task Initialization routine.

54. Virtual Memory Task Initiation proceeds with initialization functions for the Command System.

The SDAT is scanned for all devices that carry public volumes. As each public device is found, VMTI uses the ADDEV SVC to add each device to the Task Symbolic Device List. This list is used by the Resident Supervisor to determine on which devices the task is allowed to request I/O operations.

The Data Control Block (DCB) and Job File Control Block (JFCB) for the Catalog are included in each task's Initial Virtual Memory. VMTI opens a DCB for SYSSVCT -- a VI dataset containing pointers to USERCATs. It then issues FINDJFCB for SYSSVCT, opens the two SYSCAT DCBs, DDEFs USERCAT for the user, and issues FINDJFCB for USERCAT.

As part of its processing, VMTI supervises the creation of the JFCBs necessary for each task to operate in the system.

There is a special entry point in the DDEF routine that allows privileged routines to specify a User ID other than the User ID associated with the task. Thus, system data sets, which are cataloged under the user ID of TSS*****, can be accessed by privileged programs without having to first issue a SHARE command. VMTI then issues another call to DDEF to create the JFCB for the System User (SYSUSE) data set. This data set contains a list of all legal TSS/360 users and describes their attributes. Part of the LOGON command functions is to open this VISAM data set.

VMTI also issues the DDEF macro instruction to create the JFCB for the System Accounting Table. This table contains the accumulated main storage time used by tasks. At the completion of each task, this data set is opened by the LOGOFF command and is then updated by the Accounting Routine.

VMTI next issues DDEF macro instructions for the System Macro Library and System Macro Library Index. These data sets are used by the TSS/360 language processors.

VMTI issues its final DDEF to create the System Library (SYSLIB) JFCB. The LIBMAINT routine is then called by VMTI with a pointer to the SYSLIB DCB contained in the task's Initial Virtual Memory. LIBMAINT opens the System Library.

VMTI calls TIME to initialize the task timer and continues the task initialization sequence by calling VMTI II.

55. The function of VMTI II is to complete the initiation process. VMTI II first calls OPEN to open the DCB for the system message file (SYSMLF). FIND is issued to open that member. VMTI II next turns on a flag in New Task Common showing that the task is conversational. It then moves the task ID from the TSI into Task Common and New Task Common and calls LOGON.

56. The LOGON command is the mechanism by which the system validates the user's credentials and performs various initialization functions for the task. LOGON issues an ATTACH SVC, which returns the address of the TCT slot assigned to the task. The TCT contains the address of the buffer where LOGON obtains the parameters which were entered with the user's LOGON command. An initial message is issued via User Prompter to let the user know that his parameters have been received.

In order to analyze the input parameter string, LOGON calls SCAN. SCAN is a set of subroutines used by command programs to isolate and validate input parameters.

LOGON calls SCAN to validate the USERID. Once SCAN is completed, LOGON calls COMMON OPEN to open the SYSUSE data set. LOGON, issues an RCR OPEN macro using the USERID as the key, and calls VISAM Read/Write to read the legal user attributes associated with the USERID. These attributes are each checked and placed in Task Common.

Three conditions must exist before the LOGON processing for this task can be completed. The USERID of the conversational task being logged on must not already have a conversational task in the system, and the user must not have been quit from the system, and sufficient auxiliary storage must be available. Assuming these conditions exist, RCR sets the activity flag in the SYSUSE record indicating that there is now a conversational task for this USERID and increments by one of the total number of tasks currently in the system for this user. In addition, the nonconversational print flag in Task Common is turned off. This flag is used to instruct the LOGOFF routine to issue a PRINT Command to cause the SYSOUT data set to be printed for nonconversational tasks. The USERID is stored in Task Common, and an AUL entry for the task is placed in the UAL table.

RCR RATION macros are then issued to check the user's CPU time and CONNECT time to be sure he has not exceeded his ration.

Once completed, LOGON calls VISAM Read/Write to update SYSUSE. A prompt is issued (via MSGWR) if the user has a password and has defaulted it. Once the password has been verified, all other parameters are validated with system default values for those parameters omitted.

LOGON then sets a flag in Task Common indicating LOGON processing has been successfully completed. By means of the SCHED macro instruction, the task's schedule table entry is updated.

LOGON proceeds to call DDEF to create the JFCB for USERLIB. After creating the JFCB, DDEF calls LIB MAINT to build and open the DCB for the user library (USERLIB). LOGON then returns control to VMTI II after issuing the salutation message.

57. VMTI II determines the SDA for the SYSIN and SYSOUT device from the TCT. Next it sets up an SDAT entry showing the device type and maximum line length of the terminal for the buffer in GATE's PSECT. VMTI II, at this point, constructs a dummy MCB and calls LOGON2. LOGON2 initializes fields in New Task Common and issues OPEN macros for the DCBs of USERLIB

and SYSLIB. Next it issues FIND macros for the SYSPRO and SYSPRD members of USERLIB and SYSLIB, and for SYSMLF in USERLIB (SYSMLF in SYSLIB has already been opened). LOGON2 then creates the combined dictionary by copying the user profile or the system prototype profile and the user and system procedure libraries. It sets up the input and output Character Translation Tables and the Profile Character and Switch Table for GATE. Finally LOGON2 issues the SIR macro instruction to enable the Attention Handler and returns to VMTI II.

58. VAMTI II issues ZLOGON via the OBEY macro. ZLOGON, in USERLIB, is initially a null procedure to which the user can all commands he wishes to be executed at this point. VMTI II calls QLE to queue an entry to the Command analyzer and Executor and then begins a chain of returns which proceeds to VMTI, the Initial Attention Interrupt Processor, and the Task Monitor.

59. When the Command Analyzer and Executor is entered, it determines that the task is conversational and calls GATE via the GTWAR macro instruction to issue an underscore and backspace. The user is now free to enter commands.

NONCONVERSATIONAL PROCESSING

Next, a walk-through of the system operations involved in a nonconversational assembly is presented. The processing of RJE Control Cards is presented in the section on "Remote Job Entry."

At startup time the main system operator is given the opportunity to create the BULKIO task. If he declines at startup time he may create the task at any later time by issuing the ASNBD command. As operands, the operator enters the symbolic device addresses of the various unit record devices he wants used in BULKIO processing. These operands are sent via VSEND to the BULKIO task which constructs a list of S entries in the BULKCOMM table. Each of these entries represents one unit record device and contains space for input and output DCBs plus information concerning the type and availability of the device.

The BULKIO task in conjunction with the Batch Monitor provides the processing needed to create, schedule, and dispatch jobs requiring unit record devices. These jobs are initiated in response to the commands PRINT, PUNCH, RT, and WT. A job is also initiated to read cards but no command is required. The reading of cards is accomplished by loading the cards into the reader and starting the reader. The resulting asynchronous interruption is recognized as a request to read cards and the required job is initiated by the BULKIO task.

The Batch Monitor task causes creation of another Bulk I/O job to list the SYSOUT of the first Bulk I/O job.

The Batch Monitor then causes creation of the assembly task and sends a message to initialize it.

The Language Processor Control (LPC) of the Command System is loaded into the Assembler task's virtual storage when the desired language processor is requested via the LOAD command. LPC performs those functions and relationships common to all language processors and serves as the link between the language processors and the user.

The final command in all cataloged command procedure data sets is LOGOFF. The final exit for the Assembler Task is to the LOGOFF command routine, which in addition to terminating the task, requests another nonconversational Bulk I/O job to list the Assembler task's SYSOUT.

The Bulk I/O jobs to list source listings and printouts of the Assembler task's SYSOUT are subsequently initiated by the Batch Monitor Task when the PRINT command is issued. Because the processing for these jobs is in many ways similar to the read cards job, these jobs are not described in this example.

The relationships among the routines involved in the processing of this read cards job and the significant work they perform are presented in the following step by step description.

1. To initiate the reading of cards the system operator must first ensure that the BULKIO task exists. This task is created by the main operator housekeeping routine at startup time or subsequently by means of the ASNBD command. The operator must also have assigned the reader to the BULKIO

task. This is done either when the task is created or by adding the device to the TSDI of the BULKIO task at some later time by issuing the ASNBD command. If both these conditions have been met, the operator need only load the cards in the reader and start the reader.

2. The asynchronous interruption which occurs when the reader is started is received by the channel interrupt processor of the resident supervisor. The channel interrupt processor recognizes that a task, the BULKIO task, already exists for this device and simply queues a GQE in the asynchronous interrupt queue of the BULKIO task's TSI.

3. When the BULKIO task was created, an ICB was built in BULKCOMM by means of the SAEC macro instruction and enabled by the SIR macro instruction. When Task Monitor goes to service this interruption, it recognizes the existence of the routine specified in the SIR macro instruction and passes control to it.

4. The routine which services the asynchronous interruption is the Input Start routine. It deletes the ICB by means of the DIR macro instruction and calls the Input Service routine to initiate the new job.

5. Input Service reads the first card from the reader. SYSIN data sets such as the one in this example must begin with a LOGON command. Data sets are delimited by a DATASET card. When the LOGON card is read, Input Service calls Input Start at a secondary entry point.

6. Input Start validates the userid and places it in the S entry associated with the reader. A call is then made to the DDEF routine to create a JFCB defining the output data set. Input Start then opens the output data set.

7. Input Start calls Input Service which continues reading cards. Cards are read by means of the GET macro instruction and are placed in the VAM output data set by means of the PUT macro instruction. This operation continues in a loop until the LOGOFF card is read or no more buffering can be performed. When the LOGOFF card is detected, Input Closeout is called.

8. Input Closeout closes the output data set and issues a VSEND to the Batch Monitor task to create a task to process the SYSIN data set just read. Input Closeout returns to Input Service which begins looking for another task. Our example will continue with the processing of the SYSIN data set.

9. The message which Input Closeout sends to the Batch Monitor is an MCB in the form of a Batch Work Queue entry which Input Closeout had previously constructed. The VSEND is queued as an external interruption on the Batch Monitor's (Operator Task's) TSI. When the Batch Monitor task (Operator Task) is placed in execution again by the Dispatcher, Task Interrupt Control will find the interruption pending and will give control to the external interrupt processor which links to the Batch Monitor (operator) task at a point where it will create a BWQ entry.

10. At the point of entry to the Batch Monitor, a BSN is assigned, the Batch Work Queue is opened, and the new entry is written to the queue. The queue is then closed and a flag is set to indicate to the Batch Monitor Processor (Operator) that there is work in the queue.

11. When the Batch Monitor (Operator) task finds the entry in the BWQ it creates a new task to process the SYSIN data set by means of a call to Create TSI. When a task has been created, the Batch Monitor issues a VSEND to the TID of the task just created; the MCB associated with the VSEND contains the necessary information from the BWQ entry. The Batch Monitor then continues creating tasks for each entry in the BWQ. When the BWQ is empty or the maximum number of nonbackground tasks in the system is reached, the Batch Monitor calls Close Common to close the BWQ data set and returns to the Batch Monitor processor.

12. The Batch Monitor processor will initiate all possible Batch and BULKIO jobs (depending on device availability). Once it has done this, it will call the Task Monitor to queue a linkage entry to itself. This will enable the Batch Monitor to repeat the procedure of initiating tasks should any more entries be made in the BWQ between this time and the next time slice the Batch Monitor receives.

13. When the Dispatcher selects this new task for execution the SYSIN data set created by BULKIO is read. The LOGON command is read and the LOGON routine is called to process it.

14. The next command read in this example is the ASM command. When this command is processed, the Assembler will be dynamically loaded into the new task's virtual storage and will begin reading and processing the source statements which follow.

15. Following the source statements to be assembled the user may issue any of several commands. He may choose to execute the just assembled program by issuing the CALL, RUN, or LOAD and CALL commands. For this example we assume that the user wishes to check his assembly for errors and issues the PRINT command.

16. The PRINT command will cause an entry to the BULKIO preprocessor in the Command System. This routine will issue a VSENDR to the Batch Monitor task to create a BWQ entry for the print job.

17. When this is done the Batch Monitor reactivates the task which entered the PRINT command, the SYSIN task, via a VSEND. This task may now issue more commands since the printing of its assembly listing will be done by another task, the BULKIO task. Assume that this task issues LOGOFF and leaves the system.

18. When the Batch Monitor (Operator) task receives another time slice, it will find the newly created BWQ entry and will initiate it. Unlike the assembly task, no new task is created for the printing of the assembly listing since it resides on public storage. This job is performed by the BULKIO task. The Batch Monitor calls its BULKIO subroutine which calls OPEN to open the BWQ data set.

19. When the data set is open and the new entry is found, the list of S entries in BULKCOMM will be searched for an available printer. When one is found, it is marked unavailable and the information in the BWQ entry is moved to the S entry. The Batch Monitor then continues searching for more tasks to initiate.

20. When the BULKIO task is again placed in execution, its Master Services routine calls Output Service. Output Service enters a loop in whch it first issued a GET to retrieve the record from the VAM data set and then issues an MSAM PUT to write the record to the printer. This process continues until the end of the VAM data set is reached at which time MSAM FINISH is issued, a VSEND is sent to the Batch Monitor informing it that the job is complete, and the operator is also informed by means of the WTO macro instruction.

REMOTE JOB ENTRY

Remote Job Entry (RJE) provides the user at a remote terminal, all the facilities of BULKIO II and nonconversational task execution. Except for control card processing, the logic is generally common for RJE and nonconversational users. The remote station has a printer, and a card reader with which to initiate batch jobs in the same manner as the central installation. All input from remote stations is compatible with the local card reader, with the exception of the RJE control cards.

The control card processing for RJE is described in the system logic flow that follows. For nonconversational processing generally, see the section on "Nonconversational Processing."

RJE User Overview

The ASNBD command is issued to enable a line for the remote station. A JOINRJE command, issued by the system manager or administrator, verifies the station ID and adds it to the Validation Data Set partition of the Acknowledgment Data Set. QUITRJE removes the station ID from this data set.

An operand of the JOIN and PRINT commands allows the user to print his job at a station other than his own. The DIRECT command is used by the system operator to route the RJE output to a local printer or another RJE station.

The control cards entered from the remote station are the RJSTART card - identifying the station; the RJEND card - detaching the station from the system; and the CONTINUE card - which requests the system to continue, cancel, or rerun a transmission to the remote printer.

RJE Control Card Processing

If the line to the RJE station is not dedicated, the user must communicate with the Central Operator to enable the line. The operator will issue the ASNBD command with an SDA as the operand and the system flow will proceed as shown in these numbered steps (keyed to Figures 100 through 104).

1. The ASNBD command will cause a VSEND through the Resident Supervisor to the BULKIO Initialization Routine.

2. BULKIO Initialization initializes the S-entries for the input/output func-

tions of the RJE station, calls SIR to establish BULKIO Input Start as the processor for asynchronous interrupts, and issues the RJELC macro.

3. The RJELC macro expands to issue the ENABLE SVC, which is fielded in the Resident Supervisor by the Interrupt Stacker and passed to the SVC Queue Processor.

4. The SVC Queue Processor determines the task is authorized to issue the SVC, and calls RJE Line Control (RJELC).

5. RJELC determines that the function to be performed is ENABLE.

6. RJELC calls Pathfinding for the physical I/O path.

7. RJELC calls HIO to insure that the line is inactive. The completion of this HIO will cause an asynchronous interrupt, which will be passed back to RJELC through the Channel Interrupt Processor (CIP) and RJE Asynchronous Interrupt Subprocessor (RJEAIS). (This interrupt is not shown in Figure 99.)

8. SIO is called to start I/O on a Disable/Set Mode/Enable channel program.

9. Reverse Pathfinding is called to release the device path.

10. RJELC returns to the SVC Queue Processor, then returns normally through the Queue Scanner, Internal Scheduler, and Dispatcher, to BULKIO Initialization. (We assume no interrupts pending.)

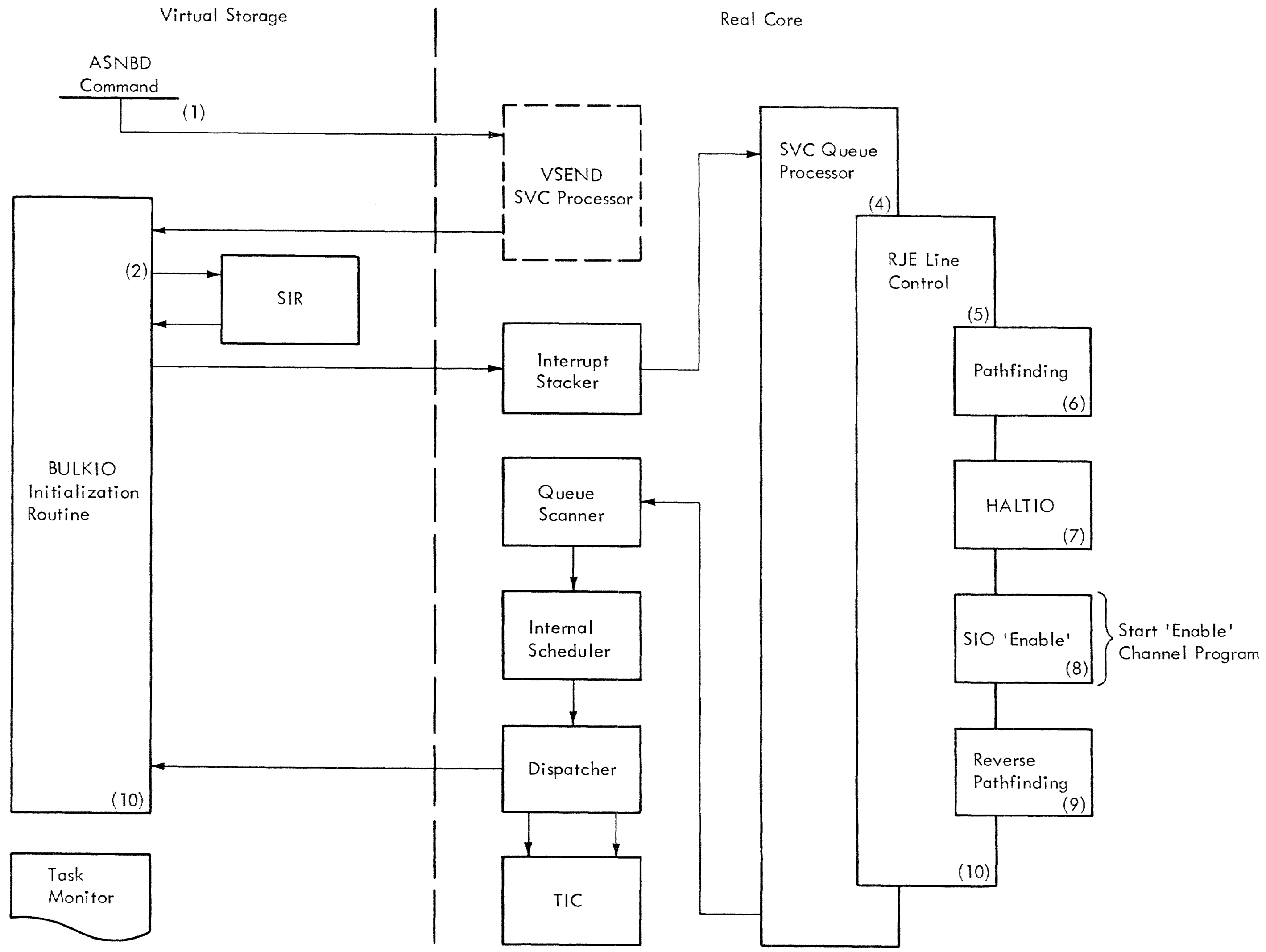


Figure 99. Enabling RJE Line

At completion of the ENABLE (immediately on a dedicated line - or by dial-up on a switched line), an asynchronous interrupt occurs and the processing continues in order to prepare the line for input from the Card Reader. (See Figure 100.)

11. The asynchronous interrupt (channel end, device end) is fielded by the Interrupt Stacker which creates the GQE and places it on the Channel Interrupt Queue.

12. The Queue Scanner finds the GQE on the Channel Interrupt Queue and invokes the CIP.

13. The CIP Calls Set Suppress Flag (SSF) to prevent another CPU from processing this queue.

14. Reverse Pathfinding is called to convert the hardware address to an SDA.

15. The CIP calls RJEAIS, which sets the code for priming the line, for bypassing HALTIO and the clearing of the device's scan table entry, then calls RJELC.

16. RJELC gets the SDA of the device and calls Pathfinding to assign the hardware path.

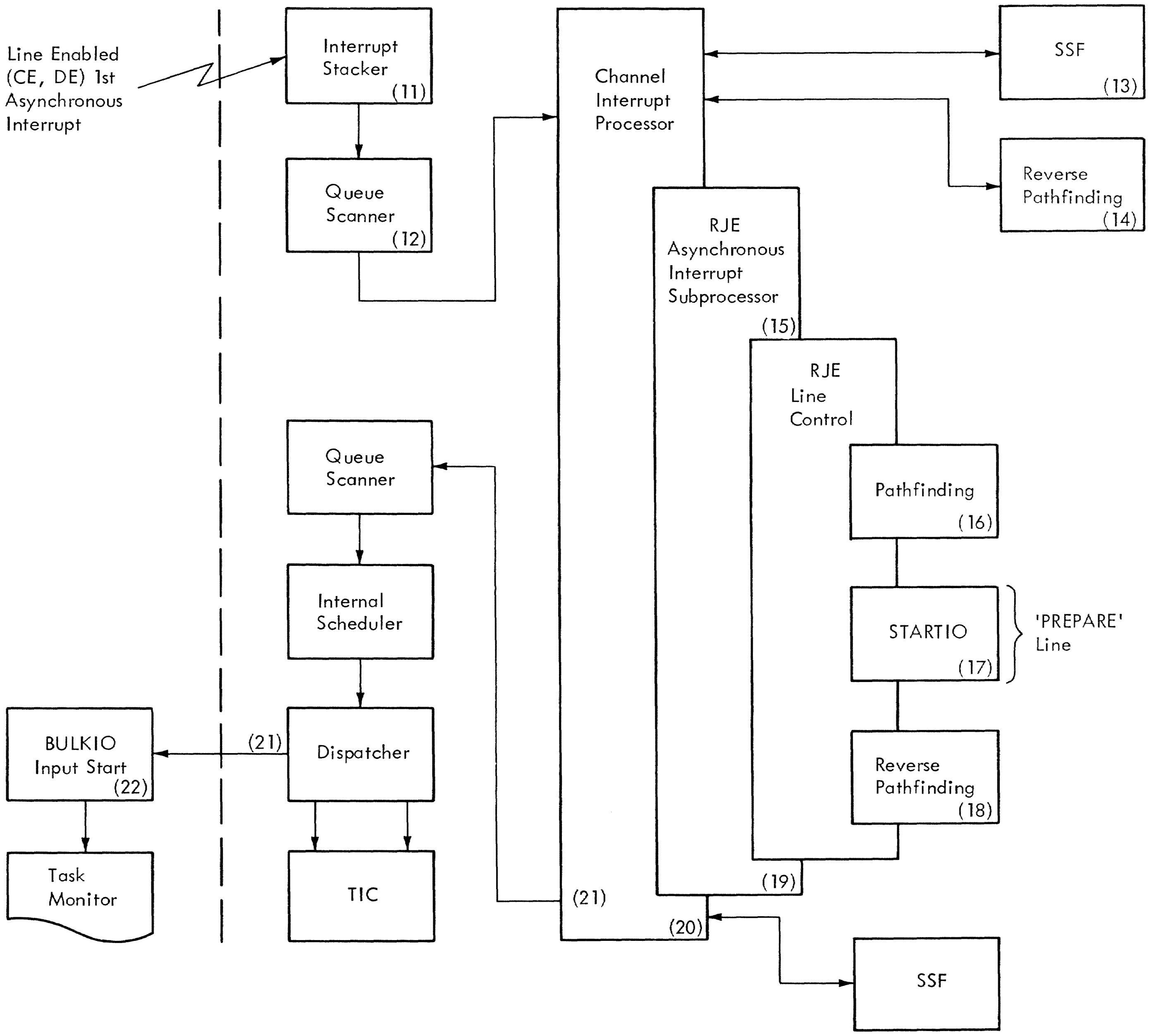


Figure 100. RJE Line Preparation

17. RJELC gets the CAW for priming the line and calls STARTIO to start the channel program for a Prepare/Read ENQ/Write ACK0 sequence.

18. RJELC calls Reverse Pathfinding to release the hardware path and then returns to RJEAIS.

19. RJEAIS restores the CIP registers (this example assumes no errors) and returns to the CIP.

20. The CIP resets the suppress flag.

21. A normal exit is made through the Queue Scanner, Internal Scheduler, Dispatcher, Task Interrupt Control, and back to the Dispatcher which will pass control to BULKIO Input Start via the Task Monitor Asynchronous I/O Interrupt Processor and Scanner/Dispatcher mechanisms.

22. BULKIO Input Start recognizes this as the first asynchronous interrupt for the RJE task, and initializes a switch which will allow the next asynchronous interrupt to be passed to BULKIO Input Service. Control is passed to the Task Monitor Scanner Dispatcher.

When the Card Reader is readied (place cards in the hopper and hit START.), an ENQ character is transmitted to the CPU. Establishment of character phase at the 2701 breaks the Prepare command, the ENQ is read, and an ACK0 is returned to the 2780 - initiating a 48-second time-out period. The channel program then terminates with a channel end/device end. When this second asynchronous interrupt is received, an MSAM Read Cards job will be initiated as follows (see Figure 101).

23. The interrupt is fielded and passed to the CIP as above. RJEAIS determines that the prime was completed normally and returns to the CIP, which passes the interrupt to BULKIO Input Start through standard Resident Supervisor linkages and the Task Monitor's Asynchronous I/O Interrupt Processor and Scanner/Dispatcher mechanisms.

24. Since the switch is initialized (see step 22), Input Start calls Input Service, which will control the MSAM card reading through the EOF.

25. Input Service issues a GET macro instruction which calls DOMSAM to initialize the Read.

26. DOMSAM calls MSAM Read/Write to set the IORCB fields and execute the IOCAL SVC, which will be handled by the Resident Supervisor.

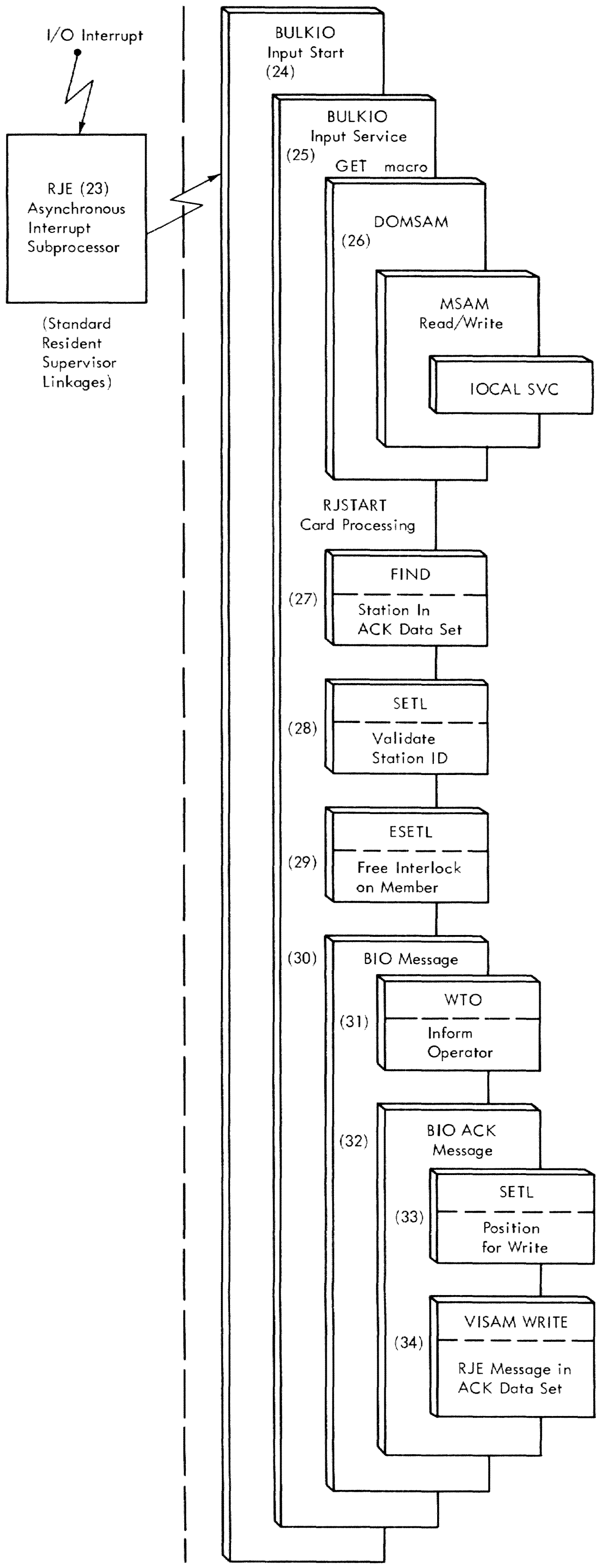


Figure 101. RJE GET Macro and RJSTART Card Processing

27. BULKIO Input Service continues issuing GETs until a return code of zero is recognized, and then searches for the RJSTART card. Cards are flushed through the reader until the RJSTART card is recognized. When Input Service recognizes RJSTART, the FIND macro is issued to locate the VALIDSTA member of the Acknowledgment Data Set.

28. A SETL macro is issued to validate the station ID, and

29. The ESETL macro is issued to free the interlock on the member.

30. BULKIO Message is passed a code indicating the System Operator and the RJE station are both to be informed that the RJSTART card has been received and validated.

31. WTO is called to inform the System Operator.

32. BULKIO ACK Message is called to prepare for the Write of the RJE acknowledgement message, by calling

33. SETL to position to the end of the last message in the Acknowledgement Data Set, and

34. VISAM Read/Write to write the message into this data set.

The RJE job is now initiated. BULKIO Input Service will control the card input, issuing a series of GET macros until EOF is recognized. An overview of LOGON card and LOGOFF card processing is shown in Figure 102.

When an EOF occurs, the RJELC routine in the Resident Supervisor is called, preparing the line with the PREPARE/READ ENQ/WRITE ACK0 channel program. Processing will subsequently be resumed at step 23 above, if the station has further card input.

The end of the terminal session for RJE is signalled by the RJEND card. Processing is as shown in Figure 103, and as follows.

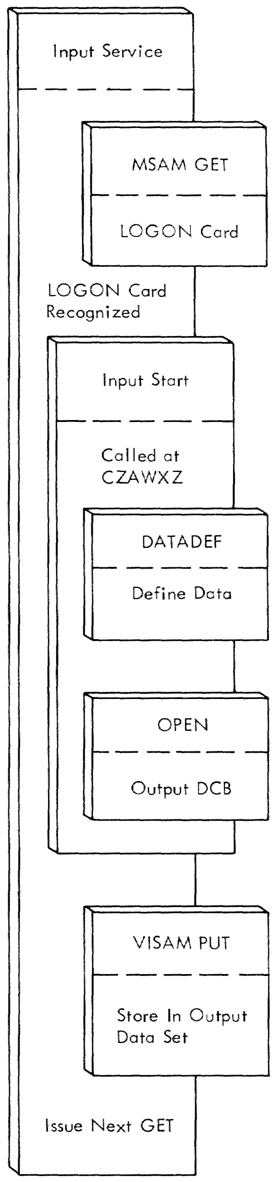
35. Input Service issues a GET macro which reads in the RJEND card. Input Service recognizes RJEND, sets the End Card flag, and issues the GET macro again.

36. A Unit Exception from the Card Reader should terminate the Read Cards job with an EOF condition.

37. Input Service recognizes EOF and calls MSAM Finish to perform housekeeping for the Card Reader.

38. Input Service cleans up the input and output S-entries, and the SDA if the line is not dedicated. A Disable/Enable (RJELC) SVC is issued to re-enable the line, completing the input portion of the RJE task.

Output for RJE, other than from the Acknowledgement Data Set, is accomplished through BIO Output Service in essentially the same manner as for any batch job. A VISAM GET on the user's data set locates the record, an MSAM PUT on the device data set locates space for the output record, and the VAM record is moved into the MSAM record with the required formatting. A HALTIO is issued terminating the PREPARE on the transmission line. A STARTIO is issued on the Write channel program as for any other IORCB. At the end of transmission to the RJE station RJELC is called to again PREPARE the line for input.



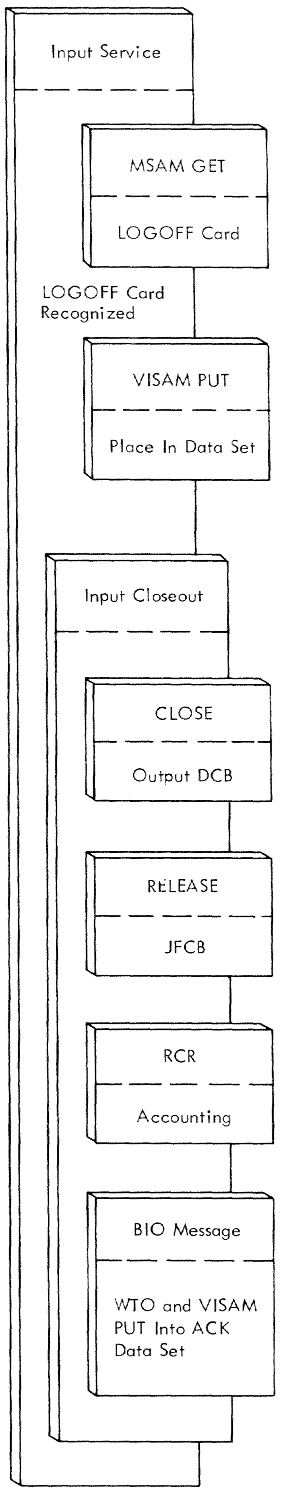


Figure 102. Processing of LOGON and LOGOFF Cards

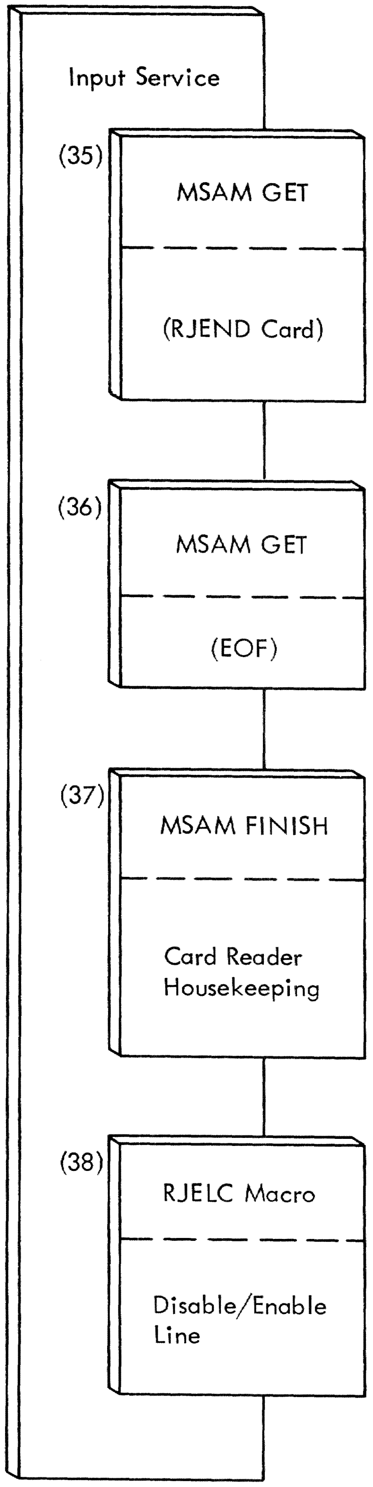


Figure 103. Termination of RJE Input Card Stream

PART III: LANGUAGE PROCESSORS AND AUXILIARY PROGRAMS

The purpose of the TSS/360 Assembler is to produce from source programs written in the Assembler language, machine language programs in a format suitable for operation under the Time Sharing System. The Assembler produces standard output and optional output.

Standard Output

Program Module Dictionary
Hexadecimal Text
External Name List

Optional Output

Internal Symbol Dictionary
Source program listing
Object program listing
Cross-reference listing
Symbol Table listing
Internal Symbol Dictionary listing
Program Module Dictionary listing

The Assembler allows source programs to be submitted in either conversational or nonconversational mode.

In the conversational mode, the assembler produces some syntax diagnostics line by line as the source program is submitted. The user may resubmit an erroneous statement or restart the assembly process. After the source program has been completely submitted, the assembler produces additional syntax diagnostics and allows the user a choice of continuing, correcting, or aborting the assembly.

In nonconversational mode, this interaction is not possible and diagnostics are produced only with the optional output listing data set.

For further information concerning use of the assembler and output from the assembler, see Assembler Programmer's Guide.

System Environment

The initial request by the user to secure the Assembler is processed by the Command Language Interpreter (CLI), which calls the Language Processor Control (LPC). The Language Processor Control calls the Assembler, which is a part of the task's Initial Virtual Memory.

The Assembler makes use of:

- Language Processor Control to supply user program source statements.
- Symbolic Library Service Routines to secure macro definitions and COPY parcels.
- Data Management services to process output list data sets and output modules.

The Assembler is called by and exits to Language Processor Control (LPC). The GETLINE function of LPC receives source language statements from the SYSIN terminal or data set and directs them to the Assembler for processing. Conversely, the symbolic listing and diagnostic messages are routed from the Assembler to the same terminal or to the task's SYSOUT data set via the PUTDIAG function of LPC.

To process COPY statements and macro instructions not defined in the user's source program, the Assembler searches macro libraries. The Assembler always searches the system macro library but may precede this with a search of a user macro library if the user has requested this option. The symbolic library service routines are used to accomplish this function.

The source program listing, Program Module Dictionary listing, cross-reference listing, Symbol Table listing, object program listing, and the Internal Symbol Dictionary listing are created if requested using Virtual Access Methods.

Virtual storage dynamically acquired by the Assembler is secured by the GETMAIN macro and released by the FREEMAIN macro.

Organization Of The Assembler

As shown in Figure 104 the Assembler is divided into four major components or phases plus an Assembler Control module which interfaces with LPC.

The principal function of the Assembler is to translate machine instruction statements which have been written in a symbolic language which has mnemonic significance to the programmer into the numeric language of the computer. This is accomplished principally by allowing alphameric symbols of the programmer's choice to represent the numerically addressed storage locations in the computer. The Assembler's primary task is to determine which symbols have been defined according to the rules of the As-

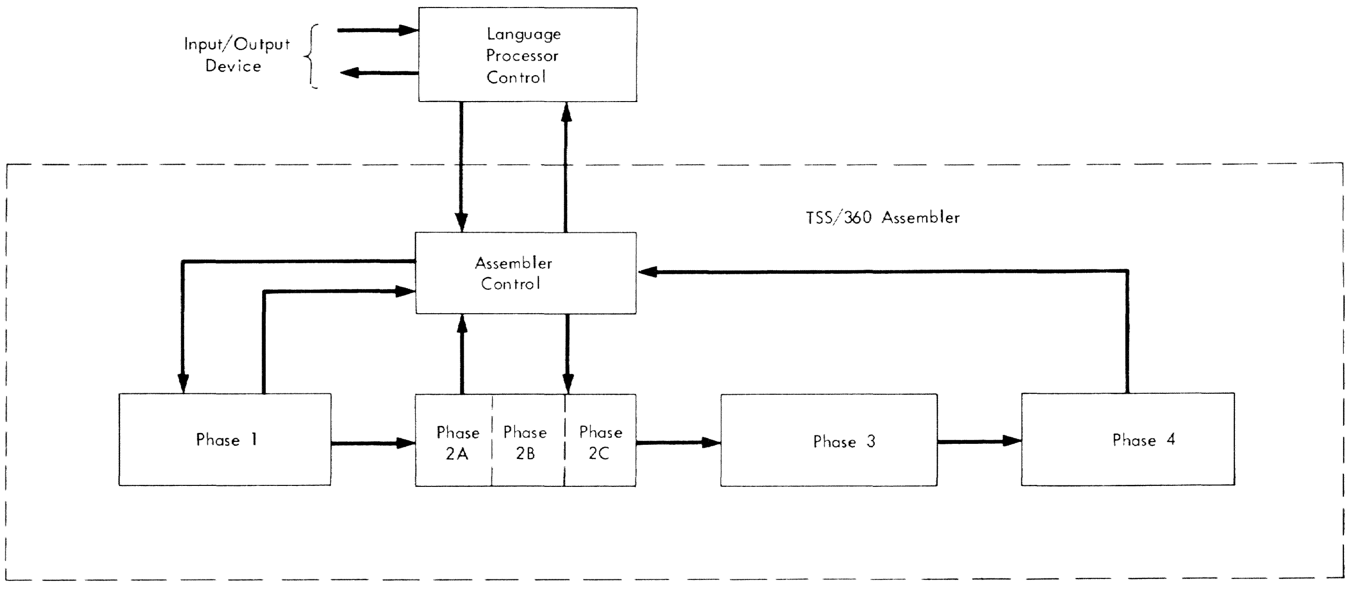


Figure 104. TSS/360 Assembler Interfaces with LPC

sembler language, assign a corresponding machine language value to the symbol, and to substitute the machine language value whenever the symbol is used in the construction of a machine language instruction.

In addition to translating machine instruction statements, the Assembler also processes Assembler instruction statements. Assembler instruction statements are requests to the Assembler to perform operations during the assembly. Machine instructions may not be generated in the assembled program as a result of assembler instruction statements. The functions provided by Assembler instruction statements are:

- Symbol Definition (Equate Symbol instruction).
- Data Definition (e.g., Define Constant instruction).
- Program Sectioning and Linking (e.g., Start Assembly instruction and Identify Entry-Point Symbol instruction).
- Base register Specification (e.g., Use Base Address Register instruction).
- Listing Control (e.g., Identify Assembly Output instruction).
- Program Control (e.g., Input Format Control instruction).

The Assembler also processes macro language and conditional assembly statements. By means of such statements, sequences of machine or assembler statements may be dynamically generated at specific points in the input to the Assembler.

The macro language permits macro instruction parameters to be identified by either positional or keyword notation. The macro language also permits a macro instruction to be utilized within a macro definition. For example, this permits a macro definition to recursively call itself. The TSS/360 Assembler macro language also permits a macro instruction to temporarily redefine a machine instruction mnemonic.

The conditional assembly statements contained in the TSS/360 Assembly Language permit the specification of arithmetic, logical and character variable symbols. Such variable symbols may be subscripted and local or global in nature. The Assembler assigns attributes such as type and length to ordinary symbols and macro instruction operands. These attributes may be referred to in conditional assembly statements. The lanugage provides for conditional and absolute branch statements and for user-controlled generation of error statements referencing macro instructions.

The method and order by which the Assembler implements these features is described in general terms below.

Syntax Analysis

In order for the Assembler to interpret a statement without ambiguity, the programmer must follow certain rules in writing the source statement with regard to separation of fields, placement of symbols and delimiters, proper choice of mnemonic operation codes, and the like. The somewhat mechanical inspection of the source statement to determine whether the rules have been observed is generally called "syntax analysis", and is the first operation performed by the Assembler on each statement. The analysis is achieved by a character-by-character scan of the incoming statement. Since this method of analysis is time consuming, the Assembler usually converts the information which has been extracted from the statement into a more convenient internal form and places it in one of the various tables which are kept for this purpose. The principal tables include a table which contains a condensed summary of each statement (the Logical Order File or LOF), and a table which contains the name and characteristics of each programmer-defined symbol (the symbol table or dictionary).

The definition of a symbol must be known to the Assembler before it can construct a machine instruction which requires the value of the symbol. However, the rules of the TSS/360 Assembler language permit a symbol to be referred to before it is defined. If the Assembler attempted to construct the machine language program concurrently with syntax analysis it would find itself frequently unable to do so for lack of information about symbols which had not yet been encountered. For this reason, construction of machine instructions is postponed until the entire source program has been syntactically analyzed and all symbols have been entered into the dictionary.

Macro Instruction Processing

A macro instruction is the invocation of a fully or partially predefined sequence of source statements through the use of a mnemonic operation code which has been declared for that purpose. The mnemonics of macro operations may be specified by the programmer himself, along with the sequence of statements which the operation represents, or, failing that, in the table-of-contents of a library of predefined macro operations which has been made available to the Assembler program. In either case, the Assembler's dictionary of symbols cannot be considered complete until the sequence of statements represented by macro instructions have been syntactically analyzed.

Macro instruction sequences may be processed either (1) prior to processing user's statements (by first searching the source program only for macro instructions and then merging their expansion into the user's statements); (2) concurrently with the user's statements (by incorporating the expansion into the program as encountered); or (3) after the user's statements. The first method is used by other System/360 assemblers. The TSS/360 Assembler, however, is committed to producing diagnostic messages of syntax errors as each source statement is submitted. This is for the benefit of a conversational user, and this requirement forces the Assembler to process the user's statements first, as received. Because system macros require the attributes of the user's symbols, and because there is no ordering rule (requiring the user's symbols to precede system macro calls), expansion of macros concurrently with the user's statement is also ruled out. Macros are therefore expanded by a second phase (Phase IIA) of the Assembler after the user's statements have been syntactically analyzed.

Expansion of source statements contained in a macro definition involves the recognition of a class of symbols (variable symbols and parameters) which are independent of the symbols used in machine language statements. Since these symbols are used only temporarily (and may be used repetitively with different meanings) it is to the assembler's advantage to maintain them in a "dictionary" which is separate from the one used for machine language symbols.

In addition, the expansion of one macro instruction frequently results in the invocation of some inner or nested macro instruction. The rules of the macro language are such that it is desirable for the assembler to maintain a separate dictionary for each nested macro level. The rules of the macro language are also such that once the instructions have been generated for a given macro level, the dictionary for that level is no longer required and can be discarded, since symbols defined at each level are independent. For this reason, macro level dictionaries are constructed linearly in working storage, and maintained by push-down-stack logic.

Since the definitions of system macros are not part of the original user's source language input, they must be retrieved from a library and added to the source program at the appropriate time. Since library retrieval is time consuming, it is desirable to avoid retrieving a macro unnecessarily and to retrieve each definition only once. This is achieved by performing library retrieval during the second phase (Phase IIA) of the Assembler. At this time those nested macro calls that are to be bypassed because of conditional assembly techniques are discarded, thus preventing their definitions from being unnecessarily retrieved. The conditional assembly instructions defined by the Assembler language allow source statements to be bypassed or included in the program. Moreover, a record is kept (in a special dictionary of macro names) whenever a definition is first retrieved. The definition is condensed into the internal form common to all statements and need not be retrieved again should the macro instruction be reinvoked. This technique prevents multiple retrievals of the same definition.

Assignment of Location Counter Values

Once the additional statements generated by macro instructions have been incorporated into the source program, all possible and potential definitions of symbols are present in the dictionary. Before machine instruction synthesis can begin, however, the (relative) machine address which each symbol represents must be determined. The value of the machine address is arrived at by maintaining a location counter for each control section in the assembly. The counter is set to zero initially and is increased at each statement by the number of bytes of machine storage represented (or bypassed) by the preceding instruction, constant, or storage reservation. Since macro instructions may generate instructions, constants, and storage reservations, the location counter cannot be assigned until macros have been expanded. In those assemblers which expand macros first, the location counter can be assigned during syntax analysis. Since the TSS/360 Assembler defers macro expansion until Phase IIA (for the reasons previously noted), location counter assignment is also deferred. To limit paging and to increase ease of maintenance, Phase IIA is limited solely to macro expansion activity, and a separate phase, Phase IIB, is used to perform the location counter assignment. As a byproduct of its principal activity, Phase IIB also resolves expressions which are dependent upon location counter values, and collects literal constants into literal pools and assigns location counter values to the literal constants.

Program Reordering

It is a requirement of TSS/360 object program modules that, to facilitate loading, all text and relocation information pertaining to a given control section be grouped contiguously in the object module. It is also a language rule that control sections may be written discontinously in the source program, and that certain statements in the language (USING, DROP, LTORG, PRINT, etc.) have effect over a range of source statements irrespective of the number of different control sections represented by that range of statements.

The TSS/360 Assembler is therefore faced with a reordering requirement. It must collect the scattered portions of a given control section without losing the effect of certain statements that are control section independent. It is the function of Phase IIC to determine whether a control section has been broken into discontinuities, and to prepare for each such break a table summarizing the effects of those statements which are independent of control section order. This analysis enables the machine instruction synthesis phase (Phase III) to collect the portions of a given control section and produce contiguous output text in the program module.

Graphically, Phase IIC would transform a sample source program containing the following USING statements and control section discontinuities from:

```
┌──────────────────────────────┐
|Section 1                     |   USING-1
├──────────────────────────────┤
|Section 2                     |
├──────────────────────────────┤   USING-2
|Section 1                     |
├──────────────────────────────┤
|Section 3                     |
├──────────────────────────────┤   USING-3
|Section 2                     |
└──────────────────────────────┘
```

to:

```
┌──────────────────────────────┐
|Section 1                     |   USING-1
|                              |
|                              |
|                              |   USING-2
|                              |
├──────────────────────────────┤
|Section 2                     |   USING-1
|                              |   USING-2
|                              |
|                              |   USING-3
├──────────────────────────────┤
|Section 3                     |   USING-2
|                              |   USING-3
└──────────────────────────────┘
```

Machine Instruction Synthesis

When the reordering requirements have been resolved, the Assembler is ready to begin the construction of machine language instructions from their source language equivalents. Phase III performs this synthesis, working from a list of control sections in such a way that each control section, however discontinously written, produces contiguous output text and relocation information for the Dynamic Loader. An expression evaluation routine, using information stored in the symbol dictionary, resolves each machine instruction operand to either a relocatable or absolute value, and the appropriate text and relocation information is entered into the object module. Source and object listings are a byproduct of this phase.

Postprocessing

When the assembly is complete and the object module has been produced, a series of postprocessing routines may be called to operate upon the dictionary and other information left by preceding phases to produce sorted listings of the symbol dictionary, cross references to symbols, and analytical printouts of the various output modules. For convenience these routines are collected into Phase IV of the Assembler.

ASSEMBLER FUNCTIONS

A brief description of each phase is given below. The Assembler Control module is described last.

PHASE I

Phase I accomplishes the following:

- Copies the source language line supplied by LPC into internal working storage.
- Maintains the sequence of the source language lines through a series of pointers.
- Establishes a partially encoded version of the source language statements to establish the logical order of assembly.
- In conversational mode, scans all statement operands and produces diagnostic messages. In batch mode, statement operands are not scanned until Phase III.
- Enters skeletal definitions for all symbols in a dictionary.
- Processes macro definitions defined in the user's source program.
- Retains statements skipped by a conditional branch instruction (i.e., AIF) in the sequential order, but does not analyze or process then unless the logical order includes them later.
- Adds statements generated by unconditional branch statements (i.e., AGO) not defined in macro definitions.
- Keeps a record of all macro instructions, global symbol declarations (e.g. GBLA), SET instructions involving global symbols, section name changes, and PRINT, USING, DROP, LTORG, and ENTRY instructions. Each of these instructions is processed further in later phases.

When the END statement is processed, control is immediately transferred to Phase IIA. If a source file is derived from a data set and has no END statement, an end-of-file indication causes an END statement to be generated, resulting in transfer to Phase IIA.

PHASE II

Phase IIA accomplishes the following:

- Expands all macro instructions.
- In conversational mode, looks for undefined symbols and gives an error indication if found.

At the end of Phase IIA control is transferred back to the LPC. In batch mode, LPC returns control immediately to Phase IIB. In conversational mode, the user may stop, correct, or continue the assembly at this point.

Phase IIB accomplishes the following:

- Processes literals and enters them in the symbol table.
- Processes all Equate [EQU] and Define Constant [DC] statements not previously resolved.
- Computes the location counter value for all symbols.

Phase IIC accomplishes the following:

- Prepares tables summarizing the USING status of all registers. Entries are made in these tables at each occurrence of a control section, USING, or DROP instruction so that the program may be processed in control section order during Phase III.
- Associates the name of each ENTRY instruction operand with the proper control section.

PHASE III

Phase III produces the following outputs, in order by control section:

- Machine language texts (left in virtual storage).

- Program Module Dictionary and list of external symbols (left in virtual storage).

- Optional listing data set, consisting of source statements, if selected, and object program, if selected.

PHASE IV

Phase IV produces the following outputs at the user's option:

- Listing data sets:

  Cross-reference index
  Sorted symbol table
  Internal Symbol Dictionary
  Program Module Dictionary

- An optional Internal Symbol Dictionary for the Program Checkout Subsystem (left in virtual storage).

Assembler Control Routine

The Assembler Control Routine acts as an interface with LPC. The Assembler has three entry points from LPC. Each entry point is to a location in the Assembler Control Routine from which control is transferred to the Assembler location where the function is accomplished. Similarly, the two exits from Assembler to LPC are also via the Assembler Control Routine.

The three entry points to the Assembler Control Routine are: (1) Phase I Control (Initiation), (2) Phase IIB Control (Continuation), and (3) abnormal termination (Early-End).

The two entry points of LPC are: (1) when the next line is desired and (2) when a diagnostic message is to be printed.

The user informs LPC that an assembly is requested through the Command Language. LPC then solicits the necessary operating parameters and enters the Assembler at Phase I Control for initialization. Then the Assembler enters LPC to obtain the first source statement and LPC returns with the statement. The Assembler processes the source statement and enters LPC for the next statement. In an error-free assembly, this process is continued until and END statement is read, at which time entry is made from Phase I Control directly to Phase IIA Control.

Upon completion of Phase IIA, control is transferred to LPC. If the assembly is in conversational mode, LPC queries the terminal user whether to continue with the assembly, correct the source program and restart, or to terminate the assembly.

If the user wishes to continue, Phase IIB Control is entered and the Assembler proceeds to completion without further conversational interaction.

If the user makes corrections and wishes to restart, LPC reenters Phase I Control to restart the assembly.

If user wishes to stop assembly, LPC enters the Assembler Control Routine at the early-end entry, which releases all dynamically acquired virtual storage and returns the call.

The processing described above is altered when LPC determines that a source line has been corrected, or the Assembler discovers a source statement error.

When the Assembler discovers an error in conversational mode, it calls LPC with a diagnostic message, and LPC transmits the message to SYSOUT. LPC then returns and Assembler again calls LPC for the next

If LPC determines that a source line has been corrected, it enters Phase I Control with a special return code and the lowest line number to which corrections have been made. If the line number which LPC returns is equal to that of the last statement processed, the Assembler processes the corrected statement and then requests the next source statement from LPC. In general if the line number is less than that of the last statement processed, the Assembler reinitializes itself and starts over again by requesting the first source statement from LPC.

If the job is to be terminated abnormally, the early-end entry of the Assembler Control Routine is called to release working storage in virtual storage. Return is then made to LPC.

The purpose of the FORTRAN IV compiler is to produce object programs, for execution under TSS/360, from source programs written in the FORTRAN language, as described in IBM System/360 Time Sharing System: IBM FORTRAN IV.

The compiler produces an object program module consisting of a Program Module Dictionary (PMD), an optional Internal Symbol Dictionary (ISD), text, and a list of external names.

In addition to the production of executable programs, the compiler also detects and gives notification of source program errors, and produces various optional documentation describing the object program. The compiler produces the following documentation:

- A listing of the source program.
- An object program storage map giving the storage layout of the object program.
- A list of source program symbols and their storage assignments.
- A cross-reference listing relating symbols and statement numbers to the source line numbers of the statements in which they were referenced or defined.
- A listing of the object module in a symbolic and numeric representation very nearly in a form that might have been produced by the Assembler.

The compiler organization and information flow are designed particularly for processing in the time sharing environment. Wherever possible, to reduce page-turning, the intermediate data have been organized and processed serially, in file form, rather than in a form requiring random access. The presence of the entire file in virtual storage ensures fast access to its contents, while repeated references to the same storage page, inherent in serial processing, relieve the Resident Supervisor of the necessity for making available a number of different pages in rapid succession.

While primarily a conventional batch processor, the compiler contains special features making it especially suitable for conversational, terminal oriented operation. The compiler syntax analysis performs a thorough error checking of the source program, statement-by-statement, as it is input through the Language Processor Control Program (LPC). Diagnostic messages are returned to the user's terminal via LPC, and each appears at the terminal following the listing of the statement in which the error was detected. At such time, LPC gives the user the opportunity to correct the error, whether it be in the last statement processed or in some earlier statement. Then LPC informs the compiler of whether a change was made and if so, which lines are affected. If only the last statement was changed, the compiler "forgets" the effect of the last statement and begins compilation with the statement replacing it. Otherwise, the compiler restarts compilation from the beginning of the source program module. In this manner the most common errors, those local to the last statement processed, may be corrected with a minimal time penalty.

After the END statement has been processed by the first phase, the compiler's second phase, during the course of its processing, may detect errors of a more global nature (e.g., undefined statement labels, illegal DO-loop flow). The resulting error messages are passed to LPC as before, but now LPC does not allow the user to supply correction lines. When the compiler's second phase is complete, LPC gives the user the opportunity to correct errors or to go on. If errors are corrected, the compiler will, under the direction of LPC, recompile from the beginning of the stored source data set, and another conversation is possible. Otherwise, compilation proceeds to termination through the remaining compiler phases.

The compiler is designed to produce compact, efficient object programs in a form suitable for loading and executing by TSS/360. All object programs (both main and subprograms) are relocatable, reentrant subroutines.

Much of the processing done during compilation is devoted to improving the speed of execution of the object program. Some of the more significant optimizations may be mentioned here. The memory organization of data is designed to maximize the sharing of base and index registers to avoid excessive reloading. Expressions occurring more than once ("common expressions") are detected and evaluated only at the first occurrence. Unchanging expressions used within an iterative loop are evaluated outside the loop. Subscripts within a loop which depend on the number of iterations are computed by recursion. Registers for frequently used address constants or subscript expressions are given permanent assignments over loops, avoiding load instructions within the loop. Common expressions or partial results are left in registers for later use after evaluation, unless or until program flow or the need for registers makes storing necessary. Instructions to be generated are selected to fit each individual situation. The order of generation within an expression is chosen to minimize the number of partial results which cannot be immediately used. Advantage is taken of commutative operations and equivalences between certain combinations of operations to obtain the most efficient code.

Organization Of The Compiler

The compiler consists of six major components: a multifunction Compiler Executive and five compiler phases. The major functions of each component are summarized here.

COMPILER EXECUTIVE ROUTINE

The Compiler Executive (EXEC) has six broad and distinct functions:

1. To interface with the compiler's environment.

2. To prepare the source statements for processing by Phase 1.

3. To control and order the operation of the phases.

4. To prepare edited lines for output.

5. To provide compiler diagnostic information.

6. To provide miscellaneous services.

All interfaces between the FORTRAN compiler and LPC, as well as interfaces with other external routines exist in the Compiler Executive Routine.

The Compiler Executive Routine may be called by LPC at either of two points and may itself call LPC at either of two points (see Figure 105).

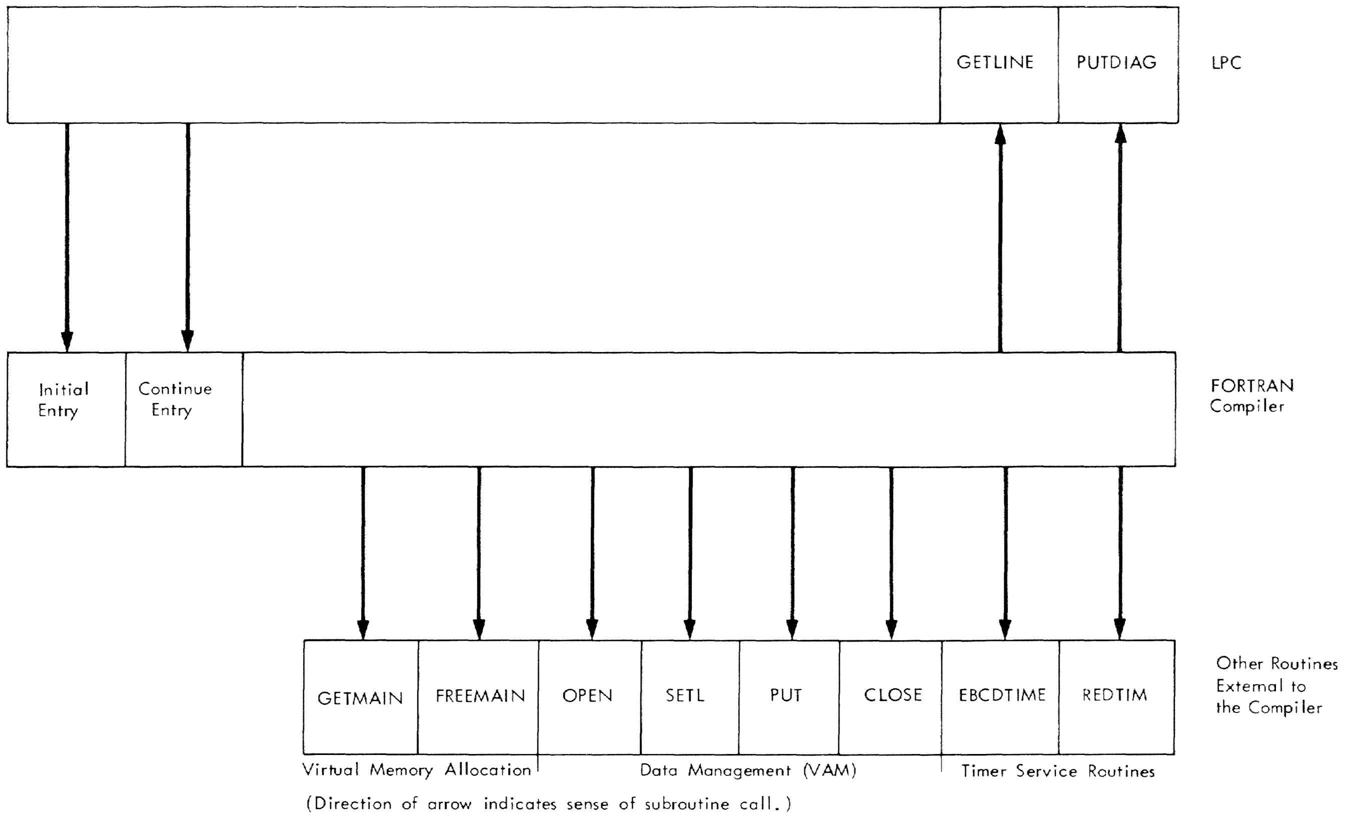


Figure 105. FORTRAN IV Compiler External Interfaces

The two compiler entry points are called INITIAL and CONTINUE. LPC calls the INITIAL entry to pass the user options to the compiler and to initiate the first stage of the compilation (Phases 1 and 2). LPC calls CONTINUE to complete the compilation after the first stage is finished. The compiler return from CONTINUE informs LPC of the extent of the elements of the output module so that LPC may dispose of them.

The compiler calls LPC at either of two points during the first stage (before the compiler returns to LPC from the INITIAL call). The first, GETLINE, is used to obtain a source line. The second, PUTDIAG, is used to pass a source error diagnostic message to LPC. PUTDIAG may also be used after the first stage.

The compiler also uses other TSS/360 service routines. The TSS/360 Data Management routines are used to maintain a List data set; the Virtual Memory Allocation routines are used to obtain space for the Symbol Table and certain other tables; and the timer service routines are used by the compiler to time stamp the listing and object program modules control sections.

A schematic diagram of the organization of the compiler is shown in Figure 106.

PHASE 1

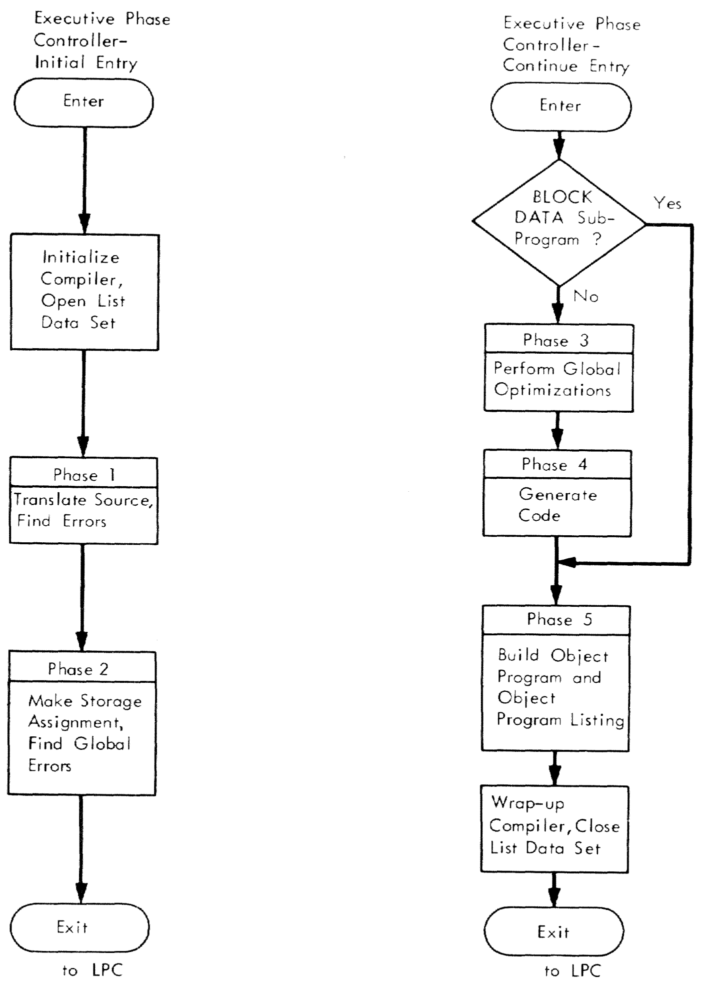
Phase 1 scans the source program a statement at a time as obtained from the Compiler Phase 1 Executive. Compiler Phase 1 analyzes the source program, detects errors, and encodes the information in various tables and files for processing by later phases of the compiler. Information generated by Phase 1 includes:

- Symbol Table information about symbolic names.

- An internal representation of the executable part of the program.

- Lists containing the information from COMMON and EQUIVALENCE statements.

Each distinct identifier or constant is given an entry in the Symbol Table. Initial values from DATA and Type statements, dimension information for arrays, NAMELIST information, and alphameric constants are stored in a table. Information concerning references to and definitions of symbols and statement numbers is stored in the Cross-reference Table. Information collected from COMMON and EQUIVALENCE statements is stored in the Storage Specification List.



Executive Phase Controller- Initial Entry
Enter
Initialize Compiler, Open List Data Set
Phase 1
Translate Source, Find Errors
Phase 2
Make Storage Assignment, Find Global Errors
Exit
to LPC
Executive Phase Controller- Continue Entry
Enter
BLOCK DATA Sub-Program ?
Yes
No
Phase 3
Perform Global Optimizations
Phase 4
Generate Code
Phase 5
Build Object Program and Object Program Listing
Wrap-up Compiler, Close List Data Set
Exit
to LPC

Figure 106. Compiler Component Organization

The most significant processing from the point of view of later optimization and code generation concerns the treatment of executable statements, statement numbers, and arithmetic expressions.

Each executable statement and statement number is placed in the Program Representation File (PRF) which, when scanned in the order it was formed, is a skeletal outline-representation of the source program. In addition to the fields that distinguish the items from each other, the PRF entries contain pointers to the appropriate Expression File (EF) entries (see below), to Symbol Table entries for variables, constants, and statement numbers, and to other PRF entries as appropriate to the individual type of entry.

Each expression is placed in the Expression Representation File (ERF) in tabular form. The ERF form of the expression is a parenthesis-free notation in which, reading from left to right, each operand occurs in the order in which it occurred in the original expression and each operation follows its associated operand pair. The form is referred to as Righthand Polish, or simply Polish.

Each of the operator items includes information about its type and code to indicate which operation is represented. Each variable or constant item includes information about its type and a Symbol Table pointer. This pointer is the means of reaching the associated Symbol Table entry and serves both to associate this term with other items representing the same variable or constant and to distinguish it from others.

PHASE 2

Phase 2 has five broad functions:

1. Assign storage locations to all source program variables that are not formal arguments of a subprogram. The effects of COMMON, equivalence, and DIMENSION statements are taken into account.

2. Detect illegal flow in DO nests and issue diagnostics.

3. Indicate that the DO-loop index variable requires materialization (must bemaintained in its storage cell) in a loop that contains an exit.

4. Detect and diagnose statement numbers (labels) that are referenced but never defined.

5. Determine definition points (points at which a value may possibly be changed) of COMMON variables and subprogram arguments.

COMMON variables are assigned storage in the order dictated by their appearance in the source program in their appropriate COMMON blocks and are given as much space as indicated by their individual DIMENSION/ Type combinations.

NonCOMMON variables which do not appear in EQUIVALENCE statements are assigned storage such that all scalars appear first, followed by all 1- then 2- dimensional arrays, etc. For any given dimensionality, variables of the same type appear together. Those requiring less storage preceding those requiring more. In this manner, a maximum of address constant sharing may be obtained in the object program.

The relative relationships of storage assignments of variables appearing in EQUIVALENCE statements is determined, and these variables are assigned storage within the appropriate COMMON block or at the end of the nonCOMMON group as required. Variables which do not occur in COMMON statements, but which appeared in EQUIVALENCE statements in conjunction with COMMON variables, are flagged as appearing in COMMON.

After a storage assignment has been made, its assignment within storage class is recorded in the Symbol Table. NonCOMMON variables are assigned storage class 6, blank COMMON storage class 9, and labeled COMMON storage class 10 to as high as 127 in the order of first appearance of the corresponding labeled blocks in the source program.

Storage classes 3 through 8 will be accumulated by Phase 5 and become the moduclasses include alphameric constants, address constants, NAME LISTs and parameter lists, nonCOMMON variables, global (unreleasable) temporary storage, and local temporary storage, in that order. The COMMON blocks (storage classes greater than 8) become individual control sections in the object program module where the block name becomes the control section name. Such control sections are combined with control sections of like name from other modules before execution during the linkage editing or loading process.

PHASE 3

Phase 3 is devoted mainly to global optimizations, being charged with the detection of common expressions, the removal of expressions from DO loops, the choice of address-modifying quantities for loop-wide register assignment, and the preparing of loop-variable-dependent subscripts for recursive evaluation. This phase also transforms storage references into machine-oriented address constants and displacements.

Phase 3 determines which arithmetic expressions need to be computed only once and then saved for later use. In addition, it determines the range of statements over which expressions are not redefined by the definition of one or more of their constituents. If the occurrence of an expression in that range is contained in one or more DO loops which are also entirely contained in that range, Phase 3 determines the outermost such loop outside which such an expression may be computed, and physically moves the expression to the front of that DO loop. Only the evaluation process is removed from the loop. Any statement number and/or store process is retained in its original position. The moved expression is linked to a place reserved for that purpose in the Program Representation File entries corresponding to the beginning of DO loops.

In the statements

```
 1  A=B+C
 2  D=A+B
 3  A=A*2
 4  Z=A+B
 5  X=B+C
```

the occurrences of the expression B+C in statements 1 and 5 are determined to be common because neither of the constituents B or C has an intervening definition. The expression identification corresponding to the plus operator will be changed from "operator" to "common expression". A common expression identifier has the properties of the original operator (e.g., here the plus operator code is retained), with the additional property that a "named" expression is identified. The common expression identifier item contains a field reserved for the expression name (actually a monotonically increased number) which will be identical only for identical expressions.

In statements 2 and 4 above the expression A+B is not a common expression because of the intervening definition of A in statement number 3. Both plus operators retain their "operator" identity. Neither becomes a "named" expression.

In the statements

```
    DO 1 I=1, 10
    A = B+C
    Y = E+F
  1 F = A
```

there are definitions of neither B nor C within the DO loop, thus, the expression B+C is given a "name", and the named expression is linked to the beginning of the DO statement, so that Phase 4 will generate the expression once, outside the loop. The occurrence of the expression inside the loop is replaced by a "residue item" that has the same "name" as the removed expression. Note that expression E+F is neither named nor removed because of the definition of F in statement 1.

Phase 3 creates two new operators, both arising only from subscripts. The first, called a base/index split operator or "? operator", is such that its right operand is a residue (computed outside a DO loop) and its left is an expression that is local to the DO loop. Phase 4 will place one quantity in a base register and the other in an index register when generating a storage reference to the subscripted quantity.

The second operator is called the recursive operator or "! operator". It has the property that its right operand is the initial value of a subscript (induction variable dependent) constituent that is to be computed recursively over a DO loop; and its left operand is the "step expression", a quantity to be added to the recursive expression after each pass through the loop. (The induction variable is the variable referenced in the DO statement of the loop. In the DO statement shown above, I is the induction variable.)

Phase 3 merges the Expression Representation File and Program Representation Files with some modification to form the Program File. This file is the primary output of Phase 3.

PHASE 4

Phase 4 selects the instructions to be generated for the executable object program and performs local optimizations such as determining best order of computation within expressions, taking advantage of existing register contents, and recognizing special situations where particularly efficient code can be produced.

Phase 4 is a collection of Program File entry processing routines, arithmetic generators tailored to the various operators and expression types, and service routines to maintain register contents, partial results, and common expressions, to select and assign registers, to determine when operands are no longer needed, to assign and release temporary storage, etc. Processing is triggered by the various file items and by the expressions they may reference.

A set of tables is maintained that reflects the contents of the various general and floating point registers at any time. When the generation of an expression is required, the register tables are searched, and if any constituent operand of the expression is in a register, it is generally used from that register, rather than from storage. Partial results are stored in temporary storage only when a register is needed for some other purpose and there is no better choice of register than the one containing the partial result, or when the partial result is a common expression which has later uses and the operation about to be performed will change the value of the register containing the common expression.

PHASE 5

Phase 5 collects the information from the various compiler-generated storage classes and forms a sharable CSECT, a PSECT, and as many COMMON CSECTs as there are declared COMMON blocks. This information and a description of internal names and labels placed in the (optional) ISD constitute the Object Program Module.

Optionally Phase 5 will also produce an assembler-like listing of the object program code obtained from the Code File, a storage map, and a cross reference listing indicating the various source program identifiers and the lines in which they were referenced or defined. The user selection of these options is passed from LPC to the Compiler Executive and thence through a compiler intercommunication table to Phase 5.

The TSS/360 PL/I compiler analyzes and processes source programs written in PL/I (See IBM System/360 Time Sharing System: PL/I Language Reference Manual and PL/I Programmer's Guide) translating them into object data sets. The code in these object data sets is not suitable for execution by TSS/360 and must be processed by the object data set converter (ODC) into TSS/360 executable code. The main services performed for the PL/I user are by the program language controller, the PL/I compiler, and the object data set converter.

PL/I Control

The PLI command invokes the program language controller (PLC) which acts as the interface with the system. PLC acts as a communications area for user-specified options, and controls the sequence of events from the invocation of the PL/I compiler.

The PL/I compiler, unlike the TSS/360 assembler and the FORTRAN compiler, cannot function until the source data set has been fully entered. Therefore, if the named data set does not exist, PLC invokes the text editor to create the PL/I source data set. Once the source data set exists, control passes back to PLC, which then calls the PL/I compiler. (See Figure 107)

PLC contains recovery facilities in case of interruptions, which permit completion to proceed from the point of interruption or from the beginning.

User options may cause PLC to act as an interface to perform any of the following functions, as well as to create the source data set mentioned above:

1. Convert separately created PL/I object data sets to TSS/360 code.
2. Combine a list of PL/I object data sets for conversion to executable code.
3. Perform multiple compilation within a single invocation of the compiler.
4. Print compiler-generated listings.

PL/I Compiler

The source data set, which is input to the compiler, is given the name the user specifies, or SOURCE.XXX. The data sets that constitute possible output from the compiler are: a list data set, named LIST.XXX(0); a load data set, named LOAD.XXX(0); and a macro data set, named MAC.XXX(0). Refer to Table 2 for compiler data sets.

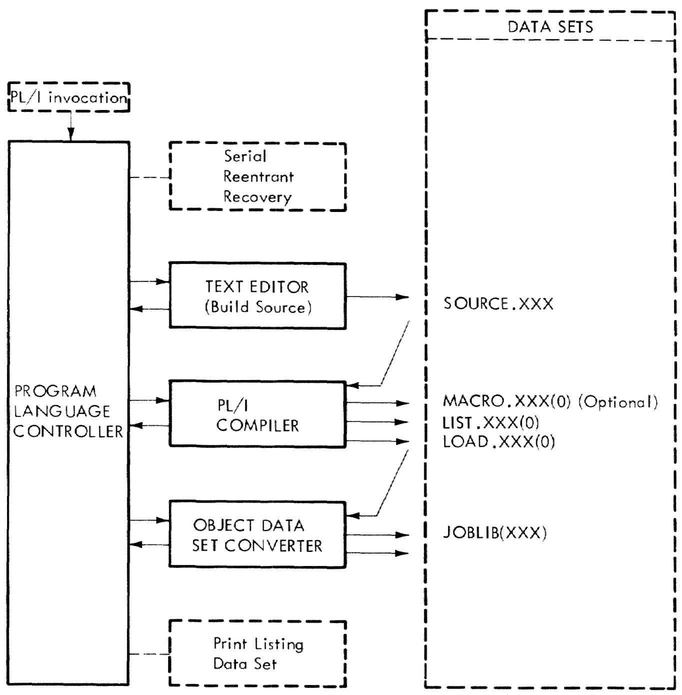


Figure 107. Program Language Controller Flow

The source program to be compiled appears as input to the compiler on the PLIINPUT data set. If one of the preprocessors is called prior to compilation, a macro data set is created with the ddname of PLIMAC. When preprocessing is completed, PLIMAC replaces PLIINPUT as input to the compiler. The PLILIST data set is opened by PLC unless the user specifies that a separate listing is unnecessary, in which case the listing is placed on SYSOUT and no record of it is retained in the system after printout.

The PLILOAD data set containing compiler output, and the PLIMAC data set containing intermediate text, are optional and are opened by control routines in the compiler. The PLIINPUT data set is always used by the compiler, and is opened by PLC.

COMPILER PHYSICAL AND LOGICAL PHASES: The PL/I compiler is comprised of 12 logical phases, each of which consists of several physical phases, and all under the control of the compiler control routines. These

Table 2. Data Sets Used by PL/I Compiler

DDNAME	DSNAME	ACCESS METHOD	COMMENT
PLIINPUT or user-supplied $$$nnnnn	SOURCE.XXX(O) or user-supplied	VISAM	Source input to compiler--user-supplied or created by text editor before compilation is initiated
PLILIST	LIST.XXX(O)	VSAM	List data set -- built unless user options indicate none is necessary
PLILOAD	LOAD.XXX(O)	VSAM	Load data set -- output from compiler and input to ODC
PLIMAC	MAC.XXX(O) or user-supplied	VISAM	Intermediate source text -- created whenever preprocessing is specified
SYSULIB or user-supplied	USERLIB or user-supplied	VPAM/ VISAM	Member of library used by macro-phase %INCLUDE verb.

compiler modules are link edited into six output modules as follows:

1. Control Output Module - contains all of the control modules except those responsible for initialization. The code in this output module is reusable; it therefore remains resident during multiple compilations.

2. Main Output Module - contains the modules responsible for initialization, along with all of the logical phases except those responsible for preprocessing, optimization (option OPT=2), and interphase dumping and tracing routines.

3. First Propreccessing Output Module - contains the modules required for macro and/or 48-character preprocessing, with the exclusion of modules which are reused in the processing of the macro option.

4. Second Preprocessing Output Module - contains those modules of the macro preprocessor that may be reused in the processing of the macro option.

5. Optimization Output Module - contains those modules which are required when the option OPT=2 is specified by the user.

6. Interphase Dumping and Tracing Output Module - contains all of the modules required for interphase dumping and tracing.

Control of the compiler is implemented as shown under Control Output Module and Main Output Module above. The logical phases of the compiler are as shown in Table 3.

The following overview of compiler flow (see Figure 108) depicts, at the system level, compiler logic including the possible use of a 48-character preprocessor. This preprocessor is called only when the compiler is in the 48-character set and the compile-time preprocessor is not needed. The user indicates this by specifying the CHAR48 option. This preprocessor is not included in the 12 logical phases above.

Object Data Set Converter (ODC)

The TSS/360 PL/I compiler produces an output load data set which contains code similar to the OS/360 PL/I object text output. To transform this load data set into a TSS/360 loadable module, the object data set converter (ODC) is used. ODC also provides some of the functions of the OS/360 linkage editor by resolving some QCONs (commonly called pseudo-registers vectors).

ODC is invoked by PLC to convert load modules if either the MERGELST or the MERGEDS parameters have been specified, or if the PL/I compiler was invoked and has created an internal merge list.

Input to ODC consists of the OS/360 object data set (normally in OS/360 used as input to the linkage editor); Output from ODC consists of the TSS/360 load modules which are STOWed into the current job library.

ODC deals with pseudo-registers as follows: It builds REFs for the pseudo-registers which must follow the other REFs in the CSD. The REF numbers assigned by ODC are independent of their position on ODC's internal text chain of REFs. Modifiers for pseudo-registers have a type-4 operation code. Each PL/I program will have its pseudo-registers resolved by the

Loader. ODC also generates a cumulative Q-length function (CXD) field in an initialization control section. This field is also resolved by the loader.

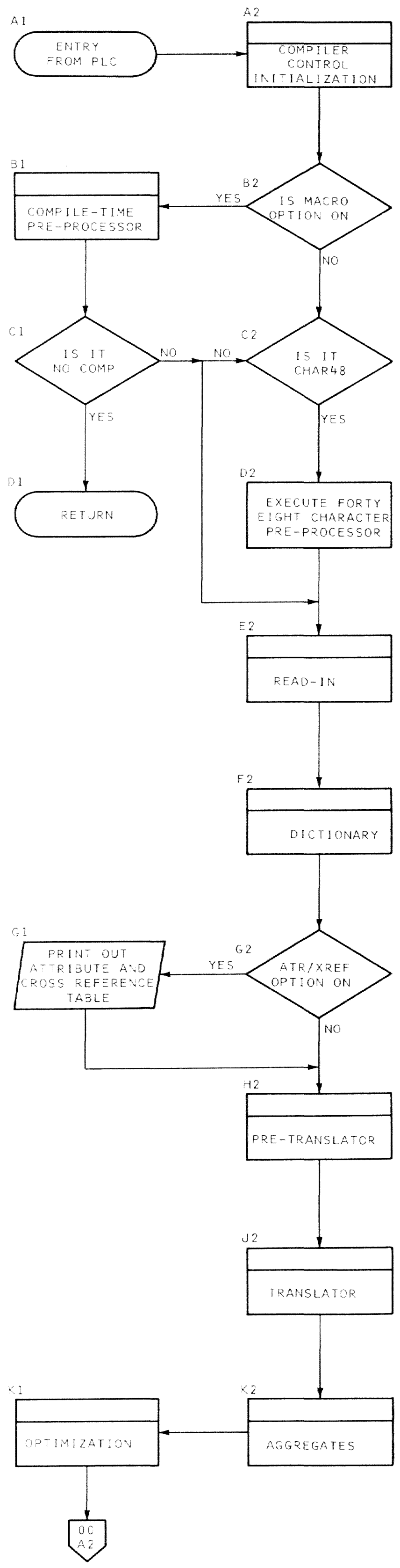
Unlike the other QCONs, the 28 standard pseudo-registers used within the library are resolved by ODC -- they do not appear as REFs nor do any pseudonyms appear as REFs. These QCONs are used by the PL/I subroutine library to ensure that reference to a program block, file, or controlled variable remains constant between PL/I modules.

Table 3. Compiler Logical Phases

Logical Phase	Function
Compile-time Processor	Reads input text, executes any compile-time statements contained in it, and modifies text as directed, producing modified text for further processing.
Read-In	Checks source program syntax and removes all superfluous characters, such as comments and non-significant blanks, from the text string.
Dictionary	Removes all BCD identifiers and attribute declarations from the source string, and replaces them by symbolic references to dictionary entries. The dictionary entries contain all the consistent declared attributes, and all the attributes specified in the language in default of source program specifications. Error messages are generated for all inconsistent attributes.
Pretranslator	Processes those features of the language that are more easily processed in their original PL/I form, than when the original syntactic form has been lost in later phases. The Pretranslator carries out these modifications which include the rearranging of the order of certain I/O statements, the creation of temporary variables for procedure arguments which are expressions, the conversion of array and structure assignments to a series of DO-loops surrounding scalar assignments, and the removal of iSUB expressions.
Translator	Converts the original PL/I syntactic form to an internal syntatic form, referred to as "triples". Triples consist of the original source program operators and operands, but rearranged so that the operations specified in the source string may be carried out in their proper order.
Aggregates	Carries out all structure and array mapping, so that elements are aligned on the correct virtual storage boundaries. When it is not possible to carry out the mapping at compilation time, such as when the aggregates contain string lengths or array bounds which are specified by expressions, object code is produced to do it at object time. This phase also checks that items DEFINED on arrays and structures can be mapped consistently.
Optimization	If optimization is requested, the optimization phases attempt to reorder triples for subscript address calculations and generate efficient pseudo-code for DO-loop control. This enables some PL/I programs to compile into faster object code at the cost of extra compile time.
Pseudo-Code	Converts the triples to a form closely resembling machine instructions, in which registers are represented symbolically, and storage locations are represented by dictionary references with offsets. The final pseudo-code version of the text also contains a number of special pseudo-code items for the guidance of later phases.

Table 3. Compiler Logical Phases (continued)

Logical Phase	Function
Storage Allocation	Searches the dictionary for all entries requiring storage, and allocates offsets to each item, either within its AUTOMATIC block, or within the STATIC storage area. Code is compiled to set up dope vectors and pointers at object time, for allocations of controlled variables and temporaries, the storage for which must be obtained during the execution of the object program. Prologue code is generated for each block of the object program.
Register Allocation	Allocates physical registers to the symbolic registers which have been requested by earlier phases, and also ensures that all the storage location offsets allocated in previous phases can be addressed by the insertion of additional instructions, where necessary.
Final Assembly	Completes the translation to machine code instructions, by calculating branch destination addresses inserted symbolically by earlier phases. Loader text is then produced for the machine instructions, constants, INITIAL values in STATIC storage, and all the constant data required for block initialization. An External Symbol Dictionary (ESD) and a Relocation Dictionary (RLD) are produced to enable the object program to be converted by the Object Data Set Converter (ODC). The Final Assembly Phase also produces a listing of the object code produced.
Error Editor	Entered at the end of every compilation. The dictionary is examined to determine whether there are any diagnostic messages to be printed out. If there are none, the compilation is terminated by the compiler control. If there are diagnostic messages to be printed out, the error dictionary entries are processed and the messages are printed. The texts of all the diagnostic messages are held in modules XG through YY.



COMPILER CONTROL PERFORMS INITIALIZATION, OVERSEES PHASE LOADING, RESOLVES SYMBOLIC TEXT AND DICTIONARY REFERENCES AND CONTROLS ALL INTERFACES BETWEEN THE COMPILER AND THE TIME SHARING SYSTEM.

THE COMPILE-TIME PRE-PROCESSOR ACCEPTS INPUT CONTAINING THE COMPILE TIME STATEMENTS OF PL/I AND PRESENTS THE COMPILER WITH THE SOURCE TEXT RESULTING FROM EXECUTION OF THESE STATEMENTS.

THE FORTY EIGHT CHARACTER SET PREPROCESSOR ACCEPTS SOURCE PROGRAMS WRITTEN IN THE FORTY EIGHT CHARACTER SYNTAX OF PL/I AND CONVERTS THEM INTO SIXTY CHARACTER SYNTAX. THIS FUNCTION IS PERFORMED BY THE COMPILE-TIME PROCESSOR WHEN BOTH MACRO AND CHAR 48 ARE SPECIFIED.

THE READ-IN LOGICAL PHASE CHECKS THE SYNTAX OF THE SOURCE PROGRAM, REMOVES SUPERFLUOUS CHARACTERS AND LEAVES CERTAIN CHAINS IN THE PROCESSED TEXT TO AID LATER PHASES.

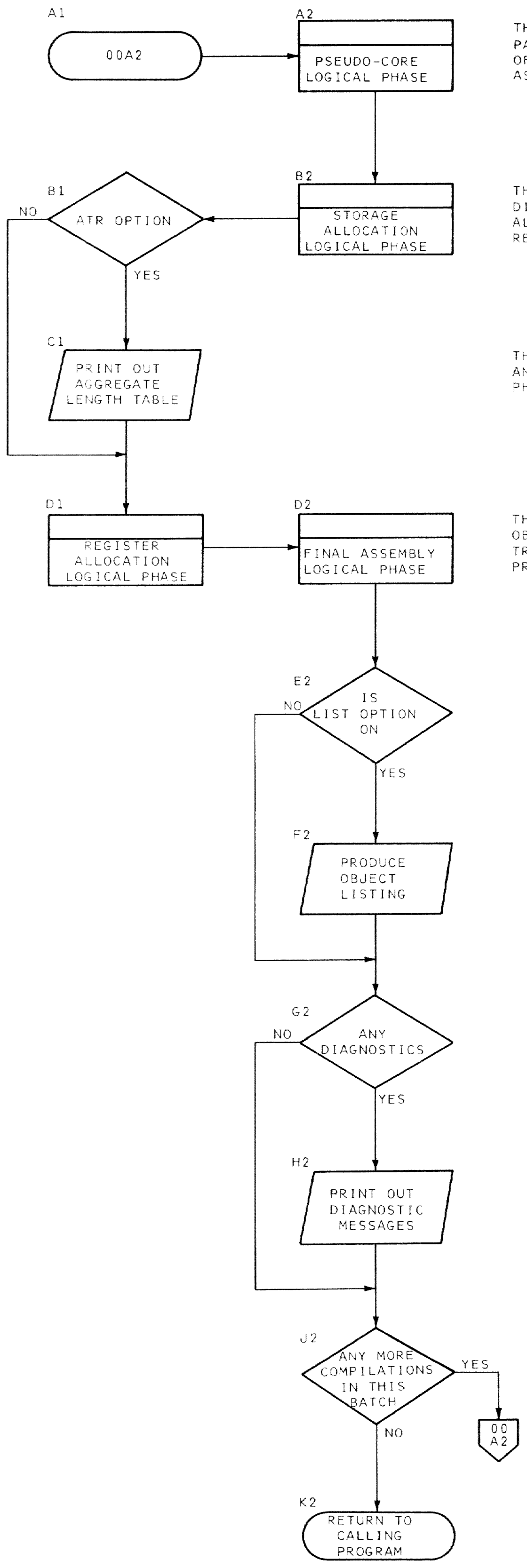
THE DICTIONARY LOGICAL PHASE CONSTRUCTS THE DICTIONARY OF IDENTIFIERS FROM INFORMATION IN DECLARE STATEMENTS AND FROM CONTEXT. IT ALSO REPLACES BCD IDENTIFIERS IN THE TEXT BY REFERENCES TO THE DICTIONARY.

THE PRE-TRANSLATOR LOGICAL PHASE MANIPULATES THE TEXT, REARRANGING I/O STATEMENTS, CREATING TEMPORARY VARIABLES WHERE PARAMETERS DO NOT MATCH THEIR CORRESPONDING ARGUMENTS, CONVERTING ARRAY AND STRUCTURE ASSIGNMENTS TO DO LOOPS, AND REMOVING ISUB EXPRESSIONS.

THE TRANSLATOR LOGICAL PHASE CONVERTS THE PL/I SOURCE TEXT TO A COMPUTER-ORIENTED FORM CALLED 'TRIPLES'. GENERIC SELECTION IS ALSO CARRIED OUT.

THE AGGREGATES LOGICAL PHASE MAPS ALL STRUCTURES AND ARRAYS TO ALIGN ELEMENTS ON CORRECT STORAGE BOUNDARIES. PSEUDO-CODE IS PRODUCED TO CARRY OUT INITIALIZATION AT OBJECT TIME.

Figure 108. Overview of Compiler Flow



A1
00A2
A2
PSEUDO-CORE LOGICAL PHASE
B1
ATR OPTION
NO
YES
B2
STORAGE ALLOCATION LOGICAL PHASE
C1
PRINT OUT AGGREGATE LENGTH TABLE
D1
REGISTER ALLOCATION LOGICAL PHASE
D2
FINAL ASSEMBLY LOGICAL PHASE
E2
IS LIST OPTION ON
NO
YES
F2
PRODUCE OBJECT LISTING
G2
ANY DIAGNOSTICS
NO
YES
H2
PRINT OUT DIAGNOSTIC MESSAGES
J2
ANY MORE COMPILATIONS IN THIS BATCH
YES
NO
00 A2
K2
RETURN TO CALLING PROGRAM

THE PSEUDO-CORE LOGICAL PHASE PERFORMS MANY PASSES OVER THE TEST. EACH PASS CONVERTS SOME OF THE TEXT CONTEXT TO A FORM ALLIED TO ASSEMBLY LANGUAGE, CALLED PSEUDO-CORE.

THE STORAGE ALLOCATION LOGICAL PHASE SCANS THE DICTIONARY AND ALLOCATES OBJECT TIME STORAGE FOR ALL IDENTIFIERS, TEMPORARIES AND ADMINISTRATIVE REGIONS. PROLOGUES ARE CONSTRUCTED.

THE REGISTER ALLOCATION LOGICAL PHASE PERFORMS AN ANALYSIS OF OBJECT TIME ADDRESSIBILITY AND ALLOCATES PHYSICAL REGISTERS IN PLACE OF SYMBOLIC ONES.

THE FINAL ASSEMBLY LOGICAL PHASE ESTABLISHES OBJECT TIME BRANCH ADDRESSES AND COMPLETES TRANSLATION TO MACHINE CODE. LOADER TEXT IS PRODUCED.

Figure 108. Overview of Compiler Flow (Continued)

The Linkage Editor, one of the IBM supplied user programs, statically combines (links) different object program modules for eventual input to the Dynamic Loader. Modules to be linked and edited by the Linkage Editor are the product of the TSS/360 FORTRAN IV and Assembler language processors, or the product of a previous Linkage Editor operation. These modules are contained in libraries (partitioned data sets). The output (object module) of a language processor may contain unresolved external references. External references are those that make references to entry point names, CSECT names, or module names that are not present in the output module created by the language processor. If possible, such symbols are resolved by the inclusion of modules from the libraries available to the Linkage Editor.

With the exception of error condition treatment, processing by the Linkage Editor is the same in conversational and nonconversational modes of operation, and does not depend on the physical origin of source statements.

RELATIONSHIP TO TSS/360

The Linkage Editor is classified as a user program, and, as such, it has the same relationship (including restrictions and conventions) to the IBM TSS/360 as any other user program. Being a shared, nonprivileged, reenterable service program, the Linkage Editor receives its inputs from virtual storage, places its output in virtual storage, and has no hardware requirements of its own. (Hardware requirements for the system data sets are under the control of catalog services and data management.)

GENERAL PROCESSING REQUIREMENTS

Certain requirements must be satisfied by the Linkage Editor user. Each library except SYSLIB and SYSULIB to be used during the Linkage Editor run must be defined by the user with a DDEF command. If the control statements are not to be input via a SYSIN device, the user must prestore them within a cataloged data set. Linkage Editor parameters must be given following a RUN LNK command when commencing a Linkage Editor run. If no parameters are supplied, the user is prompted for them. These parameters determine the characteristics of the Linkage Editor's output by specifying:

- The name of the object module to be created
- Whether the control statements are prestored
- Line and increment numbers for LPC
- The name of the library in which the object module is to be cataloged
- Version identification
- Whether an ISD is to be produced
- Whether a PMD listing is to be prepared

LPC, a component of the Command Language Interpreter, serves as the link between the user and the Linkage Editor. LPC gathers the input parameters for the Linkage Editor, loads the Linkage Editor, and passes it the parameters. The Linkage Editor calls upon LPC for source lines (control statements) and uses LPC to output diagnostic messages.

The RUN LNK command starts LPC MAIN. LPC MAIN then collects the parameters required, either by prompting the terminal user or by fetching them from SYSIN. In either case, the parameters are verified. The user is asked to correct any errors in the input parameters if the mode is conversational. Errors in the Linkage Editor input parameters processed by the LPC when the mode is nonconversational result in task termination, via the ABEND routine. LPC MAIN places the valid parameters in a list, calls the Linkage Editor, and passes the address of the list to it.

At the end of the Control Statement Processing phase, the Linkage Editor sets a return code and returns control to LPC MAIN. A code of 0 means that an object module has been created. A code of 4 means that errors in the source data set prohibited creation of an object module.

If a return code of 4 is returned after the first phase, LPC MAIN issues any remaining diagnostics and a confirmation message, if appropriate. If the mode is nonconversational, LPC MAIN terminates processing. If the mode is conversational, LPC MAIN asks the user if he wants to terminate processing or modify his source data set; the user cannot just continue if the return code is 4.

When normal processing ends, the Linkage Editor returns control from its continuation entry point to LPC MAIN. LPC MAIN now stows the object module (if one was created) and optionally issues a PRINT command to print the contents of the listing data set. LPC MAIN then returns control to its caller.

Connecting modules is one of the basic functions of the Linkage Editor. Previously assembled, compiled, or link edited programs may be included as subprograms of new, more complex programs with no unresolved symbols. Each module to be connected (linked) by the Linkage Editor may be the output of a different language processor. However, it is the user's responsibility to provide adequate linkage conventions and compatible data forms for programs composed in such a way.

Another basic function of the Linkage Editor is editing. Editing consists primarily of modifying, deleting, or replacing control sections located in any of the input modules. Both linking and editing are directed by Linkage Editor control statements. However, the Linkage Editor provides automatic editing of duplicate external symbols. That is, duplicate external symbols are removed and duplicate control sections are rejected.

During the process of linking and editing object modules, the Linkage Editor performs the following functions.

Library Calls

Object program modules are included in the output of the Linkage Editor upon request of the user. Object program modules are stored in the TSS/360 program libraries - System Library (SYSLIB), User Library (SYSULIB), and Job Libraries - as members of partitioned data sets. The Linkage Editor user must specify the ddnames of those Job libraries from which he intends to retrieve modules, prior to the Linkage Editor run, through the use of DDEF commands.

Program Modification

The Linkage Editor permits the user to statically modify modules without having to reassemble, recompile, or reexecute. The Linkage Editor can replace, delete or combine control sections as directed by control statements, and can also change the attributes of a control section. External symbols (DEFs) can be replaced, deleted, or renamed as directed by Linkage Editor control statements. Also external references can be renamed.

Programming Aids

A list of all outstanding unresolved external references (REFs), plus a list of those resolvable from the System library, is prepared upon completion of Linkage Editor processing. Also, the user may direct that an optional Program Module Dictionary (PMD) listing of the program module be prepared.

Error Detection and Messages

During Linkage Editor processing, errors or possible error conditions are detected in both conversational and nonconversational modes. Four diagnostic levels are available to TSS/360 language processors:

Diagnostic Level Value	Meaning
0	No errors
1	At least one minor error
2	At least one major error
3	A catastrophic error, no module created

Before its termination, the Linkage Editor assigns to the output module the highest diagnostic level code encountered. Only diagnostic level codes 0 and 2 are generated by the Linkage Editor. Level codes are also received as part of an input module. However, only code levels 1 and 2 may be passed on to the output module because there is no module to contain the code when the diagnostic level code is 3.

Errors detected during Linkage Editor processing do not terminate processing, but cause a response which is determined by the mode of operation. While operating in conversational mode, the Linkage Editor transmits diagnostic messages to the terminal and the user may correct the error, using applicable recovery procedures (such as modifying statements previously entered). In nonconversational mode, a standard response is made for the default condition. The diagnostic message is sent to the SYSOUT dataset and processing continues.

LINKAGE EDITOR MAJOR DIVISIONS

The Linkage Editor consists of three major divisions: Control Statement Processing, Output Processing, and Early-End Processing.

The Linkage Editor is called by Language Processor Control (LPC).

LPC receives Command System commands from the user, and after processing them, calls the Linkage Editor at one of three entry points. These entry points are

- Initiation (Control Statement Processing).
- Continuation (Output Processing).
- Early-End (Early-End Processing).

Upon request for service of the Linkage Editor by a user, LPC calls the Linkage Editor at the Initiation entry point. Control is returned to LPC after control statement processing is performed; LPC then calls the Linkage Editor at the continuation entry point to deliver the final module, after which control is returned to LPC. If it is necessary to prematurely terminate Linkage Editor processing, LPC enters the early-end entry point.

CONTROL STATEMENT PROCESSING

The control statements processed by the Linkage Editor are as follows:

INCLUDE (FORM 1): Obtains program modules from a library and places them in the output program module, linking them to any modules which have previously been included.

INCLUDE (FORM 2): Instructs the Linkage Editor to scan and include from a specified library other program modules whose entry names satisfy unresolved external references in an (as yet incomplete) output program module. If one of the newly included modules contains unresolved external references, the specified library is again scanned for modules which resolve these names. This continues until no new modules can be included.

INCLUDE (FORM 3): Functionally identical to INCLUDE (Form 2), except that the user supplies a list of external references which are not be resolved by the specified library.

RENAME: Provides a means of deleting or renaming control sections and entry names, and of renaming external references which are to be included in the output program module.

TRAITS: Provides the user with the facility to redefine attributes of a control section and causes the attributes assigned by a language processor to be removed. Thus, only those attributes mentioned in the TRAITS statement will be assigned to the specified control section.

COMBINE: Provides the ability to combine two or more control sections of a module into a single control section, thus possibly reducing the number of pages of virtual storage required.

END: Signals the end of the Linkage Editor control statement and causes the Program Library List to be searched for any unresolved references not specifically excluded.

The Linkage Editor processors used to process the control statements are: INCLUDE, RENAME, TRAITS, COMBINE, and END.

During initialization, storage required for program module processing is obtained, switches and tables are initialized, and the control statement is obtained from LPC. The control statement is checked for correctness, and its sequence of execution, which is governed by the following hierarchy, is checked.

1. INCLUDE (Form-1) control statements are executed before, between, or after any of the other control statements except END.
2. INCLUDE (Form-2 and Form-3) control statements are executed after the appearance of at least one INCLUDE (Form 1). They may never immediately follow a RENAME, TRAITS, or COMBINE control statement.
3. RENAME control statements are executed before an INCLUDE (Form-1) control statement.
4. TRAITS control statements are executed before an INCLUDE (Form-1) control statement.
5. COMBINE control statements are executed before an INCLUDE (Form-1) control statement.
6. The END control statement is executed after all other control statements are executed.

When an INCLUDE or END control statement is received in the input stream, the processor to process the control statement is called immediately. When a COMBINE, TRAITS, or RENAME statement is received, the statement is stacked until an INCLUDE statement is received. The INCLUDE statement processor is then called to initiate processing of the INCLUDE statement and to call the processors which process the stacked statements. Each processor checks the format of its related statement and then performs the required function.

The TRAITS, COMBINE, and RENAME statement processors return control to the INCLUDE statement processor after concluding processing. When all stacked statements have been processed, processing of the INCLUDE statement is completed, and control is returned to the Control Statement Input/Analyze (INANAL) Processor module.

The END statement processor receives control directly from INANAL, and after processing the END statement, returns control to the LPC.

The Form-1 INCLUDE statement causes the Linkage Editor to obtain one or more program modules from a specified library and place them in the output program module, linking them to any modules which have previously been included. The Form-2 INCLUDE statement causes the Linkage Editor to search a specified library, other than SYSLIB, and from it include other program modules whose entry names satisfy unresolved external references in an (as yet incomplete) output program module. If one of the newly included modules contains unresolved external references, the specified library is again scanned for modules which resolve these names. This continues until no new modules can be included. The Form-3 INCLUDE statement is functionally identical to the Form-2 INCLUDE statement, except that the user supplies a list of external references which are not to be resolved by the specified library. Names not appearing in this list will be resolved, if possible, by the inclusion of the program modules from the specified library. The excluded names are presumably to be resolved by subsequent INCLUDE statements. If not, they will remain unresolved in the output program module and be resolved by the Dynamic Loader when the output program module is loaded.

An ERROR Processor routine is used by the Control Statement Input/Analyze Processor routine and by the control statement processing routines to process errors resulting from incorrect statements (illegal operation symbols, illegal delimiter, etc.) and/or statements in the wrong sequence (e.g., INCLUDE statement not yet given). A diagnostic code (highest diagnostic level number) is assigned to the output module when major errors are detected in nonconversational mode. The diagnostic code is placed in the Program Module Dictionary. During the conversational mode the user is notified of an error and invited to correct the error condition.

OUTPUT PROCESSING

After all statements have been processed, LPC calls for Output Processing which generates the final output module, including PMD, text, and the ISD if required. Output Processing also prepares the external name list and PMD Listing and returns control to LPC.

EARLY-END PROCESSING

Early-end processing consists of releasing storage areas and closing any open libraries. It is entered if the Linkage Editor is to be terminated before normal completion. Return to the LPC is made with exit parameters.

TIME SHARING SUPPORT SYSTEM

The Time Sharing Support System (TSSS) provides a facility whereby the system programmer may selectively gather data for analysis of apparent system software errors, and dynamically correct those errors. This system also provides capabilities whereby the system programmer may monitor and test TSS/360. TSSS resides within the Time Sharing System but its operation is only minimally dependent upon the larger system. Conversely, when TSSS is not in use, the Time Sharing System operates without TSSS participation.

TSSS comprises two separate systems: the Resident Support System (RSS) and the Virtual Support System (VSS). They share a control nucleus that is loaded together with the Resident Supervisor by Startup. This control nucleus processes interrupts and activates either RSS or VSS, as requested. All SVC interrupts in the range 64-95 and all manual key external interrupts are delivered to the TSSS control nucleus for processing. In addition, after RSS has been activated, RSS I/O and all program interrupts are directed to the control nucleus.

RSS includes routines for independent language processing and input output that enable it to operate on a stand-alone, non-time shared basis. The RSS user is called a Master Systems Programmer (MSP). VSS is a virtual version of RSS; it executes in virtual storage with similar language processing and I/O routines. The VSS user is called a Task Systems Programmer (TSP). RSS is dependent only on hardware and on a minimal interface with TSS/360, while VSS is dependent primarily on the Resident Supervisor and minimally on the Task Monitor. VSS is independent of other virtual storage programs, both privileged and non-privileged. RSS I/O is independent of TSS/360; it performs I/O by issuing the SIO instruction. VSS I/O performs I/O operations by means of the IOCAL macro instruction.

RSS is not time sliced, whereas VSS executes within a task that is time sliced. However, VSS can call upon RSS to perform certain functions that it cannot perform for itself. In this case RSS is activated and TSS/360 execution is suspended for the time required to perform the requested operation. RSS executes in supervisor state, employing only one CPU. In a duplex system, the other CPU is placed in a wait state. VSS executes in privileged mode.

The circumstances under which RSS is activated are:

- The Master System Programmer (MSP) has initiated RSS activation or has signaled RSS to reactivate and accept input from his terminal.
- An SVC implanted by RSS, or by VSS in real storage, has been executed, which requires RSS to be active in order to process a command statement.
- VSS has signaled that it requires service from RSS.

VSS may be initiated for the purpose of connecting a Task System Programmer (TSP) to a task by:

- The VSS command of the TSS/360 command system, used at the SYSIN terminal of a logged-on task.
- The CONNECT command of RSS, used by the connected Master System Programmer, specifying an active task to which a Task System Programmer will be connected. A Task System Programmer cannot be connected to an idle terminal by the CONNECT command.
- The LOGON command of TSS/360, issued at an idle terminal, supplying parameters for connecting the Task System Programmer to a logged-on task.

TSSS LANGUAGE

The routines that make up TSSS language perform the functions requested by the System Programmer, calling upon TSSS I/O and environment routines for service as needed. The TSSS commands specify the external functions, which can be performed on request from a terminal or dynamically. These commands and their functions are as follows:

Command	Function
AT	Designates a dynamic statement and the point in TSS/360 at which execution is to occur.
CALL	Initiates the execution of a prestored set of command statements.

COLLECT	Moves data from a specified area into a specified collection area.
DEFINE	Enables the System Programmer to define temporary symbols and allocates storage when necessary.
DISCONNECT	Removes the System Programmer capability from the terminal, restores TSS/360 (except for changes made through use of the PATCH command), and permanently transfers control to TSS/360.
DISPLAY	Writes data requested by a System Programmer on his terminal.
DUMP	Writes data requested by a System Programmer on a specified output device.
END	Terminates reading of a device being used for input of pre-stored statement sets.
IF	Designates a conditional statement, whereby execution of the command statement following it is dependent on the evaluation of the predefined condition. Although IF is treated externally as a command, internal processing treats it as an operator.
PATCH	Alters the contents of a specified data field and keeps a record of the patch.
QUALIFY	Establishes implicit real storage, virtual storage, or global (i.e., affecting all users of the system) qualification for subsequent operands.
REMOVE	Deletes ATs and their associated dynamic statements, or deletes patches.
RUN	Causes control to revert to TSS/360. AT SVCs can then be executed.
SET	Alters the contents of a specified data field.
STOP	Causes TSS/360 or a specific task to halt and control to be given to the issuing system programmer.

In the general operation of TSSS language, an input command string is accepted and translated into polish notation. The elements in this polish string are then executed. Although the language routines are functionally identical for RSS and VSS, some differences in operation and output exist. A complete description of the TSSS language routines is provided in the publication _IBM System/360 Time Sharing System: Time Sharing Support System Program Logic Manual_. The following example describes a typical TSP session at a user's terminal.

A system user is experiencing difficulty in executing his program. The particular program has been run successfully in the past and the user is at a loss to explain his present difficulty. He, therefore, enlists the aid of a System Programmer at his installation.

With the user's terminal in command mode the System Programmer enters the command

```
VSS SYSPROG
```

and presses the return key. At this point the Command System processes the VSS command. TSSS subsequently connects the support system to the user's task and prints $ at the terminal; this is the TSSS invitation to enter TSSS commands.

For the purpose of this example assume that the System Programmer suspects that, after executing for some period of time, the user's PSW is being altered in such a way that processing continues but erroneous results are obtained. In an effort to confirm his suspicions the System Programmer enters the command string:

```
AT SUSPECT IF $R(10)=X'0A' DISPLAY $PSW
```

In this command string, he is using a loop counter included in the user's program to signal the expiration of a sufficient amount of time. Each time the point labeled SUSPECT is reached, TSSS will examine general purpose register 10. When that register contains the value ten, TSSS will print the current PSW at the terminal.

In discussing this example, only one task, the task to which the System Programmer is attached, is assumed, for the sake of simplicity. Not all of the several interfaces with the Resident Supervisor are fully detailed but the work which that body of code performs for the Time Sharing Support System is clearly indicated.

Operation begins with the user logged on and his task active and in command mode. The System Programmer enters the command:

```
VSS SYSPROG
```

1. This command is processed by the TSS/360 command system which issues the instruction: SVC 83. The execution

of this SVC triggers the processing which results in VSS activation.

2. The hardware interruption is received by the Resident Supervisor which recognizes it as an RSS interruption and passes control to the RSS SVC Interrupt Processor by loading a PSW from the System Table.

3. The RSS SVC Interrupt Processor recognizes the SVC code as the one for VSS activation and merely passes control to the VSS Command SVC Interrupt Processor.

4. The VSS Command SVC Interrupt Processor sets parameters locates the TSI and calls the RSS Interrupt Switching routine. The RSS Interrupt Switching Routine builds a duplicate TSI for the task and saves the current TSI in it. The routine chains the original and duplicate TSIs, sets the VSS active flag, dequeues all pending task interrupts from the original TSI and queues them on the alternate TSI, and returns control to the VSS Command SVC Interrupt Processor.

5. Next, the VSS Command SVC Interrupt Processor calls the Queue VSS Interrupt Routine which builds and enqueues an activate interrupt on the task's TSI. It accomplishes this by calls to the Resident Supervisor routines, Supervisor Core Allocation and Queue GQE on TSI. Having accomplished this, control is returned to the VSS Command Interrupt Processor.

6. The VSS Command SVC Interrupt Processor calls the Queue VSS Interrupt routine a second time at a second entry point to build and enqueue an external interrupt with a message control block (MCB) attached. By calls to Supervisor Core Allocation and Queue GQE on TSI the Queue VSS Interrupt module constructs an MCB and enqueues an external interruption GQE on the task's TSI. This MCB contains information which defines the TSP and the terminal to the system. Control is then returned to the VSS Command Interrupt Processor.

7. The VSS Command SVC Interrupt Processor exits to the Queue Scanner, which, finding no work to process, transfers control to the Dispatcher. The Dispatcher starts the task on a new time slice resulting in a task interruption, the VSS Activate interruption that was just queued by the Queue VSS Interrupt routine.

8. The occurrence of this interruption results in a call to the VSS Activate Interrupt Processor. This routine sets "VSS activation sequence in progress" and "VSS active" flags in the VSS Status Save Area, and links to the VSS Status Save Routine.

9. The VSS Status Save Routine saves the entire ISA and associated task status since its contents will be overlaid by subsequent processors. This routine returns control to the VSS Activate Interrupt Processor.

10. At this point, the Activate Interrupt Processor issues the LVPSW instruction which enables interrupts. The external interrupt enqueued by the VSS Command Interrupt Processor occurs at this point, resulting in an entry to the VSS External Interrupt Processor from the Dispatcher.

11. The VSS External Interrupt Processor tests the "VSS activation sequence in progress" flag in the VSS Status Save area and, finding it on, passes control to the VSS Activate Interrupt Processor at a secondary entry point.

12. The VSS Activate Interrupt Processor sets the "VSS activation sequence in progress" flag off and transfers the TSP terminal information from the ISA to the VSS Status Save Area. The routine then sets flags indicating that VSS is active and also indicating the mode of activation and exits to the Language Control Routine to invite input by printing $ at the terminal.

At this point VSS is active and, after the $ is printed, the System Programmer may begin issuing commands in an effort to locate the source of the trouble. In this example it was assumed that the TSP suspects that the user's PSW is being altered after a certain period of execution time. He, therefore, decides to allow the user's task to run for a short time and then to display the PSW to determine if, in fact, it has been changed. The following steps describe the processing from the point at which Language Control receives control from the VSS Activation Interrupt Processor. The function of Language Control on this, its first activation, is to invite input by printing "$" at the terminal. This involves the support system I/O routines and, therefore, a call is made to I/O Control.

13. The I/O Control Routine initializes the TSSS I/O Request Control Block (SIORCB), determines the proper access method to call (in this case the VSS Telecommunications access method), and

transfers control to it. (The SIORCB used by the Time Sharing Support System is analogous to the IORCB used by the other Time Sharing access methods.)

14. VSS Telecommunications access method builds a channel program to perform the I/O operation required (writing $ at the terminal), and transfers control to the VSS I/O Initiation/Posting routine.

15. The VSS I/O Initiation/Posting routine creates an IORCB from the information in the SIORCB created by the access method. It then issues the IOCAL macro instruction which causes the TSS/360 Supervisor to perform the actual I/O operation for it. Following the issuance of IOCAL, VSS I/O Initiation forces Time Slice End.

16. After a period of time, the I/O operation is completed (the $ has been printed at the terminal), and a channel interruption occurs. This interruption is received and enqueued on the scan table by the Interrupt Stacker. The Stacker then gives control the Queue Scanner.

17. The Queue Scanner dequeues the interruption GQE and passes it to the Channel Interrupt Queue Processor which enqueues it on the task's TSI as a VSS interruption.

18. After a time delay, during which, under normal operating circumstances, other tasks would receive a time slice, the Dispatcher gains control and, prior to setting the task in execution, calls Task Interruption Control (TIC).

19. TIC determines that an interruption is enqueued for the task and that the interruption is a VSS channel interrupt. TIC therefore switches PSWs so that the current PSW in the ISA directs the Dispatcher to the secondary entry point in VSS I/O Initiation/Posting.

20. When the Dispatcher regains control, it issues the LPSW instruction giving control to VSS I/O Initiation/Posting.

21. VSS I/O Initiation/Posting examines the ISA to determine the results of the I/O operation, and, finding the results to be satisfactory, exits to I/O Completion.

22. Following successful printing of the $, VSS I/O Completion transfers control to the I/O Calling routine -- Language Control.

At this point, VSS is prepared to accept command input from the TSP. The programmer enters the command string:

```
AT SUSPECT IF $R(10)=X'0A' DISPLAY $PSW
```

23. Language Control calls I/O Control to read the input device. (The input device terminal must be read to be able to process the command string.)

24. I/O Control sets up the SIORCB, determines that the Telecommunications access method is the correct one to use, and calls it.

25. The Telecommunications access method builds a read channel program, sets appropriate pointers and flags, and calls VSS I/O Initiation/Posting.

26. VSS I/O Initiation/Posting sets up the IORCB and issues the IOCAL macro instruction. Processing continues as in steps 16 through 22 above except for the fact that this is a read operation.

27. I/O completion returns to the I/O Editor following a read operation with TAM as the access method.

28. The I/O Editor converts the device standard character codes to EBCDIC, and exits directly to Language Control.

29. Language Control now has a command string in EBCDIC. In order to convert this to executable code, Language Control calls the Source to Polish module.

30. Source to Polish converts the input string to polish notation which is a string of symbols, literals, and operators that defines the operations to be performed and the order in which they are to be performed.

31. When Source to Polish is finished it returns to Language Control which immediately calls Scan Control.

32. Scan Control examines the Polish String, determines the proper keyword execution subroutine (in this case the AT Command Processor), and passes control to it.

33. The AT Command Processor saves the original instruction code located at the point where the AT SVC (SVC 80) is to be implanted, implants the AT SVC, stores the source statement, (i.e.,

the command string to be dynamically executed) at the end of the AT table, and returns control to the Scan Control routine with a return code of 4 and a zero in GPR 0.

34. The Scan Control routine returns a code of zero to the Language Control routine indicating that processing is complete and further input can be accepted.

35. Language Control requests further input by repeating steps 13 through 22 to print $ at the TSP's terminal. At this time the TSP has completed his input of diagnostic commands and simply issues the command:

    RUN START

    which causes the user's task to begin execution from its first instruction labeled START.

36. Processing of the RUN command is the same as for the AT command (steps 25 through 34 above).

37. The RUN Command Processor computes the start address at which processing is to be resumed and inserts that address as the current PSW in the VSS save area. Control is then returned to Scan Control with a return code of eight.

38. Scan Control returns control to the Language Control routine with an indication that a RUN command has been processed (RC=8).

39. Language Control converts the return code of eight to a return code of zero and returns to the VSS Activate Interrupt Routine.

40. VSS Activate passes control and the return code to the VSS Restore Status routine.

41. Restore Status restores the saved ISA and remotely executes SVC 82.

42. The execution of this SVC causes an entry to the RSS SVC Interrupt Processor via the Resident Supervisor Interrupt Stacker. The SVC Interrupt Processor transfers control to the VSS Exit routine (a real core module).

43. The VSS Exit routine calls the RSS Interrupt Switching routine, and passes it a deactivation indicator.

44. Interrupt Switching restores the task to its original state, i.e., the state prior to VSS activation, resetting the "VSS active" flag and restoring the original TSI. The routine then returns to VSS Exit.

45. VSS Exit stores the task status from the VSS paging buffer into the XTSI, and exits to the Queue Scanner.

46. The Queue Scanner, finding no work, calls the Dispatcher, which sets the task in execution.

Each time the instruction labeled SUSPECT is executed, VSS is reactivated by execution of the SVC. The first nine times this occurs, the IF clause returns a false indication and TSS/360 is reactivated after the instruction overlaid by the AT SVC is executed. The tenth time the AT SVC is encountered, the execution of the IF clause returns a true indication and the remainder of the command string is executed. The processing of these commands is described below:

47. The occurrence of the SVC interruption results in an entry to the TSS Interrupt Stacker. The Stacker recognizes the interruption as one belonging to TSSS and transfers control to the RSS SVC Interrupt Processor.

48. The RSS SVC Interrupt Processor determines that the VM AT Execution SVC Processor should receive control, and calls that routine.

49. The AT Execution SVC Processor provides the processing required to reactivate VSS. The first step in this reactivation is to call the RSS Interrupt Switching routine.

50. Interrupt Switching saves the input TSI by building a copy of it, dequeues all pending interruptions for the task, and returns to VM AT Execution.

51. VM AT Execution calls the Queue VSS Interrupt routine to enqueue a code 2 interruption on the task's TSI. When this is accomplished, Queue VSS Interrupt returns control to VM AT Execution.

52. VM AT Execution now calls the Queue VSS Interrupt routine at a secondary entry point to construct an MCB and enqueue an external interrupt. The MCB contains information which defines the terminal and TSP to the system. When control returns, VM AT Execution exits to the Queue Scanner.

53. Processing continues as in steps 7-12 above in order to reactivate VSS. At the end of the reactivation process, the VSS Activate Interrupt Processor,

recognizes a code 2 interrupt and exits to the VSS AT SVC Processor.

54. The AT SVC Processor locates the source string which was stored in step 35 above, reads the string into the Language Control input buffer setting the "input in storage" flag, and calls Language Control.

55. Language Control bypasses the read instruction to I/O Control and calls Source to Polish and Scan Control as above to convert the input string to executable form. Among the subroutines which Scan Control uses in this conversion is one called Operator Functions which performs such functions as comparison and subscripting. This subroutine also treats the IF command as an operator and makes the comparison specified. On each of the first nine passes through this loop the comparison fails and scan control ignores the remainder of the input string. On the tenth pass the comparison results in the setting of a "true" indicator and Scan Control continues scanning the polish string. If this is not the 10th time, step up to #69 and continue.

56. Scan Control next encounters the DISPLAY command and calls the DUMP and DISPLAY Command Processor to perform the necessary operations. DUMP/DISPLAY first calls VSS Real Core Access to get a page of real core.

57. VSS Real Core Access tests the get/put indicator, implants the proper SVC (SVC 65) at the beginning of the VSS paging buffer and executes it remotely. Note that VSS may not access real core directly, and that RSS must perform this function.

58. The SVC interruption is received by the Interrupt Stacker which issues the LPSW instruction to load the appropriate RSS PSW from the system table and transfer control to the RSS SVC Interrupt Processor.

59. The SVC Interrupt Processor sets the "activation in progress" indicator, performs an LRA on the addresses passed to it from VSS, and exits to the RSS SVC Service Processors routine.

60. The interrupt processor for SVC 65 sets up the proper parameters and calls the RSS Real Core Access routine.

61. RSS Real Core Access determines that the area to be displayed is in main storage and moves it into the RSS paging buffer. RSS Real Core Access then returns to the RSS SVC Service Processor for SVC 65.

62. The SVC Service Processor moves the requested page from the RSS paging buffer into the VSS paging buffer and exits to RSS Exit which deactivates RSS and restores control to TSS/360 via LPSW.

63. When the task within which VSS has been activated receives its next time slice, control is returned to the VSS Real Core Access routine at the next sequential instruction following the execution of the SVC 65. The requested page of real storage is in the VSS paging buffer and VSS Real Core Access then returns to the DUMP/ DISPLAY Command Processor.

64. DUMP/DISPLAY initializes an SIORCB to write the contents of the buffer at the terminal and calls I/O Control.

65. As in steps 13-22, the VSS I/O routines cause the contents at the buffer to be written at the terminal.

66. When the DISPLAY has been completed (the I/O system returns to the DISPLAY command processor), the DISPLAY command processor returns to Scan Control which returns to Language Control to invite new input.

67. Language Control, recognizing AT mode, does not invite new input but returns to the VSS AT SVC Processor.

68. The VSS AT SVC Processor, recognizing that no more ATs are to be processed at this time implants the original, overlaid TSS/360 instruction in a special CSECT and implants an SVC 84 immediately following it. It then sets the current PSW instruction counter (IC) to point to the new location and returns to the VSS Activate Interrupt processor with an implied RUN return code. The RUN return code is processed as shown in steps 25-34.

69. The current PSW now contains the address of the overlaid instruction. When the task within which VSS has been activated is dispatched, this instruction will be executed, and immediately thereupon the SVC 84 will be executed, causing reactivation of VSS as shown in steps 49-55.

70. The AT SVC Processor, recognizing an SVC 84 (a "return" SVC) will restore the PSW IC, remove the SVC 84 from the special CSECT and return a RUN return

code to the VSS Activate Interrupt Processor.

Now do steps 42-48

Control has been returned to TSS/360.

(Note, however, that the AT SVC is still there and will be executed each time it is encountered.)

UTILITY PROGRAMS

The utility programs are a group of stand-alone programs that support the time sharing system but operate outside of the system. The utility programs will run on a Model 67 in the standard PSW mode. The utility programs include:

- DASDI - Direct Access Storage Device Initialization. Initializes the IBM 2311 and IBM 2314 disk storage units in either SAM or VAM format, and the IBM 2301 drum storage unit in paging format; assigns alternate tracks, if any defective tracks are discovered, to the SAM formatted disks; and makes defective pages unavailable for VAM formatted disks.
- DUMP/RESTORE - transfers the contents of a direct access device to a tape or another direct access device of the same model. Capable of restoring the data from the tape to a direct access device.
- DIRECT ACCESS PRINT - prints the contents of direct access devices.
- STAND ALONE CORE DUMP - prints the contents of main storage.
- PATFIX - prints diagnostic and/or updates the Page Assignment Tables.

SYSTEM GENERATION AND MAINTENANCE

The generation and maintenance of IBM TSS/360 is a process by which a specific installation may initialize or alter its system with respect to its hardware configuration, task management requirements, and command language default options. The generation process is also used to initialize those system tables, control blocks, and data sets which are universal and common to all installations.

SYSTEM GENERATION

The system generation process is composed of three major parts:

- System Build (SYSBLD)
- STARTUP
- System Generation (SYSGEN) and supporting macro instructions

System Build (SYSBLD) consists of a SYSBLD PRELUDE and SYSBLD proper. The prelude is read in from the IPL volume by the Initial Program Load (IPL) hardware micro program. This micro program is is activated when the operator depresses the CPU console LOAD key. The sole function of the prelude is to read in SYSBLD proper.

SYSBLD is a resident, nonreenterable, standalone utility which operates outside of the TSS/360 environment. The primary function of SYSBLD is to set system tables with data required for the operation of STARTUP.

SYSBLD locates the operator's terminal and interrogates the operator for certain parameters. From the parameters given, SYSBLD initializes:

- The Pathfinding Tables
- The Symbolic Device Allocation Table
- The Configuration Control Block
- The User Table (SYSUSE)
- The Accounting Table (SYSACCT)
- The Catalog (SYSCAT)

The last function of SYSBLD is to reread the prelude, alter it, and rewrite it to the IPL Control Volume. The prelude is altered to read in the STARTUP module instead of SYSBLD. Finally, the operator is notified to perform an initial program load that causes STARTUP PRELUDE to be read in.

STARTUP

STARTUP, like SYSBLD, consists of a prelude and STARTUP proper. STARTUP is called by means of an initial program load, but it may be called later by the Reconfiguration routine of the Resident Supervisor. In this latter case STARTUP is used to perform a system restart following the occurrence of a system error (see "Error Handling").

The functions of STARTUP PRELUDE are:

- Determines the physical address of the IPL volume.
- Determines the physical address of the operator's terminal.

- Identifies its own CPU.
- Generates a list of partitioned or defective main storage pages.

The physical address of the IPL volume is either passed to STARTUP PRELUDE by the Reconfiguration routine or is stored as an I/O interrupt code by the IPL micro program.

STARTUP PRELUDE puts the CPU into the wait state and waits for an asynchronous interrupt caused by the operator pressing the REQUEST key on his terminal.

STARTUP PRELUDE then searches for a location into which to read the Configuration Control Block (CCB) data set. The CCB is a group of tables containing installation dependent information describing the hardware configuration. The main storage location of the CCB must be dynamically determined for the following two reasons. First, care must be taken not to overlay a Prefixed Storage Area (PSA). Second, it is not necessary that the storage units be assigned contiguously. The beginning address for each storage unit is independently set by a Floating Storage Address switch and, hence, any given address may not be addressable by the system.

In a multiple CPU environment, STARTUP PRELUDE determines which PSA is active (see "Prefixed Storage Area") and determines the identification of the CPU that is executing STARTUP PRELUDE . This is done by placing each CPU's identification into the CPU's primary and alternate PSAs, using normal addressing. STARTUP PRELUDE then directly addresses location zero and identifies the CPU and active PSA.

In a duplex or half-duplex environment, STARTUP PRELUDE does a configuration analysis to determine if any units are partitioned or if any configuration switches are invalid.

STARTUP PRELUDE then generates a list of partitioned or defective main storage pages, locates STARTUP proper on the IPL volume, reads in STARTUP Proper into consecutive pages, and transfers control to STARTUP proper.

A basic function of STARTUP proper is to link load Initial Virtual Memory and the Resident Supervisor. These components are contained on the IPL control volume. However, prior to the link load phase of STARTUP, the operator is asked whether he wishes to establish a library hierarchy for including installation supplied modules to the link loading process. If not, modules are link loaded from the IPL control volume according to load lists contained therein. (The operator is asked for the codes of those modules or functions to be excluded from loading -- the last byte of the last word of each entry in the load list contains a preassigned code). If the library hierarchy option is taken, the operator designates a private volume which contains the data sets to be used as libraries. Then the link load phase of STARTUP proper uses a library search analagous to that used by the Dynamic Loader as follows: Observing the order of library hierarchy, STARTUP proper locates the appropriate load list and then, for each CSECT in the load list, retrieves the module defining the named CSECT from the first library in the hierarchy which contains the CSECT. After the link load phase of STARTUP proper is completed, the operator is permitted to remove the private volume. The data sets used as libraries during STARTUP are called "delta" data sets. They are defined to be TSS/360 VPAM data sets with sequential members. All delta data sets to be used in a single STARTUP session must be completely contained within one private volume.

Startup link-loads control sections in the order in which they are named in the load list. In contradistinction to dynamically loaded modules, link-loaded control sections of like attributes are allocated virtual storage on the next available double word boundary whenever this will not cause a control section to cross a page boundary. Otherwise, control sections are allocated on page boundaries as in dynamic loading. Thus, the order in which names appear in the load list controls which control sections are contained within any IVM page. System efficiency can be enhanced by organizing the load list in such a fashion that the number of pages required for system functions (such as opening a data set) are minimized.

Additional functions of STARTUP proper include:

- Determines auxiliary device on which to place Initial Virtual Memory (IVM),
- Constructs a skeletal Shared Data Set Table (SDST).
- Constructs a skeletal XTSI which describes the status and location of IVM in its segment and page tables.
- Constructs Shared Page Tables for public CSECTs.
- Initializes the skeletal ISA.
- Checks to see if Interval Timer has been activated. If not, STARTUP prompts for that action.

- Writes the pages containing IVM PSECTs, the IVM Task Dictionary, the skeletal Shared Data Set Table, the skeletal ISA, and the skeletal XTSI out to the auxiliary paging device (usually a disk).
- Places slot and head numbers into XPT and XSPT for drum processing.
- Writes the pages containing IVM (read-only) CSECTS to the primary paging device (usually a drum).
- Constructs a storage Map for the Dynamic Loader.
- Initializes the Core Block Table.
- Sets the storage protection keys.
- Starts up other CPUs when they are present.
- Initializes the System Table.
- Creates the Main Operator task.
- Enqueues an Attention Interrupt GQE on the Main Operator task's TSI.
- Exits to the Resident Supervisor.

SYSTEM GENERATION MACRO INSTRUCTIONS

Once STARTUP completes its processing, an operable time sharing system exists and the following functions may be performed within the time sharing environment.

The balance of the system generation process is devoted to defining the specific system configuration and parameters. Once STARTUP has provided the basic system, the system manager logs on and may join other users. Specifically, he joins a system programmer whose duty is to define the configuration and system parameters.

By the use of specific macro instructions, the system programmer compiles a source module that defines:

- The number of channel control units
- The address of multiplexor and selector channels
- The number and model of storage units
- The characteristics of each CPU
- The type, model, and address of device control units
- Groups of similar devices connected to each channel
- The symbolic and actual addresses of operator consoles
- Command language default options
- The scheduling algorithm parameters
- The limit on size and number of tasks allowed to operate concurrently
- Virtual storage allocation and manipulation parameters

The last macro instruction, GENSCB, in this source module generates the system control blocks which contain the parameters specified in the other macro instructions. Using the object module produced from these source statements, the system programmer can now use the system maintenance procedure to update the system. The parameters or the new data sets specified are made a part of the system thereby replacing or augmenting portions of the existing system.

APPENDIX A: PROGRAM LOGIC MANUALS

This section contains a list of the TSS/360 Program Logic Manuals and identifies the manuals in which major system components are described.

System Control Blocks, Form Y28-2011

Resident Supervisor, Form Y28-2012

- Interrupt Stacker
- Queue Scanner
- Queue Control Subroutines
- Queue Processors
- Page Handling Subroutines
- I/O Service Subroutines
- Storage Allocation and Release Subroutine
- Dispatcher
- Major Error Recovery Routines

Command System Form Y28-2013

- Command Controller
  - Command Analyzer and Executor (CA&E)
  - SCAN
  - GATE
  - User Prompter
  - Virtual Memory Task Initialization (VMTI)
- Interruption Processors
- Command Routines
- System Operation and Administrator Routines
- Language Processor Control (LPC)

Program Control System, Form Y28-2014

System Generation and Maintenance, Form Y28-2015

- SYSBLD PRELUDE
- SYSBLD
- STARTUP PRELUDE
- STARTUP

Access Methods, Form Y28-2016

- General Services
  - GETPOOL, FREEPOOL, GETBUF, FREEBUF
- Open Processing
- Close Processing
- BSAM
- QSAM
- MSAM
- TAM
- RTAM
- MTT
- IOREQ
- VAM
- Posting

System Service Routines, Form Y28-2018

- Catalog Services
- External Storage Allocation
- Device Management
- Virtual Storage Allocation
- Symbolic Library Services
- Control Section Store

FORTRAN IV, Form Y28-2019

FORTRAN IV Library, Form Y28-2020

Assembler, Form Y28-2021

Time Sharing Support System, Form Y28-2022

Linkage Editor, Form Y28-2030

Dynamic Loader, Form Y28-2031

- The Loading Process
- The Unloading Process
- Loader LOGOFF routine
- Library Search Hierarchy Maintenance

Independent Utilities, Form Y28-2039

- Direct Access Storage Device Initialization
- Dump Restore
- Core Dump
- Direct Access Dump
- VAM Utility Program

Task Monitor, Form Y28-2041

- Interrupt Processors
- Queue Linkage Entry
- Scanner-Dispatcher
- Specify Interrupt Routine
- Delete Interrupt Routine

On Line Test Control Program, Form Y28-2042

Operator Task Bulk I/O, Form Y28-2047

- Operator Task
- Batch Monitor
- Bulk I/O Preprocessor
- Bulk I/O Task
- Operator Commands
- Batch Work Commands

PL/I Compiler, Form Y28-2051

PL/I Library, Form Y28-2052

APPENDIX B: CONTROL BLOCK SUMMARY

This section contains a list of the important system control blocks and a brief description of each.

Figure 109 shows schematically the flow of information among control blocks.

RESIDENT SUPERVISOR CONTROL BLOCKS

ASAT--Auxiliary Storage Allocation Table: Contains lock byte, symbolic device address for first drum directory, pointer to next drum directory, disk symbolic device address and pointer to disk directory.

AST--Auxiliary Segment Table: In XTSI for each task. Indicates classification of segment (shared or private) and location of segment on auxiliary storage.

CBT--Core Block Table: Main mechanism for keeping track of main storage use and availability. One entry for each main storage page (4096 bytes), set up in available and pending chains. Position of each entry references main storage page. Each entry (20 bytes) contains TSI pointer, virtual memory address and chain pointers to additional blocks for task. Startup assigns pages to Resident Supervisor which are not available to user Core Allocation. Those pages temporarily used (assigned by Supervisor Core Allocation) are assignable. This table is maintained by User Core Allocation and Release.

DAIB--Direct Access Interface Block: Maximum size, one page. Serves as interface between Page Direct Access Queue and Page Direct Access Interrupt subroutines. In supervisor storage, contains one DAIB for each paging GQE and exists for duration of paging operation. Contains save area, seek and search arguments, PCB pointers, and actual channel program.

DICB--Drum Interface Control Block: One page per drum. A work area for the Drum Queue Processor.

GQE--Generalized Queue Entry (64 bytes): Represents a unit of work to be performed by supervisor. Summary of information pertinent to five types of interruptions. Contains pointer to TSI; pointers to GQE chain; pointers to PCB, IORCB, GQE movement information; symbolic device address.

IOPCB--I/O Paging Control Block: A parameter list which provides communication link between VAM and the Supervisor. Contains PGOUT SVC, number of entries in external storage address list, virtual storage address, external storage address list, symbolic device address and external page number.

IORCB--Input/Output Request Control Block: Basic control element for any I/O operation. Maximum size: 1920 bytes, wholly within one page. Contains user ID, symbolic device address, the channel program, pseudo-CCW list (with logical addresses), self-buffer area or page list, pointer to DCB. The I/O CALL SVC is imbedded within the IORCB to force it to be in main storage when the SVC is executed. Transferred between virtual storage and the Resident Supervisor.

MCB--Message Control Block: Varies from 16 to 1920 bytes. For inter-task communication, used by VSEND SVC users: Device Management, Batch Monitor, CLI. Contains length of message, type of message, whether reply is expected or whether this is a reply, task ID of sender and receiver, and contains VSEND SVC. If reply is expected, contains address of MEB (Message Event Control Block).

Pathfinding and Reconfiguration Tables:
Channel Table
Control Unit Table
Device Group Table
Multiplexor Channel Table
Symbolic to Actual Address Conversion Table
Selector Channel Table

PCB--Page Control Block (64 bytes): Controls paging. Pointed to by GQE. Contains virtual memory address, external address, main storage address, flags, PCB chain address.

PSA--Prefixed Storage Area (4096 bytes): Contains: pointer to current task (for that CPU), PSW area, CPU Logout area, Channel Logout area, Supervisor Private Working Storage, CPU status table, Damage Report (SERR), Intercom routine, Error Recovery Control Table (SERR), SIPE and CRM work area.

PGT--Page Table: One per segment table entry. Describes page locations in storage.

RSPI--Resident Shared-Page Index: Records the status of SPTs. Indicates location in storage, in-transit condition (table coming in or going out), length of table, and indicates GQE chain of TSIs waiting for an in-transit condition to end. Used by

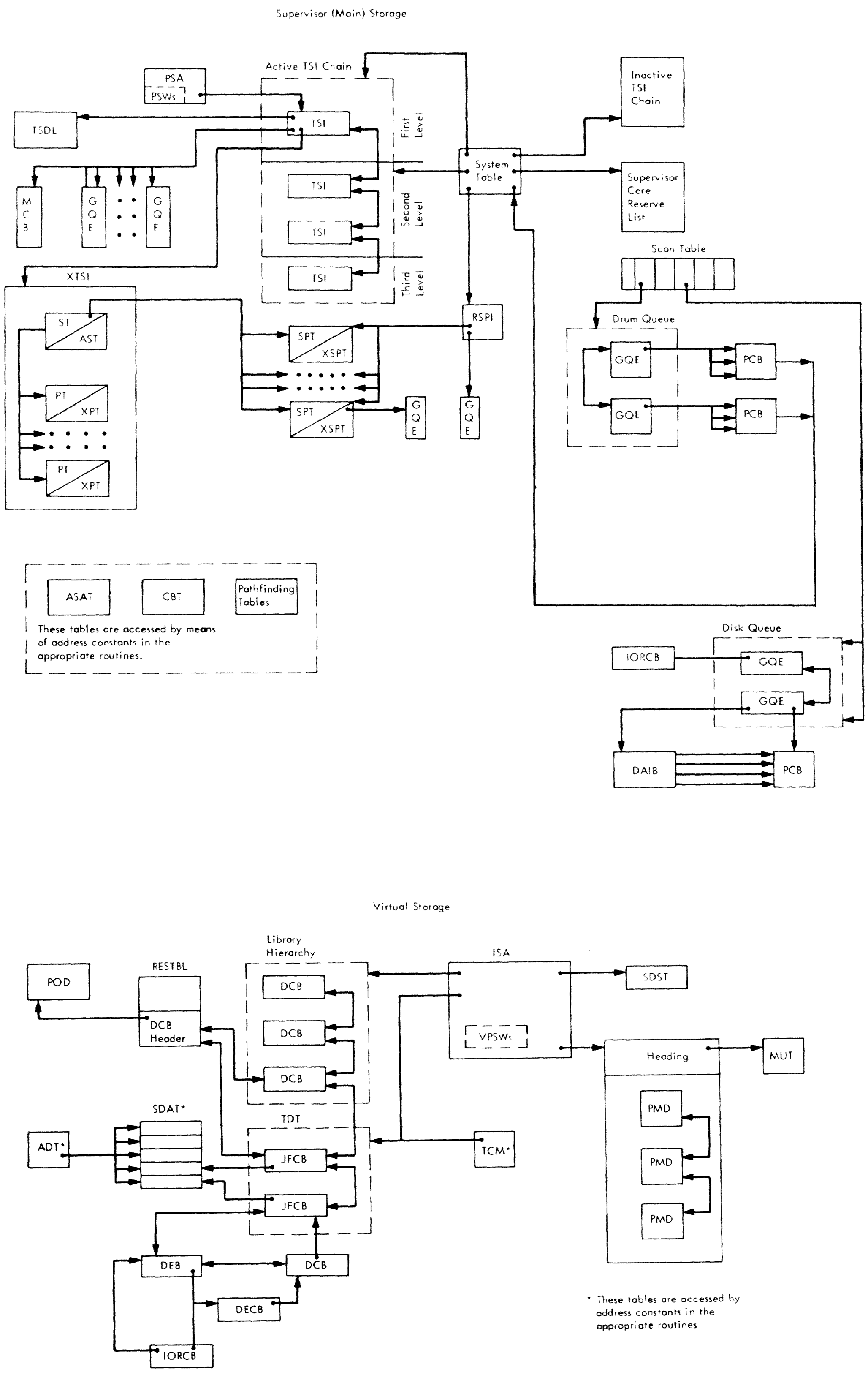


Figure 109. Relationship Between Control Blocks

ADSPG, Page Posting, Page Turning and Timer Interrupt Processor.

RSV--Reserve List of Pages Table (20 bytes): Contains up to five addresses of main storage pages held in reserve for supervisor core allocation.

SCANT--Scan Table: A common anchor point for all GQEs. Queue Summary. Pointers to first and last GQE and location of queue processor. Flags: suppress (don't work on GQEs), processor lock byte (one CPU at a time). 16 bytes per entry, one entry for each device or supervisor facility.

SPT--Shared Page Table: Basic element for hardware lookup of shared pages. Entries identical to page table entries.

SGT--Segment Table: Groups of sixteen 4-byte entries. There is one entry for each segment of the program. Each entry contains a pointer to the beginning of a page table and the count of the number of entries in that page table.

STE--Schedule Table: Contains fields which control the order in which tasks are brought into the dispatchable list and conditions under which the task must leave.

SYS--System Table: Contains various system and installation parameters. Anchor point for the chain of TSIs (pointer to first task in active list) and for RSPI. Contains time-of-day clock. Parameters and pointers used by scheduling algorithm, such as commutator and front wall, reside here.

TSI--Task Status Index (128 bytes): Contains userid, privilege, priority, terminal address, pointers to XTSI, next TSI in chain, GQEs for task interruptions, task device list, SYSIN, and SYSOUT device addresses. Flags: ready, page wait, await, in execution, time-slice-end. Also, contains XTSI external location; lock byte to prevent two CPUs in single task; count of pages used last time slice, task ID (TID), BSN, I/O awaiting paging pointer, quantum counter.

XPT--External Page Table: Describes auxiliary storage. Contains 2301, 2311 or 2314 storage addresses for each virtual storage page. Contains page status, unprocessed flag, page hold flag, hold count.

XSPT--External Shared Page Table (12-byte entries): Contains external location of page, write count, GQE chain address, and flags: update in place, preferred paging device, page changed, assigned, shared, unprocessed, held, auxiliary storage, protect class and in-transit indications.

XTSI--Extended Task Status Index: Contains segment and page tables, estimated run time, save area for all registers, current PSW.

VIRTUAL MEMORY CONTROL BLOCKS

Available Device Table: Used by Device Management as a directory of SDAT entries.

BWQ--Batch Work Queue: Stores requests for nonconversational tasks until they can be initiated and contains record of active nonconversational tasks.

Catalog SBlock: The catalog SBlock is the basic unit of storage within the catalog data set. SBlocks are chained together to make up indexes, generation indexes, data set descriptors, sharing descriptors, or sharer lists. Data is retrieved from the catalog via catalog services in the form of SBlocks.

DCB--Data Control Block: Major means of communication between Data Management and the user. Also provides the principal means of achieving device independent coding.

DCB (VAM): Composed of five parts. DCB common; transfer list for macros common to VAM; organization independent working storage, extended VISAM transfer list or extended VSAM working; and extended VISAM working storage.

DCB (VISAM)--Indexed Sequential Data Control Block: Primary user communication with data set. Contains DCB macro parameters for VISAM: (V or F), logical record length (max. 4000), record key position (RKP), key length (KEYLEN) max. 255, address of end-of-data-set routine (EDDAD), address of synchronous error exit (SYNAD) routine, percent pad to be left in page.

DEB--Data Extent Block: This table provides the required attributes of both a data set and the device on which the volume for that data set resides. It also contains pointers to other control blocks associated with the data set (CHATDT, CHADCB, and unchecked CHADECS). If a direct access volume is used, then it contains CHADEB. Also contains information about the volume extents.

DECB--Data Event Control Block: The DECB is a table of information pertaining to the status of an I/O operation. Information in the DECB is set by macro supplied parameters and the posting routine. Information in the DECB is used by the problem program and by the Read/Write, check and control (CNTL) routines.

DSCB--Data Set Control Block: Used to describe data sets. Has the following types:

Types 1 and 3 Data Set Control Blocks make up that part of the volume Table of Contents that describes the residence of SAM data sets.

Type 4 Data Set Control Block describes the Volume Table of Contents and also contains the device constants for the volume it resides on.

Type 5 DSCBs are used for direct access device space management (DADSM). They contain information of the availability of SAM volume.

Types E and F Data Set Control Blocks compose that part of the volume used to describe the residence of VAM data sets.

Hash Tables: Privileged and Nonprivileged System Hash Table and User Hash Table: The Hash Tables contain the heads of a number of search chains which expedite the locating of a specific external name. The tables are indexed by a hash function derived from the name to be found.

ICB--Interrupt Control Block: The user's means of specifying the interruption information needed to make an interruption handling routine available to the Task Monitor. Built by the SPEC, SSEC, SEEC, SAEC, STEC, and SIEC macro expansions. The ICB can point to a Communications Area (COM) (in the user virtual storage) when parameters and data are presented to the interrupt handling routine represented by this ICB. The ICB is made available to the Task Monitor via the SIR macro, and a Request Entry is placed in the ITB as a result of SIR. The ICB is a part of the problem program's virtual storage.

ISA--Interrupt Storage Area: The task's analogous "PSA" storage area. (Segment 0, Page 0). Contains old and new VPSWs for each of the six types of task interruptions. Contains three save areas and Virtual Memory Allocation tables.

ISD--Internal Symbol Dictionary: This table is created by the language processors and used by the Program Checkout Subsystem to process checkout statements with the user's internal symbols.

ITB--Interrupt Table: A block of virtual storage built up by the Task Monitor as it creates REs, and QEs for interruption processing. The size of the ITB depends on the number of REs and QEs that it contains.

JFCB--Job File Control Block: Created by DATADEF. Contains volume numbers and other information about a data set for use by the access methods and volume mounting routines.

LDS--Line Data Set Format (VISAM): Maximum 132 chars./record. Format: 4 bytes-length of record, 7 bytes-line no., 1 byte-card reader/ keyboard, 120 bytes-text.

MAP--Memory Map Table: Contains an entry for each control section involved in allocation. The entry contains the virtual storage address of the control section origin and the virtual storage address of the corresponding control section dictionary (CSD). MAP is used to locate the PMD of modules which have issued loads and unloads.

MEB-- Message Event Control Block: Controls reply to an intertask message in conjunction with MCB. Inbedded AWAIT or TWAIT SVC is remotely executed when task wants to delay processing until message is received. Also contains event completion bit, receiving task ID, sending task address of MCB.

MSG--System Message Record: Describes logical record format of VISAM SYSMSG data set (a collection of all system messages issued by Command Language System to users). Max. 262 bytes. Accessed by MSGWR routine, which can insert variable text.

PAT -- Page Assignment Table: Indicates the availability of pages on a VAM volume. The pages may be available, assigned, or in error.

PMD--Program Module Dictionary: Each PMD consists of one module heading plus as many Control Section Dictionaries (CSDs) as there are control sections in the module. Address pointers in the module are initially relative to the beginning of the module itself, except where otherwise specified.

POD--Partitioned Organization Directory: Located in user's virtual storage. Relates member names to the position of members within the data set and defines each member's attribute.

PVT -- Public Volume Table: A list of all devices in the system which have been designated as public devices.

QE--Queue Entry: Built by the Queue Linkage Entry routine in the Task Monitor each time an interruption occurs and there is a routine defined to handle the interruption. The QE is queued on the Request Entry for that routine. The QE contains the necessary interruption information from the VPSW and the sense and status information from the ISA required by the Task Monitor's

Scanner-Dispatcher at dispatch time. Some of the information in the QE is moved to a user defined Communication Area (COM) at dispatch time so that he may analyze the conditions and status at the time of the interrupt. The QE represents the occurrence of an interruption of the type specified in the RE to which the QE is attached (chained).

RDS -- Region Data Set: Maximum 256 characters/record. Format: 4 byte-length of record, 1 byte-card reader/keyboard, 7 byte key length, 0-247 byte region name, unused bytes-text.

RE--Request Entry: Built by the SIR (Specify Interrupt Routine) module in the Task Monitor. Each time an ICB (Interrupt Control Block) is made available to the system via the SIR macro instruction, a Request Entry is built and queued on the appropriate entry in the Task Monitor's Interrupt Table (ITB). The RE contains all necessary information regarding the priority, status and mode of operation for the routine specified in the ICB. The RE is the means of placing the ICB data into the Task Monitor ITB and is dynamic with the task. ICBs on the other hand, are a static part of the problem program. The RE points to the QEs on its chain and contains the priority and masking conditions of the interruption handling routine the RE represents.

RESTBL--Relative External Storage Correspondence Table: Variable length. Contains information on external location of VAM data set pages. Created at open time from DSCBs.

RESTBL--Relative External Storage Correspondence Table Header: Contains information describing RESTBL itself. Pointers to beginning and end of data set and pointer to end of assignable pages.

Entries (4 bytes): One entry per VAM data set page. Each entry contains flags, device and external page number (if page defined). Flags: not in use; no page assigned.

RQU--Request Queue: Used by Device Management to suspend a task requiring an unavailable device until device becomes available.

SAT--System Accounting Table: Contains the accumulated CPU time used by tasks. A separate record is kept for all tasks that log on under the same charge number and user identification. The SAT resides on a system residence volume as a VISAM data set with a key of charge number, userid. Entries are added by the Accounting Routine the first time it encounters a new charge number, userid combination. Updates are made by the Accounting Routine at the completion of each task.

SDAT--Symbolic Device Allocation Table: Describes the characteristics and status of each separately allocable I/O device in system. Consists of an 8-byte header and a variable number of 64-byte entries.

SDST -- Shared Data Set Table: Used for controlling the use of shared data sets and shared data set members. Contains count of current users and the names of data sets.

SLX -- Symbolic Library Index: For retrieval of information from a symbolic library. DDname-INDEX. Sequential organization. Contains names and aliases of all parcels in the associated library. For example, an SLX may contain a list of macro names (GETMAIN, GET, PUT, etc.) and pointers to the first statement of macro dfinition in the library associated with the SLX.

TCM -- Task Common Table: Contains those system values referenced in a single task by more than one Command Language object module. Task Common is a control section (PSECT). It has user read only protection status (storage key B) which implies that nonprivileged routines may read the page but cannot write into it, while privileged routines may do both reading and writing. It is necessary that TCM remain in virtual storage from LOGON to LOGOFF time. For this reason, its control section definition is included as part of task initiation.

TDT -- Task Data Definition Table: The TDT contains one JFCB for each data set in use by a particular task.

TDY -- Task Dictionary Table: Contains information needed to load or unload modules in a particular task. Contains headings, two hash tables, the Memory Map Table, and a PMD for each module loaded. The TDY is initialized by STARTUP and maintained by the Dynamic Loader.

User Table: VISAM data set describing attributes of all legal users. Contains one entry for each user joined to the system. The entries are in userid sequence and are variable in length, up to a maximum of 256 bytes. Entries are added by the JOIN Command and removed by the QUIT Command (except for the entries for the main system operator and the system manager, which are created at System Build time).

VPSW -- Virtual Program Status Word: For each task, analogous to machine PSW. Represents state of machine when task was interrupted. Contains logical instruction address, instruction length, interrupt code, storage keys and task masks.

This appendix lists all the modules in the Time Sharing System for which descriptive flowcharts exist and gives the form number of the PLM in which each flowchart can be found. The modules are listed alphabetically by module ID. See Appendix A for the PLM or system component name associated with the form number.

CEAAF	Queue GQE on TSI.............	Y28-2012
CEAAG	Start I/O....................	Y28-2012
CEAAJ	Dequeue I/O Requests.........	Y28-2012
CEAAL	Purge........................	Y28-2012
CEAAQ	Paging Failure Recovery......	Y28-2012
CEAAS	Alternate Path Retry.........	Y28-2012
CEAAV	Same Path Retry..............	Y28-2012
CEAA0	IOCAL........................	Y28-2012
CEAA1	Pageout Service..............	Y28-2012
CEAA2	Task Interrupt Control.......	Y28-2012
CEAA3	I/O Device Queue Processor...	Y28-2012
CEAA4	Channel Interrupt Processor..	Y28-2012
CEAA5	Pathfinding..................	Y28-2012
CEAA6	Page Direct Access Queue Processor....................	Y28-2012
CEAA7	Page Direct Access Interrupt Subroutine..................	Y28-2012
CEAA8	Page Drum Queue Processor....	Y28-2012
CEAA9	Page Drum Interrupt Processor....................	Y28-2012
CEABA	RJE Asynchronous I/O Interrupt....................	Y28-2012
CEABB	RJE Synchronous I/O Interrupt....................	Y28-2012
CEABC	RJE Line Control.............	Y28-2012
CEABE	External Machine Check Interrupt Processor..........	Y28-2012
CEAHQ	SVC Queue Processor..........	Y28-2012
CEAHQA	Add Pages....................	Y28-2012
CEAH2	Add Shared Pages.............	Y28-2012
CEAH8	Check Protection Class.......	Y28-2012
CEAIC	Inter-CPU Communication......	Y28-2012
CEAIR	Recovery Nucleus.............	Y28-2012
CEAIS	System Error Processor.......	Y28-2012
CEAKD	Dispatcher...................	Y28-2012
CEAKE	Entrance Criteria............	Y28-2012
CEAKI	Internal Scheduler...........	Y28-2012
CEAKR	Create Real Time Interrupt...	Y28-2012
CEAKT	Timer Interrupt Queue Processor....................	Y28-2012
CEAKZ	Rescheduling.................	Y28-2012
CEAJI	Interrupt Stacker............	Y28-2012
CEAJQ	Queue Scanner................	Y28-2012
CEAJSF	Set Suppress Flag............	Y28-2012
CEAL1	Core Control Subroutines.....	Y28-2012
CEAMC	Task Initiation..............	Y28-2012
CEAMD	Delete TSI...................	Y28-2012
CEAMP	Page Posting.................	Y28-2012
CEANA	Program Interrupt Queue Processor....................	Y28-2012
CEANB	User Core Allocation.........	Y28-2012
CEANG	Segment Block Remover........	Y28-2012
CEAP7	AWAIT SVC Processor..........	Y28-2012
CEAQ7	Connect Segment to Shared Segment......................	Y28-2012
CEAR4	Terminal SVC Processor.......	Y28-2012
CEASS	TSS Dynamic Status...........	Y28-2012
CEAS7	Set Real Time Interval.......	Y28-2012
CEATC	Terminal Control Subprocessor.................	Y28-2012
CEBCP		

	Core Dump	Y28-2039
CEBDI	Direct Access Storage Device Initialization (DASDI)	Y28-2039
CEBDP	Direct Access Dump (DADUMP)	Y28-2039
CEBDR	DUMP/RESTORE	Y28-2039
CEBVP	VAM2UT	Y28-2039
CEHAC	RSS Channel Interrupt Processor	Y28-2022
CEHAD	RSS I/O Interrupt Processor	Y28-2022
CEHAE	RSS External Interrupt Processor	Y28-2022
CEHAP	RSS Program Interrupt Processor	Y28-2022
CEHAQ	TSP Asynchronous Interrupt Processor	Y28-2022
CEHAS	RSS SVC Interrupt Processor	Y28-2022
CEHBD	RSS Disconnect	Y28-2022
CEHBE	RSS Exit	Y28-2022
CEHBL	RSS Loader	Y28-2022
CEHBT	RSS External Page Location Address Translator	Y28-2022
CEHBU	RSS Unloader	Y28-2022
CEHCA	RSS Real Core Access	Y28-2022
CEHCB	RSS Virtual Memory Access	Y28-2022
CEHCC	RSS Inter-CPU Communication	Y28-2022
CEHCF	Find TSI	Y28-2022
CEHCH	RSS Status Save	Y28-2022
CEHCM	RSS Message	Y28-2022
CEHCQ	Queue VSS Interrupt	Y28-2022
CEHCS	RSS Interrupt Switching	Y28-2022
CEHDA	Virtual Memory AT Execution SVC Processor	Y28-2022
CEHDE	VSS Exit	Y28-2022
CEHDL	LOGON RSS/VSS SVC Processor	Y28-2022
CEHDR	RSS SVC Service Processor	Y28-2022
CEHDV	VSS Command SVC Processor	Y28-2022
CEHEA	RSS I/O Control	Y28-2022
CEHEB	RSS I/O Initiation	Y28-2022
CEHFA	RSS Direct Access Device Access Method	Y28-2022
CEHFB	RSS Console Access Method	Y28-2022
CEHFC	RSS Sequential Access Method.	Y28-2022
CEHFD	RSS Telecommunications Access Method	Y28-2022
CEHFE	RSS I/O Editor	Y28-2022
CEHGA	RSS Direct Access Device Error Recovery	Y28-2022
CEHGB	RSS Console Error Recovery	Y28-2022
CEHGC	RSS Sequential Access Device Error Recovery	Y28-2022
CEHGD	RSS Telecommunications Error Recovery	Y28-2022
CEHGE	RSS Error Scan and Recovery	Y28-2022
CEHGF	RSS Storage Print Routine	Y28-2022
CEHHA	RSS I/O Completion	Y28-2022
CEHJA	RSS AT SVC Processor	Y28-2022
CEHJF	RSS $AT/$PATCH Format	Y28-2022
CEHJH	RSS $STATUS/$TASK Format	Y28-2022
CEHKA	RSS AT Command Processor	Y28-2022
CEHKC	RSS Collect Command Processor	Y28-2022
CEHKD	RSS DUMP/DISPLAY Commands Processor	Y28-2022
CEHKE	RSS DEFINE Command Processor	Y28-2022
CEHKL	RSS CALL/END Commands Processor	Y28-2022
CEHKM	RSS DISCONNECT Command Processor	Y28-2022
CEHKN	RSS RUN Command Processor	Y28-2022
CEHKP	RSS PATCH Command Processor	Y28-2022
CEHKQ	RSS QUALIFY Command Processor	Y28-2022
CEHKR	RSS REMOVE Command Processor	Y28-2022
CEHKS	RSS SET Command Processor	Y28-2022
CEHKT	RSS STOP Command Processor	Y28-2022
CEHKW	RSS CONNECT Command Processor	Y28-2022

CEHLA	RSS Operator Functions	Y28-2022
CEHLC	RSS Language Control	Y28-2022
CEHLL	RSS Literal Resolution	Y28-2022
CEHLP	RSS Source to Polish	Y28-2022
CEHLS	RSS Scan Control	Y28-2022
CEHMA	RSS Address to Symbol Resolution	Y28-2022
CEHMM	RSS Memory Map Format	Y28-2022
CEHMS	RSS Symbol Resolution	Y28-2022
CEHWS	VSS Symbol Resolution	Y28-2022
CEIAA	STARTUP	Y28-2015
CEIAP	SYSBLD/STARTUP Prelude	Y28-2015
CEIDA	SYSGEN	Y28-2015
CEIFA	SYSBLD	Y28-2015
CEKAB	Extract Source Character (ESC)	Y28-2019
CEKAC	Statement of Identification (SID)	Y28-2019
CEKAD	Phase 1 Main Loop (PH1M)	Y28-2019
CEKAE	Assemble Components (ACOMP)	Y28-2019
CEKAF	Array Dimension SpecificationsProcessor	Y28-2019
CEKAG	Subscript Processor (SUBS)	Y28-2019
CEKAH	Initial Value Data Specification Processor (IDATA)	Y28-2019
CEKAI	Expression Processor (EXPR)	Y28-2019
CEKAJ	Statement Label Processor (LABEL)	Y28-2019
CEKAK	Assignment Statement Processor (EQUA)	Y28-2019
CEKAL	END Statement Processor (END)	Y28-2019
CEKAM	External Statement Processor	Y28-2019
CEKAN	Conversion Subroutine (CNVRT)	Y28-2019
CEKAQ	GO TO Statement Processor (GOTO)	Y28-2019
CEKAR	IF Statement Processor (IF)	Y28-2019
CEKAS	Type Statements Processor (TYPE)	Y28-2019
CEKAU	DIMENSION Statement Processor (DIMN)	Y28-2019
CEKAV	COMMON Statement Processor (COMM)	Y28-2019
CEKAX	Executable Statements, Pass 2 (EXEC2)	Y28-2019
CEKAY	EQUIVALENCE Statement Processor (EQUI)	Y28-2019
CEKAZ	DO Statement Processor (DO)	Y28-2019
CEKBA	Begin Loop Processor (BGNLP)	Y28-2019
CEKBB	End Loop Processor (ENDLP)	Y28-2019
CEKBC	ASSIGN Statement Processor (ASSI)	Y28-2019
CEKBD	File Control Statement Processor (FCON)	Y28-2019
CEKBE	Input/Output Statement Processor (RWIO)	Y28-2019
CEKBF	FORMAT Statement Processor (FORM)	Y28-2019
CEKBG	PAUSE, STOP, RETURN Statement Processor (PSR)	Y28-2019
CEKBH	NAMELIST Statement Processor (NAML)	Y28-2019
CEKBI	BLOCK DATA Statement Processor (BLDA)	Y28-2019
CEKBK	Statement Function Definition (SFDEF)	Y28-2019
CEKBL	Statements Function Expansion (SFEX8)	Y28-2019
CEKBM	DATA Statement Processor (DATA)	Y28-2019
CEKBN	IMPLICIT Statement Processor (IMPL)	Y28-2019
CEKBQ	Fallthrough Determination (FALTH)	Y28-2019
CEKBS	Subprogram Entry Statement Processor (SUBE)	Y28-2019
CEKBT	Subprogram Entry Statements, Pass 2 (SUBE2)	Y28-2019
CEKBV	CALL Statement, Pass 2 (CALL2)	Y28-2019
CEKBW		

I/O List
Processor (IOLST)............ Y28-2019
CEKBX
Function Classifier (FNCLS).. Y28-2019
CEKBY
Library Function Selector
(LIBN)....................... Y28-2019
CEKCA
Diagnostic Message Generator
(ERR)........................ Y28-2019
CEKCB
Constant Arithmetic
Subroutine (ARITH)........... Y28-2019
CEKCC
Label String Processor
(LBSTR)...................... Y28-2019
CEKCD
Format Label Processor for
I/O Statements (FLABL)....... Y28-2019
CEKCE
Read Transfer Processor
for I/O Statements (RTRAN)... Y28-2019
CEKCF
FORMAT or NAMELIST
Processor (FNAME)............ Y28-2019
CEKCG
Term Processor (TEMPRO)...... Y28-2019
CEKCH
File Real Constant (FLRC).... Y28-2019
CEKCI
Insert Variable in Symbol
Table (IVST)................. Y28-2019
CEKCJ
Check Limits (CKLIM)......... Y28-2019
CEKCL
Initial Value Processor
(IVAL)....................... Y28-2019
CEKCN
Decimal to Binary Integer
Conversion (ICNV)............ Y28-2019
CEKCP
Decimal to Floating Binary
Conversion (FCNV)............ Y28-2019
CEKCR
Actual Argument Service
Routine (AARG)............... Y28-2019
CEKCS
Constant Arithmetic
Interrupt (CHKINT)........... Y28-2019
CEKJB
Process Label References
and Definitions (FSCAN)...... Y28-2019
CEKJC
Memory Assignments for
Variables (VSCAN)............ Y28-2019
CEKJD
Label Reference Processor
(RTN1)....................... Y28-2019
CEKJE
Label Reference Processor
(LAB)........................ Y28-2019
CEKJH
Diagnostic Message
Generator (DX)............... Y28-2019
CEKKA
Acquire Entry From Compute
and Removal Table............ Y28-2019
CEKKB
Polish Expression Generator
Routine...................... Y28-2019
CEKKC
End Loop PRF Entry Routine... Y28-2019
CEKKE
Expression Scan Routine...... Y28-2019
CEKKF
Push Down Primitive Operand
Routine...................... Y28-2019
CEKKG
Variable Compute Point and
Remove Level Routine......... Y28-2019
CEKKH
Trial File Manipulation
Routine...................... Y28-2019
CEKKI
Expression Removal and
Commonality Determination
Routine...................... Y28-2019
CEKKJ
Check Commonality............ Y28-2019
CEKKK
Establish Common Expression
Routine...................... Y28-2019
CEKKL
Operand List Expression
Formation Routine............ Y28-2019
CEKKM
Subscript Expression Revision
Routine...................... Y28-2019
CEKKN
Canonical Form Routine....... Y28-2019
CEKKO
Save Popularity Counts for
Register Assignment.......... Y28-2019
CEKKP
Search and Insert Triads..... Y28-2019
CEKKR
Phase 3 Master
Control Routine.............. Y28-2019
CEKKU
PRF Processing Routine....... Y28-2019
CEKKV
Begin Loop 1 PRF Processor... Y28-2019
CEKKW
Begin Loop 2 PRF Processor... Y28-2019
CEKLA
Label Common Expressions..... Y28-2019
CEKLB
File Constant and Covering
Adcon........................ Y28-2019
CEKLD
Expunge a Removable
Subexpression................ Y28-2019
CEKLE
File CRT Entries............. Y28-2019
CEKLF
Copy and Edit an Expression.. Y28-2019
CEKLI
Loop Test-Expression
Generator.................... Y28-2019
CEKMA
Real Divide Generator
(RDIV)....................... Y28-2019
CEKMB
Real Multiply
Generator (5RMUL)............ Y28-2019
CEKMC

	Real Plus Generator	Y28-2019
CEKMD	Integer Divide Generator (IDVDE)	Y28-2019
CEKME	Integer Multiply Generator (IMPLY)	Y28-2019
CEKMF	Integer Plus Generator (IPLUS)	Y28-2019
CEKMG	Complex Plus Generator (CPLUS)	Y28-2019
CEKMH	Relational Expression Generator (RLTNL)	Y28-2019
CEKMI	Logical Expression Generator (ANDOR)	Y28-2019
CEKMJ	Equation PF Entry Processor (EQUAT)	Y28-2019
CEKMK	External Function Generator (FUNC)	Y28-2019
CEKML	Expression Tree Builder (TRBLD)	Y28-2019
CEKMM	Make Initial Assignment to General Register (ASAR)	Y28-2019
CEKMN	Make SYNONYM Assignment to General Register (ASARS)	Y28-2019
CEKMP	Make Synonym Assignment to Floating Register (ASFRS)	Y28-2019
CEKMQ	Select Floating Register (SELFR)	Y28-2019
CEKMR	Search General Registers (FNDAR)	Y28-2019
CEKMS	Search Floating Registers (FNDFR)	Y28-2019
CEKMT	Find Temporary Storage (FNDWS)	Y28-2019
CEKMU	Maximum Operator Generator (MAX)	Y28-2019
CEKMV	Memory Access Routine (MEMAC)	Y28-2019
CEKMW	Operand Processing Routine (OPND)	Y28-2019
CEKMX	Release Temporary Storage (RLSWS)	Y28-2019
CEKMY	Result-Register Operand Processing Subroutine (RSLT)	Y28-2019
CEKMZ	Local Branch Generator (SADDR)	Y28-2019
CEKNA	General Register Availability for Integer Divide (SELGD)	Y28-2019
CEKNB	Determine Availability of Register for Multiplication (SELGM)	Y28-2019
CEKND	Select Operand Routine (SELOP)	Y28-2019
CEKNE	Weight Subroutine (WGHT)	Y28-2019
CEKNF	Select Position for Operand (SLPOS)	Y28-2019
CEKNG	Select Single General Register (SELSR)	Y28-2019
CEKNH	Select Even/Odd General Register Pair (SELDR)	Y28-2019
CEKNI	Code File Output Subroutine (INSOT)	Y28-2019
CEKNJ	Comma Operator Processing Subroutine (COMMA)	Y28-2019
CEKNK	Arithmetic IF PF Entry Processor (AIF)	Y28-2019
CEKNL	Logical IF PF Entry Processor (LIF)	Y28-2019
CEKNM	Begin Loop 1 PF Entry Processor (BL1)	Y28-2019
CEKNN	Begin Loop 2 PF Entry Processor (BL2)	Y28-2019
CEKNO	Begin Loop 3 PF Entry Processor (BL3)	Y28-2019
CEKNP	End Loop PF Entry Processor	Y28-2019
CEKNQ	Assigned GO TO PF Entry Processor (AGO)	Y28-2019
CEKNR	Computed GO TO PF Entry Processor (CGO)	Y28-2019
CEKNS	ASSIGN PF Entry Processor (ASSGN)	Y28-2019
CEKNU	Referenced Label PF Entry Processor (LABEL)	Y28-2019
CEKNV	Labeled Branch Generator (LBL)	Y28-2019
CEKNW	Arithmetic Expression Generation (AGEN)	Y28-2019
CEKOB	Common Expression Usage Count	Y28-2019
CEKOC	Operand Status Routine	

	(KEY)	Y28-2019
CEKOD	Entry Point Processor (ENT)	Y28-2019
CEKOE	RETURN Processor (RTRN)	Y28-2019
CEKOF	Complex Multiply Generator (CMUL)	Y28-2019
CEKOG	Complex Divide Generator (DDIV)	Y28-2019
CEKOH	I/O Statement PF Entry Processor (RD)	Y28-2019
CEKOI	I/O List Element PF Entry Processor (ILIST)	Y28-2019
CEKOJ	End List PF Entry Processor (NDLST)	Y28-2019
CEKOK	STOP and PAUSE Statement PF Entry Processor (STOP)	Y28-2019
CEKOL	CALL Statement Processor (CALL)	Y28-2019
CEKOM	Open Function Control Routine (DCOM)	Y28-2019
CEKOM2	Open Function Processing Routine (OPEN6)	Y28-2019
CEKON	Register Memory Clear Routine (FLUSH)	Y28-2019
CEKOR	Single Operand Locating Routine (KEY1)	Y28-2019
CEKOS	Operand Fetch Complement/Store Routine (FETCH)	Y28-2019
CEKOT	Open Function Control Routine (OPEN1)	Y28-2019
CEKOU	Open Function Processing Routine (OPEN2)	Y28-2019
CEKOV	Add by Load Address (LADDR)	Y28-2019
CEKOW	Select One Operand in a Register (SLONE)	Y28-2019
CEKOX	Open Function Processing Routine (OPEN3)	Y28-2019
CEKOY	Open Function Processing Routine (OPEN4)	Y28-2019
CEKOZ	Open Function Processing Routine (OPEN5)	Y28-2019
CEKSB	Object Program Module Builder (BUILD)	Y28-2019
CEKSC	Common Control Section Generator (CMSEC)	Y28-2019
CEKSD	Preset Data Processor (SPECS)	Y28-2019
CEKSF	Code Control Section Generator (COSEC)	Y28-2019
CEKSG	PSECT Builder (PRSEC)	Y28-2019
CEKSH	Internal Symbol Dictionary Generator (ASSIST)	Y28-2019
CEKSJ	Symbol Table Sort (SYMSRT)	Y28-2019
CEKSK	Cross Reference List Routine (CRFSRT)	Y28-2019
CEKSL	Constant Conversion (CONCV)	Y28-2019
CEKTA	Phase Controller (PHC)	Y28-2019
CEKTC	Get Next Source Statement (GNSS)	Y28-2019
CEKTD	Process Terminal Modifications (MOD)	Y28-2019
CEKTE	Receive Diagnostic Message (RDM)	Y28-2019
CEKTF	Constant Filers (CONFIL)	Y28-2019
CEKTH	Master Input/Output Routine (MIO)	Y28-2019
CEKTI	Analyze Console Source Line (ANALYZ)	Y28-2019
CEKTJ	Inspect a Console Character (INSCON)	Y28-2019
CEKTK	Move a Line to a List Data Set (LDMOVE)	Y28-2019
CEKTL	Build a List Data Set Buffer (BUILD)	Y28-2019
CEKTM	Flush a List Data Set Buffer (FLUSH)	Y28-2019
CEKUA	Executive Overall Flow	Y28-2019
CEVAC	Assembler Control	Y28-2021
CEVAD	Address Constant Processor	Y28-2021
CEVAN	ANOP Instruction Scan	Y28-2021
CEVBS	Basic Scan Routine	Y28-2021
CEVCC	CCW Text Processor	Y28-2021
CEVCD	CSD Processor	Y28-2021
CEVCE	SPACE Instruction Scan	Y28-2021
CEVCN	CNOP Instruction Scan	Y28-2021
CEVCP		

Substitution Control
Routine...................... Y28-2021
CEVCS
Constant Scan Routine........ Y28-2021
CEVCT
Control Section Instruction
Scan......................... Y28-2021
CEVCW
CCW Instruction Scan......... Y28-2021
CEVCY
COPY Instruction Scan........ Y28-2021
CEVDC
DC/DS Instruction Scan....... Y28-2021
CEVDF
Macro Definition Processor... Y28-2021
CEVDP
DC Constant Processor........ Y28-2021
CEVDR
DROP Statement Processor..... Y28-2021
CEVDX
Diagnostic Message
Processor.................... Y28-2021
CEVEJ
EJECT Instruction Scan....... Y28-2021
CEVEP
Entry Point Processor........ Y28-2021
CEVEQ
Assign Value to Name
Routine...................... Y28-2016
CEVET
Interface with EBCDTIME
Macro........................ Y28-2021
CEVEV
Expression Evaluation........ Y28-2021
CEVEY
ENTRY Instruction Scan....... Y28-2021
CEVGB
Binary Self-defining Term
Generator.................... Y28-2021
CEVGC
Character Self-defining Term
Generator.................... Y28-2021
CEVGD
Decimal Self-defining Term
Generator.................... Y28-2021
CEVGH
Hexadecimal Self-defining
Term Generator............... Y28-2021
CEVGL
Global/Local Symbol Scan..... Y28-2021
CEVGN
Location Counter Reset
Routine...................... Y28-2021
CEVGO
AGO/AIF Instruction Scan..... Y28-2021
CEVGP
Identify Operation Code...... Y28-2021
CEVGV
Get Relocatable Value
Routine...................... Y28-2021
CEVGW
Interface with VISAM PUT
Macro........................ Y28-2021
CEVIC
ICTL Instruction Scan........ Y28-2021
CEVIQ
ISEQ Instruction Scan........ Y28-2021
CEVKM
Lookup Dictionary Item....... Y28-2021
CEVLG
LTORG Instruction Scan....... Y28-2021
CEVLM
Macro name Item Look-up...... Y28-2021
CEVLP
Dictionary Lookup and Put.... Y28-2021
CEVLS
Object Program Listing
Routine...................... Y28-2021
CEVLT
Literal Text Processor....... Y28-2021
CEVMC
MACRO Instruction Scan....... Y28-2021
CEVMD
PMD Listing Routine.......... Y28-2021
CEVMN
MNOTE Instruction Scan....... Y28-2021
CEVMO
Machine-Operations
Processor.................... Y28-2021
CEVMP
Machine Instruction Scan..... Y28-2021
CEVMX
MEND/MEXIT Instruction Scan.. Y28-2021
CEVND
END Instruction Scan......... Y28-2021
CEVPA
Phase I Control.............. Y28-2021
CEVPB
Phase IIA Control............ Y28-2021
CEVPC
Phase IIB Control............ Y28-2021
CEVPD
Phase IIC Control............ Y28-2021
CEVPE
Phase III Control............ Y28-2021
CEVPF
Phase IV Control............. Y28-2021
CEVPL
Literal Pooling Processor.... Y28-2021
CEVPM
Macro Parameter Processor.... Y28-2021
CEVPR
PRINT Instruction Scan....... Y28-2021
CEVPS
Parameter Item Analyzer...... Y28-2021
CEVPV
Output Relocatable Value
Routine...................... Y28-2021
CEVQU
EQU Instruction Scan......... Y28-2021
CEVRD
Obtain Next Source
Statement.................... Y28-2021
CEVRF
Macro Reference Processor.... Y28-2021
CEVRG
ORG Instruction Scan......... Y28-2021
CEVRL
Literal Resolution
Processor.................... Y28-2021
CEVRS
Alignment Resolution
Routine...................... Y28-2021
CEVSA
ISD Listing Routine.......... Y28-2021
CEVSD

	ISD Processor	Y28-2021
CEVSE	SET Statement Scan	Y28-2021
CEVSL	Literal Operand Scan	Y28-2021
CEVSP	Substitute into Operation Code	Y28-2021
CEVSR	Symbol Table Edit Routine	Y28-2021
CEVSS	String Substitution Routine	Y28-2021
CEVST	Statement Analyzer	Y28-2021
CEVSX	Source Listing Processor	Y28-2021
CEVSY	Define Location Symbol	Y28-2021
CEVTI	TITLE Instruction Scan	Y28-2021
CEVTK	Macro dictionary lookup	Y28-2021
CEVTM	Macro name Item Insert	Y28-2021
CEVTP	Macro dictionary Insert	Y28-2021
CEVUD	USING/DROP Instruction Scan	Y28-2021
CEVUP	USING Table Processor	Y28-2021
CEVUV	Using-Register Routine	Y28-2021
CEVXF	Cross Reference Listing Routine	Y28-2021
CEVXN	EXTRN Instruction Scan	Y28-2021
CEYTS	Linkage Editor	Y28-2030
CFADA	LPCMAIN	Y28-2013
CFADB	GETLINE	Y28-2013
CFADC	PUTDIAG	Y28-2013
CGCCA	Allocate Module	Y28-2031
CGCCB	Select HASH	Y28-2031
CGCCC	SRCHPACK	Y28-2031
CGCCE	Resolve Symbol	Y28-2031
CGCCH	Load PMD	Y28-2031
CGCCJ	Fix PMD	Y28-2031
CGCCK	Attach Text	Y28-2031
CGCCL	Fix	Y28-2031
CGCCN	Add PMD	Y28-2031
CGCCO	Drop PMD	Y28-2031
CGCCP	Reject DIAG	Y28-2031
CGCCR	BISEARCH	Y28-2031
CGCCT	PCSA	Y28-2031
CGCCU	Check DEF Legal	Y28-2031
CGCCV	Link DEFs	Y28-2031
CGCCY	Define REF	Y28-2031
CGCDA	Modify MUT Counts	Y28-2031
CGCDB	Delete Caller MUTEs	Y28-2031
CGCDC	Delete Selected MUTEs	Y28-2031
CGCDD	Modify Use Counts	Y28-2031
CGCDE	Test User Counts	Y28-2031
CGCDG	Add MUTE	Y28-2031
CGCDPR	Loader PROMPT	Y28-2031
CGCKA	SYSINDEX	Y28-2018
CGCKB	SYSXBLD	Y28-2018
CGCKC	SYSEARCH	Y28-2018
CGCQC	Q-CHAIN	Y28-2031
CGCRQ	RESOLVE Q-REF	Y28-2031
CHCBD	I/O Initialization and Machine Interrupt Enabler	Y28-2020
CHCBE	Interrupt and Machine Indicator Subprograms	Y28-2020
CHCBZ	Error Processor (LIBER)	Y28-2020
CHCIA	I/O Initialization	Y28-2020
CHCIB	DCB Maintenance	Y28-2020
CHCIC	I/O Control	Y28-2020
CHCID	NAMELIST Processor	Y28-2020
CHCIE	List Item Processor	Y28-2020
CHCIF	FORMAT Processor	Y28-2020
CHCIH	Integer Output Conversion	Y28-2020
CHCII	Real and Integer Input Conversion	Y28-2020
CHCIJ	Real Output Conversion	Y28-2020
CHCIM	Complex Input Conversion	Y28-2020
CHCIN	Complex Output Conversion	Y28-2020
CHCIO	Alphameric Input Conversion	Y28-2020
CHCIP	Alphameric Output	

	Conversion	Y28-2020
CHCIQ	Logical Input Conversion	Y28-2020
CHCIR	Logical Output Conversion	Y28-2020
CHCIS	General Input Conversion	Y28-2020
CHCIT	General Output Conversion	Y28-2020
CHCIU	List Termination	Y28-2020
CHCIV	Dump Routine	Y28-2020
CHCIW	Exit Routine	Y28-2020
CMASA	SERR Bootstrap	Y28-2012
CMASD	Checker Program	Y28-2012
CMASF	Restore and Validate	Y28-2012
CMASG	Instruction Retry	Y28-2012
CMASH	CPU Memory Checkout 1	Y28-2012
CMASI	CPU Memory Checkout 2	Y28-2012
CMASJ	CPU Memory Checkout 3	Y28-2012
CMASN	EREP67	Y28-2018
CMATC	OLTS Print	Y28-2042
CMATD	OLTS Compare	Y28-2042
CMATE	OLTS Conversion	Y28-2042
CMATF	OLTS: Setup Control, Attention Processor, Section Selection, Routine Selection, Job Option Selection, Sequencer	Y28-2042
CZAAC	SCAN	Y28-2013
CZAAD	MSGWR	Y28-2013
CZAAF	VMTI	Y28-2013
CZABA	Batch Monitor	Y28-2047
CZABB	EXECUTE	Y28-2013
CZABC	BACK	Y28-2013
CZABD	Bulk I/O Preprocessor	Y28-2047
CZABF	Read Tape (RT)	Y28-2047
CZABG	PRINT	Y28-2047
CZABH	PUNCH	Y28-2047
CZABI	Write Tape (WT)	Y28-2047
CZABJ	CANCEL	Y28-2013
CZABK	Assign Batch Device (ASNBD)	Y28-2047
CZABL	TWAIT	Y28-2047
CZABQ	XWTO	Y28-2047
CZABS	QUIT/JOIN RJE	Y28-2013
CZABX	LABEL	Y28-2047
CZACA	Main Operator Control Program (MOCP)	Y28-2047
CZACB	Main Operator Housekeeping Routine (MOHR)	Y28-2047
CZACD	REPLY	Y28-2047
CZACF	Message/Broadcast	Y28-2047
CZACG	FORCE	Y28-2047
CZACN	SHUTDOWN	Y28-2047
CZACP	ABEND (Completion Code 1)	Y28-2013
CZACQ	ABEND (Completion Code 2)	Y28-2013
CZACR	ABEND (Completion Code 3)	Y28-2013
CZACS	PAIR	Y28-2013
CZADF	DATA	Y28-2013
CZAEA	DDEF	Y28-2013
CZAEB	FINDJFCB	Y28-2013
CZAEC	FINDDS	Y28-2013
CZAEG	MODIFY	Y28-2013
CZAEH	LOCFQN	Y28-2013
CZAEI	CATALOG	Y28-2013
CZAEJ	ERASE/DELETE	Y28-2013
CZAEK	JOBLIB/DDNAME?	Y28-2013
CZAEL	DSS?/PC?	Y28-2013
CZAEM	LINE?	Y28-2013
CZAEN	RET	Y28-2013
CZAET	VAM Tape	Y28-2013
CZAFH	PERMIT	Y28-2013
CZAFI	SHARE	Y28-2013
CZAFJ	RELEASE	Y28-2013
CZAFK	JOIN	Y28-2013
CZAFL	QUIT	Y28-2013
CZAFM	LOGON	Y28-2013

CZAFN	LOGOFF	Y28-2013
CZAFS	CDD	Y28-2013
CZAFU	SECURE	Y28-2013
CZAFV	CDS	Y28-2013
CZAGB	USAGE	Y28-2013
CZAGC	UPDTUSER	Y28-2013
CZAGD	FLOW	Y28-2013
CZAHA	DIAGNO	Y28-2013
CZAHB	IAIP	Y28-2013
CZAHC	XIP/XIIS	Y28-2013
CZAHD	XIMS	Y28-2013
CZAMA	SET	Y28-2013
CZAMB	BRANCH	Y28-2014
CZAMC	STOP and GO	Y28-2014
CZAMD	DISPLAY and DUMP	Y28-2014
CZAME	IF	Y28-2014
CZAMF	AT	Y28-2014
CZAMG	CALL	Y28-2014
CZAMH	Expression Scan (EXPSCAN)	Y28-2014
CZAMI	Form Data Field Definition (DATAFLD)	Y28-2014
CZAMJ	Subscript to Polish (SUBPOL)	Y28-2014
CZAML	Form Data Location Definition (DATALOC)	Y28-2014
CZAMO	Form External Symbol Definition (EXTERNAL)	Y28-2014
CZAMQ	Scan Field to Delimiter (SCANFLD & GETCHAR)	Y28-2014
CZAMR	QUALIFY	Y28-2014
CZAMS	REMOVE	Y28-2014
CZAMT	UNLOAD	Y28-2014
CZAMZ	User Control	Y28-2013
CZANA	Phase II Control (PHASE2)	Y28-2014
CZANF	Code Generator (CODEGEN)	Y28-2014
CZANG	Subscript Generator (SUBGEN)	Y28-2014
CZANH	Combine Constants (COMCON)	Y28-2014
CZANI	Operator Code Generator (OPGEN)	Y28-2014
CZANT	Load Operand (LOADOP)	Y28-2014
CZANV	Base Register Assignment (GETBASE)	Y28-2014
CZANW	Issue Diagnostics (DIAGNO)	Y28-2014
CZANX	User Prompting (PROMPT)	Y28-2014
CZAOA	Evaluate Module Name (VALMODE)	Y28-2014
CZAOB	Evaluate Symbol (VALSYM)	Y28-2014
CZAOD	GETREG	Y28-2014
CZAPB	Phase III Control (PCSPUT)	Y28-2014
CZAPC	Location Table Scan (FINDLOC)	Y28-2014
CZAPG	Symbol Generator (SYMGEN)	Y28-2014
CZAPI	FORMDIAG	Y28-2014
CZAPK	Saved Instruction Execution (SAVIX)	Y28-2014
CZAPN	Call Generated Code (GENCALL)	Y28-2014
CZAQA	Display/Dump Control (DISPDUMP)	Y28-2014
CZAQB	Process Phrase List (NEXTLIST)	Y28-2014
CZAQC	Process Display List (NEXTITEM)	Y28-2014
CZAQD	Process Next ISD Entry	Y28-2014
CZAQF	Display Register (DISREG)	Y28-2014
CZAQG	Display Simple Variable (SIMVAR)	Y28-2014
CZAQH	Convert Item by Data Type (ADDITEM)	Y28-2014
CZAQI	Display an Instruction (DISINST)	Y28-2014
CZAQJ	Display an Array (DISARAY)	Y28-2014
CZAQK	Display a Line of an Array (DISALINE)	Y28-2014
CZAQM	Display a Range in Hexadecimal (DISHEX)	Y28-2014
CZAQN	Display a Hexadecimal Line	

	(DISHLINE)	Y28-2014
CZAQQ	DISRHEAD	Y28-2014
CZAQR	Format a Symbol (DISYM)	Y28-2014
CZAQU	Output a Line (DISOUT)	Y28-2014
CZAQV	Real Number Conversion (REALCON)	Y28-2014
CZASA	Command Analyzer and Executor	Y28-2013
CZASB	Attention Handler	Y28-2013
CZASC	Source List Handler	Y28-2013
CZASD	Source List Handler	Y28-2013
CZASE	Virtual Memory Error Recording Edit and Print (VMEREP)	Y28-2013
CZASF	REGION	Y28-2013
CZASG	DATA LINE	Y28-2013
CZASH	REVISE	Y28-2013
CZASJ	INSERT	Y28-2013
CZASK	EXCERPT	Y28-2013
CZASL	EXCISE	Y28-2013
CZASM	CONTEXT	Y28-2013
CZASN	LOCATE	Y28-2013
CZASP	LIST	Y28-2013
CZASQ	CORRECT	Y28-2013
CZASR	UPDATE	Y28-2013
CZASS	Transaction Table Updater	Y28-2013
CZAST	MATCH	Y28-2013
CZASU	NUMBER	Y28-2013
CZASV	STET	Y28-2013
CZASY	Drum Access Module (DRAM)	Y28-2018
CZASZ	PROFILE HANDLER	Y28-2013
CZATA	OLTAM EXIO and Data Integrity	Y28-2042
CZATB	OLTAM Posting	Y28-2042
CZATC	GATE	Y28-2013
CZATD	VMTI-2	Y28-2013
CZATE	Procedure Expander	Y28-2013
CZATF	SYSXPAT	Y28-2013
CZATG	OLTS Device Allocation	Y28-2042
CZATH	KEYWORD	Y28-2013
CZATI	CHGPAS	Y28-2013
CZATJ	User Prompter	Y28-2013
CZATP	PROCDEF	Y28-2013
CZART	SYNONYM/DEFAULT	Y28-2013
CZATS	Text Editor Controller	Y28-2013
CZATU	MCAST	Y28-2013
CZAVB	TIME	Y28-2013
CZAVR	VSS	Y28-2013
CZAWA	Bulk I/O ABEND	Y28-2047
CZAWC	FORTRAN Data Set Converter	Y28-2047
CZAWM	Bulk I/O Message Handler	Y28-2047
CZAWN	ACK Data Set Handler	Y28-2047
CZAWS	Bulk I/O Initialization	Y28-2047
CZAWT	Bulk I/O Control	Y28-2047
CZAWU	Bulk I/O Attend Alert	Y28-2047
CZAWV	Bulk I/O Input Closeout	Y28-2047
CZAWW	Bulk I/O Master Service	Y28-2047
CZAWX	Bulk I/O Input Start	Y28-2047
CZAWY	Bulk I/O Output Service	Y28-2047
CZAWZ	Bulk I/O Input Service	Y28-2047
CZAXX	RPS/CVV	Y28-2013
CZBLT	String Comparison	Y28-2013
CZBSE	Edit Initialization	Y28-2013
CZBTA	TRIN	Y28-2013
CZBTB	LOGON2	Y28-2013
CZBTC	PRMPT Command	Y28-2013
CZCAA	MTREQ	Y28-2018
CZCAB	BUMP	Y28-2018
CZCAC	PAUSE	Y28-2018
CZCAD	RELEAS	Y28-2018
CZCAM	MOUNT VOL	Y28-2018
CZCCD1	Loader Logoff	Y28-2031

CZCCD2
Loader Release............... Y28-2031
CZCCD4
Loader Cleanup............... Y28-2031
CZCDH1
LIBE MAINT................... Y28-2031
CZCDL1
Explicit Link................ Y28-2031
CZCDL2
HASH Search.................. Y28-2031
CZCDL3
LIBE Search.................. Y28-2031
CZCDL4
Page Relocation.............. Y28-2031
CZCDL5
MAPSEARCH.................... Y28-2031
CZCDL6
Set Search Flags............. Y28-2031
CZCDU1
Explicit Unlink.............. Y28-2031
CZCDU2
Delete Module................ Y28-2031
CZCEA
ALLOCATE..................... Y28-2018
CZCEB
SAM Search................... Y28-2018
CZCEF
DSCBREC...................... Y28-2018
CZCEG
GIVBK SAM.................... Y28-2018
CZCEH
VOLSRCH...................... Y28-2018
CZCEI
Virtual Memory Input Error
Recovery..................... Y28-2016
CZCEJ
ESA Lock..................... Y28-2018
CZCEK
ADDSCB....................... Y28-2018
CZCEL
FINDEXPG..................... Y28-2018
CZCEM
READWRIT..................... Y28-2018
CZCEN
RELEXPG...................... Y28-2018
CZCEQ
VAMINIT...................... Y28-2018
CZCES
SCRATCH...................... Y28-2018
CZCEW
WRITDSCB..................... Y28-2018
CZCEX
EXTEND....................... Y28-2018
CZCEY
DUPOPEN...................... Y28-2016
CZCEZ
DUPCLOSE..................... Y28-2016
CZCFA
ADDCAT....................... Y28-2018
CZCFB
EVV.......................... Y28-2013
CZCFD
DELCAT....................... Y28-2018
CZCFF
Search SBLOCK................ Y28-2018
CZCFG
Get SBLOCK................... Y28-2018
CZCFI
INDEX........................ Y28-2018
CZCFL
LOCATE....................... Y28-2018
CZCFO
OBTAIN/RETAIN................ Y28-2018
CZCFS
SHARE........................ Y28-2018
CZCFT
DELVAM....................... Y28-2016
CZCFU
SHAREUP...................... Y28-2018
CZCFV
UNSHARE...................... Y28-2018
CZCFX
CATFLUSH..................... Y28-2018
CZCFZ
RENAME....................... Y28-2018
CZCGA
Virtual Memory Allocation.... Y28-2018
CZCHA
Small Virtual Memory
Allocation................... Y28-2018
CZCHB
CLOSE Command................ Y28-2013
CZCJA
STIMER - TTIMER.............. Y28-2041
CZCJC
Cleanup...................... Y28-2041
CZCJD
Delete Interrupt Routine..... Y28-2041
CZCJI
Interrupt Inquiry (INTINQ)... Y28-2041
CZCJL
Leave Privilege.............. Y28-2041
CZCJQ
Queue Linkage Entry (QLE).... Y28-2041
CZCJS
Specify Interrupt Routine
(SIR)........................ Y28-2041
CZCJT
Program, SVC, Timer, External,
Asynchronous I/O, Synchronous
I/O & VSS Interrupt Processors,
Data Set Paging, Scanner
Dispatcher................... Y28-2041
CZCJY
Set Clock.................... Y28-2041
CZCJZ
Cancel Clock................. Y28-2041
CZCKZ
CSECT Store.................. Y28-2018
CZCLA
Open Common.................. Y28-2016
CZCLB
Close Common................. Y28-2016
CZCLD
Forced End of Volume......... Y28-2016
CZCMA
Get a Buffer................. Y28-2016
CZCMB
Get a Buffer Pool............ Y28-2016
CZCMC
MSAM open.................... Y28-2016
CZCMD
Set Unit Record.............. Y28-2016
CZCME
DOMSAM....................... Y28-2016
CZCMF

	MSAM READ/WRITE	Y28-2016
CZCMG	MSAM Posting and Error Retry	Y28-2016
CZCMH	MSAM Finish	Y28-2016
CZCMI	MSAM Close	Y28-2016
CZCNA	Free a Buffer	Y28-2016
CZCNB	Free a Buffer Pool	Y28-2016
CZCOA	Open VAM	Y28-2016
CZCOB	Close VAM	Y28-2016
CZCOC	MOVEPAGE	Y28-2016
CZCOD	Insert Page/Delete Page	Y28-2016
CZCOE	Request Page	Y28-2016
CZCOF	Insert	Y28-2016
CZCOG	Reclaim	Y28-2016
CZCOH	Interlock	Y28-2016
CZCOI	Release Interlock	Y28-2016
CZCOJ	FIND	Y28-2016
CZCOK	STOW	Y28-2016
CZCOL	Search	Y28-2016
CZCOM	Extend POD	Y28-2016
CZCON	Relocate Members	Y28-2016
CZCOO	Get Page Number	Y28-2016
CZCOP	OPEN Sequential	Y28-2016
CZCOQ	CLOSE Sequential	Y28-2016
CZCOR	GET	Y28-2016
CZCOS	PUT	Y28-2016
CZCOT	Set Location	Y28-2016
CZCOU	PUT Exchange	Y28-2016
CZCOV	Flush Buffers	Y28-2016
CZCOX	POD?	Y28-2015
CZCPA	PUT	Y28-2016
CZCPB	GET	Y28-2016
CZCPC	Set Location	Y28-2016
CZCPE	READ/WRITE Delete Record	Y28-2016
CZCPL	Add Directory Entry	Y28-2016
CZCPI	Get Page, End Sequential, Release Exclusive	Y28-2016
CZCPZ	OPEN Indexed Sequential	Y28-2016
CZCQA	CLOSE Indexed Sequential	Y28-2016
CZCQE	Search SDST	Y28-2016
CZCQI	Expand RESTBL	Y28-2016
CZCQK	VAM Data Management Error Processor	Y28-2016
CZCQQ	VAM Interlock Reset	Y28-2016
CZCQV	CONVERT	Y28-2013
CZCRA	BSAM READ/WRITE	Y28-2016
CZCRB	Tape Positioning, Card Reader Stacker Select, Printer Channel Skipping	Y28-2016
CZCRC	Check I/O	Y28-2016
CZCRG	Backspace	Y28-2016
CZCRH	DA Error Retry	Y28-2016
CZCRM	Logically Reposition Tape or DA Data Set	Y28-2016
CZCRN	Note ID of Last Record Read or Written	Y28-2016
CZCRP	SAM Posting and Error Retry	Y28-2016
CZCRQ	FINDR	Y28-2016
CZCRR	RELFUL	Y28-2016
CZCRS	FULREL	Y28-2016
CZCRX	Virtual Memory Error Recording	Y28-2018
CZCRY	VMSDE	Y28-2018
CZCSA	QSAM	Y28-2016
CZCSB	IOREQ	Y28-2016
CZCSC	IOR OPEN	Y28-2016
CZCSD	IOR CLOSE	Y28-2016
CZCSE	IOREQ Posting	Y28-2016
CZCTA	SYSTIME	Y28-2018
CZCTC	Terminal Task Control	Y28-2016
CZCTR	RTAM Error Recording Interface	Y28-2018
CZCWA	ASCII Translation and	

	Conversion	Y28-2016
CZCWB	Build or Modify Common Portion of DEB	Y28-2016
CZCWC	SAM CLOSE	Y28-2016
CZCWD	DA OPEN	Y28-2016
CZCWL	Build or Extend DA DEB	Y28-2016
CZCWM	Message and ABEND Processing	Y28-2016
CZCWO	SAM OPEN Mainline	Y28-2016
CZCWP	Tape Positioning	Y28-2016
CZCWR	Read Format 3 DSCBs	Y28-2016
CZCWT	Tape OPEN	Y28-2016
CZCWV	Volume Sequence Convert	Y28-2016
CZCWX	Tape Volume Label Processor	Y28-2016
CZCWY	Tape Header/Trailer Label Processor	Y28-2016
CZCXD	DA Output EOV	Y28-2016
CZCXE	Mainline EOV	Y28-2016
CZCXI	DA Input EOV	Y28-2016
CZCXN	DA Input Label Processor	Y28-2016
CZCXO	Tape Output EOV	Y28-2016
CZCXS	Set DSCB	Y28-2016
CZCXT	Tape Input EOV	Y28-2016
CZCXU	DA Output Label Processor	Y28-2016
CZCXX	Concatenation Processor	Y28-2016
CZCYA	TAM OPEN	Y28-2016
CZCYG	TAM CLOSE	Y28-2016
CZCYM	TAM READ/WRITE	Y28-2016
CZCZA	TAM Posting	Y28-2016
CZHNE	VSS External Interrupt Processor	Y28-2022
CZHNM	VSS Message	Y28-2022
CZHNP	VSS Program Interrupt Processor	Y28-2022
CZHNV	VSS Activate Interrupt Processor	Y28-2022
CZHPA	VSS Real Core Access	Y28-2022
CZHPB	VSS Virtual Memory Access	Y28-2022
CZHPR	VSS Restore Status	Y28-2022
CZHSA	VSS I/O Control	Y28-2022
CZHSB	VSS I/O Initiation/Posting	Y28-2022
CZHTA	VSS Direct Access Device Access Method	Y28-2022
CZHTB	VSS Console Access Method	Y28-2022
CZHTC	VSS Sequential Access Method	Y28-2022
CZHTD	VSS Telecommunications Access Method	Y28-2022
CZHTE	VSS I/O Editor	Y28-2022
CZHUA	VSS Direct Access Device Error Recovery	Y28-2022
CZHUB	VSS Console Error Recovery	Y28-2022
CZHUC	VSS Sequential Access Device Error Recovery	Y28-2022
CZHUD	VSS Telecommunications Error Recovery	Y28-2022
CZHUE	VSS Error Scan and Recovery	Y28-2022
CZHVA	VSS I/O Completion	Y28-2022
CZHWA	VSS Address to Symbol Resolution	Y28-2022
CZHWM	VSS Memory Map Format	Y28-2022
CZHXA	VSS Operator Functions	Y28-2022
CZHXC	VSS Language Control	Y28-2022
CZHXL	VSS Literal Resolution	Y28-2022
CZHXP	VSS Source to Polish	Y28-2022
CZHXS	VSS Scan Control	Y28-2022
CZHYA	VSS AT Command Processor	Y28-2022
CZHYC	VSS COLLECT Command Processor	Y28-2022
CZHYD	VSS DUMP/DISPLAY Command Processor	Y28-2022
CZHYE	VSS DEFINE Command Processor	Y28-2022
CZHYL	VSS CALL/END Commands Processor	Y28-2022
CZHYM	VSS DISCONNECT Command Processor	Y28-2022
CZHYN		

VSS RUN Command Processor....	Y28-2022
CZHYP	
VSS PATCH Command Processor..	Y28-2022
CZHYQ	
VSS QUALIFY Command Processor....................	Y28-2022
CZHYR	
VSS REMOVE Command Processor....................	Y28-2022
CZHYS	
VSS SET Command Processor....	Y28-2022
CZHYT	
VSS STOP Command Processor...	Y28-2022
CZHZA	
VSS AT SVC Processor.........	Y28-2022
CZHZF	
VSS $AT/$PATCH Format........	Y28-2022
CZHZH	
VSS $TASK Format.............	Y28-2022
CZUBC	
TSU Module...................	Y28-2047
CZUFA	
DMPRST.......................	Y28-2047
CZUFO	
PATCLEAR.....................	Y28-2047
CZUPF	
PATFIX.......................	Y28-2047
CZUFY	
User Catalog Scan............	Y28-2018
CZUFX	
DSCB Catalog Recovery........	Y28-2018
CZUVA	
VAMACC.......................	Y28-2013

Where more than one page reference is given, the major reference is first.

GY28-2009-2

IBM S/360 Sys Log Sum   Printed in U.S.A.   GY28-2009-2

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

IBM Technical Newsletter

File Number	S360-36
Re. Order No.	GY28-2009-2
This Newsletter No.	GN28-3146
Date	September 15, 1970
Previous Newsletters Nos.	None

IBM SYSTEM/360 TIME SHARING SYSTEM
SYSTEM LOGIC SUMMARY
PROGRAM LOGIC MANUAL

This Technical Newsletter, a part of Version 8, Modification 0 of the IBM System/360 Time Sharing System, provides replacement pages for the subject publication. These replacement pages remain in effect for all subsequent versions or modifications unless specifically altered. Pages to be inserted and/or removed are listed below:

7-10	191-192
15-16	195-198.1
27-30	209-210
37-38	215-216.1
59-60.1	271-272
111-120.1	275-276.1

A change to the text or a small change to an illustration is indicated by a vertical line to the left of the change; a changed or added illustration is denoted by the symbol (•) to the left of the caption.

Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Dept. 643, Neighborhood Road, Kingston, N. Y. 12401

PRINTED IN U.S.A.

IBM Technical Newsletter

File Number	S360-36
Re: Order No.	GY28-2009-2
This Newsletter No.	GN28-3193
Date	September 15, 1971
Previous Newsletter Nos.	GN28-3146

IBM SYSTEM/360 TIME SHARING SYSTEM SYSTEM LOGIC SUMMARY PROGRAM LOGIC MANUAL

This Technical Newsletter, a part of Version 8, Modification 1 of the IBM System/360 Time Sharing System, provides replacement pages for the subject publication. These replacement pages remain in effect for all subsequent versions or modifications unless specifically altered. Pages to be replaced are:

13- 14	133-134
41- 42	157-162
65- 66.1	181-182
73- 74	191-192
111-112	195-196
115-116	205-208
119-120	229-232
120.1 (Deleted)	255-258
129-130.1	287-296

A change to the text or a small change to an illustration is indicated by a vertical line at the left of the change.

Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Programming Publications, Dept. 643, Neighborhood Road, Kingston, N.Y. 12401

IBM Technical Newsletter

File Number	S360-36
Base Publication No.	GY28-2009-2
This Newsletter No.	GN28-3219
Date	February 1, 1972
Previous Newsletters	GN28-3146 GN28-3193

IBM System/360 Time Sharing System:
System Logic Summary

This Technical Newsletter provides replacement pages for the subject publication. Pages to be inserted and/or removed are:

111-112
115-116

A change to the text is indicated by a vertical line to the left of the change.

Summary of Amendments

Inaccuracies in the description of the Schedule Table Entry have been corrected.

IBM Corporation, Dept. 643, Neighborhood Road, Kingston, N. Y. 12401

PRINTED IN U S A