File No. S360-36
Order No. GC28-6720-4

OS

IBM Systems Reference Library

IBM System/360 Operating System

MVT Guide

OS Release 21

This publication describes the MVT (multiprogramming with a variable number of tasks) configuration of the operating system control program. It contains a description of the MVT control program, options available with MVT, information on optimizing performance, information on modifying the system, and a description of the internal logic of the MVT configuration. General descriptions are provided of the operating environment of the control program, of the initial program loading procedure, and of the job management, task management, data management, volume management and recovery management functions.

The MVT configuration of the control program is designed for use with System/360 Models 40, 50, 65, 75, 85, 91, 95, and 195; with the System/360 Model 65 Multiprocessing System; and with System/370 Models 145, 155, 165, and 195. The minimum main storage is 262,144 (256K) bytes, or 524,288 (512K) bytes in the case of the Model 65 Multiprocessing system.

Information in this publication for TSO with the Model 65 Multiprocessing (M65MP) configuration is for planning purposes until that item is available.

Fifth Edition (March, 1972)

This is a major revision of, and obsoletes, GC28-6720-3. Section IV of this publication contains a major quantity of the information formerly in **IBM System/360 Operating System: System Programmer's Guide**, GC28-6550. The publication, **IBM System/360 Operating System: Data Management for System Programmers**, GC28-6550, contains the remaining **System Programmer's Guide** material. Changes to text and illustrations are indicated by a vertical line to the left of the change. A summary of major changes appears after the table of contents.

This edition applies to Release 21 of IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, refer to the latest IBM System/360 SRL Newsletter, GN20-0360, for the editions that are applicable and current.

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

Preface

This guide explains how to use the MVT (multiprogramming with a variable number of tasks) control program. The MVT control program performs system functions, such as input/output operations and supervision of jobs, and processes up to 15 jobs concurrently.

This guide is intended for new users as well as users familiar with the MVT control program. It presents material in an introduction, five sections, and six appendices.

There is a Task Directory at the end of each section. The Task Directory contains a list of topics discussed in the section along with headings under which the topics can be found. Users with a specific topic in mind who cannot find the topic in the Table of Contents should look in the Task Directory.

The Introduction explains the scheduling, main storage organization, system generation, and uses of an MVT system.

Section I describes the MVT control program to new users or to those who wish to review MVT. This section describes storage organization, supervisor state and program state, initial program loading and nucleus initialization programs, and the five major functions of the control program. Experienced users do not need to read this section.

Section II describes briefly the options available for MVT. Planning and managerial personnel can use this information when expanding their system with the MVT options.

Section III explains device requirements, system generation macro instructions, and planning aids. Application programmers will find this information helpful to run their programs quickly and efficiently. Planning personnel can use this information to optimize system performance according to the demands of their installation.

Section IV lists the standard IBM cataloged procedures for readers, writers, and initiators, and explains how to modify the control program. Planning personnel need this information to tailor cataloged procedures to their installation's requirements.

Section V explains in detail the five major routines of the MVT control program. A new user should fully understand the first four sections of this guide before reading this section. An experienced user could go directly to this section.

Appendix A explains how to use certain system macro instructions.

Appendix B describes the control character transformations for the standard output writer.

Appendix C explains how to use the RESERVE macro instruction with the Shared DASD (Direct Access Storage Device) option.

Appendix D lists and states the full names of common acronyms found in this guide.

Appendix E names the program logic manuals that discuss the MVT control program. This appendix includes a short abstract of each manual.

Appendix F explains the different SVC routines and lists them by number.

As a prerequisite, users must have read

IBM System/360 Operating System: Introduction, GC28-6534.

A basic knowledge of the job control language is assumed throughout this mamual; for job control language information, see the publication **IBM System/360 Operating System: Job Control Language Reference,** GC28-6704.

Additionally, the following publications provide related information:

IBM System/360 Operating System:

Basic Telecommunications Access Method, GC30-2004

Data Management Services, GC26-3746

Data Management for System Programmers, GC28-6550

Graphic Programming Services for IBM 2250 Display Unit, GC27-6909

Graphic Programming Services for IBM 2260

Display Station (Local Attachment), GC27-6912

Introduction to Main Storage Hierarchy Support for IBM 2361 Models 1 and 2, GC27-6942

Operator's Procedures, GC28-6692

Operator's Reference, GC28-6691

Principles of Operation, GA22-6821

Storage Estimates, GC28-6551

Supervisor Services and Macro Instructions, GC28-6646

System Generation, GC28-6554

System Management Facilities, GC28-6712

Time Sharing Option Guide, GC28-6698

User's Guide for Job Control from the IBM 2250 Display Unit, GC27-6933

Utilities, GC28-6586

This publication also refers to the following program logic manuals (PLMs):

IBM System/360 Operating System:

Initial Program Loading/Nucleus Initialization Program Program Logic Manual, GY28-6661

Input/Output Supervisor Program Logic Manual, GY28-6616

Machine Check Handler for IBM System/360 Model 65 Program Logic Manual, GY27-7155

Machine Check Handler for IBM System/360 Model 85 Program Logic ManuL, GY27-7184

Machine Check Handler for IBM System/370 Model 145 Program Logic Manual, GY27-7237

Machine Check Handler for IBM System/370 Models 155 and 165 Program Logic Manual, GY27-7198

MVT Job Management Program Logic Manual, GY28-6660

MVT Supervisor Program Logic Manual, GY28-6659

Time Sharing Option (TSO) Control Program Program Logic Manual, GY27-7199

Contents

Figures

Summary of Amendments for GC28-6720-4 OS Release 21

Alternate Path Retry (APR)

APR includes the VARY PATH function as standard for both MVT and M65MP.

GENERALIZED Trace Facility (GTF)

Describes the tracing and editing abilities of GTF.

Changed list IEARSV00 to support Open/Close/EOV

Modules in the list IEARSV00 changed.

Disposition of Temporary Dedicated Data Sets

Clarification of submitting separate jobs instead of job steps in a job when using temporary dedicated data sets.

MVT Options List

Lists the options available for the MVT user.

System Programmer's Guide Chapters

Section II contains the following **System Programmer's Guide** chapters:

The Shared Direct Access Device Option
The Time Slicing Feature

Section IV contains the following **System Programmer's Guide** chapters:

Using the Link Pack Area
Job Queue Format
Output Separation
Writing System Output Writer Routines
Adding SVC Routines to the Control Program
Message Routing Exit Routines
Handling Accounting Routines
Writing Rollout/Rollin Installation Appendages
The Must Complete Function
The PRESRES Volume Characteristic List

Appendix A contains the following **System Programmer's Guide** chapter:

System Macro Instructions

Appendix B contains the following **System Programmer's Guide** chapter:

Control Character Transformations (from Writing System Output Writer Routines)

Appendix C contains the following **System Programmer's Guide** chapter:

RESERVE Macro Instruction Used with the Shared DASD Option (from The Shared Direct Access Device Option)

System/370 Model 195

The Model 195 can also function as a System/370 CPU.

Channel Check Handler (CCH) for Model 65 Multiproccessing and System/370 Model 145

CCH must be specified for M65MP; CCH standard with System/370 Model 145.

3420 and 3803 Support

3420 and 3803 support changes the operation of Error Volume Analysis (EVA) routines. This support increases the number of output writers for DSO. Cataloged procedures include the new devices in their parameters.

TSO Multiprocessing

TSO operates with M65MP.

Status Display Support

Bit 0, Byte 1 of the UCMDESCD field has assigned a descriptor code of 9.

Elimination of IEFSD165

IEFSD165 no longer used in the list IEFSD061.

Reserving Initiator Queue Records

Adds additional considerations for reserving initiaor queue records.

Model 65 Multiprocessing Use of the Shared DASD Option

M65MP can use the Shared DASD Option.

Summary of Amendments for GC28-6720-3 OS Release 20.1

Time Sharing Option (TSO)

Adds documentation of the Time sharing Option.

SYSOUT Data Blocding for FORTRAN G

Changes the record format for the SYSPRINT data set from RECFM=FBA to RECFM=FSBA Nstandard blocks).

New CPU Support

Adds references to System/370 Model 145.

New Device Support

Adds references to the IBM 2305 Fixed Head Storage Facility, IBM 2319 Direct Access Storage Facility, IBM 2319 Direct Access Storage Facility, IBM 3211 Printer, and IBM 3330 Disk Storage Drive.

Resident BLDL Table

Can contain directory entries from SYS1.SVCLIB as well as SYS1.LINKLIB.

Recovery Management

Revises the discussion of recovery management to include information formerly contained in the publication in the publication **IBM System/360 Operating System: Concepts and Facilities**, GC28-6535.

Summary of Amendments for GC28-6720-2 OS Release 20

Model Dependency for Models 155 and 165

Documents IBM System/370 Models 155 and 165 CPUs.

Models 155 and 165 Recovery Management Support

Adds Models 155 and 165 information for Machine Check Handler (MCH) facility.

Channel Check Handler with Model 155 and 2880 channels

Adds Model 155 and 2880 channels for Channel Check Handler (CCH) facility.

Time of Day

Adds documentation on the Time of Day (TOD) support under System/370.

MVT Redocumentation

This edition of **MVT Guide** obsoletes **MVT Control Program Logic Summary, Prigram Logic Manual**, GY28-6658: all information in the specified Program Logic Manual is now contained herein.

Introduction

The multiprogramming with a variable number of tasks (MVT) control program reads one or more job streams. This allows the CPU (central processing unit) to process several jobs concurrently. Long-running jobs can be processed along with short-running jobs. Batch jobs can be processed concurrently more quickly than if each job were run sequentially. Foreground jobs can be run along with background jobs.

The control program schedules jobs by priority, and supervises the concurrent processing of as many as 15 jobs. A job is the major unit of work that the computing system processes. Users define jobs to the system by using Job Control Language (JCL). The control program processes tasks, which are the major units of work processed by the CPU. Jobs can be further sub-divided into job steps. Users define job steps with JCL. The control program processes each job step as a separate task.

How Does the MVT Control Program Work?

The MVT control program reads in a stream of jobs, but instead of processing them immediately, it schedules, or queues them on a direct access storage device, as illustrated by Figure 1.

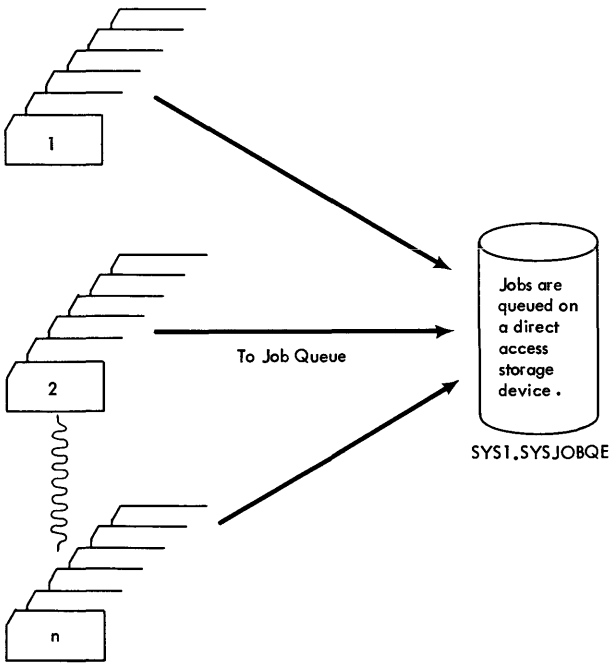


Figure 1. The MVT Control Program Reads a Stream of Jobs to the Input Work Queue

Each job in the job stream has a priority between 0-13. These priorities can be user-specified by means of JCL parameters. Jobs in the job stream belong to one of 15 job classes, identified with the alphabetic characters A through O, where each job class contains jobs in order of priority. Figure 2 illustrates job priorities.

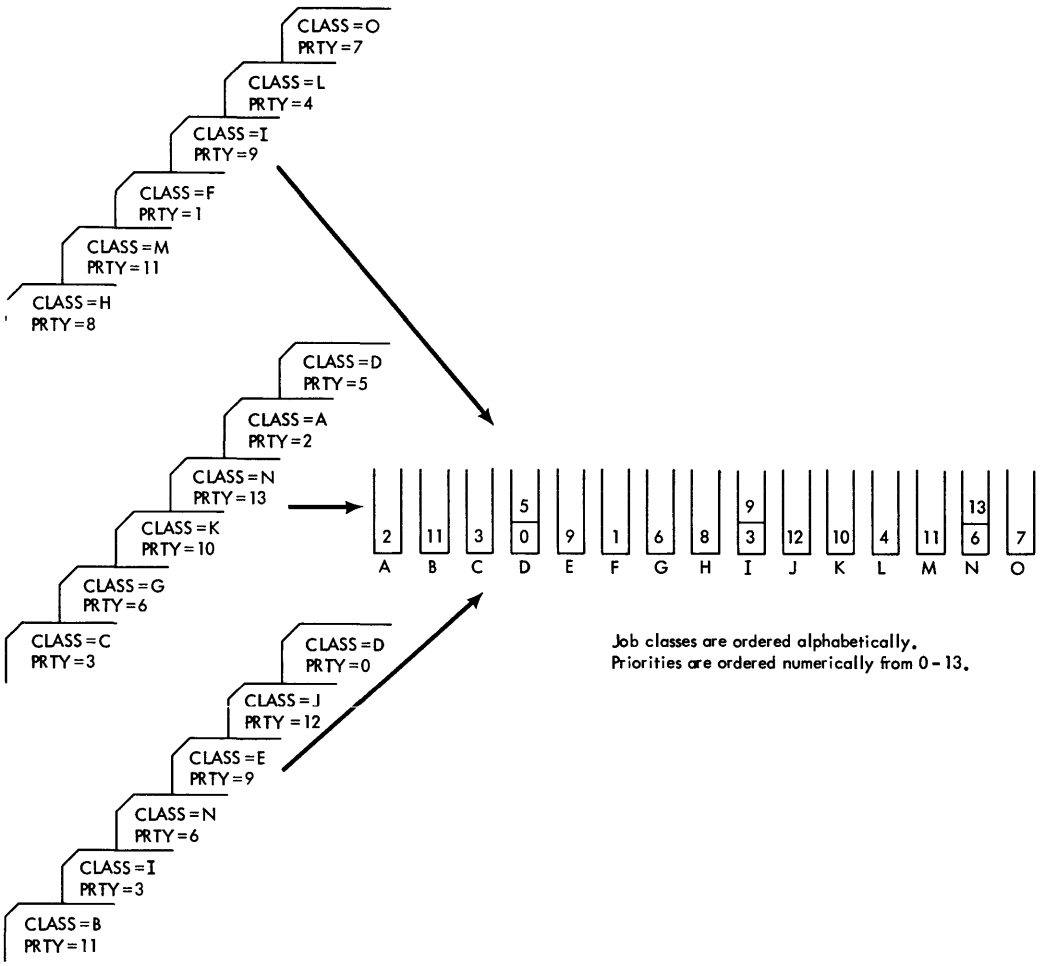


Figure 2: Job Classes and Job Priorities

The actual queueing process is complex The operator starts a reader/interpreter with a START READER command. The reader/interpreter -- a routine in job management -- begins reading and interpreting JCL, and queueing jobs. It reads the JCL, determining from it classes and priorities for the job. The reader/interpreter then queues jobs from the job stream into work queues. Figure 3 illustrates jobs being read into an input data set and being processed. The initiator/terminator, in response to a START command, allocates necessary resources to the job using the information provided by the JCL. Resources allocated include data sets and main storage needed for control blocks. At this time, data management routines assign necessary space for output data sets needed when the job terminates. However, if the system does not have sufficient resources, the job will wait for the resources to become available, or, if the job waits too long, the operator may cancel it.

Since the system has a finite amount of resources, and each job competes for those resources, the MVT control program must resolve the demands for system resources. The control program uses priority to resolve demands for system resources; it also uses interruptions.

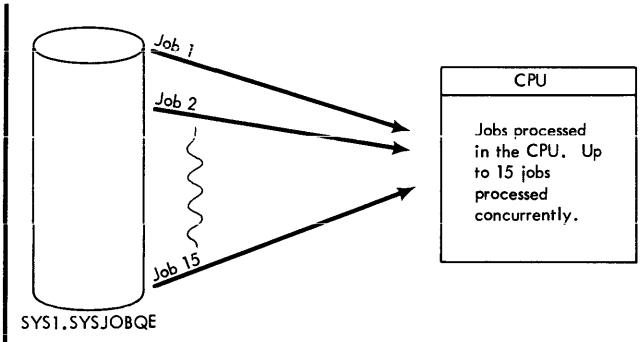


Figure 3. Jobs are Stored on SYS1.SYSJOBQE and then Processed

The initiator/terminator uses a macro instruction, ATTACH, to logically connect the job to the control program. At this point, the control program recognizes the job as a task (or as several tasks, depending on the processing involved). A control block, the task control block (TCB) represents the task to the control program. These TCBs go onto the TCB queue. Tasks ready to be processed are dispatchable and the TCBs representing them in the TCB queue indicate that they are dispatchable. Conversely, tasks not ready for processing are non-dispatchable and the TCBs representing them in the TCB queue indicate this. Following an interruption, the Supervisor gains control of the CPU from the task being processed. There are five kinds of interruptions:

- Supervisor Call interruptions
- Timer/external interruptions
- I/O interruptions
- Program interruptions
- Machine Check interruptions

The Supervisor handles each interruption and then checks the TCB queue for the highest priority dispatchable task. This highest priority task may or may not be a different task than the one that lost control.

The output can be written during program execution by a direct system output writer, or the output from each task can go to an output data set on an intermediate direct access storage device. Figure 4 illustrates this. After the task finishes processing, the Supervisor removes the TCB from the TCB queues, and the initiator/terminator deallocates the resources that the task used. Control blocks in the output work queue point to the intermediate output data. These control blocks have the same priority and class as the jobs they represent. MVT supports up to 36 output classes, designated A through Z and 0 through 9. The operator controls output by starting an output writer for a specified class. The output writer writes the jobs from the output data set to a user-specified device.

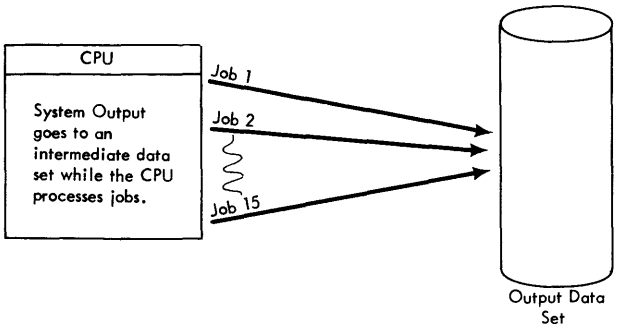


Figure 4. Processed Jobs Written to an Output Data Set

Main Storage Areas in MVT

Main storage in the MVT environment consists of four areas, as illustrated in Figure 5:

- Fixed area
- System queue area (SQA)
- Dynamic area
- Link pack area (LPA)

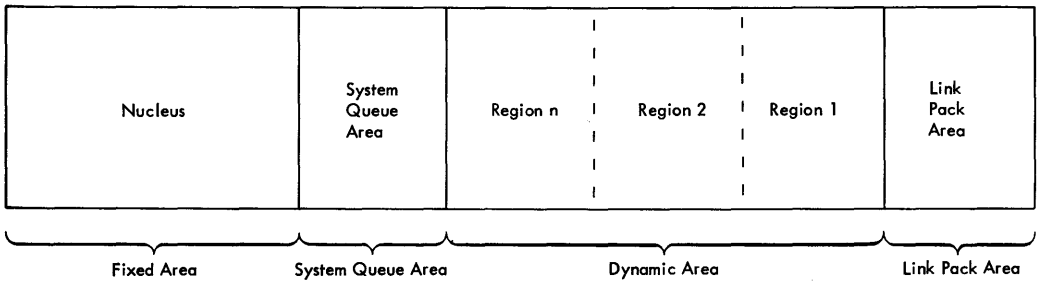


Figure 5. Main Storage Areas.

The Fixed Area: The fixed area contains the nucleus and the SVC and I/O Supervisor transient areas. This fixed area cannot be altered.

The System Queue Area: The system queue area contains the control blocks and tables needed by the control program.

The Dynamic Area: The dynamic area consists of regions; the areas where the CPU processes jobs. The regions in the dynamic area have no fixed size; they vary in size to accommodate the job that occupies them. A job's size can be described with parameters in the JCL, or a default value can be used for job size. The control program will assign a region of the proper size. The dynamic area can contain up to 15 jobs, one in each region, if the space exists. Thus, an installation operating under an MVT control program can process several jobs of various sizes concurrently.

The Link Pack Area: The link pack area contains routines frequently used by the control program, such as error recovery routines and selected SVC routines.

Major Components of the MVT Control Program

The MVT control program consists of five functions performed by five major groups of routines:

1) Job management routines direct and control the flow of jobs. Job Management also directs the use of the CPU, programming resources, and main storage.

2) Task management routines supervise the execution of all work done in the system. They also control the allocation of system resources on a priority basis.

3) Data management routines move information between main storage and auxilliary storage. They direct the usage of auxilliary storage devices and control input/output operations.

4) Volume management routines determine the operational quality of tape devices containing data used by the control program.

5) Recovery management routines record data and attempt retry operations after machine malfunctions.

Figure 6 illustrates the five functions of the control program and shows their relationship to each other.

These five routines reside in either the fixed area or the dynamic area of main storage; not all the routines reside in main storage at one time. The routines that reside in the nucleus provide supervisory functions; they can bring other non-resident portions of the control program into main storage. A protection key of 0 protects these routines.

The non-resident routines usually operate in the problem state in various locations in main storage. These routines may or may not be protected from alteration.

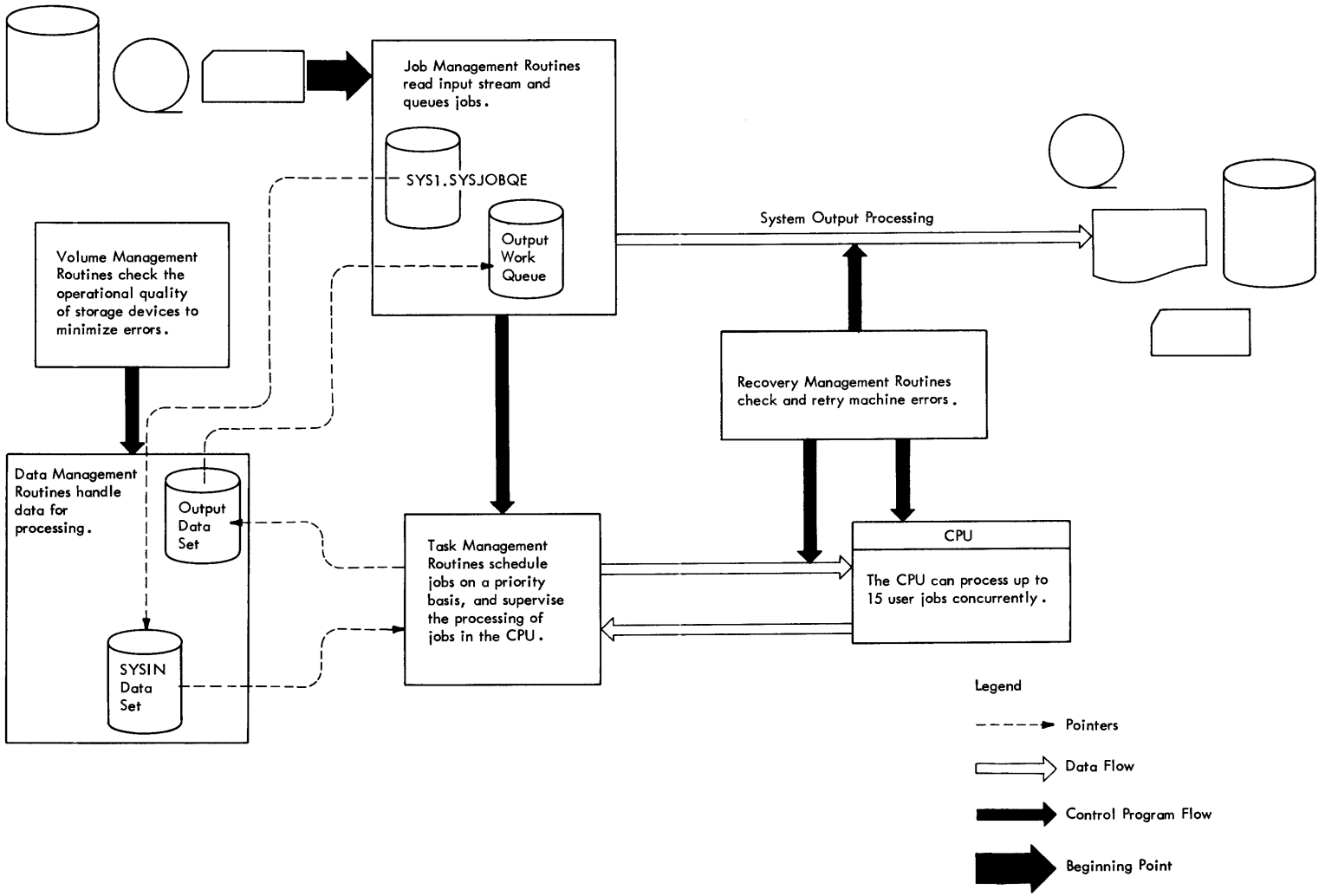


Figure 6: MVT Control Program Routines

How is an MVT System Generated?

The process of constructing, or generating, an MVT system is called system generation. The two-part process of preparing a system for actual work is called initial program loading and nucleus initialization. System generation only occurs for one of three reasons:

- To add new devices to a system
- To add a new or different nucleus to an existing system
- To initiate a new system

Initital program loading and nucleus inititalization occur every time the system is restarted after being shut down.

The system generation (SYSGEN) process combines separate IBM-supplied modules into a cohesive system after the user determines device and programming requirements. A special group of macro instructions compiles these modules into a system.

The initial program loading (IPL) routines load the control program into main storage. The nucleus initialization program (NIP) prepares main storage for processing jobs.

Section I: The MVT Control Program

This section provides a general description of MVT; it explains the MVT control program and how it works. The major components of the control program and their functions are described. Section V: Logic Summary, describes the internal logic of the MVT control program. Detailed descriptions of the implementation of the control program functions are in the program logic manuals listed in Appendix E.

Operating Environment

The control program routines operate in the supervisor state or the problem state. Therefore, the use of certain control and I/O instructions is restricted to certain routines. Secondly, organization and assignment of main storage in the operating system is based on the protection feature of System/360. The protection feature allows sections of main storage to be reserved for use only by certain routines. The operating states and the protection feature are described in the **Principles of Operation** publication. This chapter describes:

- Routines in the supervisor state
- Routines in the problem state
- Orgainization of main storage

Routines in Supervisor State

Certain routines of the control program operate in supervisor state. These routines have exclusive control over the privileged functions. They can execute a special group of instructions called privileged instructions, which perform functions such as starting I/O operations, enabling and disabling interruptions, and changing storage protection keys. In other words, control program routines in the supervisor state perform both supervisory and service functions. Some routines in the supervisor state are resident, some are nonresident. Thus, the control program has exclusive control over privileged functions, and it can insure the integrity of all the program and data in a multiprogramming environment.

Resident Routines

The resident routines of the control program (the nucleus) are loaded into main storage during the initial program loading (IPL) procedure, and are never overlaid by another part of the operating system. The nucleus contains all the task management routines (except for some nonresident SVC routines), one job management routine, the I/O supervisor and BLDL routine of data management and the resident recovery management routines. The routines in the nucleus operate under program status words (PSWs) with protection keys set to zero. The nucleus is one load module that is a member of the NUCLEUS partitioned data set (SYS1.NUCLEUS).

The routines in the nucleus perform primarily supervisory functions. The service routines of the nucleus are the resident SVC routines.

SVC routines are entered as a result of SVC interruptions, and perform control program services. There are four types of SVC routines, and two are resident and two are non-resident:

- Type 1 SVC routines, which are part of the nucleus and are disabled (masked) for all interruptions except machine-check interruptions.
- Type 2 SVC routines, which are part of the nucleus but may be enabled (interruptable) for part of their operation.

Nonresident Routines

The nonresident control program routines that operate in supervisor state are types 3 and 4 SVC routines, I/O error-handling routines, and parts of one recovery management routine.

The non-resident SVC routines are:

- Type 3 SVC routines, which may be enabled, and are not larger than 1024 bytes.
- Type 4 SVC routines, which may be enabled, and are larger than 1024 bytes. They are brought into main storage in segments of 1024 bytes or less.

The nonresident SVC routines reside in the SVCLIB partitioned data set (SYS1.SVCLIB), and operate either from areas defined in the nucleus called SVC transient areas or from the link pack area. Like the resident routines, nonresident SVC routines operate under PSWs with protection keys of zero.

As long as an SVC routine (or a module of an SVC routine) is in a transient area, that copy is used as many times as it is requested.

The I/O error-handling routines reside on SYS1.SVCLIB and operate either from the I/O supervisor transient area or from the link pack area. They are called by the I/O supervisor, and either correct an error that occurred during an I/O operation, or post a code for an access method routine.

The machine-check handler (MCH) routine of recovery management is partially nonresident. Its nonresident modules reside in SYS1.SVCLIB, and operate from its own transient area in the nucleus. The MCH routine attempts to recover from a machine check interruption so that processing can continue.

The dynamic device reconfiguration (DDR) routine of recovery management is mainly nonresident; however, a portion of the DDR routine is resident (DDR SYSRES).

Routines in the Problem State

The control program routines that operate in the problem state are job management routines and the access method routines. These routines operate from the dynamic and link pack areas. Job management routines operate under PSWs having protection keys of zero because these routines store data in the nucleus and system queue area. Access method routines operate under the same PSWs as their callers. The job management routines reside on SYS1.LINKLIB, the access method routines on SYS1.SVCLIB.

Organization of Main Storage

The relative positions of the four areas of main storage, and the program or data that occupies these areas is shown in Figure 7.

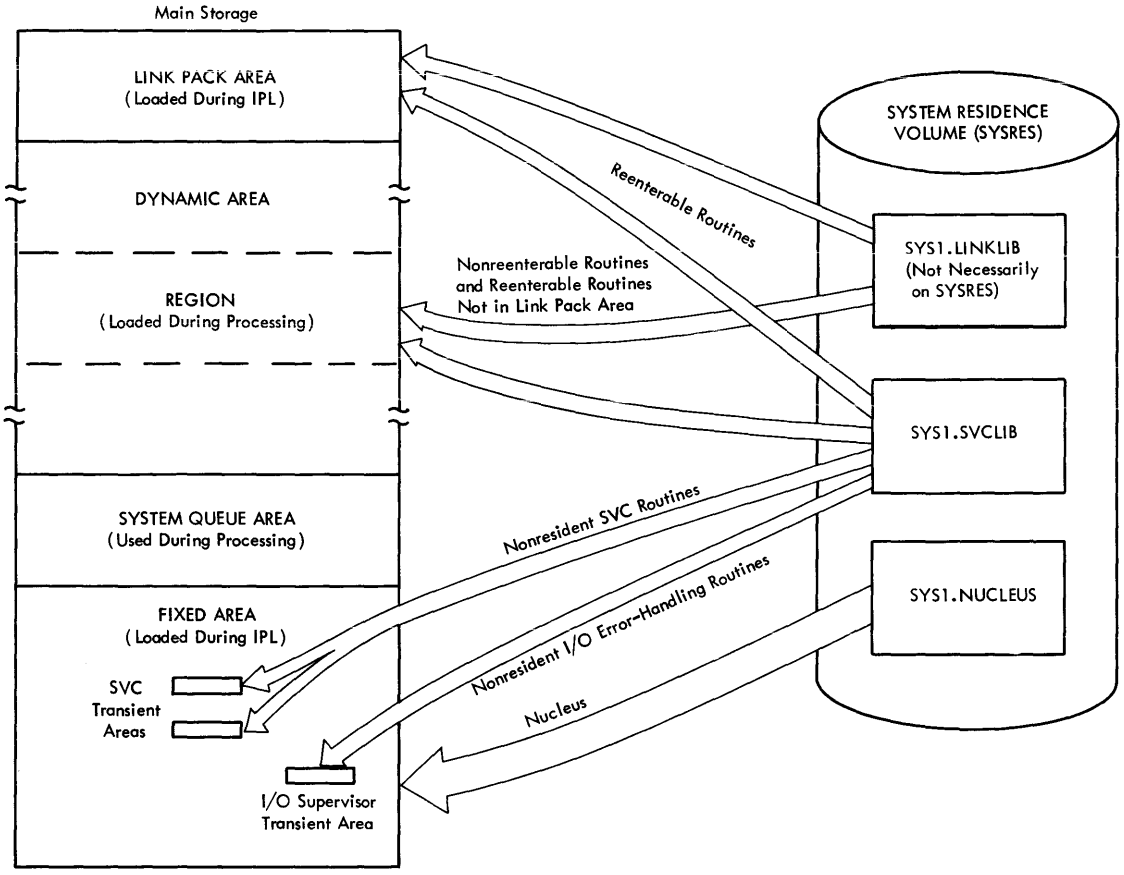


Figure 7. Areas and Contents of Main Storage

Fixed Area

The fixed area is that part of main storage into which the nucleus is loaded at IPL time. The storage protection keys of the fixed area are zero so that its contents can be modified by the control program only. The fixed area also contains small areas called transient areas into which certain nonresident routines are loaded when needed.

Transient areas are defined in the nucleus, and embedded in the fixed area. There are two types of transient areas: SVC transient areas and the I/O Supervisor transient area; these areas are used by nonresident SVC routines and nonresident I/O error-handling routines, respectively. Like the rest of the routines in the fixed area, the transient area routines operate with protection keys of zero. All routines that operate from transient areas reside on SYS1.SVCLIB.

SVC Transient Areas

An SVC transient area is 1024 bytes in length and is reserved for nonresident SVC routines. In the MVT configuration, the number of SVC transient areas is specified at system generation. The minimum number is two. When a nonresident SVC routine (or a module of a nonresident SVC routine) is required and is not already in the link pack area or one of the SVC transient areas, the routine or module is read into an available transient area. If no SVC transient area is available, and if none can be made available, the task for which the SVC routine was called is put in the wait state until an area becomes available. A transient area is

available when it is empty or when the SVC routine that occupies it has completed its operation.

If no SVC transient area is available for an SVC routine, a transient area currently being used can be appropriated. An area is appropriated when the task requiring the area has a higher priority than all the tasks that are currently using the area. When several transient areas fall into this category, the area appropriated is the one having the lowest priority user. The appropriated transient area is loaded with the SVC routine for the higher priority task. The SVC routine that is overlaid because of this higher-priority requirement is later reloaded so that it can complete its operation.

I/O Supervisor Transient Area

There is one I/O supervisor transient area. It is 1024 bytes in length and is reserved for nonresident I/O error-handling routines that are brought into main storage for the I/O supervisor.

System Queue Area

The system queue area is adjacent to the fixed area and provides the main storage space required for tables and queues built by the control program. The nucleus initialization program (NIP) sets up the system queue area. Its storage-protection key is zero so that it can be modified by control program routines only. The data in the system queue area indicates the status of all the tasks and many of the resources in the system.

Link Pack Area

The link pack area contains reenterable routines that reside on SYS1.LINKLIB and SYS1.SVCLIB, track addresses of other routines on these two libraries, and serially-reusable I/O error recovery routines that reside on SYS1.SVCLIB. The routines in the link pack area are used for all the tasks that require them, and need not be loaded into the various regions of main storage. The list of track addresses (the BLDL list) reduces the time required to find the listed routines on SYS1.LINKLIB and SYS1.SVCLIB. These routines are loaded into the region of the task that requires them. Types 3 and 4 SVC routines and I/O error recovery routines in the link pack area operate in supervisor state. The others generally operate in the same state as the routines that called them. The organization of the list and routines in the link pack area is shown in Figure 8.

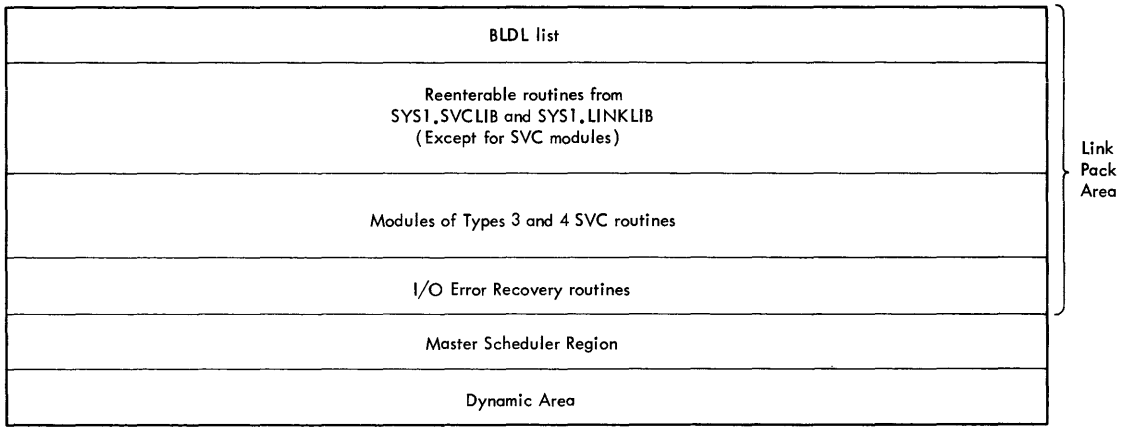


Figure 8. Upper Main Storage After IPL

You select the routines that you want in the link pack area by creating lists of the desired routine names. At IPL time, the NIP program loads the indicated routines starting at the highest part of main storage and working downward. After nucleus initialization, the contents of the link pack area cannot be modified unless the IPL procedure is repeated. If IBM 2361 Core Storage and main storage hierarchy support are included in the system, a secondary link pack area may be created in hierarchy 1 to contain other nonresident SVC and reenterable modules. These will also be loaded by the nucleus initialization program at IPL time.

In addition to user-specified routines, several system-specified job management and access method routines also reside in this area.

Dynamic Area

The dynamic area fulfills the main storage requirements of job steps and system tasks. This area is all the main storage between the link pack area and the system queue area. As job steps and system tasks are initiated, storage from the dynamic area is allocated to them in blocks called regions.

The dynamic area may be expanded by the inclusion of IBM 2361 Core Storage, an extension of IBM 2050, 2065, or 2075 Processor Storage. Main Storage hierarchy support for IBM 2361 Models 1 and 2 is a control program option that permits selective access to either the processor storage (identified as hierarchy 0) or IBM 2361 Core Storage (identified as hierarchy 1) portions of main storage. When the main storage hierarchy support option is selected, a region may be defined to consist of two parts: the first located in hierarchy 0 and the second located in hierarchy 1. If IBM 2361 Core Storage is not included in the system and a region is defined to exist in two hierarchies, a two-part region is established within processor storage. The two parts are not necessarily contiguous.

Normally, all storage requested by programs of a given step or task is assigned from its region, although the rollout/rollin feature does provide the capability of acquiring temporary additional storage.

Regions are assigned from the highest available block of dynamic area storage that is large enough to fill the request. A region is assigned when the step or system task is initiated, and, remains assigned for the life span of the step or task. Any released region becomes part of the free storage in the dynamic area and is again available for allocation to a job step or system task.

The region for the master scheduler task is assigned at IPL time by the nucleus initialization program. This region is adjacent to the link pack area (see Figure 8), and remains assigned for as long as the control program is in storage.

A region assigned to a job step has a protection key value from 1 to 15 associated with it. As 2K blocks of storage within this region are assigned for the step, the protection key of each block is set to the associated value. (Storage assigned to subpool 252 is an exception; its key remains set to zero.) Unassigned blocks of storage within a job step region have their protection keys set to zero.

Regions assigned to system tasks have protection keys of 0. Protection keys of dynamic area storage that is not part of any region are also set to 0.

Generalized Trace Facility

The Generalized Trace Facility (GTF) program service assists problem determination and analysis by tracing system events, user events, or both. GTF records and formats trace data on tape, direct access storage, or internally in the GTF region. GTF can trace:

- I/O interruptions (including program controlled interruptions), both for event classes or for an individual device
- SIO operations, both for event classes or for an individual device
- SVC interruptions, both for event classes or by individual SVC numbers
- Program interrruptions, both for event classes or for an individual interrupt code
- External interruptions
- Task switches by the system dispatcher
- User events

GTF performs a wider variety of tracing operations than the OS trace option, as well as the same operations. Unlike the OS trace option, GTF is part of the MVT control program, occupying less than 1000 bytes when inactive and the specfied region size when active. The OS trace option can be included in a system with GTF; but starting GTF suspends the operation of the trace option.

GTF has a macro to simulate the System/370 monitor call (MC) instruction. This allows GTF to be used with System/360 CPUs, and on System/370 CPUs without the monitor call instruction.

GTF comprises two major functions:

- Generalized trace function
- Trace edit function

Generalized Trace Function

An operator-issued START command initiates the Generalized Trace function as a system task in a region (which must be in hierarchy 0 if the installation uses main storage hierarchy support). The operator can specify in the START command the:

- Tracing in main storage or for an external device
- Suspension of ABDUMP formatting of trace buffers
- Timestamping -- or recording the time of the occurence of an interrupt -- for every logical trace record

Once the operator starts the trace task, he replies to the message "SPECIFY TRACE OPTIONS", requesting the:

- Tracing of event classes
- Tracing of single events
- Immediate termination of GTF on error condition in GTF
- Tracing of user-oriented data events

Parameters for the TRACE keyword can be listed in the SYS1.PARMLIB data set, making it unnecessary for the operator to issue parameters after starting the generalized trace function.

GTF traces an event after receiving control from the interrupt handler. GTF gathers information about the interruption, and records it on the trace data set or maintains it internally.

Trace Edit Function

IMDPRDMP formats GTF trace data sets with the EDIT control verb, using either the trace data set or core image dumps generated by IMDSADMP or by the System Dump Facility as input. To aid in problem determination, keywords in the EDIT control verb allow the user to select material from the GTF trace data set, such as jobnames, TCB addresses, and single or multiple events in an event class rather than using than the entire trace data set.

The **Service Aids** SRL describes both the generalized trace function and the Edit function in detail.

Loading and Initializing the Control Program

Before the operating system can be used, the nucleus must be loaded from the system residence volume into main storage, and initialized. These functions are performed by the initial program loading (IPL) program, and the nucleus initialization program (NIP) respectively. These programs are described in the **Initial Program Loader and Nucleus Initialization Program PLM.**

Loading the Nucleus

Part of the initialization procedure for the system residence volume was the placing of two IPL records at track 0, cylinder 0 preceding the standard volume label.

To load the nucleus, you specify the system residence volume and press the LOAD button on the console. This action causes the first of the IPL records to be read into location 0 of main storage and to be given CPU control. This record reads the second IPL record which, in turn, reads the IPL program into main storage.

The IPL program clears and determines the size of main storage, and sets all the storage protection keys to zero. If IBM 2361 Core Storage is part of an MVT system that includes main storage hierarchy support, the IPL program determines the boundary between the processor storage (hierarchy 0) and IBM 2361 Core Storage (hierarchy1) portions of main storage. The IPL program then relocates itself into the upper portion of main storage, clears its old location, and loads the nucleus into the lower portion of main storage. After completing the operation, the IPL program passes control to the nucleus initialization program.

Initializing the Nucleus

The nucleus initialization program (NIP) is a control section included with the nucleus when the system is generated. NIP initializes tables in the nucleus, determines addresses and storage boundaries of routines and tables in the nucleus, and checks and sets the interval timer. NIP converts any time-slice intervals to timer units, obtains a primary/master console, determines the hard copy log requirements and obtains a hard copy log if required. Nip defines the boundaries of the system queue area, loads the link pack area, and assigns the region of the master scheduler task. NIP also initializes the SYS1.DUMP data set that is used by the damage assessment routine (DAR) to write a core image dump when a system failure occurs. NIP then passes control to a routine of the master scheduler task.

The initializing routines of the master scheduler task initialize the input and output work queues, open the SYS1.LOGREC data set, initialize the system log and the remaining consoles in the system and execute any automatic commands specified during IPL. If system environment recording (SER) routines were specified, the initializing routines select the appropriate SER routine for the model and load it into the nucleus. If system management facilities (SMF) are included with the system, the initializing routines also create an SMF task, place it into a wait state, and open the SYS1.MAN data set. After the master scheduler has completed its initializing functions, it is placed in the wait state. If the automatic commands did not result in any tasks that can now be performed, the system is placed in the wait state.

System Restart

If during later processing a system failure requires that the IPL procedure be repeated, the operator can preserve at least part of the contents of the input and output work queues. This preservation of queues allows the system to be restarted after the IPL procedure is complete without rereading all the jobs that were in the system when the failure occurred. During the new IPL procedure, the system restart routines purge the queues of unprocessable entries. The queue entries that are preserved are:

- Those representing jobs that are enqueued but not yet started.
- Those representing started jobs that can be restarted.
- Those representing system messages and SYSOUT data sets.

The remaining queue entries for jobs being processed when the failure occurred are purged from the queues, and a message is issued to the operator indicating that these jobs must be resubmitted. Any temporary data sets of these jobs are purged, but system messages and SYSOUT data sets that are complete are written.

System restart routines are described in the **MVT Job Management PLM**.

Nucleus Initialization in a Multiprocessing Environment

The initial program loading (IPL) process is unchanged for MVT with Model 65 multiprocessing, but the nucleus initialization program (NIP) has some additional functions.

The NIP program begins by determining if the system is to be operated in a two-CPU mode, a one-CPU mode, or a partitioned mode.

If the system is to be operated in a two-CPU multisystem mode, during the first of two phases NIP initializes the first CPU, loads a PSW into location zero of the second CPU, and them issues an IPL signal for the second CPU. During the second phase, initialization is similar to that for the first CPU, except that the interval timer for the second CPU is given a very high value. This timer is not used, but is reset with the same high value every time the timer for the first CPU is reset. This ensures that the second timer will cause no interruptions. After the second CPU is initialized, NIP passes control to a routine of the master scheduler task. After master scheduler routines have performed their initialization functions, the system is ready to begin processing jobs.

Job Management

Job management routines process communications from the programmer and the operator to the control program. This processing falls into two categories: **command processing** and **job processing.**

Command processing is the reading, scheduling, and executing of operator commands issued via either a console device or an input job stream. Job processing is the reading and interpreting of control statements, the initiating of job steps defined in these statements, and the writing of system messages and system output (SYSOUT) data sets from the intermediate volumes on which they were originally placed.

Job management routines perform several tasks to accomplish command processing and job processing. The job management tasks are referred to as **system tasks** to distinguish them from the user tasks that are performed by processing programs. The job management tasks and the routines that perform them are described in the **MVT Job Management PLM.**

When a system task is created, it is assigned a region of main storage. The routines for this task that are not in the link pack area are loaded into and operate from the region. Regions of system tasks have storage protection keys of zero. Routines that perform system tasks (unlike those that perform user tasks) operate under a PSW with a protection key of zero so that they can write not only in their regions but also in the system queue area.

Task Management

Task management routines control the allocation and use of CPU, main storage, and programming resources. The task management functions are:

- Interruption supervision. The analysis of interruptions to determine what supervisor processing is required.
- Task supervision. The recording of what tasks are currently in the system, their status, priorities, the programs they require, and the order in which these tasks are to be performed.
- Main storage supervision. The allocating and freeing of main storage, and recording of what use is being made of any portion of main storage.
- Contents supervision. The loading of programs into storage, the recording of what programs are currently in main storage, and what characteristics these programs possess.
- Timer supervision. The setting and maintaining of the interval timer from information provided in timer macro instructions.

Except for some nonresident SVC routines, the task management routines are part of the nucleus. All task management routines operate in the supervisor state under a PSW with a protection key of zero.

The task management routines are described in the **MVT Supervisor PLM.**

Data Management

Data management routines:

- Assign and release space on direct access volumes.
- Maintain the catalog.
- Perform the I/O support (open, close, end-of-volume) processing.
- Process I/O operations.

The first three of these functions are performed by type 3 and 4 data management SVC routines; they reside on SYS1.SVCLIB and operate from either the link pack area or the SVC transient areas. These routines are invoked via SVC instructions that are either coded directly or generated as part of macro instruction expansions in the calling program. When the SVC instruction is executed, the resulting SVC interruption causes control to pass to the SVC interruption handler. The desired SVC routine is brought into an SVC transient area (unless a copy is already in one of the areas or the link pack area), a request block for the SVC routine is built and enqueued to the appropriate TCB, and the SVC routine is given control. Upon completion, the SVC routine returns control to the supervisor.

The fourth function, processing of I/O operations, is performed by access method routines and the I/O supervisor. The access method routines reside on SYS1.SVCLIB and operate from either the link pack area, or the region of their associated task. These routines are loaded by the open SVC routine, and are entered via a branch instruction that is part of the expansion of the macro instruction for that access method.

The I/O supervisor is part of the nucleus. The portion of the I/O supervisor that processes I/O operations is a type 1 SVC routine invoked by an EXCP macro instruction normally issued by an access method routine.

Volume Management

One of the major factors affecting an operating system is the condition of volumes stored on a medium subject to deterioration with use. Magnetic tape is such a medium. During use, it is stretched, flexed, and rubbed, causing its oxide coating to crack or to be eroded. Eroded particles of oxide, fingerprints, and dust contaminate its surface, multiplying erosion, and breaking contact between the tape and the read/write station. In time, there will be failures in the read and write processes.

A rapidly rising rate of read and write errors, if detected, would signal the probability of a deteriorating tape. If the failure rate could be monitored, it would be possible to judge the condition of the volume and rescue its contents, either by reconditioning, by transfer to a different volume, or by a combination of both processes, before system performance could be seriously affected.

Read and write errors for a given volume can be monitored by a facility called OS Volume Statistics which supports tape volumes. This facility has two options, error statistics by volume (ESV), and error volume analysis (EVA). Both options can be specified at the same time.

Recovery Management

When a machine malfunction occurs, recovery management routines record critical machine and program data, and (in some cases) attempt to recover from the error. Depending on the specific routine and type of error, recovery takes place at one of four levels:

1. Functional recovery -- resumption of the task at the point where the error occurred. Machine or recovery management facilities correct storage errors, retry unsuccessful instructions and I/O operations.
2. System recovery -- termination of the task affected by the error, permitting system operation to continue.
3. System-supported restart -- re-IPL using system job and data queues preserved by system restart facilities.
4. System repair -- total system halt for manual repairs, aided by recovery management records.

Recovery management records are written in SYS1.LOGREC, a dedicated data set on the system residence volume. They can be edited and printed by use of the IFCEREP0 service aid program, described in the **Service Aids** publication.

Task Directory for Section I: The MVT Control Program

For information about:

- Main storage organization, see
  Fixed Area
  System Queue Area
  Link Pack Area
  Dynamic Area

- Operating in the supervisor state, see
  Resident Routines
  Nonresident Routines

- Tracing events in the control program, see
  Generalized Trace Facility

- Nucleus loading, see
  Loading the Nucleus

- Nucleus initialization, see
  Initializing the Nucleus
  Nucleus Initialization in a Multiprocessing Environment
  System Restart

Section II: MVT Options

This section discusses in general terms, the options available with MVT that provide user services and help optimize performance; the discussions are not intended to be comprehensive, and you will need to refer to other publications. The **System Generation** publication explains how to include these options into the control program.

Additional Pairs of Transient Areas

One pair of supervisor (SVC) transient areas is always provided in an MVT system; optionally, additional pairs may be added. When a non-resident SVC routine is required during job execution, it is loaded into an available transient area. If no transient areas are available, then the task requiring the routine is placed in a wait state until one becomes available.

Alternate Path Retry (APR) -- Standard with M65MP

The alternate path retry (APR) option allows an I/O operation that has developed an error on one channel path to a device to be retried on another channel path to the same device. This can be done only if another channel path has been assigned to the device performing the I/O operation. APR also provides the capability to vary a path to a device online or offline by use of the VARY command.

APR can handle:

- Up to four paths to one device
- Two paths to a CPU for a multiprocessing system
- The ninth drive on a 2314

While it is not module dependent, APR only performs it function usefully in a system that has the channel check handler (CCH) and alternate paths to one or more I/O devices.

The operation of the selective retry function of APR, done in conjunction with the I/O supervisor, is automatic. The VARY path function can be initiated by entering the VARY PATH command in the input stream or at the console.

BLDL Table Made Resident

Any or all of the SYS1.LINKLIB directory entries can be made resident in fixed main storage. The user can modify this list to fit his requirements. If he creates a list of his own, the operator communication option in the SUPRVSOR macro instruction must be specified so that he can have his list brought in during system initialization.

The standard list of SYS1.LINKLIB directory entries, IEABLD00, can be made resident. This BLDL list has nine entries. To use the BLDL list, the operator communication option must be specified at system generation time in the OPTIONS parameter of the SUPRVSOR macro instruction. This will cause the "SPECIFY SYSTEM PARAMETERS" message (IEA101A) to specify a different BLDL list to be used during the loading of the nucleus.

Channel-Check Handler (CCH)

CCH intercepts channel-check conditions, performs an analysis of the environment, and facilitates recovery from channel-check conditions by scheduling device-dependent error

recovery procedures by the input/output supervisor, which will determine whether the failing channel operation can be retried. If CCH is not present in the system, one of the other recovery management facilities receives control and writes an error record for the channel failure. In this case, the error causes system termination.

This feature is optional in the System/360 Models 65, 75, and 91 if the models are specified in the CENPROCS macro instruction.

It is automatically included in the System/360 Models M65MP, 85, and 195 and System/370 Models 145, 155, 165, and 195 if the models are specified in the CENPROCS macro instruction.

Checkpoint/Restart Facility

The checkpoint/restart facility expands the use of the restart capabilities that are provided by the RD parameter that can be specified in either the JOB or EXEC statements. The RD parameter permits execution of jobs to be automatically restarted at a job step after abnormal termination occurs.

The checkpoint/restart facility enables the user to write checkpoint macro instructions (CHKPT) at various points in his program in order to record job status information. Then, when an ABEND occurs, his program can be automatically restarted at the last of these points. Or, restart can be deferred until a later time, when the job can be resubmitted and the RESTART parameter in the JOB statement is used. The RD parameter can also be used to suppress partially or totally the checkpoint/restart facility.

The following restrictions apply to the establishment of a checkpoint when using the CHKPT macro instruction.

- When the checkpoint is established, the job step must comprise a single task. The job step task must be the only task when the job step is restarted.
- A checkpoint cannot be established by an exit routine that returns control to the control program.
- If a STIMER or WTOR macro instruction has been issued, a checkpoint cannot be established before the time interval is completed or the operator's reply is received.

In order to use the checkpoint/restart facility, the user must indicate that he plans to use it at system generation time in the RESIDNT parameter of the SUPRVSOR macro instruction. The basic modules required from the SVC library (SYS1.SVCLIB) for the checkpoint/restart facility will then be loaded automatically at NIP time. In the programs that contain CHKPT macro instructions, a checkpoint data set and work area must be defined. The checkpoint/restart cataloged procedure (IEFREINT) must be in SYS1.PROCLIB either before or after system generation.

Additional modules from the SVC library will be required if chained scheduling or track overflow are going to be used. The user obtains the additional modules by constructing his own access method option list (IEAIGGxx) and includes this in the parameter library (SYS1.PARMLIB). In order to use his own access method list the operator communication option must be specified at system generation time in the OPTIONS parameter of the SUPRVSOR macro instruction. This will cause the "SPECIFY SYSTEM PARAMETERS" message (IEA101A) to be printed during NIP and provides the operator with the opportunity to specify a different access method option list to be used during the loading of the nucleus.

Consoles--Alternate and Composite Console Options

One primary console must always be specified for any operating system except the M65MP system. M65MP must have two primary consoles specified except when the multiple console option (MCS) is specified. (See the description in "Consoles - Multiple Console Support (MCS)".) One alternate console can be specified, or two for the M65MP system, when MCS is not selected. A composite console (e.g., a card reader and a printer) can be specified as a primary or an alternate console. The composite console is considered as one console even though it may be two different hardware devices.

The following guidelines must be used when MCS is not selected:

- A primary console must be specified in the SCHEDULR macro instruction.
- A composite console can be used as a primary or an alternate console.
- For M65MP, no more than two of the total number of consoles specified can be composite consoles.
- When a graphic device is going to be active as a console, a device that produces printed output must be specified.

Consoles--Multiple Consoles Support (MCS)

The user must specify the multiple console support (MCS) option to have two or more consoles active during execution time. For M65MP systems, if you want to direct messages to or receive commands from more than one console, you must specify MCS.

The following guidelines must be followed with MCS:

- One console must be specified in the SCHEDULR macro instruction and is called the "master" console.
- An alternate console for the master console must be specified in the ALTCONS parameter of the SCHEDULR macro instruction.
- A SECONSLE macro instruction must be coded defining the alternate as a secondary console.
- Additional secondary consoles can be defined with SECONSLE macro instructions -- up to a maximum of 31 secondary consoles.
- For all consoles for which no alternate console is specified, the master console is automatically assigned as the alternate.

A hard copy log can be specified either at system generation time or by the operator during system initialization or execution time. A hard copy log is required when there is more than one active console during initialization or execution time, or when there is an active display console. The hard copy log can be the system log that is contained on SYS1.SYSVLOGX and SYS1.SYSVLOGY or it can be a console with output capability. If the log is required, the system records the operator commands, the system commands and responses, and the messages with the routing codes of 1, 2, 3, 4, 7, 8, and 10 on the hard copy log. Additional messages can be recorded if desired.

Routing codes and descriptor codes are required for all messages handled by a system using MCS. Messages that already exist can be assigned routing codes at system generation time, or, by default, they will be sent to the master console.

Routing codes are assigned to all new operator messages (WTO and WTOR). They designate what function the message is connected with and determine where a message will be sent. A system generation parameter provides the ability to supply routing codes to all operator messages that already exist and do not have a routing code.

Each console is assigned one or more routing codes. The routing codes assigned to a console are matched to the routing codes assigned to a WTO or WTOR message. If there is a match, the message is sent to the console. There are some messages that are not routed by the routing code, e.g., a message that is broadcast to all active consoles.

Descriptor codes must be specified for all new operator messages. They are specified in the WTO or WTOR macro instructions. They designate how a message is to be printed or displayed.

All commands have been arranged by function into four command code groups: informational, system control, I/O control, and console control.

An exit is provided, just before the routing codes of a message are checked, to enable the user to supply his own routine to add, delete, or change routing and descriptor codes.

The following guidelines must be used:

- If HARDCOPY=SYSLOG is specified in the SCHEDULR macro instruction during system generation, then at IPL time the operator must change the HARDCPY parameter to refer to the address of an operator console that has output capability. The device should not be the master console. The HARDCPY specification can be changed back after the message IEE141I has been received. (For detailed operating instructions see the **Operator's Reference** publication.)
- A master console must be specified in the CONSOLE keyword parameter of the SCHEDULR macro instruction.
- An alternate console must be specified in the ALTCONS keyword parameter SCHEDULR macro instruction.
- The alternate must be defined in the CONSOLE parameter of a SECONSLE macro instruction.
- A console with at least printed output capability must be specified as the hard copy log. Although the system log is not a console it can be used even though it does not directly produce printed output.
- A record of the operator commands, the system commands and responses, and routing codes 1, 2, 3, 4, 7, 8, and 10 should be maintained.
- Up to 31 secondary consoles can be specified with SECONSLE macro instructions. They can all have alternate consoles specified. If no alternate is defined, then the master console automatically becomes the alternate.
- A 2250 Display Unit can be specified as a master, secondary, or alternate console.
- Any number of consoles can be composite consoles.
- Routing and descriptor codes are assigned to all new operator messages that are written.

Conversational Remote Job Entry (CRJE) Facility

The conversational remote job entry (CRJE) facility provides remote access to the operating system from printer-keyboard terminals. Authorized terminal users can conversationally prepare and update programs and data, submit them for OS background processing, and receive the output either at the central installation or at the remote terminal.

Conversational remote job entry (CRJE) requires the basic telecommunication access method (BTAM) routines. Background execution of CRJE-submitted jobs is accomplished concurrently with normal batch processing under the supervision of the OS job management routines. The valid CRJE terminal user is one that has been defined in the system at CRJE assembly time in the CRJEUSER macro instruction or has been added to the system by the central operator using the USERID central command.

The terminal user can insert, replace, delete, or change information to be submitted in jobs by using the CRJE data set updating facilities. He can have PL/I or FORTRAN source statements checked for syntax errors before submitting the job. The syntax checking program(s) are included at system generation time by the CHECKER macro instruction.

The terminal user can inquire about the status of the system or remotely submitted jobs. There is also a message facility for two-way communication between terminal users, and between terminal users and the central operator.

CRJE is specified at system generation time in order to have the necessary modules included in the system. After generation, create the specific CRJE system required for the installation. There are three macro instructions available for this job -- CRJELINE, CRJETABL, and CRJEUSER. Set up a job that includes the CRJE macro instructions necessary to specify the system; users may include their own routines. The assembler translates these macro instructions and creates the required modules. The linkage editor incorporates the modules into the operating system.

SYS1.MACLIB must be in the operating system so that the assembler can expand the macro instructions. SYS1.TELCMLIB must be in the system to hold some of the CRJE load modules as well as the telecommunication subroutines. Enough system queue space must be specified in the CTRLPROG macro instruction during system generation to handle the necessary CRJE space requirements.

Decimal Similation Option for Model 91

The decimal simulation option provides the Model 91 with the ability to handle decimal arithmetic instructions; the Model 91 is not equipped with decimal arithmetic instruction circuitry. This option requires both a long execution time and that the CPU not be operational during the simulation. The universal instruction set, which is standard for the Model 91, includes only the EDIT and EDMK decimal instructions; any other decimal instruction is simulated. This option should be specified if COBOL, PL/1, or RPG is to be included in the system, or if decimal arithmetic instructions are to be used in assembler language.

Direct Access Volume Serial Number Verification

The user can add direct access volume serial number verification to his new system. If he does, the volume serial number of a direct access device is checked after an unsolicited device end interrupt condition has been corrected and the volume has been put back on line again.

When an unsolicited device end interrupt is received from a direct access device, the I/O supervisor (IOS) will insure that the volume serial number of the mounted volume agrees with the volume serial in the unit control block (UCB).

The coding to do the checking will be included at system generation time unless NODAV is specified in the OPTIONS keyword parameter of the SUPRVSOR macro instruction.

Dynamic Device Reconfiguration (DDR) -- Standard for M65MP

The dynamic device reconfiguration option allows a demountable volume to be moved from one device to another and repositioned if necessary, without abnormally terminating the job or redoing IPL. A request to move a volume may be initiated by either the system or the operator and the volume may be a system residence volume or any other volume.

The system transfers control to the DDR routines when a permanent I/O error occurs. These routines then determine if another device of the same type is available to which the volume can be moved. When another device is available the system requests a volume swap by issuing a message to the operator. The operator must answer this message by entering a SWAP command.

Sometimes the operator will determine that a volume needs to be swapped. He can initiate this action by entering a SWAP command.

The DDR routines will be used if:

- DDR, DDRSYS, or DDRNSL have been specified in the OPTIONS keyword parameter of the SUPRVSOR macro instruction during system generation.
- The device that has a permanent I/O error is a 2311, 2314, 2321, 3330, any 2400 series or 3400 series magnetic tape drive, a card reader, a printer, or a card punch. No teleprocessing devices are supported. Any device for which shared DASD has been specified can only be demounted and remounted on the same device. The DDR routines can be used for the unit record devices only if the operator issues the request by means of the SWAP command.
- The type of permanent I/O error is supported. The ones that are NOT supported are: wrong length record, no record found, unit exception, program check, protection check, IOB intercept condition, backing to load point, or when the permanent I/O error is caused by the channel program.

Notes:

- The user should not code specific unit addresses in programs that will be processed on a system that has DDR.
- The direct access serial number verification routines must be in the system that has the DDR routines.

For FETCH: When I/O errors occur while the FETCH routines are addressing the SVCLIB, the DDR system residence routines receives control, and, if possible, requests a swap. In order for this to occur, OPTIONS=DDRSYS must have been specified in the SUPRVSOR macro instruction and the conditions listed above must exist.

For DDR System Residence Routines: When these routines are specified in the OPTIONS keyword parameter of the SUPRVSOR macro instruction, another keyword parameter, ALTSYS, must also be specified.

If high availability is important to the installation, a duplicate system residence volume would be advisable. However, in order to use such a volume, writing on the system residence volume would have to be prohibited except to the SYS1.LOGREC data set.

The system residence device specified during system generation can be changed at IPL time by the operator. OPTIONS=COMM must be specified in the SUPRVSOR macro instruction during system generation in order to be able to make this change.

For Nonstandard Labels: If the user desires DDR and has nonstandard magnetic tape labels, OPTIONS=DDRNSL must be specified. A nonstandard label routine must be given the name

NSLREPOS. This routine can either be added during system generation using the SVCLIB macro instruction, or it can be linkage edited into SVCLIB after the system generation process is completed.

For DDR When EXCP is Used: When the EXPC macro instruction is used to address magnetic tape drives in a program that will run under a system with DDR, REPOS=Y or N must be coded in the DCB macro instruction to indicate whether an accurate block count is being maintained.

Graphic Programming Services

The graphic programming services handle graphic input and output and a set of problem-oriented routines that are used as building blocks in the construction of graphic processing programs. In addition, the graphic subroutine package (GSP) allows the FORTRAN IV or PL/I F progammer to use the graphic programming services.

The problem oriented routines are generalized routines that generate graphic instruction for displaying various images and alphameric information on the IBM 2250 Display Unit. These routines function as part of the problem program and are reached by a CALL or LINK macro instruction.

Indexed Sequential Access Method (ISAM)

The indexed sequential access method can be included in the new system so that tasks can use the basic indexed sequential access method (BISAM) or the queued indexed sequential access method (QISAM). If the user plans to use the SVC 2B (CIRB) in his programs, then BDAM, ISAM, or BTAM must be specified. (The MACLIB macro instruction must also be specified if CIRB is included.)

Job Step Timing

Each job step can be timed and the time limits enforced. The amount of time used is recorded after a job step is finished. In addition, the following are included in this option: the data plus the time of day, changing the time at midnight, and being able to request, check, and cancel intervals of time. (See the description of "Timing options" later in this section.)

Main Storage Hierarchy Support

Main storage hierarchy support provides selective access to either processor storage or IBM 2361 core storage.

Main storage is divided into two blocks known as hierarchies; hierarchy 0 is assigned to processor storage and heirarchy 1 to the 2361. Program controlled interrupt (PCI) must always be specified. (See the description of "Program Controlled Interrupt (PCI)" later in this section.)

For MVT systems, the hierarchy structures is maintained even though there may not be a 2316 Core Storage Unit on the system.

Program Controlled Interrupt (PCI)

The program controlled interrupt (PCI) facility permits the program to cause an I/O interrupt during execution of an I/O operation. PCI provides a means of altering the program of the

progress of chaining during an I/O operation. It also permits programmed dynamic main-storage allocation.

A routine, PCI fetch is able to bring a program into main storage with only one seek of the disk if:

- A buffer is always available for relocation dictionaries.
- No errors occur during the I/O operation.
- No cylinders are crossed while bringing in the program.
- The speed of the central processing unit allows PCI to modify the channel command word before it reaches the channel.

An additional WAIT and seek are required each time a buffer is not available. A seek required each time an error occurs or a cylinder is crossed. If the speed of the central processing unit does not allow PCI to perform its function in time, the number of seeks needed by the standard fetch are required.

Reenterable Load Modules Made Resident

Reenterable load modules from the SYS1.LINKLIB and SYS1.SVCLIB can be made resident. MVT or M65MP systems can have modules from either or both libraries made resident in the link pack area.

There are standard lists that are used during IPL time to place the load modules from the libraries into the fixed portion of main storage; IEAIGG00 for SYS1.LINKLIB and IEARSV00 for SYS1.SVCLIB. If the user desires to create his own list, then the operator communication option (OPTIONS=COMM) must be specified in the SUPRVSOR macro instruction. This will cause the message (IEA101A) to print out "SPECIFY SYSTEM PARAMETERS". Then the operator will provide the unique identification for the list. The reenterable load modules pointed to by the list will be loaded into main storage at IPL time.

Remote Job Entry (RJE) Facility

The remote job entry (RJE) facility provides a method of entering jobs from remote work stations into the job stream. Once the jobs have been entered execution proceeds under the supervision of the operating system. Any output data sets created by a remotely submitted job that the user wants returned are placed in a separate output class and then sent to the remote user.

The RJE facility operates on a computer-based telecommunications system that has the System/360 Operating System and requires the basic telecommunications access method (BTAM) routines. RJE is specified at system generation time in order to have the necessary modules included in the system.

After generation the user must create the specific RJE system required for his installation. There are four macro instructions available for this job -- RJETERM, RJELINE, RJEUSER, and RJETABL. The user sets up a job that includes the RJE macro instruction necessary to specify his system and may include user-written routines. The assembler translates these macro instructions and creates the required modules. The linkage editor incorporates the modules into the operating system. SYS1.MACLIB must be in the operating system so that the assembler can expand the macro instructions. SYS1.TELCMLIB must be present in the operating system also to hold some of the RJE load modules as well as the telecommunication subroutines.

Enough system write-to-operator (WTO) buffers must be specified in the WTOBFRS parameter of the SCHEDULR macro instruction during system generation so that an RJE task

will not have to wait to display a message. If a wait occurs, a work station time-out could result. A recommended value for the number of buffers is twice the number of telecommunication lines in the system.

Rollout/Rollin Option

A job can temporiarily expand its specified region. A job step's region size can be based on a minimum actual requirement, rather than a maximum.

When a job step needs more main storage, an attempt is made to obtain unassigned storage; if none is available, another job step is rolled out -- that is, its entire region is transferred to secondary storage -- and its storage is made available to the first job step. When released by the first job stop, the additional storage is again available as unassigned storage, if that was its source, or to receive the rolled-out job step, which is transferred back into main storage (rolled-in). Through job control you specify jobs eligible to be rolled out or to cause rollout. Exits are provided at key decision points where installation-written routines can be added to expand, redirect, or limit the feature's operation.

The data set SYS1.ROLLOUT must be cataloged in the new system before IPL.

The Shared Direct-Access Device Option

The Shared DASD option allows computing systems to share direct access storage devices. Systems can share common data and consolidate data when necessary; no change to existing records, data sets, or volumes is necessary to use the facility. However, reorganization of volumes is may be be desirable to achieve better performance.

The following control units and devices are supported by the Shared DASD option:

1. IBM 2841 Storage Control Unit equipped with two-channel switch -- IBM 2311 Disk Storage Drive, 2303 Drum Storage, and 2321 Data Cell.
2. IBM 2314 Direct Access Storage Facility equipped with the two-channel switch -- IBM 2314 disk Storage Module.
3. IBM 2314 Direct Access Storage Facility combined with the IBM 2844 Auxilliary Storage Control -- IBM Disk Storage Module. Device reservation and release are supported by this combination with or without the presence of the two-channel switch. Two channels -- one from System A and one from System B -- may be connected to the combination. In addition, the two-channel switch may be installed in either or both of the control units, thus permitting as many as four systems to share the devices.
4. IBM 2820 Control Unit with two-channel switch -- IBM 2301 Drum Storage.
5. IBM 2835 Storage Control Unit with two-channel switch -- IBM 2305 Fixed Head Storage Facility.
6. IBM 3830 Storage Control Unit with two-channel switch -- IBM 3330 Disk Storage Drive.

Alternate channels to a device from any one system may only be specified for the IBM 2314 Direct Access Storage Facility, or the IBM 3330 Storage Unit.

The Shared DASD option requires that certain combinations of volume characteristics and device status be in effect for shared volumes of devices. One of the following combinations must be in effect for a volume of device:

System A	Systems A, B, C
1. Permanently resident	Permanently resident
2. Reserved	Reserved
3. Removable	Offline
4. Offline	Removable or reserved

If a volume/device is marked removable on any one system, the device must be in offline status on all other systems. The mount characteristic of a volume and/or device status may be change on one system as long as the resulting combination is valid for other systems sharing the device. No other combination of volume characteristics and device status is supported or detected if present.

The RESERVE macro instruction is issued by a task to reserve a device for use by a particular system. The RESERVE macro instruction protects the issuing task from interference by other tasks in the system. Each task issuing the RESERVE macro instructions must also use the DEQ macro instruction to release the device.

The RESERVE instructions for the same resource without an intervening DEQ will result in an abnormal termination unless the second one specifies the keyword parameter RET=. Termination routines in all operating system confiruartions will release devices reserved by a terminating task.

Operating system configurations do not have to be identical to share a data set. The only additional equipment needed for the Shared DASD option is either a two-channel switch or a 2844 Auxilliary Control unit. The user must also observe certain restrictions about the data sets that are shared. The following data sets cannot be shared:

SYS1.SVCLIB	SYS1.SYSJOBQE
SYS1.NUCLEUS	PASSWORD data set
SYS1.LOGREC	SYSCTLG (on system residence volume)
SYS1.SYSVLOGX	SYS1.ROLLOUT
SYS1.SYSVLOGY	SYS1.ACCT
SYS1.MANX	SYS1.MANY
SYS1.DUMP	

SYS1.LINKLIB (can only be shared when the 2 systems are same type)

Volume handling on the Shared DASD option must be clearly defined since operator actions on the sharing system must be performed in parallel. You should make sure that following rules are in effect when using the Shared DASD option:

1. Operators should initiate all shared volume mounting and dismounting operations. The system will dynamically allocate devices unless they are in reserved or permanently resident status and Only the former can be changed by the operator.

2. Mounting and dismounting operations must be done in parallel on all sharing systems. A VARY OFFLINE must be effected on all systems before a device may be dismounted.

3. Valid combinations of volume mount characteristics and device status for all sharing systems must be maintained. To IPL a system, a valid combination must be established before device allocation can proceed. This valid combination is established either by:

   a. Specifying mount characteristics of shared devices in PRESRES.
   b. Varying all sharable devices off line prior to issuing START commands and then following parallel mount procedures described in the chapter "How to use the Shared DASD Option" in the **Operator's Guide**.

Note: The Set-Must-Complete (SMC) parameter available with the ENQ macro instruction may also be used with RESERVE.

Note: If a restart occurs when a RESERVE is in effect for devices, the system will not restore the RESERVE; the user's program must reissue the RESERVE.

System Management Facilities (SMF)

The System Management Facilities (SMF) are a group of routines that collect and record data about how the system and the I/O devices were used by the jobs and the job steps. For the M65MP systems these routines collect and record data about the use of CPUs, channels, and storage, as well as I/O devices. The data that is collected by the SMF routines is put on one or two data sets (SYS1.MANX and SYS1.MANY) -- one if magnetic tape is used, or two if direct access devices are used. Six exits are provided so that the user can supply his own exit routines to supplement the SMF option. The data collected by the user can be recorded on his own or the SMF data sets.

In order to use SMF, the user must specify the ACCTRTN parameter in the SCHEDULR macro instruction and the TIMER parameter in the SUPRVSOR macro instruction at system generation time. A definition list (SMFDEFLT) should be placed in the parameter library (SYS1.PARMLIB) before the first IPL. (This list can be put in either before or after system generation.) The definitions in the list provide the factors that determine which functions SMF will perform and whether any of the six exits (IEFUJV, IEFUJI, IEFUSI, IEFACTRT, IEFUTL, IEFUSO) are going to be used. If the user has written one or more routines to supplement SMF, they may be placed in SYS1.CI505 before system generation is started.

The SMF macro instruction (SMFWTM) and the SMF dump routine (IFASMFDP) are included automatically at system generation time as part of the SMF routines. The macro instruction is used to write the user's data records onto the SMF from the SMF I/O buffer. The dump routine should be used, if the data sets are on direct access devices, to dump the contents to magnetic tape. A sample program (TESTEXIT) to test the SMF routines and any user written routines is provided in the sample library (SYS1.SAMPLIB) of the starter operating system.

Telecommunications Access Method (BTAM, QTAM, and TCAM Optional)

The telecommunications access method can be included in the newsystem in the new system so that tasks can use the basic telecommunications access method (BTAM) or the queued telecommunications access method (QTAM). BTAM may be used with any control program; QTAM can be used with MVT or M65MP systems. If CIRB (SVC 2B) is desired in the new system, BDAM, ISAM, or BTAM must be specified.

The Time Sharing Option (TSO)

The IBM System/360 Operating System Time Sharing Option (TSO) adds general purpose time sharing to the facilities already available through the MVT configuration of the control program. As a result, the system provides a number of new capabilities:

- It gives users access to the system through a command language which is entered at remote terminals -- typewriter-like keyboard-printer or keyboard-screen devices connected through telephone or other communication lines to the computer.
- It gives those who may not be programmers the use of data entry, editing, and retrieval facilities.

- It makes the facilities of the operating system available to programmers at remote terminals to develop, test, and execute programs conveniently, without the job turnaround delays typical of batch processing. Both terminal-oriented and batch programs can be developed at terminals.

- It allows the management of an installation to dynamically control the use of the system's resources from a terminal.

- It creates a time-sharing environment for terminal-oriented applications. Some applications, such as problem-solving languages, terminal-oriented compilers, and text-editing facilities, are available as IBM Program Products. Installations can add others suited to their particular needs.

As far as the user is concerned, the distinctive feature of a time-sharing system is the way in which it "converses" or interacts on a step-by-step basis with him as he does his work. He is prompted for information the system needs to execute his job, he receives responses to his requests for action, and he is notified of errors the system detects, so that he can take corrective action.

A major consideration in the design of TSO is ease of use. The way in which a user communicates with the system is simplified to encourage people who may not be programmers to take advantage of the speed and versatility of a computing system to solve their problems. There are four ways in which TSO achieves this goal:

- The physical medium is familiar and easy to use.
- A terminal user defines his work in a language that is uncomplicated and natural to him.
- If a user does not know how to define his work to the system, he can type HELP and receive information pertinent to the type of operation he is trying to perform.
- The system keeps the terminal user aware of what is happening, so he knows what to do next.

For the data processing center, TSO is compatible with operating system standard formats and services, while providing the facilities needed for various time sharing and terminal-based applications.

A time sharing system reduces delays in receiving results. A larger number of jobs share resources of the system concurrently, and the execution of each is controlled primarily by a remote terminal user. Thus **time sharing** can be defined as the shared, conversational, and concurrent use of a computing system by a number of users at remote terminals.

The system resources shared by the time sharing jobs (foreground jobs) entered from the terminal are also shared by batch jobs (background jobs) that are being processed at the same time.

Because time sharing is carried out within the framework of MVT job and task management, the foreground and background environments are compatible. TSO uses the same data formats, programming conventions, and access methods as the rest of the operating system. The programming languages and service programs available with TSO are compatible with their background counterparts.

Certain facilities are unavailable to foreground jobs, although they remain available to background jobs. These include:

- The basic telecommunications access method (BTAM).
- The graphic access method (GAM).
- The EXCP equivalents of the BTAM, QTAM, and GAM access methods.

- Main storage requests for hierarchy 1 (all foreground requests for main storage are allocated in hierarchy 0).
- Use of job control language in the foreground for other than single-step jobs (the TSO command language is used to provide the equivalent of multi-step jobs).
- Checkpoint/Restart facilities (foreground requests for checkpoint are ignored).
- Rollout/Rollin option.

The TSO command language is also generally compatible with the Conversational Remote Job Entry (CRJE) command language. Programs can be developed in the foreground and stored in background libraries. These programs are compatible with other operating system programs. Most problem programs can be executed in either the background or the foreground without revision or recompilation.

TSO is not necessarily intended to be used as a dedicated time-sharing system, that is, a system on which only time-sharing operations take place. TSO augments the facilities already available with the operating system: batch processing, teleprocessing, and other data processing activities can take place concurrently on the same system.

TSO is an extension of the MVT configuration of the control program on System/360 Models 50 through 195, or System/370 Models 145, 155, and 165. The minimum machine configuration for System/360 models must include 384K of main storage, the required I/O devices for MVT, plus at least one each of the following:

- A terminal (IBM 1050, 2260, 2741, or AT & T Teletype (trademark of Teletype Corporation, Skokie, Illinois) Model 33 or 35 KSR).
- A transmission control unit (IBM 2701, 2702, or 2703).
- Sufficient direct access storage space (IBM 2301, 2303, 2305, 2314, 2319, or 3330) for swap data sets, command libraries, and system data sets.

In a System/360 with 384K of main storage, TSO is a "dedicated" time sharing system. In other words, with 384K the system can run as a time sharing system or as a batch job processing system, but not both at the same time. To run both time sharing and batch jobs concurrently or to execute on System/370 models, at least 512K of main storage is required. (At least 128K of main storage is required for system generation.)

The TSO control program, which is an extension of the MVT control program, consists of many routines, each of which performs functions to support time-sharing operations. These routines have been grouped, by function, into seven basic TSO control program components:

- Time Sharing Control Task (TSC).
- Terminal Input/Output Coordinator (TIOC).
- Region Control Task (RCT).
- Logon/Logoff Scheduler.
- Time Sharing Dispatcher.
- Time Sharing Driver (TSD).
- Time Sharing Interface Program (TSIP).

Detailed information on the concepts, features, and capabilities of TSO is provided in the **Time Sharing Option Guide**; detailed information on the control program is provided in the **Time Sharing Option (TSO) Control Program PLM**.

The Time Slicing Facility

The user can establish a group of tasks (called the time-slice group) that share the use of the CPU, each for the same, fixed interval of time. All member tasks are given an equal slice of CPU time, and no task within the group can monopolize the CPU.

The time slicing option is included in the system to provide a method of controlling response time of a task. However, since it is being implemented in a priority dispatcher, any task of a higher priority than that of the time-slice group will be dispatched first, if it is ready. Time-slicing applies only to the problem program priorities, 0-13. Priorities 14 and 15 are reserved for the system and cannot be time sliced. Therefore, the response time of a time-slice task can be affected by the processing of system tasks, such as readers, writers, master scheduler, etc., which will always run at a higher priority than the time-slice group. To guarentee response time, the time-slice group should be defined in the high dispatching priority.

Time-slicing operates within the structure of the current dispather. A priority is assigned to a group of tasks that are to be time sliced. The time slicing occurs among the tasks in the group only when the priority level of the group is the highest priority level that has a ready task. Each task in the group is dispatched for the specified time slice. The time slicing continues until all tasks are waiting, or a task of higher priority than that of the group becomes ready.

The dispatcher will recognize that a priority level being time sliced; it will determine which task within the group is to be dispatched and then dispatch that task for the maximun time interval. If the time slice task loses control prior to the expiration of its interval (because an implicit or explicit wait is issued, or because a higher priority task becomes ready), the remainder of the time is not saved. That is, when control returns to the time-slice group, the next ready task in the group is given control, not the interrupted task.

The time slicing facility is especially useful in a graphics environment or in any application of a conversational nature where concurrent tasks may involve conversation between the user and the problem program through a terminal. Establishing a time-slice group within this environment enables those tasks to be performed with a uniform response time.

The group of tasks to be time-sliced and the lenght of the time slice are specified by the installation at system generation time. This can be modified at system initialization time with the TMSL parameter in response to the system message "SPECIFY SYSTEM PARAMETERS". The modifications are limited by the number of groups specified during system generation. Any task in the system that is not defined within the time-slice group is dispatched under the current priority structure; that is, the task is dispatched only when it is the highest priority ready task on the TCB queue.

A single time-slice group can be defined by associating the time-slice group with job dispatching priorities, either in the system generation statements or in an operator reply at IPL time. However, if a step dispatching priority is stated in the EXEC statement of a step (the DPRTY=entry), then the value of that priority determines whether of not the step is a member of the time-slice group.

Time slicing is invoked in the PRTY parameter of the JOB statement. New tasks can become associated with the time-slice group by using the ATTACH or the CHAP macro instructions. Tasks created with the ATTACH macro instruction have the same characteristics for time-slicing as the tasks which created them. Tasks can be changed into or out of time-slicing groups with the CHAP macro instruction.

Note: Where job priorities differ by 1, dispatching priorities differ by 16. Jobs changes by the CHAP macro instruction from one priority time-slicing group to another group must have thier dispatching priority changed by 16, and not 1. The publication **Supervisor Services and Macro**

Instructions describes task priorites and the formulas used to derive the dispatching priority from the job priority.

Timing Options

The INTERVAL Option

The JOBSTEP Option

INTERVAL and JOBSTEP Required

These options may be selected when an interval timer is included in the central processing unit. There are two levels of interval timer support that may be specified:

- **Internal Timing** (INTERVAL) provides the ability to request, check, and cancel time intervals with the STIMER and TTIMER macro instructions, plus the ability to change the time at midnight. This level of support also includes the facilities provided by the TIME macro instructions.
- **Job Step Timing** (JOBSTEP) provides the ability to time each job step and enforce the time limits. This level of support also includes the facilities provided by the TIME, STIMER, and TTIMER macro instruction. (See "Job Step Timing Option" in this section.)

If no timing options are specified, then just the time of day is available.

If SMF(System Management Facilities) is to be included, TIMER=JOBSTEP must be specified in the SUPRVSOR macro instruction.

For MVT or M65MP systems INTERVAL or JOBSTEP must be specified. The storage required is included in the basic fixed requirement.

Trace Option

A tracing routine that aids in debugging and maintenance can be added to the system. The tracing routine stores information pertaining to start I/O (SIO) instruction execution, supervisor (SVC) interruptions, external interruptions, program check interruptions, and I/O interruptions in the trace table. When the table has been completely filled, the next new entry in the table will overlay the first entry, the next one overlays the second entry, etc.

During system generation, only the size of the table is specified. However, when this system generation parameter is specified, the trace program routines are also included as part of the control program.

Type 3 and 4 SVC Routines Made Resident

Modules of type 3 and 4 supervisor (SVC) routines can be made permanently resident in the fixed area of storage.

Type 3 and 4 SVC modules are loaded and made resident at IPL time. When this option is specified, the transient SVC table option must also be specified. The SVC table is a table containing the relative track address of all transient SVCs. This table is also stored in the resident portion of the control program.

The names and sizes of the type 3 and 4 SVC routine modules are given in the appendix of **Storage Estimates**. (See also the preceding description "Transient SVC Table Made Resident".)

During a nucleus generation this option can be added or deleted from the options specified during a complete system generation. But the transient SVC table option will have to be specified the same way it was specified in the last complete generation.

User-Added SVC Routines

User-written supervisor (SVC) routines can be added to the control program.

All of the SVC routines, whether they are to be transient or resident, must be listed in the operand of the SVCTABLE system generation macro instruction.

Any resident SVC routines that are to be added must be specified in the system generation RESMODS macro instruction. The fixed storage requirement is increased by the total of the sizes of the routines that are going to be added plus the size of the control information.

Any transient SVC routines that are to be added must be specified in the SVCLIB system generation macro instruction in the operand. In this case, only the size of the control information is added to the fixed storage requirements.

Non-standard error routines can be one of the types of routines that are added. User-written routines must have a value from 220 to 229. This value is the suffix of the name IGE00 by which the error routine is contained in SYS1.SVCLIB.

Volume Statistics Facility

The volume statistics facility is used only for magnetic tape volumes with or without labels and provides two functions. Either one or both functions can be specified at system generation time in the SCHEDULR macro instruction. One function is error statistics by volume (ESV) and is intended primarily to be used with labeled volumes. It will handle unlabeled volumes if the serial number is given to the operating system. Statistics about the number of read or write errors and the system and unit on which it is located are recorded.

The other function is error volume analysis (EVA) and is intended primarily to be used for unlabeled or non-standard labeled volumes. It monitors the number of read or write errors based on the limits the user provides at system generation time.

The error statistics by volume (ESV) routines collect a set of statistics for each labeled tape volume during any interval that the volume is open. An unlabeled tape volume can be handled if the serial number has been supplied to the operating system.

If ESV=SMF is specified at system generation time, the statisitcs are accummulated on the system management facility (SMF) data sets, SYS1.MANX or SYS1.MANY. ACCTRTN=SMF should be specified in the SCHEDULR macro instruction, but if it is not coded it is assumed. If any subparameter for ACCTRTN other than SMF is specified, it is ignored and SMF is assumed. The TIMER keyword parameter is also required in the SUPRVSOR macro instruction. The IFHSTATR utility program is used to print the ESV records, record 21, from an SMF data set that is on magnetic tape. If SYS1.MANX is on tape, no transfer is required. But if the SMF data sets are on a direct access device, the user must dump them onto tape in order to be able to extract the ESV records. The SMF dump program, IFASMFDP, is used to transfer the data from SYS1.MANX and SYS1.MANY to tape.

If ESV=CON is specified or if ESV is not coded, an abridged version of the statistics is printed on the console. This occurs at end-of-volume or when the tape is closed.

The user can provide his own recording routine. ESV=CON must be specified or the keyword parameter can be omitted since the default is CON. The UCBs, in the proper format, will be constructed at system generation time. He can provide his own method, using SVC 91, specify his own record format, and select his own recording data set. If he uses the SMF record 21 format instead of his own, he can use the IFHSTATR utility to print the statistics.

The error volume analysis (EVA) routines monitor the number of read and write errors for unlabeled or non-standard labeled tape volumes. The user provides the maximum limits for read errors and/or write errors and, if the maximum is reached or exceeded, a message, IEA620I, is printed on the console.

Task Directory for Section II: MVT Options

For information about:

- Increasing system response, see
  Additional Pairs of Transient Areas
  BLDL Table Made Resident
  Main Storage Hierarchy Support
  Reenterable Load Modules Made Resident
  The Time Slicing Facility
  Type 3 and 4 SVC Routines Made Resident
  User-Added SVC Routines

- Recovering from machine malfunctions, see
  Alternate Path Retry
  Channel-Check Handler
  Checkpoint/Restart Facility
  Direct Access Volume Serial Number Verification
  Dynamic Device Reconfiguration
  Volume Management Facilities

- Increasing user convenience, see
  Consoles -- Alternate and Composite Console Option
  Consoles -- Multiple Console Support
  Conversational Remote Job Entry
  Decimal Simulation Option
  Graphic Program Services
  Remote Job Entry (RJE) Facility
  Rollout/Rollin Option
  The Shared Direct-Access Device Option
  The Time Sharing Option

- Checking system activities, see
  Job Step Timing
  System Management Facilities (SMF)
  Trace Option
  Timing Options

- Using access methods, see
  Indexed Sequential Access Method (ISAM)
  Telecommunications Options

Section III: Planning For MVT

This section contains information that will help you optimize MVT performance; when planning a new MVT configuration, be familiar with most of the section.

The sections on job classes, output classes, and priorities are planning aids; much of the performance of an MVT configuration depends on how well users plan these.

MVT Requirements

The MVT configuration of the control program is designed for use with the IBM System/360 Models 40, 50, 65, 75, 85, 91, 95, and 195, the System/370 Models 145, 155, 165, and 195, or the Model 65 Multiprocessing System (M65MP).

Configuration Requirements

The minimum hardware configuration requirements are:

- 256K bytes of main storage (512K with Model 65 Multiprocessing).
- Two direct-access storage devices (not including 2302 Storage Unit).
- One IBM 1052 Printer-Keyboard, for Models 40, 50, 65, 75, 85, 91, or 195. On Models 50, 65, 75, 85, 91, and 195 a 2250 Display Unit Model 1, a 2250 Display Unit Model 3, or a 2260 Display Station may be substituted for the 1052 Printer-Keyboard. The Model 85 may also have its own console, the IBM 5450.
- One IBM 3210 or 3215 Printer-Keyboard, for Model 145 or 155.
- One IBM 3066 System Console, for Model 165.
- One card reader or tape device.
- One card punch or tape device.
- One printer or tape device.

Storage Requirements

The **Storage Estimates** publication contains the actual figures needed to calculate the storage requirements of the installation.

System Generation Requirements

Generating an MVT control program is the same process essentially as generating MFT; you may use an existing MVT configuration as the generating system or the IBM-supplied starter system. When you plan your system generation you should review carefully the section "Preparation for System Generation" in the **System Generation** publication and the section "Storage Requirements."

The CTRLPROG and SCHEDULR sysgen macro instructions are used at system generation to specify an MVT control program. The following tables show the parameters that may be specified with these macros.

CTRLPROG Macro Instruction

MACRO INSTRUCTION	PARAMETER	REQUIRED	SPECIFIES
CTRLPROG	TYPE	Yes	Names type of control program
	MAXIO	Yes	Number of I/O operations that can be processed at one time
	QSPACE	No	Size of the system queue area in 2K blocks (value of 10 is default)
	ADDTRAN	No	Number of additional pairs of transient areas
	OPTIONS	No	Rollout/rollin
	TMSLICE	No	Time slicing for all jobs at certain priority or priorities
	HIARCHY	No	Storage hierarchies for the system
	OVERLAY	No	Asynchronous overlay supervisor (assumed for MVT)
	FETCH	No	PCI fetch (assumed for MVT)

The following example shows how the CTRLPROG macro instruction might be used to specify MVT.

```
CTRLPROG      TYPE=MVT,MAXIO=30,QSPACE=15,ADDTRAN=2,TMSLICE=(10,
              SLC-512,7,SLC-256)
```

MAXIO=30
The maximum number of I/O operations that can be handled simultaneously is 30.

QSPACE=15
Fifteen 2K blocks are specified for the system area.

ADDTRAN=2
There will be a total of six transient areas: the original pair plus two more pairs.

TMSLICE=(10,SLC-512,7,SLC-256)
All tasks with a priority of 10 can have control of the CPU for a maximum of 512 milliseconds at a time; those with a priority of 7 can have control for 256 milliseconds at a time.

The rollout/rollin function is not specified; the asynchronous overlay supervisor and PCI fetch are assumed.

SCHEDULR Macro Instruction

MACRO INSTRUCTION	PARAMETER	REQUIRED	SPECIFIES
SCHEDULR	TYPE	Yes	Type of control program
	CONSOLE	Yes	Address of primary console
	ALTCONS	No	Address of alternate console
	OPTIONS	No	System log, and inclusion of remote job entry and conversational remote job entry
	STARTR	No	Automatic START reader command after IPL
	STARTW	No	Automatic START writer command after IPL
	ACCTRTN	No	User-supplied accounting routine or system management facility
	VLMOUNT	No	Automatic volume recognition
	TAVR	No	Standard density for magnetic tapes used with automatic volume recognition
	STARTI	No	Automatic START initiator command after IPL
	WTOBFRS	No	Number of buffers to be used by WTO routines
	REPLY	No	Number of reply queue elements to be used by WTO routines
	PROCRES	No	Address of the device on which SYS1.PROCLIB resides
	JOBQRES	No	Address of the device on which SYS1.SYSJOBQE resides
	JOBQFMT	No	Size of each logical track for SYS1.SYSJOBQE
	JOBQLMT	No	Number of 176-byte records in SYS1.SYSJOBQE to be reserved for each initiator started
	JOBQTMT	No	Number of 176-byte records in SYS1.SYSJOBQE to be reserved for the termination of jobs that require more records for initiator than specified in JOBQLMT
	JOBQWTP	No	Number of records in SYS1.SYSJOBQE to be reserve for Write-to-Programmer messages for a job
	WTLCLSS	No	Default classname for SYSOUT for Write-to-log messages.
	WTLBFRS	No	Size of buffer area used as temporary storage for WTL messages
	INITQBF	No	Size of initiator queue buffer
	MINPART	No	Minimum region size to initiate a job
	CONOPTS	No	Multiple console support
	ROUTCDE	No	Routing codes master console is authorized to receive
	OLDWTOR	No	Routine codes for WTO and WTOR messages with no routing codes assigned
	HARDCPY	No	Hard copy log
	EVA	No	Use of and threshold values for error volume analysis facility
	ESV	No	Destination of volume error statistics records

```
SCHEDULR TYPE=MVT,CONSOLE=01A,ALTCONS=(I-00C,O-00D),STARTR=A-00E,
               STARTW=A,00F,WTOBFRS=75,,REPLY=75,STARTI=AUTO,
               WTLBFRS=20,JOBQFMT=20,JOBQLMT=80,JOBQTMT=80,
               INITQBF=15,MINPART=67
```

The following example illustrates the use of the SCHEDULR macro instruction to specify the MVT job scheduler.

CONSOLE=01A
Specifies that the address of the primary console is 01A.

ALTCONS=(I-00C,O-00D)
A composite console is used as the alternate console; its input address is 00C and its output address is 00D.

STARTR=A-00E
The START READER command will be executed automatically after IPL on the device at address 00E.

STARTW=A-00F
The START WRITER command will be executed automatically after IPL on the device at address 00F.

WTOBFRS=75
The maximum number of buffers used by the WTO routines is 75.

REPLY=75
The maximum number of reply queue elements used by the WTOR routines is 75.

STARTI=AUTO
The START INITIATOR command will be issued automatically after IPL.

WTLBFRS=20
The maximum number of buffers for the WTL messages is 20.

JOBQFMT=20
There will be 20 176-byte records for each logical track in the job queue.

JOBQLMT=80
Eighty 176-byte recores will be reserved for each initiator started.

JOBQTMT=80
Eighty 176-byte records will be reserved for the termination of jobs that require more than 80 records for initiation.

INITQBF=15
The initiator will use a 15K area to keep the most frequently used job queue records in its main storage. (The MINPART specification that follows must include provision for this area.)

MINPART=67
Sixty-seven 1024-byte blocks are required to process a job (15K is for the job queue buffers and 52K is for the initiator itself).

The classname for WTL messages is L (the default). SYS1.SYSJOBQE and SYS1.PROCLIB are located on the system residence device (default).

Planning Aids

A key to optimizing performance in MVT is avoiding wasteful contention for resources; jobs that require the same resources (main storage, I/O devices, data sets) should not be run concurrently, or one job will wait in main storage without executing until the other job frees the required resource.

Job Classes

In MVT you can control not only the priority of jobs but the actual mix of jobs in the computer; this is done by assigning jobs to job classes. You specify the job class on the JOB statement (CLASS=jobclass), and when an initiator that can handle that particular class is started, the jobs will be processed in priority order.

There are no absolute rules for assigning job classes and some experimentation is necessary. Generally, jobs of similar characteristics should be assigned to the same class; for example, if you have several jobs that require large blocks of storage, you would not want to tie up all of main storage by having these jobs running concurrently. You might assign them all to class B (or C or D -- whatever you choose -- class names have no inherent meaning); then if you start only **one** initiator that can handle class B jobs, you will never have more than one of these large jobs in main storage at once. This way you may leave sections of storage free to process other jobs; for example, at the same time you might have a small, short-running job and an I/O bound job in main storage. You might assign jobs that require less than one minute to class C; jobs with high I/O might be assigned to class D.

Class B = jobs with large main storage requirements.

Class C = jobs of less than one minute running time.

Class D = jobs with high I/O requirements.

With these assignments the operator might issue these commands:

START INIT,,,BCD

START INIT,,,CDB

START INIT,,,DCB

If the three initiators are processing jobs with the same priority and all necessary resources (I/O devices, data sets, etc.) are available, then three jobs, one from each of the three different classes, would run concurrently. If a job from one of the classes has higher priority than the others, it will be initiated first, assuming all necessary resources are available.

When the operator starts an initiator, he indicates what class or classes it can handle (START INIT,,,A); an initiator can handle up to eight job classes, and as many as fifteen initiators can be running at once (this means that you can have a maximum of 15 user-program regions operating concurrently in addition to system tasks). You might have initiators dedicated to I/O bound jobs, CPU jobs, long running jobs, etc. You may wish to simplify the operator's job by having mnemonically named catalog procedures for these initiators as explained in Section IV.

If your system uses automatic volume recognition (AVR), you can improve efficiency by assigning jobs that require non-resident volumes to the same class. With AVR, if volumes required by a job are not mounted at allocation time (and the DEFER subparameter is not specified), no other jobs can be initiated until the required volumes are mounted by the operator. Thus, if you assign jobs that will require non-resident volumes to the same class, only one initiator will be issuing mount messages and it will be easier for operators to anticipate which volumes to mount. Also, you can effect performance improvements by assigning these jobs the same priority (see the topic "Job Priorities"). If they run at the same priority, they will probably be initiated on a first-in-first-out basis, and the operator can anticipate which volumes to mount next.

The job class is specified as a parameter on the JOB statement. Its format is:

```
//jobname  JOB  CLASS=jobclass
```

Jobclass may be any single letter from A-O. If no job class is specified, the default will be class A. (You can ignore job class altogether by not using the CLASS parameter, and all jobs will run as class A jobs.)

Always try to have an initiator running that can handle all job classes in the system. If a job is submitted and there is no initiator running for the job class, it will remain in the input queues indefinitely or until the operator checks the input queues and discovers the job (by means of the "DISPLAY N" command).

Job Priority

While job classes control which jobs are to be initiated, job priorities control the order of the jobs in each class. Priorities can range from 0-13 (13 being highest). You assign the priority as a parameter on the JOB statement as follows:

```
//jobname  JOB  PRTY=priority
```

where priority can be any decimal integer from 0-13. If no priority is specified, the default priority is assumed; you set the default priority in your reader catalog procedure (see the topic "Reader Procedures" in Section IV). There are no absolute rules for assigning job priorities; this will depend on the job mixes that you run and which jobs require fastest turnaround.

The actual value used by the supervisor to determine which task receives control of the CPU next is dispatching priority. It may be calculated from the job priority stated on the JOB statement, or you may state dispatching priority explicitly on the EXEC statement using the DPRTY parameter. (See the **Job Control Language Reference** publication for details on how to use this parameter.) Dispatching priorities should be assigned with the assumption that tasks of higher priority will be given control when competing with tasks of lower priority. Tasks with a large number of input/output operations should be assigned higher dispatching priorities than tasks with little input/output since the higher priority I/O tasks will be in a wait state more often and during those times lower priority tasks can have control of the CPU. Also, if a subtask (a task attached by another task) must complete before the originating task can complete, the subtask should be assigned a priority which will eliminate as nearly as possible a long wait time for the originating task.

SYSOUT Classes

Output from problem programs is assigned to an output class which will be processed by output writers. You may use a maximum of 36 SYSOUT classes named with single letters (A-Z) or digits (0-9) for output class names. The names have no inherent meaning but are simply used to group output of similar characteristics. Writers are assigned to process only certain classes of output; you may assign these classes in your output writer cataloged procedure, or the operator may assign them as parameters in the START command. (See the section "Writer Procedures" for information on controlling the characteristics of output writers.)

Careful planning of output classes can improve throughput at an installation. By assigning jobs to carefully planned output classes and making sure that operators know which writers to run at all times, you can make sure that all output devices are used constantly and that there is no wasteful contention for these devices. It is essential that if output is assigned to a class for which no writer is started, it will remain indefinitely in the output queue occupying direct access space that might be needed by another program.

In planning your SYSOUT classes you should consider the following:

1. A special output class might be reserved for very high priority jobs. For instance, only jobs requiring turnaround time of less than an hour might be assigned a class of A.

2. If your system has a universal character set printer that will be used as an output device, you might assign a separate output class to each character set image stored in the system library. This will minimize the changing of printer chains and trains.

3. If your system has a printer with a forms control buffer (FCB), you might assign a separate output class to each FCB image. This will minimize the changing of printer forms.

4. Although a writer may handle as many as eight different output classes, it can only write output on one device. For example:

   START WTR,00E,,ABCDEF

   The writer started would handle six different classes, but all these classes would be written on the printer at the specified address 00E. If one of the classes is to be written on tape, it could not be handled by this writer. Direct system output can handle only one output class per device.

5. The standard IBM-supplied writer can write output to the following: 1403, 1442, 1443, 2400, 2400-1, 2400-2, 2400-3, 2400-4, 2520, 2540, 3211, 3400-2, 3400-3, or 3400-4.

You specify the output class as a parameter on the DD statement defining your output data set; its format is SYSOUT=x where x is any single letter (A-Z) or digit (0-9); however, you should avoid using the digits whenever possible as they are reserved for future system use.

System messages that are generated during the execution of a program must also be routed to an output device; since you will want messages to appear with their program output, you should assign messages to the same message class as the output.

The message class is assigned as a parameter of the job statement; its format is

MSGCLASS=x

where x is any single letter (A-Z) or digit (0-9).

If no message class is specified, the default class specified in the RDR procedure is used.

System Output Writers

A writer may handle as many as eight different output classes and will process the classes on a priority basis, always first handling the highest priority job of the first job class named in either the START command or in the cataloged procedure. The following commands might be used to start output writers:

START WTR,,,A
START WTR,,,B
START WTR,,,ABC

In this case you would be starting three writers. The first writer would handle only job class A, the second only job class B, and the third would handle job classes A, B, or C. The third writer will always select an output class A job if one is available and if not, a class B job, and if neither A nor B is available, then a class C job. After a writer selects a job class for processing, if there are jobs ready for writing in that particular class, it writes them in priority order. After writing of a particular job output begins, no other work can be processed by the writer until that output is finished. If no output is ready for writing, the writer will remain in storage until it is stopped by a command from the operator, or until output is available for processing. Detailed format requirements for the START command are given in the **Operator's Reference** publication.

Following are some examples of output classes:

Class	Destination	Writer Name
B	Printer 1 (online)	WTB
C	Printer 2 (online)	WTC
D	Printer 3 (offline)	WTD
E,F,G	Tape	WTEFG

In the preceding example, four writers are used: WTB will write class B output on a printer, WTC will write class C output on a printer, WTD will write class D output on tape (for eventual offline printing), and WTEFG will write classes E, F, or G output on tape. One or all of these writers might be operating concurrently in the system depending on what jobs are being processed and what I/O devices are available. If there is no work for a writer, one should not be started, since it will occupy space that another program might use.

Direct System Output Writers

Direct system output writers write problem program and system messages, produced by the initiator, directly to system output devices. Valid output devices are: printer, punch, and magnetic tape.

Direct system output writers are started by the operator. Selection of the various direct system output writers is made when the initiator first selects a job from the input queue. The selected writers for the job remain in effect for the duration of that job. They are used whenever a job step has a SYSOUT class equal to the OUTCLASS assigned to a selected direct system output writer. SYSOUT classes not qualifying for direct system output (DSO) will be spooled in the normal fashion.

When the problem program writes its output, the output will go directly to the output device assigned to the direct system output writer. System output writers can handle as many as eight different job classes; each of the job classes must be specified in the START command. For example, if the operator enters the command,

START DSO,283,,(JOBCLASS=ABC,OUTCLASS=B)

any job running with a jobclass of A, B, or C and an output class of B will have its ouput written directly to tape device 283, if the device was selected for the job.

The job class and output class of a direct system output writer can be changed with the MODIFY command. For example, if the operator enters the command,

MODIFY 283,JOBCLASS=DE,OUTCLASS=A

any direct system output writer writing to output device 283 will process jobs with a jobclass of D or E, and an output class of A.

Direct system output writers can be stopped by a STOP command. A STOP or MODIFY command for DSO will be queued, and will gain control when the respective device is available for selection. When the command gains control, all initiators are blocked from DSO selection, and will wait until the completed processing of that request.

A user-supplied DSO procedure may be used, but it must execute the IBM-supplied direct system output writer.

Region Size

You may specify through a job control language parameter the region size to be used by a job or job step. By providing careful guidelines to programmers for specifying region size, you can increase the number of jobs running in your system at one time (this, of course, requires that you have planned the job classes and initiators to run these jobs). You specify the region size to be used by a job (and all the job steps within that job) by using the REGION parameter on the JOB statement. If you use the REGION parameter on the EXECUTE statement, the size that you specify will be used only by the job step. If you specify a region size on both the JOB statement and the EXECUTE statement, the JOB statement size will override the EXECUTE specification. If you omit the REGION parameter from both, then the default value specified in the input reader cataloged procedure (see the section "Readers") will be used.

You specify the region size thus:

REGION=nnnnnK

nnnnn is the number of 1024-byte areas to be allocated to the job or job step.

In choosing region sizes for jobs, you should be careful not to make regions wastefully large; a region is allocated for the duration of a job or job step, and if there is unused storage within the region, it cannot be used by another job until termination.

The smallest region that you use in MVT is 52K, the minimum initiator region size. If you include the terminator modules of the initiator in your link pack area, however, these modules will only require work area within problem program regions. To improve system performance and reduce the dynamic storage required by the initiator, it is also recommended that you place the device name table and device mask table in the link pack area. You may then specify regions as small as 12K.

Inadequate region size specification for your job or job step results in an abnormal termination.

The **Job Control Language Reference** publication contains further detail about the use of the REGION parameter.

Additional Transient Areas

There is one pair of transient areas embedded in the nucleus. These areas are used for nonresident SVC routines as they are needed; they are loaded into the transient areas, used by the tasks that require them, and left in main storage until the transient area they occupy is needed for other nonresident routines. They can be overlaid if the transient area they occupy is needed for non-resident SVC routines called by a higher priority task. If a task calls a non-resident SVC routine and no transient area is available or none can be made available, the task is put in a wait state until an area is available.

You can increase the number of pairs of transient areas in the nucleus (each pair occupies 2,990 bytes) at system generation time if you need to reduce access time on direct access devices. The transient areas, in effect, extend the link pack area but give added flexibility, since routines are moved in and out of transient areas as necessary.

Avoiding Main Storage Fragmentation

The order in which operators start readers, writers, and initiators often affects the amount of main storage available. Tasks are assigned main storage beginning at the high end of the dynamic area and each task receives the highest available contiguous block of storage. If a contiguous block of storage is not available and none can be made available for a task , then the task must wait. Therefore, it is important that main storage not be broken into many small fragments of space too small for any task.

Long-running jobs should be started first so that they occupy the high end of storage; otherwise they might occupy large blocks of middle storage with small blocks of unused space existing at the high and low ends of storage.

Generally writers, which are long running tasks, should be started before readers and initiators, which run intermittently. If you are running graphics or telecommunications applications, however, these jobs should be started first since they are likely to be long-running, continuous jobs.

Placing System Libraries on Direct Access Devices

Several factors must be considered when putting system libraries (SVCLIB, MACLIB, LINKLIB, PROCLIB, PARMLIB, and SYSJOBQE) on direct access storage devices. If you put all six libraries on the same device, throughput will be decreased because of excessive arm interference. You should balance libraries on devices and balance devices on channels. Ideally, each library would be on a different device and each device on a different channel. In installations with smaller systems, you should put SYSJOBQE and LINKLIB on the same direct access device on channel 1, and SVCLIB, PROCLIB, PARMLIB, and MACLIB on another device on channel 2.

When you put more than one library on a 2311, 2314, 2319, or 3330 direct access device, you can reduce arm movement by placing the volume table of contents (VTOC) approximately midway between the first and last cylinder being used. The libraries, starting with the most frequently referenced, can then be alternately placed on both sides of the VTOC with the least frequently referenced libraries furthest from the VTOC.

System Restart

When it is necessary to shut down the system (end-of-shift, end-of-day, normal maintenance, or system malfunction), system restart allows the system to resume operation without your having to reenter jobs that have been enqueued. Information concerning jobs on the input, hold, and output queues, and jobs in interpretation, initiation, execution, or termination is preserved for use when the system is reloaded. When the system is restarted, the operator receives messages describing the status of each job in the system. If a job was being interpreted, its jobname is written out at the console. If a job was under control of an initiator, its jobname and stepname are given. In addition, the operator is informed of whether allocation was being performed for the job or whether the job was being executed or terminated.

When the system must be restarted, jobs that were being interpreted are run out and must be reentered in the input stream. All jobs that were enqueued on their appropriate job class, hold, or output class queues, remain there for subsequent processing when the system is restarted. No operator action is required.

Jobs that were dequeued from an input queue may under some circumstances be successfully completed; otherwise, the partial output of the job is routed to the output queue. The circumstances under which the job may be completed are:

- If the entire job (not job step) was being terminated, system restart completes the termination.
- Any job that was in step termination at the time of restart will be executed starting at the next step of the job after system restart is complete.
- Any job which can be restarted will be restarted (after completion of system restart) at the current step if the following conditions are met:

  a. The step had completed the allocation phase.
  b. The system restart abnormal termination code of FF3 is designated as a restartable code.
  c. The step is designated as restartable by the programmer.
  d. The operator replies yes to the verification request to restart the job.

Output jobs that were being processed by an output writer are reenqueued for reprocessing of data sets which had not been written completely at the time the system was shut down. No

operator action is required. All system messages and data sets that had not been processed are written by the first eligible output writer started.

Information on restarting the system is given in the **Operator's Procedures** publication.

Task Directory for Section III: Planning For MVT

For information about:

- Requirements for using MVT, see
  MVT Requirements
  Configuration Requirements

- Requirements for generating an MVT system, see
  CTRLPROG Macro Instruction
  SCHEDULR Macro Instruction

- Using output writers, see
  SYSOUT Classes
  System Output Writers
  Direct System Ouptut Writers

- Using job classes and priorities, see
  Job Classes
  Job Priorities

- Using main storage, see
  Storage Requirements
  Region Size
  Additional Transient Areas
  Avoiding Main Storage Fragmentation

- Using direct access storage devices, see
  Configuration Requirements
  Placing System Libraries on Direct Acess Storage Devices

Section IV: Modifying the System

This section describes how the user can modify an MVT system to suit the needs of his installation. The topics covered in this section are:

- Standard IBM cataloged procedures
- Using the link pack area
- Job queue format
- Output separation
- Writing system output writer routines
- Adding SVC routines to the control program
- Message routing exit routines
- Handling accounting routines
- Writing rollout/rollin installation appendages
- The must complete function
- The PRESRES volume characteristic list

Standard IBM Cataloged Procedures

After you plan your job classes and output classes and have some idea of what job mixes you will try to run concurrently, you can plan your catalog procedures for readers, writers, and initiators. You can simplify the operator's job and save time by having several procedures for each of these and naming them so that operators can tell what they do by their names. For instance, a reader that reads from tape with a blocking size of 3200 could be named RT32. An output writer that writes class A output to a printer could be WTA. An initiator cataloged procedure that processes class A FORTRAN jobs might be IFRTA. By thus naming procedures, when one of the system tasks stops or is waiting for work, the operator will know what task is involved by its name.

Generally, in your cataloged procedure you should be as specific as possible in providing parameters so the operator need only type the START command and the name of the procedure. If you wish to change parameters (for instance the blocksize in a reader procedure), you should use symbolic parameters. The operator can type in the changed parameter, and it will override the one in the procedure. The START command can also start problem programs, but SMF (System Management Facilities) will not be recorded, nor will Checkpoint/Restart be done for these jobs.

In choosing blocking factors for your readers and writers, you should refer to the topic "SYSIN and SYSOUT Data Blocking" to see what blocksizes are acceptable to certain processors and utilities. Where several blocksizes are available, you should let the amount of storage available determine the blocksize. In general, larger blocksizes mean more efficient processing; however, as blocksize increases, available storage for other jobs decreases, and the job mixes that you want may become a controlling factor. Region size specifications for jobs must reflect increases in blocksize.

Reader Procedures

IBM supplies three cataloged procedures for readers. You can use these procedures, modify them by overriding parameters, or you can write your own procedures. A cataloged procedure for readers requires four job control statements:

- An EXEC statement where the stepname IEFPROC specifies the reader.
- A DD statement named IEFRDER to provide the reader with a description of the input stream.
- A DD statement named IEFPDSI to describe the procedure library.
- A DD statement IEFDATA to define the spooling, or concurrent peripheral operation (CPO), data set that is used for intermediate storage of input stream data. (The attributes of the spooling data set must not be changed for a checkpoint restart if the data set was open and not completely read. The extents and number of extents do not have to remain the same.)

The three readers provided by IBM are essentially the same except for different blocking factors. They are shown in the following examples.

The standard reader procedure supplied by IBM is named RDR. It specifies a block size of 80 bytes for the concurrent peripheral operation data set.

```
//IEFPROC     EXEC      PGM=IEFIRC,REGION=48K,                          X
//                      PARM='80103005001024905010SYSDAbbbE00001A'
//IEFRDER     DD        UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,         X
//                      DISP=OLD,                                       X
//                      DCB=(BLKSIZE=80,BUFL=80,                        X
//                      BUFNO=1,RECFM=F)
//IEFPDSI     DD        DSNAME=SYS1.PROCLIB,DISP=SHR
//IEFDATA     DD        UNIT=SYSDA,                                     X
//                      SPACE=(80,(500,500),RLSE,CONTIG),               X
//                      DCB=(BLKSIZE=80,LRECL=80,BUFL=80,               X
//                      BUFNO=2,RECFM=F,DSORG=RD)
```

The two other cataloged procedures for readers supplied by IBM provide blocking for spooling, or concurrent peripheral operation, data sets of 400 (RDR400) and 3200 (RDR3200).

```
//IEFPROC     EXEC      PGM=IEFIRC,REGION=50K,                          X
//                      PARM='80103005001024905010SYSDAbbbE00001A'
//IEFRDER     DD        UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,         X
//                      DISP=OLD,                                       X
//                      DCB=(BLKSIZE=80,LRECL=80,BUFL=80,               X
//                      BUFNO=1,RECFM=F)
//IEFPDSI     DD        DSNAME=SYS1.PROCLIB,DISP=SHR
//IEFDATA     DD        UNIT=SYSDA,                                     X
//                      SPACE=(80,(500,100),RLSE,CONTIG),               X
//                      DCB=(BLKSIZE=400,LRECL=80,BUFL=400,             X
//                      BUFNO=2,RECFM=FB,DSORG=PS)

//IEFPROC     EXEC      PGM=IEFIRC,REGION=52K,                          X
//                      PARM='80103005001024905010SYSDAbbbE00001A'
//IEFRDER     DD        UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,         X
//                      DISP=OLD,                                       X
//                      DCB=(BLKSIZE=80,LRECL=80,BUFL=80,               X
//                      BUFNO=1,RECFM=F)
//IEFPDSI     DD        DSNAME=SYS1.PROCLIB,DISP=SHR
//IEFDATA     DD        UNIT=SYSDA,                                     X
//                      SPACE=(80,(500,12),RLSE,CONTIG),                X
//                      DCB=(BLKSIZE=3200,LRECL=80,BUFL=3200,           X
//                      BUFNO=1,RECFM=FB,DSORG=PS)
```

To create your own reader procedure, you can use the IBM-supplied procedures as examples. The statement requirements are explained in the following paragraphs.

The EXEC Statement

The EXEC statement specifies the reader and its region size. It also passes a set of parameters to the reader. Its format is:

```
//IEFPROC     EXEC      PGM=IEFIRC, REGION=nnnnnK,                    X
                        PARM='bpptttooommmiiicccrlsssssssaaaaefh'
```

The step name must be IEFPROC as shown. The parameter requirements are

PGM=IEFIRC
specifies the reader. Its name is IEFIRC.

REGION=nnnnnK
specifies the region size for the reader. The value nnnnn represents a number from one to five digits that is multiplied by K (1024 bytes) to designate the region size. The region requirement depends on the size of the buffers and the reader modules (if any) in the link pack area. An insufficient size specification will result in an abnormal termination. If you have a blocked procedure library, the region size will have to be increased by the block size rounded off to the next highest multiple of 2K. This allows for the increase in buffer size. If double buffering is used, the region size must be increased by twice the block size, rounded to the next highest multiple of 2K.

PARM='bpptttooommmiiicccrlsssssssaaaaefh'
is a set of parameters for the reader. This parameter field must consist of 35 characters. Their meanings are:

b
any character from 0 through 9 or A through F that indicates whether the job step can be rolled out (see the section "Rollout/Rollin") by another job step, whether it can cause rollout of another job step, whether an account number is required, and whether a programmer name is required. The following chart shows the meaning of each possible character.

Character	Can Step Be Rolled Out	Can Step Cause Rollout?	Accn' Info Required?	Pgmr Name Required
0	no	no	no	no
1	no	no	no	yes
2	no	no	yes	no
3	no	no	yes	yes
4	no	yes	no	no
5	no	yes	no	yes
6	no	yes	yes	no
7	no	yes	yes	yes
8	yes	no	no	no
9	yes	no	no	yes
A	yes	no	yes	no
B	yes	no	yes	yes
C	yes	yes	no	no
D	yes	yes	no	yes
E	yes	yes	yes	no
F	yes	yes	yes	yes

pp
two numeric characters from 00 to 14 indicating the default priority for jobs read from this input stream. Priority 14 should be avoided because it is used by the system to expedite the processing of certain jobs. When no priority is specified in the JOB statement, this default priority is assigned to the job.

ttt
three numeric characters indicating the default for the maximum time (in minutes) that each job step may run.

ooo
three numeric characters indicating the default for the primary number of tracks assigned for SYSOUT data sets. This primary allocation should meet most of your needs, so that secondary allocation will not usually be needed.

mmm
three numeric characters indicating the default for the secondary number of tracks assigned for SYSOUT data sets.

iii
three numeric characters under 255 indicating the dispatching priority of this reader while it is processing JCL statements.

ccc
three numeric characters indicating the default for the region size (specified as a number of 1024 byte blocks) assigned to job steps read from this input stream.

r

a numeric character from 0 to 3 that specifies the disposition of commands read from the input stream. The reader, if r is:

0 - passes the command to the command scheduling routine to be executed.

1 - displays the command (via a WTO macro instruction), and passes it to the command scheduling routine to be executed.

2 - displays the command (via a WTO macro instruction), asks the operator whether the command should be executed (via a WTOR macro instruction), and passes the command to the command scheduling routine if the operator replies yes.

3 - ignores the command (treated as a no operation).

The WTO and WTOR macro instructions issued by the reader are sent to the primary console in systems without the multiple console support option and to the MCS master console in systems with the MCS option.

l

a numeric character 0 or 1 which specifies the bypass label processing option. 0 signifies that the bypass label processing parameter in the label field of a DD statement is to be ignored. The label parameter is processed as no label. 1 signifies that the bypass label processing is not to be ignored. The label parameter is processed as it appears.

ssssssss

eight alphameric characters specifying the default device for SYSOUT. This becomes the UNIT subparameter in the DD statement defining SYSOUT (if the UNIT field is omitted from the DD statement). If the designation is fewer than eight characters, the ssssssss field must be padded to the right with blanks.

This default device can be specified by its address, group, or type. However, the UNIT=type form may cause all units of that type to be used for system output, since the device allocation program spreads the data sets among all candidate devices. To reserve some devices for private volumes, you should define a UNIT group which is a subset of the available direct access devices. You may specify the name SYSOUT as the default unit name for the system output data sets if it was specified at system generation; when this default is used, a unit count of 1 is implied. (UNITNAME SYSOUT is fully explained in the **System Generation** publication.)

aaaa

four hexadecimal numbers from 0000 to E000 indicating which operator command groups are to be executed if read from this input stream. This parameter is valid only for systems with the multiple console support option. In MVT systems without the multiple console support option, this parameter is set to 'E000' permitting all commands except HALT, MODE, and SWAP to be entered into the input stream. In systems with the multiple console support option four blanks default to 'E000'.

The following table shows the operator commands that are affected by the aaaa parameter with MCS. The commands are grouped by function. If the command is in a group authorized by the aaaa parameter, it is processed. If the command is not authorized by the aaaa parameter, it is ignored and an error message is sent to the master console.

Note: Informational commands (Group 0) are always valid when entered into the input stream.

Bit settings for the aaaa parameter are

Byte	Bits	Bit Settings	Meaning
0	0	1	Group 1 commands executed
(aa)	1	1	Group 2 command executed
	2	1	Group 3 commands executed
	3-7	00000	Reserved
1			
(aa)	0-7	00000000	Reserved

Example: If you wish to authorize commands from command groups 2 and 3 to be executed when entered into the input stream, code the aaaa parameter: "6000"

Command Group	Function	Commands		
0	Informational	BRDCST	LOG	REPLY
		DISPLAY	MSG	SHOW
1	System Control	CANCEL	MODIFY	SET
		CENOUT	QUIESCE	START
		HALT	RELEASE	STOP
		HOLD	RESET	USERID
		MODE		WRITELOG
2	I/O Control	MOUNT	UNLOAD	VARY *
		SWAP		
3	Console Control	VARY *		
1,2,3	Master Console	All commands are valid, plus		
			VARY MSTCONS	
			VARY HARDCPY	
			VARY CPU	
			VARY STOR	
			VARY CH	

Note: VARY (Group 2) is accepted only to VARY a non-console device online or offline. VARY (Group 3) provides only for console switching and console reconfiguration or secondary consoles.

ef

MSGLEVEL value in absence of a value in the JOB statement. If there is no MSGLEVEL= parameter in the JOB statement, job control statements and allocation/termination messages are recorded in the system output data set according to the value of the ef parameter. The values and their effects are

e

Kinds of job control statements recorded.

- 0 - JOB statement only.
- 1 - Input statements, cataloged procedure statements, and symbolic parameter substitution values.
- 2 - Input statements only.

A blank defaults to a value of 0.

f

Kinds of allocation/termination messages recorded.

0 - None, except in the case of an abnormal termination. (In that event, all messages are recorded.)
1 - All.

A blank defaults to a value of 1.

h
MSGCLASS Default Value (A-Z, 0-9). If there is no MSGCLASS keyword parameter in the JOB statement, job control statements and allocation/termination messages are recorded according to the message class specified by this character. If the character is blank or absent, A is the default class.

DD Statement for the Input Stream

Your procedure for the reader must include a DD statement that describes the input stream. The format for this statement is:

```
//IEFRDER     DD   UNIT=device,LABEL=(type),VOLUME=SER=SYSIN,          X
//                 DCB=(list of attributes)[,DSNAME=name,             X
//                 DISP=OLD]
```


This DD name must be IEFRDER as shown. The IEFRDER statement can be overridden with a START command. The parameter requirements are as follows:

UNIT=device
specifies the device from which the input stream is to be read. This can be any device supported by the queued sequential access method (QSAM). The device can be specified by its address, type, or group.

LABEL=(,type)
describes the data set label (needed only for tape data sets). If this parameter is omitted, a standard label is assumed.

VOLUME=SER=SYSIN
specifies the volume containing the input stream. This parameter is required for magnetic tape or direct access volumes. The serial SYSIN is recommended for identification of this volume, but other serials can be used.

DCB=(list of attributes)
specifies the characteristics of the input stream and the buffers. If the BLKSIZE, LRECL, and BUFL subparameters are not specified, an 80-byte value is assigned to each. Other subparameter fields may be specified as needed; otherwise, the QSAM default attributes are assigned, as follows:

BUFNO - two buffers
RECFM - U-format, with no control characters
TRTCH - odd parity, no data conversion, and no translation
DEN - lowest density

The time required for I/O operations is reduced by using chained scheduling; with this technique several I/O operations are chained together and a series of separate read or write operations is issued as one continuous operation. Chained scheduling can be used only with simple buffering. Each data set for which chained scheduling is specified must be assigned at

least two, and preferably three, buffers. Chained scheduling is discussed in the **Data Management Services** publication.

DSNAME=name
specifies the name of the input stream data set to be read, this keyword should be used only with direct access or tape input stream.

DISP=OLD
specifies that the input stream is an existing data set.

DD Statement for the Procedure Library

Your procedure for the reader must include a DD statement that defines the procedure library. This statement must follow the IEFRDER statement which describes the input stream. The format for this statement is:

```
//IEFPDSI      DD   DSNAME=SYS1.PROCLIB,DISP=SHR
```

This DD name must be IEFPDSI as shown. The parameter requirements are as follows:

DSNAME=SYS1.PROCLIB
identifies the procedure library. To concatenate other data sets with the system library, you may follow the IEFPDSI DD statement with other unnamed DD statements, thus expanding the system procedure library.

DISP=SHR
specifies that the procedure library is an existing data set and can be shared with other tasks.

DD Statement for the Spooling Data Set

Your procedure for the reader/interpreter must include a DD statement that defines the spooling, or CPO (concurrent peripheral operation) data set. Two DCB parameters (BLKSIZE, and buffer number) may be overridden by parameters in the input stream on DD * and DD DATA statements. The CPO data set is used for intermediate storage of input stream data. The format for this statement is:

```
//IEFDATA      DD   UNIT=device,                                    X
//                  SPACE=(units,(quantities)[,RLSE,CONTIG]),       X
//                  VOLUME=SER=volser,DISP=(status,disp),           X
//                  DCB=(list of attributes),DSORG=PS
```

This DD name must be IEFDATA as shown. The parameter requirements are as follows:

UNIT=device
specifies one or more direct access devices on which data sets from the input stream will be written. If more than one device is provided, the different data sets are not necessarily written in a continuous manner from device to device. Instead, the different data sets might be spread among the available devices according to a reader algorithm based on priorities and optimum access. If you want all the input stream data sets written on the same device, use the VOLUME parameter (described below) in this DD statement to identify the specific volume. The DEFER option must not be used.

CAUTION: Do not use UNIT group names unless the request is for no more than one device, or the group is defined to have devices of only one type.

SPACE=(units,(quantities),RLSE,CONTIG)
specifies space allocation for the direct access volume. The optional RLSE subparameter releases all unused space to the system when the data set is closed. The optional CONTIG subparameter ensures that space is allocated in contiguous tracks or cylinders.

VOLUME=SER=volser
identifies a specific direct access volume. This parameter is not required, but you can use it to cause all input stream data sets to be written on the same volume. You should also use this parameter if you specify the DISP parameter.

DISP=(status,disp)
specifies the status and disposition of the CPO data set. This parameter is not required, but can be used to bypass the first space allocation (as explained above). To do this, specify the parameter as DISP=OLD. The system then assumes that the data set exists, and does not allocate space for the reader/interpreter program. Subsequently, the reader/interpreter forces a DISP=(NEW,PASS) status for the CPO data set so that space is allocated on it for recording the input stream data sets.

DCB=(list of attributes)
specifies the characteristics of the spooling data set and the buffers to be used by the data set. The RECFM and the LRECL subparameters cannot be overridden and should not be specified. The values for these subparamaters are RECFM=FB and LRECL=80. The BLKSIZE and BUFL sybparameters must be specified in the IEFDATA DD statement. The BLKSIZE and BUFNO values may be overridden by specifying them on a DD * or DD DATA statement in the reader input stream. However, the BLKSIZE and BUFNO values on the IEFDATA statement are always used as upper limits. Thus, if the overriding statements exceed these limits, the IEFDATA values are used. (For a more detailed explanation of how to override these parameters, see the **Job Control Language** publication.) The BUFNO and RECFM subparameters, if not specified, assume the QSAM default attributes of two buffers.

BUFNO -- two buffers.
RECFM -- U-format, with no control characters.

DSORG=PS
must be coded as shown.

Automatic SYSIN Batching (ASB)

Readers read and interpret job control language statements and place SYSIN data sets on direct access devices for later processing. The interpreting of job control language statements often requires only a small proportion of the total time used by the reader but the reader remains resident even when inactive. You may therefore save space by separating the interpreting of job control statements from the storing of SYSIN data sets. If the two functions are separated, the interpreter portion of the reader does not have to be resident at all times and will be called into storage only after a certain number of job control language statements have been collected. Separating the two functions of the reader is called **automatic SYSIN batching (ASB)**.

IBM supplies a cataloged procedure that provides automatic SYSIN batching; this procedure is named RDRA and is invoked by a START command. The procedure is shown and described in the following text; by using it as a model, you may write your own procedure, coding the parameters suited to your installation.

```
//IEFPROC      EXEC       PGM=IEFVMA,REGION=16K,
//  PARM='80103005001024905030SYSDAbbbE00001A,
                          1101207004E000SYSDAbbbERDRH'
//IEFRDER      DD         UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,
//  DCB=BLKSIZE=80,RECFM=F,BUFL=80,BUFNO=10)
//IEFPDSI      DD         DSNAME=SYS1.PROCLIB,DISP=SHR
//IEFDATA      DD         UNIT=SYSDA,SPACE=(3200,(15,15),RLSE,CONTIG),
//  DCB=(BLKSIZE=3200,BUFNO=2,RECFM=FB,BUFL=3200)
```

The EXEC statement for the SYSIN batcher is similar to the EXEC statement for the standard reader/interpreter. It specifies the SYSIN batcher program, the MVT region size and passes a set of parameters to the program. The format is as follows:

```
//IEFPROC      EXEC PGM=IEFVMA,REGION=nnnnnK,
//  PARM=('bpptttooommmiiicccrlsssssssssaaaaefh',
//                          'ejjaarratabaaaddddddddgk')
```

The step name must be IEFPROC, as shown. The parameters are as follows:

PGM=IEFVMA
specifies the automatic SYSIN batcher program. It must be IEFVMA as shown.

REGION=nnnnnK
specifies the region size for the automatic SYSIN batch reader. The value nnnnn represents a number from one to five digits that is multiplied by K (K=1024 bytes) to designate the region size. The region requirement depends on the size and number of input buffers and ASB reader modules (if any) in the link pack area. The algorithm for estimating the required region is in the "Estimating the Dynamic Main Storage Requirement" section of the **Storage Estimates** publication. An insufficient size specification will result in abnormal termination.

PARM=('bpptttooommmiiicccrlsssssssssaaaaefh',
these parameters are the same as those used in the ordinary (non-batching) procedure. See the preceding description for an explanation of these parameters.

e
a numeric character from 0 to 3 that ordinarily specifies the disposition of commands read from this input stream. The SYSIN batcher, if e is:

0 -- executes the command.
1 -- displays the command (via a WTO macro instruction), and executes it.
2 -- displays the command (via a WTOR macro instruction),but does not execute it until advised by the operator.
3 -- ignores the command (treated as no operation).

jj
two numeric characters which indicate the number of jobs to be read by the automatic SYSIN batch reader before interpreting the job control language and putting the jobs onto the job input queue for execution.

aa
the number of logical tracks on SYS1.SYSJOBQE which can be used by the automatic SYSIN batch reader for later interpretation. The **Storage Estimates** publication contains information on estimating SYS1.SYSJOBQE work space.

rra
three decimal numbers. The size of the region, in K bytes (1024 bytes), the Automatic SYSIN batch reader routine is to obtain for the reader. The reader uses the region to read in tracks of job control statements that the ASB reader has written in the SYS1.SYSJOBQE data set. The size of the region should complement the number specified for the next parameter, the ta parameter. The **Storage Estimates** publication describes how large a region is required for the interpreter routine.

ta
two decimal numbers. Number of logical tracks of job control statements that are to be read in from the SYS1.SYSJOBQE data set into the region defined by the previous parameter, the rra parameter. The reader interprets one at a time, the statements blocked by the automatic SYSIN batch reader. The **Storage Estimates** publication describes how to determine the number of tracks of SYS1.SYSJOBQE data to be kept ready in main storage.

baaa
this parameter is the same as the aaaa parameter in the non-batching procedure.

dddddddd
the unit type or name of the direct access device where the SYSIN batcher is to temporarily store all SYSIN data. This must be the same as that indicated in the IEFDATA UNIT parameter.

g
the numeric character 0 or 1, where 1 is specified only if the SYSIN data set is to contain binary data as input to a 709/7090/7094/7094 II integrated emulator job.

k
main storage hierarchy to be used in loading the interpreter subroutine of the ASB reader.

& RDRH -- A value may be assigned in the operator START command. If none is given there, the default value in the PROC statement of this procedure is used. The default value may be changed by supplying a PROC statement with another value.

0 -- Use hierarchy 0 main storage.
1 -- Use hierarchy 1 main storage.

The DD statements are the same as those described for the standard reader, with the following exceptions:

IEFRDER (DD statement for the input stream)
The parameter requirements are the same as those for the reader/interpreter, except for the DCB parameter. This parameter specifies the characteristics of the input stream and the buffers.

If the BLKSIZE and BUFL subparameters are not specified, an 80-byte value is assigned to each. LRECL need not be specified because fixed length 80-byte records are the only input accepted by the ASB reader. Other subparameter fields may be specified as needed; otherwise, the QSAM default attributes are assigned as for the reader/interpreter.

IEFDATA (DD statement for the CPO data set)
If the BLKSIZE and BUFL subparameters are not specified, an 80-byte value is assigned to each. LRECL need not be specified because fixed length 80-byte records are the only input accepted by the ASB reader. The BLKSIZE and BUFNO parameters may be overridden by specifying them on a DD * or DD DATA statement in the reader input stream. However, the BLKSIZE and BUFNO values on the IEFDATA statement are always used as upper limits. Thus, if the overriding statements exceed these limits, the IEFDATA values are used. In addition, the ASB reader always uses one buffer for IEFDATA. Therefore, the BUFNO value specified applies only as a default.

Initiator Procedures

You may write different initiator cataloged procedures for the different types of jobs that initiators will handle. You might have a FORTRAN initiator (an initiator to handle the job classes to which FORTRAN jobs are assigned), a COBOL initiator, an I/O bound job initiator, or a CPU bound initiator.

A cataloged procedure for an initiator requires only one job control statement: an EXEC statement. Additional DD statements may be optionally added so that specific control volumes will be mounted when an initiator is started.

An EXEC statement with the step name IEFPROC specifies the initiator program and any job classes to be associated with the initiator if the START command does not specify job classes. Optional DD statements specify control volumes to be allocated to the initiator task.

The standard initiator cataloged procedure supplied by IBM is named INIT. The procedure is:

```
//IEFPROC     EXEC     PGM=IEFIIC,PARM='A,LIMIT=13'
```

If you write your own initiator procedure you must follow the format for the standard procedure. The statement requirements are explained in the following paragraphs.

The EXEC Statement

The EXEC statement specifies the initiator program and passes a set of parameters to it. The format for the EXEC statement is:

```
//IEFPROC     EXEC     PGM=IEFIIC,PARM='x[(n)][,x2[(n )]...[,LIMIT=K]
```

The step name must be IEFPROC, as shown. The parameter requirements are as follows:

PGM=IEFIIC
specifies the initiator program. The name of the program must be IEFIIC, as shown.

PARM='x[(n)][,x_1[(n_1)]...[,LIMIT=K]]'

x - Job class. (Letter A - O.) (One to eight job classes may be named.)

n - (0 - 15), a **force value** priority at which all jobs from the preceding class will be run.

K - (0 - 15) The priority above which no jobs will be run by this initiator.

If the START command for an initiator includes any job class references, all definitions in the cataloged procedure are voided.

The LIMIT= entry in the cataloged procedure means that no job may be run at a priority higher than the value indicated by K. The force value in (n above) is used for a job **unless** it is greater than the limit value (K above). You may not always specify a force value (n) priority. If you do not, priority is determined by the following order as long as the limit value, K, is not exceeded:

- The EXEC statement
- The JOB statement
- The cataloged reader procedure

If a job class is assigned a force priority, it overrides the priority indicated in any of the above three sources.

DD statements for control volumes are optional. The standard procedure INIT does not include a DD statement for a control volume. This optional facility is discussed next.

Mounting Control Volumes

A control volume that will be referred to during a catalog search can be mounted before the search begins. DD statements for control volumes may be included in initiator procedures cataloged in the procedure library (SYS1.PROCLIB). Such DD statements cause direct access volumes to be mounted and allocated for the life of the initiator. This facility is particularly useful when control volumes will be needed for departmental job batches.

Initiation by an initiator with a DD statement for a control volume ensures that the control volume will be mounted prior to a catalog search from the catalog on the system residence volume to the catalog on the control volume for a specified data set. If such DD statements for control volumes are not included in initiator procedures, an attempt will be made to mount a required control volume if a catalog search could not be completed during allocation for a step. However, when control volumes are mounted in this manner, they are eligible for demounting immediately after the catalog search has been completed and will not necessarily remain mounted for the life of the job or job step requiring them.

Initiator Action

By starting an initiator that includes a DD statement for a control volume, mounting is requested before the initiator is allowed to start initiating jobs. If the volume is already mounted, the initiator proceeds with initiation.

When a STOP command is issued for the started initiator and the volume is demountable and PRIVATE, it will be demounted if no other job steps or initiators are allocated to the volume. The volume then would stay mounted until the last job step using it terminates or the initiators using it are stopped, at which time the volume would be demounted.

DD Statement Formats

As many volumes may be defined by DD statements in the initiator procedure as the user finds useful. The format follows the specifications contained in the **Job Control Language Reference** publication. The following is an example of a DD statement that could be included in an initiator procedure for a control volume:

```
//ddname DD   VOLUME=(PRIVATE,SER=ser#),UNIT=  {address}  ,DISP=OLD
                                                {TYPE   }
                                                {group  }
```

VOLUME=(PRIVATE,SER=ser#),
specifies the volume serial of the control volume. PRIVATE ensures that this volume will not be used to satisfy job step data set requests unless requested by the specific volume serial number. Also, unless already mounted and permanently resident or reserved, the volume will be demounted when the initiator is stopped, when last used by job steps being processed by other initiators, or when other initiators allocated to the volume are stopped.

UNIT= { address } { type } { group }
specifies the unit address, unit type, or group on which the control volume is to be mounted.

DISP=OLD
specifies that a temporary data set will not be allocated to the volume. A dsname will be generated for this data set and when the initiator is stopped, a message will be written on the system output data set that this data set (generated name) has been kept. This message can be ignored as no action needs to be taken.

Dedicated Data Sets

Dedicated data sets save the time taken repeatedly to allocate (and deallocate) space used only temporarily during a job step. A dedicated data set is allocated space when the initiator is started and belongs to the initiator. Every job step running under that initiator can use the dedicated data set as a temporary data set. If you use dedicated data sets for temporary data sets, the checkpoint/restart facility is internally suppressed. To dedicate any data set quickly to successive jobs or job steps, you add a DD statement to the initiator procedure. An initiator procedure (INITD) for use of dedicated data sets with processor programs has been added to the system. To save repeated catalog searches, you may also dedicate system library data sets.

The dedicated data sets feature has been implemented by adding code to the allocation routine that, before allocating space for a temporary data set, attempts to relate a request for a temporary data set with a dedicated data set. If the space required for the temporary data set fits within the dedicated data set, the dedicated data set space is used. If not, normal allocation takes place. The same criterion will be used with presently coded requests for temporary data sets. That is, if the space requested is within the range of the dedicated data set, it will be used.

How to Dedicate a Data Set

You dedicate a data set by adding a DD statement (for each data set to be dedicated) to the initiator procedure. The unit must be a DASD; the space may be for a sequential or partitioned data set. (See the publication **Storage Estimates**, the chapter "Job Step Initiation Requirement," for details on the number of DD statements per initiator.) Each DD statement must be of the form:

```
//ddname      DD   UNIT=unitparms,VOL=volparms,
                 SPACE=(kind,(amount,increment,dirblks)),
                 DISP=(new,delete)
```

ddname
A user-supplied ddname must be given to identify the DD statement. The ddname is used (in the form DSNAME= & ddname) in the DD statement of the problem program job step which is to make use of the dedicated data set.

unitparms
Parameters that describe the unit to be used for the dedicated data set. The unit must be a DASD. The AFF= and DEFER unit parameters may not be used. The unit parameters specified here override those of the job step DD statement for which the dedicated data set is used.

volparms
Volume parameters. A volume may be specified for each unit specified in the preceding unit parameter entry. The volume parameters specified here override those of the job step DD statement for which the dedicated data set is used.

(kind,(amount, increment,dirblks))
Type and size of space (in terms of CYL, TRK, avgbl, or ABSTR) to be allocated to the data set. If ,dirblks is omitted, the data set request implies sequential organization. If,,dirblks is used, the data set request implies partitioned organization. If the dedicated data set is going to reside on an IBM 2301, or 2303 Drum Storage device, do not request space in cylinders.

When a dedicated data set with partitioned organization reaches an EOV condition, the initiator must be restarted. The DD statement in the problem program job step that is to use a dedicated data set must describe a problem program data set of the same organization as the dedicated one. Increments, once allocated, remain allocated until the initiator stops.

new,delete
These disposition parameters may either be coded explicitly or may take effect by default, if the DISP= entry is ommitted.

The effect of new is that the data set is freshly allocated from any available space on the volume, each time a START initiator operator command is used or the system is restarted.

The effect of delete is that the data set is not kept when the initiator is stopped and the space is available for reallocation to other jobs.

DSNAME

The allocation procedure for an initiator pre-allocated data set is the same as for any temporary data set. This procedure is simplest with no dsname= entry in the DD statement. That results in a system assigned data set name of the form:

SYSnumber.Rnumber.procname.RVnumber.

You may also code DSNAME= & name, DSNAME= & & name, or DSNAME=name. These names will override those used in the job step DD statement for which the dedicated data set is used.

DCB parameters:

DCB parameters specified here have no effect.

How to Use a Dedicated Data Set

If you want a dedicated data set to be used temporarily in a job step, define the temporary data set in a DD statement of the form:

```
//ddname      DD   DSNAME=&ddname,
//                 SPACE=(avgbl,(amount,increment,dirblks)),
//                 UNIT=unitparms,DISP=(new,delete),DCB=dcbparms
```

& ddname

name of the DD statement for the dedicated data set, preceded by an & sign.

(avgbl,(number,increment,dirblks))

Space request, in terms of average block length only, needed for this temporary data set.

An attempt to allocate the dedicated data set will be replaced by the normal allocation procedure if one of the following conditions is encountered:

- If the total space (primary and increments) requested here exceeds the total space (primary and increments) available to the dedicated data set.
- If the use of ,dirblks (presence or absence) differs from that in the DD statement of the dedicated data set, (or if ISAM is specified).
- If the use of ,dirblks requested here exceeds the space for ,dirblks specified in the dedicated data set.
- If the space request is shown in other than average block length.
- Although the total space (primary and increments) requested here is compared to the total space (primary and increments) available to the dedicated data set, the primary quantity in the DD statement of the initiator procedure will be allocated to the data set, and not the primary quantity requested here. If a secondary quantity is specified, it will override the secondary quantity specified in the initiator procedure's DD statement.

unitparms

Unit parameters describe the unit to be used for the temporary data set, if the dedicated data set is not used. Here, the unit may be a magnetic tape unit, as well as a DASD.

(new,delete)
These disposition parameters must either be coded explicitly or may take effect through default.

dcbparms
DCB parameters required for the temporary data set. Unless specified, you may find that a previous user has left the dedicated data set with undesired DCB parameters.
If a secondary increment is coded in the SPACE parameter, the DCB subparameter BLKSIZE should be coded since the system will use it to calculate the number of tracks required to fulfull the secondary quantity request.

Procedure INITD

Language processor programs, such as FORTRAN compilers, make much use of temporary data sets. To permit ready use of the dedicated data set feature with IBM-supplied processor procedures, IBM supplies the initiator procedure INITD. It becomes part of the system by inclusion in the SYS1.PROCLIB at system generation time.

INITD is an initiator procedure that dedicates five utility data sets commonly used with IBM-supplied processor procedures. To use the dedicated data set facility with these procedures, start the INITD inititator.

Before including the INITD procedure in the system, review the space allocations, unit specifications, and ddnames used in the procedure against the system's requirements. If they are significantly different, code your own.

Presently existing procedures can be used under the INITD initiator without changes. Procedures designed for the dedicated data set feature remain in operation without the presence of the dedicated data set feature. In short, the procedure will run under any initiator regardless of whether that initiator has dedicated data sets.

The INITD procedure is:

```
                        Procedure:  INITD
//IEFPROC     EXEC PGM=IEFIIC,PARM='A,LIMIT=13'
//SYSUT1      DD   DSNAME=&UT1,SPACE=(1700,(200,100),,CONTIG),UNIT=SYSDA
//SYSUT2      DD   DSNAME=&UT2,SPACE=(1700,(200,100),
                   UNIT=(SYSDA,SEP=SYSUT1)
//SYSUT3      DD   DSNAME=&UT3,SPACE=(1700,(200,100)),
                   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2))
//SYSUT4      DD   DSNAME=&UT4,SPACE=(460,(700,100)),
                   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2,SYSUT3))
//LOADSET     DD   DSNAME=&LOADSET,UNIT=(SYSDA,SEP=SYSUT1),
                   SPACE=(3600,(100,10))
```

Each statement of the INITD procedure is explained in detail in the following. In addition to describing the reason for or effect of the use of a prarmeter, the description distinguishes between those parameters that must be coded as shown and those that you may override or substitute for.

The EXEC Statement

The EXEC statement for the procedure is:

```
//IEFPROC     EXEC PGM=IEFIIC,PARM='A,LIMIT=13'
```

IEFPROC
The step name. Must be coded as shown.

EXEC
The job control statement name. Must be coded as shown. Defines the beginning of a job step.

PGM=IEFIIC
The program to be executed in this job step. IEFSD060 is the name of the initiator program. Must be coded as shown. Whether dedicated data sets are used depends on the DD statements that follow, not on the name of the program.

PARM='A,LIMIT=13'
Parameter list for the initiator program. A is the class of jobs to be processed, LIMIT=13 is the dispatching priority limit for this initiator. Both of these values can be overridden by values used with the START command for the initiator.

DD Statement for the Dedicated Utility Data Sets

There are four DD statements in the INITD procedure that allocate space to four commonly used utility (or scratch) data sets. The statements are:

```
//SYSUT1      DD   DSNAME=&UT1,SPACE=(1700,(200,100),,CONTIG),UNIT=SYSDA
//SYSUT2      DD   DSNAME=&UT2,SPACE=(1700,(200,100),
                   UNIT=(SYSDA,SEP=SYSUT1)
//SYSUT3      DD   DSNAME=&UT3,SPACE=(1700,(200,100)),
                   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2))
//SYSUT4      DD   DSNAME=&UT4,SPACE=(460,(700,100)),
                   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2,SYSUT3))
```

DSNAME
The leading & sign marks the name as that of a temporary data set.

SPACE=
The first three data sets will be assigned space that can accomodate 200 blocks of 1700 bytes. When that space is exhausted, additional space will be allocated for 100 blocks at a time. Additionally, for the first data set, SYSUT1, all the primary space is to be allocated space for 700 blocks of 460 bytes initially. When exhausted, space is to be allocated for 100 blocks at a time.

UNIT=
Space is to be allocated from direct access storage devices. If possible, each data set is to be on a separate device from every other data set to avoid contention for the device.

DD Statement for the LOADSET Data Set

In the INITD procedure, the dedicated data set for the object module -- the LOADSET data set -- is defined as follows:

```
//LOADSET     DD   DSNAME=&LOADSET,SPACE=(3600,(100,10)),
                   UNIT=(SYSDA,SEP=SYSUT1)
```

LOADSET
DDName of the dedicated data set.

DD
Data definition statement.

DSNAME= & LOADSET
A temporary data set.

SPACE=(3600,(100,10))
Space allocation commonly used in compilers.

UNIT=
Space is to be allocated on a DASD but not the same one as the SYSUT1 data set.

Use of Dedicated Data Sets by Processing Programs for Utility Data Sets

Presently, processor programs show the temporary nature of the utility data sets by omitting a DSNAME= entry. If these DD statements are revised with the addition of a DSNAME= & name entry, the system will attempt to use dedicated data sets of the INITD program for job steps processed under that initiator. To illustrate the necessary change, let us look at a present DD statement and the change required. The following is a DD statement from the COBECLG procedure for which a temporary data set will be allocated:


```
//SYSUT1      DD   UNIT=SYSDA,SPACE=(1024,(200,65))
```

The temporary character of this data set is shown by the absence of a DSNAME= entry. To force consideration of the dedicated data set, assuming that the step is running under the INITD procedure, add a DSNAME= & name (or & & name) entry referring to the dedicated data set to be considered for use:

```
//SYSUT1      DD   UNIT=SYSDA,SPACE=(1024,(200,65)),DSNAME=&SYSUT1
```

With the addition of the dedicated data set feature, the allocation program now first searches the DD statements in the initiator procedure for an already existing data set with a DD name like that following the & sign (the symbolic name). If the allocation program finds such a data set, it next determines whether the organization (sequential, partitioned) of the dedicated data set is the same as that of the temporary data set and whether the total space requirements (primary and increments) of the temporary data set fall within the total space allocation of the dedicated data set. If there is no dedicated data set with the symbolic name, the orgainizations are not the same, or the temporary space does not fit within the dedicated space, the initiator will attempt normal allcoation. It is for the latter event that unit parameters should be present.

System Library Data Sets as Dedicated Data Sets

System library data sets, such as the COBOL library, may be referred to repeatedly in a batch of jobs. To save allocating the system data set in each job and step, the system data set can be dedicated in an initiator procedure. Caution must be exercised when dedicating system libraries or other non-temporary data sets. The DD statement in the initiator procedure must have the disposition specified as old or share and keep to prevent the deletion of the data set when the initiator is stopped. In the same manner, the disposition on the job step DD statement referring to the dedicated library must also be old or share and keep or pass to allow the dedication to take place without a space comparison. The example data set references are as follows.

The following is the DD statement in the COBECLG procedure that results in the allocation of the COBOL library to the job step calling the procedure:

```
//SYSLIB       DD   DSNAME=SYS1.COBLIB,DISP=(SHR,KEEP)
```

The explicit data set reference (DSNAME=SYS1.COBLIB) requires a search of the catalog in each job step using the procedure. To save the repeated catalog search, move the DD statement to the initiator procedure and replace it in the COBECLG procedure with a DD statement in which the DSNAME= & name entry refers to the ddname of the dedicated data set. Allocation treats this as a dedication request, dedicated if so found. The new DD statement in the COBECLG procedure, after adding the present one to the initiator, is:

```
//SYSLIB       DD   DSNAME=&SYSLIB,DISP=(SHR,KEEP)
```

The result is one catalog search per initiator start instead of one catalog search every job step. However, keep in mind that this COBECLG procedure requires the initiator with the dedicated data set. Using this modified procedure with an unmodified initiator will result in failure to allocate.

Disposition of Temporary Dedicated Data Sets

Allocation/termination routines do not delete temporary dedicated data sets at the end of each job step, but, instead, keep them until the initiator stops; this occures even if there is a specification of DISP=(NEW,DELETE) or DISP=(MOD,DELETE) on the DD statement for the data set. Therefore, if you attempt to use such a data set a second time in the same job, it will contain data from the previous use. This can be a problem if you are using cataloged procedures and run the same procedure twice within the same job. For example: assume that you use the procedure PL1LFLCLG twice within the same job and it uses a dedicated data set with a disposition of (MOD,PASS) for the compile step and (OLD,DELETE) for the linkage edit step. When the procedure is entered for the second time, the object module produced by the second compile step will be placed in back of the object module produced by the first compile step. Since both object modules are assigned identical names by the compiler, only the first will be linkage edited.

You can avoid this problem by not using dedicated data sets for jobs that run the same cataloged procedure twice. Alternatively, using DISP=(NEW,DELETE), you could submit each cataloged procedure as separate jobs instead of submitting them as separate job steps within the same job.

Use the following chart to determine the disposition, by allocation/termination, of temporary data sets.

If you code	**Allocation/termination treats it as:**
DISP=	
NEW	OLD
OLD/SHR	OLD
MOD	MOD
,DELETE	KEEP
,PASS	PASS
,KEEP	KEEP

System Output Writers

A cataloged procedure for output writers requires two job control statements: an EXEC statement and a DD statement.

An EXEC statement with the step name IEFPROC specifies the output writer program.

A DD statement named IEFRDER defines the output data set. The standard output writer procedure supplied by IBM is named WTR; it is shown in the following sequence:

```
//IEFPROC     EXEC      PGM=IEFSD080,REGION=20K,                    X
//                      PARM='PA'
//IEFRDER     DD        UNIT=1403,VOLUME=(,,,35),                   X
//                      DSNAME=SYSOUT,DISP=(NEW,KEEP),              X
//                      DCB=(BLKSIZE=133,LRECL=133,BUFL=133,        X
//                      BUFNO=2,RECFM=FM)
```

When creating your own output writer procedure, you must conform to the procedure format and the statement requirements. Use the IBM-supplied procedure as an example. The statement requirements are explained individually in the following paragraphs.

SEC IV

The EXEC Statement

The EXEC statement specifies the output writer program and its region size. It also passes a set of parameters to the output writer program. The format for the EXEC statement is:

```
//IEFPROC     EXEC      PGM=IEFSD080,REGION=nnnnnK,                 X
//                      PARM='cxxxxxxxx,seprname'
```

The step name must be IEFPROC, as shown. The parameter requirements are as follows:

PGM=IEFSD080
: specifies the output writer program. The name of the program must be IEFSD080, as shown.

REGION=nnnnnK
: specifies the region size for the output writer. The value nnnnn represents a number from one to five digits that is multiplied by K (K=1024 bytes) to designate the region size. The region requirement depends on the size of the buffers, the data set writer used, and which modules of the output writer (if any) are in the link pack area. The complete algorithm for estimating the required region is contained in the "Estimating the Dynamic Main Storage Requirement" section of the **Storage Estimates** publication. An insufficient size specification will result in an abnormal termination.

PARM='cxxxxxxxx,seprname'
: is a set of parameters for the output writer program. The first part of this parameter field can contain from two to nine characters. The second part of this parameter field, if specified, is separated from the first part by a comma, and contains a program name from one to eight characters. Both parts of this parameter field are explained below.

  c
  : an alphabetic character, either P (for printer) or C (for punch), that specifies the type of control characters for the output of the writer.

xxxxxxxx
from one to eight (no padding required) single-character class names for system output. These specify the type of output that the writer can process, and also establish the priority of the output classes, with the highest priority on the left. If class name parameters are included in the START command, they override this entire set of class names in the cataloged procedure.

seprname
the name of the program (up to eight characters) that provides job separation in the output data set. The named program must reside in the link library (SYS1.LINKLIB). You can specify the name IEFSD094 to use the output separator supplied by IBM, or you can specify the name of your own program. This subparameter may be omitted, in which case no output separator is used.

DD Statement for the Output Data Set

Your procedure for the output writer must include a DD statement that defines the output data set. The format for this statement is:

```
//IEFRDER     DD        UNIT=device,LABEL=(,type)                    X
//                      VOLUME=(,,,volcount),                        X
//                      DSNAME=anyname,DISP=(NEW,KEEP)               X
//                      DCB=(list of attributes),                    X
//                      UCS=(code[,FOLD][,VERIFY]),                  X
//                      FCB=(image-id [,ALIGN ])
                                      [,VERIFY]
```

This DD name must be IEFRDER as shown. The parameter requirements are as follows:

UNIT=device
specifies the printer, magnetic tape, or card punch device on which the output data set will be written. The devices that can be used are: 1403, 1442, 1443, 2400, 2400-1, 2400-2, 2400-3, 2400-4, 2520, 2540, 3211, 3400-2, 3400-3, or 3400-4.

LABEL=(,type)
describes the data set label (needed only for tape data sets). If this parameter is omitted, a standard label is assumed.

VOLUME=(,,,volcount)
limits the number of tape volumes that can be used by this writer during its entire operation (from the time it is started to the time it is stopped). This parameter is not required for printer or card punch devices.

DSNAME=anyname
specifies a name for the output data set (tape only, for label purposes), so that it can be referred to by subsequent job steps. This name is also necessary for specification of the KEEP subparameter in the DISP field.

DISP=(NEW,KEEP)
specifies the KEEP subparameter to prevent deletion of the output data set (tape only) at the conclusion of the job step.

DCB=(list of attributes)
specifies the characteristics of the output data set and the buffers. The BLKSIZE and LRECL subparameter fields must be specified in all cases. The BUFL subparameter field, if not specified, is calculated on the basis of the BLKSIZE value. Other subparameter fields may be specified as needed; otherwise, they will assume the QSAM default attributes which are

BUFNO --three buffers for the 2540 device, two buffers for all other devices.
RECFM --U-format, with no control characters.
TRTCH --odd parity, no data conversion, and no translation.
DEN -- lowest density.

UCS=(code[,FOLD][,VERIFY])
specifies the code for a universal character set (UCS) image that will be loaded into the UCS buffer. FOLD causes bits 0 and 1 to be ignored when comparing characters between the UCS buffer and the print line buffer. This option allows lowercase alphabetic characters to be printed in uppercase by an uppercase print chain or train. VERIFY causes the specified UCS image to be printed for verification by the operator. The UCS parameter is optional, and is valid only when the output device is a 1403 or 3211.

FCB=(image-id[,ALIGN / VERIFY])
causes the forms control buffer (FCB) image with the specified image-id to be loaded into the FCB. One of two optional parameters, ALIGN or VERIFY, can be coded. Either parameter allows the operator to align forms. In addition, VERIFY causes the specified FCB image to be printed for visual verification. The FCB parameter is valid only when the output device is a 3211.

For the processing of output jobs that require special chains for printing, you should have specific classes for each different chain. You can specify the desired chain in your writer procedure, and when that writer is started the chain will be loaded automatically. (Printers used with special chains should be named with esoteric device names as defined at system generation time.)

The following sequence is an example of a writer cataloged procedure for the P11 chain.

```
//IEFPROC     EXEC     PGM=IEFSD080,REGION=20K,                        X
//                     PARM='PDEG,IEFSD094'
//IEFRDER     DD       UNIT=SYSPR,DSNAME=SYSOUT,FCB=(STD2,ALIGN),      X
//                     UCS=P11,                                        X
//                     DISP=(,KEEP),DCB=(BLKSIZE=133,BUFL=133,         X
//                     LRECL=133,BUFNO=2,RECFM=FM)
```

If the output device is a 3211, a UCS or FCB image can be loaded dynamically between the printing of data sets. Therefore, a mixture of data sets using different images in a single output class is allowed; however, this may require mounting trains and changing forms, and may not be desirable. When the output device is a 1403, the UCS image is specified at START WTR time and cannot be changed until the writer is stopped; all data sets within an output class must be printed using the same train. This parameter cannot be overridden for a specific data set when using the (asynchronous) Sysout Writer. The FCB image is ignored when the 1403 is specified.

Command Chaining

By using command chaining you may reduce the amount of CPU time used by a writer; this is done by having the SYSOUT writer intercept PUT instructions and execute an EXCP only when all of a chain of buffers are full. This command chaining is provided if the writer procedure specifies all of the following conditions:

1. It uses more than three buffers.
2. It uses machine control characters in writing to the output print or punch device.
3. It does not use PCI.
4. The output device is a printer or punch.

Direct SYSOUT Writers

The direct SYSOUT writer is an option that results in writing output directly from (synchronously with the execution of) the problem program. It requires two job control statements: an EXEC statement and a DD statement.

- The EXEC statement is named IEFPROC.
- The DD statement is named IEFRDER and describes the ultimate output data set.

The procedure supplied by IBM is named DSO and is described in the following. If you wish to create your own procedure, follow its format.

```
                              Procedure:  DSO
//IEFPROC     EXEC      PGM=IEFDSO,REGION=8K,PARM=(PA,,A)
//IEFRDER     DD        UNIT=2400,DSN=SYSOUT,DISP=(NEW,KEEP),
                        LABEL=(,SL),VOL=(,,,05),DCB=(BUFNO=3)
```

The statement requirements are explained individually in the following paragraphs.

The EXEC Statement

The EXEC statement specifies the direct SYSOUT writer and the space it requires to start in MVT. It is also used to give the writer program necessary operating information.

```
//IEFPROC     EXEC      PGM=IEFDSO,REGION=8K,
                        PARM=(cx,seprname,jjjjjjjj)
```

IEFPROC
Name of the EXEC statement.
Required as shown.

IEFDSO
Name of the writer program.

REGION=8K
Space required by IEFDSO to start in MVT.

PARM=
Information for the IEFDSO program.

c

A letter, P for printer or C for card punch, that describes the ultimate hard-copy medium. Must be given.

x

The SYSOUT class to be processed. If stated here, and in the START command, the latter rules. If not stated here, must be given in the START command.

,seprname

Output separation program name. May be omitted, but comma must be written if other items follow. IEFSD094 - Name of the IBM-supplied separator program.

,jjjjjjjj

Jobclasses to be processed. From zero to eight letters (A - O) showing the job classes to be processed.

If any job classes are named in the START command, they overrule all stated here. If none are named here they must be given in the START command.

The DD Statement

This DD statement describes the kind of volume to be used and the format of the data set.

```
//IEFRDER     DD   UNIT=name,DSN=anyname,DISP=(NEW,KEEP),LABEL=(,SL),
                  VOL=(,,,volcount),DCB=(list),UCS=(code[,FOLD][VERIFY]
                                    [,ALIGN ]
                  FCB=(IMAGE-ID[,VERIFY ])
```

IEFRDER

Name of the DD statement.
Required as shown for IEFDSO.

name

Any form of unit identification may be used, for example, 00E, 2400, or TAPE.
Multiple parallel units (UNIT=2400,2) cannot be used.

DSN=anyname

Name of a non-temporary data set.
A name must be given.
If stated here and in the START command also, the latter rules.
The name is used in the disposition messages at step termination, and must be used to identify the data set if it is to be printed later from tape.

DISP=(NEW,KEEP)

Required disposition.

LABEL=(,SL)

If DSO is being used to write to magnetic tape, standard label tapes are required. The label description may be stated explicitly or may be omitted, in which case SL is assumed.

,,,volcount
1 - 225.
The maximum number of volumes a data set to be processed by this writer will have. Determines the amount of job queue space allocated to each SYSOUT data set processed by this writer. After the first 5 volumes, each subsequent 15 require another job queue record. If omitted, 1 is assumed.
If stated here and also in the START command, the latter rules. This value cannot be given in a DD statement of a job to be processed.

list
The following DCB parameters gain control only if they are not also given in the SYSOUT DD statement or in the DCB macro instruction (that is, default values can be stated in this procedure):

```
BFALN, BFTEK, BUFL, BUFNO, BLKSIZE, LRECL, RECFM, NCP, HIARCHY, UCS.
```

The following DCB parameters, if stated here, override all except those given in a Start command:

```
CODE, DEN, MODE, OPTCD, PRTSP, STACK, TRTCH.
```

The FUNC parameter, when coded here, pertains only to system messages, not data set output. Punch (P) and interpret (I) are the only valid subparameters for system message processing.

UCS=(code[,FOLD][,VERIFY])
A UCS image can be specfied if the device is a UCS printer. the specified code is a one to four character name that identifies the UCS image.

Fold and VERIFY are optional. If the UCS parameter is specified in the START command, that specification will be used instead of the specification in this procedure.

FCB=(image-[,ALIGN][,VERIFY]
An FCB image load can be specified if the output device is a 3211 printer. The specified image-id is a one to four character name that identifies the FCB image.
ALIGN or VERIFY is optional, but only one can be coded. If the FCB parameter is specified in the START command, that specification will be used instead of the specification in this procedure.

Note: UCS and FCB images established in the DSO procedure or in the START command are maintained from job to job until one or both are overridden by a subsequent DD statement or SETPRT macro instruction. If this happens and the new image is a default image, it is maintained until another image is specified. If the current image is not a default, the original image established in the START command or the DSO procedure will be used.

Choosing Direct SYSOUT Writers

Since you have a choice between direct system output writers and system output writers, the following factors should be considered when deciding which type of writer to use:

- A direct system output writer does not require its own region. This can be an advantage to the user that has a 256K system.

- Each direct system output writer can process only one output class and needs one I/O device assigned to it. In a system with several active initiators, there will probably not be enough I/O devices to run direct system output writers for all output classes. In this case, it is possible to have jobs running with more than one output class. A direct system output writer could handle one output class and system output writers could handle the others.

- Direct system output cannot handle output from system tasks, jobs canceled while on the input or hold queue, and jobs failed by the reader. It is necessary to start a system output writer to handle these types of output.

- System output writers and direct system output writers could be used together. If the output queue is filled, direct system output writers could be started. This would allow the system output writers to clear the output queue, without stopping work in the problem program regions.

The installation should assign specific output classes and devices to be handled by direct system output writers. In addition to better control over the installation, the following considerations also apply:

- The system output writer attempts to correct certain error conditions (for example, invalid control characters); the direct system output writer does not have this facility. Therefore, a job that runs successfully under the system output writer might abnormally terminate when using the direct system output writer.

- If a direct system output writer was active at checkpoint time, it must be active and allocated to the same device type when a restart is attempted. If certain devices are not reserved for direct system output, the operator will have to stop any writer going to the needed device and start the necessary direct system output writer. He will then have to restore the previous configuration when the restart job is completed.

- Output from the direct system output writer cannot be stopped without canceling the entire job. Therefore, for example, it would be desirable to assign dumps or punched output that might not be needed to classes being handled by the regular system output writer.

Also, it is strongly recommended that job separators be used with direct system output writers; otherwise, output from more than one job may run together on a page.

SYSIN and SYSOUT Data Blocking

Blocking input stream records reduces the time required to access and process them. If the records are blocked, auxiliary storage space is conserved since more records can occupy the same amount of space. There are two types of blocking for the input reader:

- input to the reader from the job stream
- output from the reader (input to the processing program)

If the input is to be blocked, you specify the number of buffers and their sizes on the IEFRDER statement in a reader cataloged procedure. This is dependent on the type of input device used. These can be overridden by specifying the new buffer sizes and number in the START command for the reader.

The output from the reader (the input stream data that is transferred from the input stream to a direct access device) can be blocked. You indicate the block size in the IEFDATA statement in the reader cataloged procedure.

Blocking system output will improve performance since it also reduces disk arm interference; of course, at the same time it requires additional main storage allocation for the problem program region and the output writer. The additional space required in each case is equal to the logical record length times the blocking factor plus the input buffer space. You specify this blocking in the program which writes the output data set on a direct access device (for later writing by the writer), and you should consider it when specifying region size in the writer procedure.

In planning your blocking for the reader you can determine the block sizes accepted by the various processors and utilities by using the following chart.

Data Blocking Accepted by Processors under MVT		LRECL RECFM BLKSIZE		
Processor	SYSPRINT	SYSPUNCH	SYSIN (IEFDATA)	SYSLIN (≤3200)
American National Standard COBOL	121 FBA FT	80 FB FT	80 FB FT	80 FB FT
Assembler F	121 FBM FT	80 FB FT	80 FB FT	80 FB FT
FORTRAN E (with PRFRM option)	121 FM 121	80 F 80	80 FB FT	80 FB FT
FORTRAN G	120 FBSA FT	80 FB FT	80 FB FT	80 FB FT
FORTRAN H	137 VBA FT	80 FB FT	80 FB FT	80 FB FT
PL/I F	125 VBA FT	80 FB FT	<100 FB FT	80 FB FT
Linkage Editor E15,E18 Linkage Editor F44 Linkage Editor F88,F128	121 FM 121 121 FM,FBM 605 121 FM,FBM FT≤4840			80 F,FS 80 80 F,FS,FB,FBS 400 80 F,FS,FB,FBS 3200
Sort	 U 120		80 FB FT	
RPG	121 FA 121	80 F 80	80 FB FT	80 F 80
Utilities	121 FBA FT	 NA	80 FB FT	

F=Fixed; FA=Fixed, USASI control characters; FB=Fixed blocked; FBA=Fixed blocked, USASI control characters; FBSA=Fixed blocked, standard blocks, USASI control characters; FBM=Fixed blocked, machine control characters; VB=Variable blocked; VBA=Variable blocked, USASI control characters; FT=Full track; U=Undefined

Cataloged Procedure Examples for a Complete Installation

The following examples show how an installation may design and use catalog procedures to increase throughput and simplify the job of the MVT operator. The procedures were written for an installation which does batch job processing primarily. Its job mix is approximately:

```
FORTRAN, PL/I        50%
COBOL                25%
RPG                  15%
other                10%
```

The majority of the jobs will accept blocked SYSIN data. The FORTRAN and PL/I jobs require small amounts of main storage for the go phase and have a short execution time. Most of the COBOL and RPG jobs require extended execute time and large amounts of main storage.

The sample procedures include:

READER:
- RA - Automatic SYSIN Batch Reader
- RB - Reader for blocked SYSIN with chained scheduling
- RU - Reader for unblocked SYSIN with chained scheduling

WRITERS:
- WC - Writer with command chaining capability
- WU - Writer for special chains

INITIATORS:
- ICR - Initiator to initiate COBOL and RPG jobs (alias IRC)
- IFP - Initiator to initiate FORTRAN and PL/I jobs (alias IPF)

There are also compile-link edit-execute procedures for each of the two initiator procedures.

RA - Automatic SYSIN Batch Reader Procedure

```
//ASB          PROC      BKS=3200,SS="20,20",REG=16K
//IEFPROC      EXEC      PGM=IEFVMA,REGION=&REG,                          X
//  PARM='80103005001024905030SYSDA    E00001,1101210004E000SYSDA  0'
//IEFRDER      DD        UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,          X
//                       DCB=(BLKSIZE=80,RECFM=F,BUFNO=10,BUFL=80)
//IEFPDSI      DD        DSNAME=SYS1.PROCLIB,DISP=OLD
//IEFDATA      DD        UNIT=SYSDA,SPACE=(&BKS,(&SS)),                   X
//                       DCB=(BLKSIZE=&BKS,BUFNO=2,RECFM=FB,BUFL=&BKS)
```

RA- ASB Reader Procedure

RA, the cataloged procedure for the automatic SYSIN batch reader requests space for the SYSIN data without the CONTIG or RLSE parameters; this may require more direct access space but will improve overall performance. The symbolic parameters allow the space allocated and number of blocks allocated for SYSIN and the size of the blocks to be varied at START time by the operator. The procedure specified twenty blocks of 3200 bytes for primary and secondary allocation. Programs which use these SYSIN data sets should allocate a region size large enough to accomodate the buffer lengths.

RB - Reader for Blocked SYSIN Data

```
//            PROC      REG=60K,BKS=3600,SS='15,15'
//IEFPROC     EXEC      PGM=IEFIRC,REGION=&REG,                          X
//  PARM='80103005001024905010SYSDA    E00001'
//IEFRDER     DD        UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,          X
//  DCB=(BLKSIZE=80,RECFM=FB,BUFL=80,BUFNO=5,OPTCD=C)
//IEFPDSI     DD        DSNAME=SYS1.PROCLIB,DISP=OLD
//IEFDATA     DD        UNIT=SYSDA,SPACE=(&BKS|(&SS)),                   X
//                      DCB=(BLKSIZE=&BKS,LRECL=80,RECFM=FB.             X
//                      BUFL=&BKS,BUFNO=2)
```

RB - Reader Procedure for Blocked SYSIN

The procedure RB reads jobs with blocked SYSIN data; the operator can change the size and number of blocks allocated for SYSIN. The procedure requests a primary space allocation of fifteen 3600-byte blocks. The region parameter may also be changed to accommodate different SYSIN buffer and block sizes. The chained scheduling option is used to improve performance in the same manner as command chaining is used with the writer.

RU - Reader Procedure for Unblocked SYSIN

```
//IEFPROC     EXEC      PGM=IEFIRC,REGION=50K,                           X
//  PARM='80103005001024905010SYSDA    E00001'
//IEFRDER     DD        UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,          X
//  DCB=(BLKSIZE=80,BUFL=80,RECFM=F,BUFNO=10,OPTCD=C)
//IEFPDSI     DD        DSNAME=SYS1.PROCLIB,DISP=OLD
//IEFDATA     DD        UNIT=SYSDA,SPACE=(80,(500,500),RLSE,CONTIG),     X
//  DCB=(BLKSIZE=80,BUFL=80,BUFNO=2,RECFM=FB,LRECL=80)
```

RU - Reader Procedure for Unblocked SYSIN

This procedure is for a reader which handles jobs with unblocked SYSIN. Chained scheduling is used in the procedure.

WC - Writer with Command Chaining

The writer procedure WC uses command chaining which provides better performance by obtaining a full chain of buffers before writing to the device. Additionally, symbolic parameters in the procedure allow the operator to specify in the START command that output is to be routed to a printer, tape, or punch (command chaining will function only with punch or printer). When punch output is routed to a tape (as an intermediate device), the PARM field of the procedure must contain a **C** instead of a **P** to specify card punch control characters.

```
//WC          PROC      REG=22K,BN=5,PC=P
//IEFPROC     EXEC      PGM=IEFSD080,REGION=&REG,                        X
//                      PARM='&PC.AEFG'
//IEFRDER     DD        UNIT=1403,DSNAME=SYSOUT,DISP=(,KEEP),            X
//                      VOLUME=(,,,20),LABEL=(,NL),                      X
//                      DCB=(BLKSIZE=133,BUFL=133,LRECL=133,             X
//                      RECFM=FM,BUFNO=&BN)
```

WC - Writer Procedure with Command Chaining

WU - Writer for Special Chains

The writer procedure WU is only for printer output requiring special UCS print chains; the print chain specified in the procedure is the PN chain (which will be the default chain if the operator's START command does not specify a chain). The UCS chain and a unit name (e.g., SOUT) must have been defined at system generation time to identify the devices that can handle UCS chains. The writer will write output classes J, K, and L which have been previously defined by the installation to contain all output requiring special chains. The symbolic parameters in the procedure allow the operator to specify any special print chains in his START command.

The WU catalog procedure could be duplicated (without the symbolic parameters) for each UCS chain to be used in which case it should be named with the name of the chain; for example, the following procedure could be named WPN.

```
//WU          PROC     REG=20K,CHN=PN
//IEFPROC     EXEC     PGM=IEFSD080,REGION=&REG,                    X
//                     PARM='PJKL,IEFSD094'
//IEFRDER     DD       DSNAME=SYSOUT,DISP=(,KEEP),                  X
//                     UNIT=SOUT,UCS=&CHN,                          X
//                     DCB=(BLKSIZE=133,LRECL=133,BUFL=133,         X
//                     RECFM=FM,BUFNO=2)
```

WU - Writer Procedure for Special Chains

Initiator Catalog Procedures

The following two initiator procedures use dedicated data sets; the necessary work files for the compilers are allocated at START time and remain allocated as long as the initiators are active. The two procedures are for:

1. An initiator to handle job classes D and E which will include ANS COBOL and RPG compile-link edit-execute jobs.
2. An initiator for job classes F and G which will include FORTRAN H and PL/I compile-link edit-execute jobs.

The following initiator procedure for classes D (primary) and E (secondary is named ICR (Initiator for COBOL and RPG) and may be stored under the alias IRC.

```
//IEFPROC     EXEC     PGM=IEFIIC,PARM='DE,LIMIT=8'
//SYSABEND    DD       SYSOUT=A,SPACE=(TRK,(1,10))
//LOADSET     DD       UNIT=SYSDA,SPACE=(80,(500,100))
//SYSUT4      DD       SPACE=(460|(700,100)),                       X
//                     UNIT=(SYSDA,SEP=(LOADSET))
//SYSUT3      DD       UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2      DD       SPACE=(460,(700,100)),                       X
//                     UNIT=(SYSDA,SEP=(SYSUT2,SYSUT3))
```

ICR - Initiator Procedure for COBOL and RPG

The following examples are compile-link edit-execute procedures for American National Standard COBOL and RPG; these procedures may be used for jobs initiated by any initiator. For ease of programming, the procedures are named the same as the IBM-supplied procedures, but are actually modified to handle the dedicated data sets.

```
//COB          EXEC     PGM=IKFCBL00,REGION=86K,PARM=SUPMAP
//SYSPRINT     DD       SYSOUT=A
//SYSUT1       DD       DSNAME=&SYSUT1,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2       DD       DSNAME=&SYSUT2,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3       DD       DSNAME=&SYSUT3,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4       DD       DSNAME=&SYSUT4,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSLIN       DD       DSNAME=&LOADSET,UNIT=(SYSDA,SEP=SYSUT1),     X
//                      DISP=(MOD,PASS),SPACE=(80,(500,100))
//LKED         EXEC     PGM=IEWL,REGION=96K,PARM='LIST,XREF,LET',    X
//                      COND=(5,LT,COB)
//SYSLIB       DD       DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1       DD       DSNAME=&SYSUT1,UNIT=SYSDA,SPACE=(1024,(50,20))
//SYSPRINT     DD       SYSOUT=A
//SYSLMOD      DD       DSNAME=&GOSET(RUN),DISP=(,PASS),UNIT=        X
//                      (SYSDA,SEP=SYSUT1),SPACE=(1024,(50,20,1))
//SYSLIN       DD       DSNAME=&LOADSET,DISP=(OLD,DELETE)
//             DD       DDNAME=SYSIN
//GO           EXEC     PGM=*.LKED.SYSLMOD,COND=((5,LT,COB),(5,LT,LKED))
```

COBUCLG - Compile-Link Edit-Execute Procedure for ANS COBOL

```
//RPG          EXEC     PGM=IESRPG,REGION=52K,PARM='NODECK,LOAD,LIST'
//SYSPRINT     DD       SYSOUT=A
//SYSPUNCH     DD       SYSOUT=B
//SYSUT3       DD       DSNAME=&SYSUT3,UNIT=SYSDA,SPACE=(600,(100,20))
//SYSUT2       DD       DSNAME=&SYSUT2,UNIT=SYSDA,SPACE=(600,(100,20))
//SYSUT1       DD       DSNAME=&SYSUT1,SPACE=(600,(100,20)),         X
//                      UNIT=(SYSDA,SEP=(SYSUT2,SYSUT3))
//SYSGO        DD       DSNAME=&LOADSET,UNIT=SYSDA,DISP=(MOD,PASS),  X
//                      SPACE=(80,(200,50))
//LKED         EXEC     PGM=IEWL,REGION=96K,PARM='XREF,LIST,LET',    X
//                      COND=(9,LT,RPG)
//SYSPRINT     DD       SYSOUT=A
//SYSLMOD      DD       DSNAME=&GOSET(RPG),UNIT=SYSDA,               X
//                      DISP=(,PASS,DELETE),SPACE=(1024,(50,20,1))
//SYSLIN       DD       DSNAME=&LOADSET,DISP=(OLD,DELETE)
//             DD       DDNAME=SYSIN
//SYSUT1       DD       DSNAME=&SYSUT4,SPACE=(1024,(50,20)),         X
//                      UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN))
//GO           EXEC     PGM=*.LKED.SYSLMOD,COND=((9,LT,RPG),(5,LT,LKED))
```

RPG - Compile-Link Edit-Execute Procedure for RPG

The initiator procedure for job classes F (primary) and G (secondary) is for FORTRAN H and PL/I jobs. The procedure allocates the necessary data sets for both FORTRAN (SYS1.FORTLIB) and PL/I (SYS1.PL1LIB). By allocating these data sets at initiation time (with the SHR parameter), the procedure saves the time required for a catalog search for these data sets during job execution. This time saving can be significant when running many short, fast compile-link edit-execute jobs. The procedure is named IFP and may be stored under the alias IPF.

```
//IFP         PROC     CLS=A
//IEFPROC     EXEC     PGM=IEFIIC,PARM='FG,LIMIT=12"
//SYSABEND    DD       SYSOUT=&CLS|SPACE=(TRK,(1,10))
//SYSLIBF     DD       DSNAME=SYS1.FORTLIB,DISP=(SHR|KEEP)
//SYSLIBP     DD       DSNAME=SYS1.PL1LIB,DISP=(SHR,KEEP)
//LOADSET     DD       UNIT=SYSDA,SPACE=(400,(200,50))
//SYSUT1      DD       UNIT=SYSDA,SPACE=(1024,(200,20)),          X
//                     SEP=(SYSLIBP,LOADSET)
//SYSUT3      DD       UNIT=SYSDA,SPACE=(80,(250,250)),           X
//                     SEP=SYSUT1
```

IFP - Initiator Procedure for FORTRAN and PL/1

The following compile-link edit-execute procedures for FORTRAN H and PL/I will execute correctly only when they are run under an initiator which has the library data sets allocated to it. Otherwise, the load step will fail since a new temporary data set will be allocated to the SYSLIB DD card for DSNAME= & SYSLIBF or DSNAME= & SYSLIBP and the library data set will not exist. In the GO steps for both procedures SYSOUT data is blocked. The caller of the procedure can override the specified blocksize by specifying his own LRECL; blocksize will then be adjusted at OPEN time to the next lowest multiple or LRECL. This will occur with direct access SYSOUT data sets in BSAM, or with fixed length QSAM records. For ease of programming, the procedures are named the same as the IBM-supplied procedures, but are actually modified to handle the dedicated data sets.

```
//FORT        EXEC     PGM=IEKAA00,REGION=228K
//SYSPRINT    DD       SYSOUT=A
//SYSPUNCH    DD       SYSOUT=B
//SYSLIN      DD       DSNAME=&LOADSET,UNIT=SYSDA,DISP=(MOD,PASS),  X
//                     SPACE=(400,(200,50),RLSE)
//LKED        EXEC     PGM=IEWL,REGION=96K,PARM=(MAP,LET,LIST),     X
//                     COND=(4,LT,FORT)
//SYSLIB      DD       DSNAME=&SYSLIBF,DISP=SHR
//SYSPRINT    DD       DSNAME=&SYSOUT=A
//SYSLMOD     DD       DSNAME=&GOSET(MAIN),UNIT=SYSDA,DISP=(,PASS), X
//                     SPACE=(3072|(30,10,1),RLSE)
//SYSUT1      DD       UNIT=SYSDA,DSNAME=&SYSUT1,SEP=SYSLMOD        X
//                     SPACE=(1024,(200,20))
//SYSLIN      DD       DSNAME=&LOADSET,DISP=(OLD,DELETE)
//            DD       DDNAME=SYSIN
//GO          EXEC     PGM=*.LKED.SYSLMOD|COND=((4,LT,FORT)(4,LT,LKED))
//FT05F001    DD       DDNAME=SYSIN
//FT06F001    DD       SYSOUT=A,DCB=(BLKSIZE=2660,LRECL=133)
//FT07F001    DD       SYSOUT=B,DCB=(BLKSIZE=2600,LRECL=80)
```

FORTHCLG - Compile-Link Edit-Execute Procedure for FORTRAN H

```
//PL1L        EXEC     PGM=IEMAA,REGION=52K,PARM='LOAD,NODECK'
//SYSPRINT    DD       SYSOUT=A
//SYSLIN      DD       DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,  X
//                     SPACE=(80,(250,100))
//SYSUT1      DD       DSNAME=&SYSUT1,UNIT=SYSDA,SEP=SYSLIN,        X
//                     SPACE=(1024,(60,60),,CONTIG),DCB=BLKSIZE=1024
//SYSUT3      DD       DSNAME=&SYSUT3,UNIT=SYSDA,SEP=SYSUT1,        X
//                     SPACE=(80,(250,250)),DCB=BLKSIZE=80
//LKED        EXEC     PGM=IEWL,REGION=96K,PARM='XREF,LIST',        X
//                     COND=(9,LT,PL1L)
//SYSLIB      DD       DSNAME=&SYSLIBP,DISP=SHR
//SYSUT1      DD       DSNAME=&SYSUT1,UNIT=SYSDA,SEP=SYSLIB         X
//                     SPACE=(1024,(200,20)),DCB=BLKSIZE=1024
//SYSLMOD     DD       DSNAME=&GOSET(GO),UNIT=SYSDA|DISP=(MOD,PASS),X
//                     SPACE=(1024,(50,20,1),RLSE),SEP=SYSUT1
//SYSPRINT    DD       SYSOUT=A
//SYSLIN      DD       DSNAME=&LOADSET,DISP=(OLD,DELETE)
//            DD       DDNAME=SYSIN
//GO          EXEC     PGM=*.LKED.SYSLMOD,COND=((9,LT,PL1L),        X
//                     (9,LT,LKED))
//SYSPRINT    DD       SYSOUT=A,DCB=(BLKSIZE=2660,LRECL=133)
```

PL1LFCLG - Compile-Link Edit-Execute Procedure for PL/I

Using the Link Pack Area (MVT)

The link pack area is always present in main storage, and, as a minimum, always contains a group of system-specified load modules concerned with job management processing. The link pack area can be extended to contain:

- Load modules of nonresident SVC routines
- Load modules of nonresident error recovery procedures
- Other reenterable load modules from the system linkage library (SYS1.LINKLIB) and SVC library (SYS1.SVCLIB)
- (SYS1.LINKLIB) and SVC library (SYS1.SVCLIB)
- A table (the BLDL table) containing directory entries of load modules in the linkage library (SYS1.LINKLIB) and SVC library (SYS1.SVCLIB)

Essentially, the link pack area in MVT configurations is the counterpart of the MFT configuration residency options. If the system includes IBM 2361 Core Storage and main storage hierarchy support, a secondary link pack area can be created in hierarchy 1 that will contain nonresident SVC load modules and/or other reenterable load modules from the SVC library.

Select the load modules to be made resident and the linkage library load modules whose directory entries are to appear in the BLDL table. Indicate your choices to the system through lists of the load module names placed in the system parameter library (SYS1.PARMLIB). Standard (default) and alternative lists may be made up for each category.

During the initial program loading (IPL) process the nucleus initialization program places the specified load modules in the link pack area and constructs the BLDL table. The load modules and BLDL table remain, unchanged, in the link pack area until the next IPL procedure is performed. The resident access method routines, the resident SVC routines, and the resident BLDL table entries can be used by all tasks; the resident error recovery procedures are used by the I/O supervisor.

Procedure for Using the Link Pack Area

The following material provides guidelines for the use of the link pack area.

Initialization

When the system is generated, indicate whether to extend the link pack area to include nonresident SVC routines, nonresident error recovery proceudres, other reenterable load modules, the BLDL table, or any combination of these. The **System Generation** publication describes the procedure (the SUPRVSOR macro instruction).

To exercise full control over the content of the link pack area (except for the mandatory modules which are always loaded) include the operator communication facility at system generation. The **System Generation** publication describes the procedure (the SUPRVSOR macro instruction). The operator communication facility enables you to respecify the content of the link pack area at each IPL.

Creating Parameter Library Lists

The IEBUPDTE utility program places load module name lists in the parameter library. To avoid the duplicate loading into either the link pack area or dynamic main storage of modules

already resident in the link pack area, the ADD utility control statement must show all the ALIAS names of the load module being placed in the link pack area.

Updating the system data set SYS1.PARMLIB should not be attempted while other jobs are operative. The recommended procedure is described in the **Operator's Reference** publication.

The nucleus initialization program (NIP) will search the system catalog to locate the SYS1.PARMLIB data set. If it is not found in the catalog, SYS1.PARMLIB is assumed to reside on the IPL volume. If no VTOC entry can be found, the operator will receive message IEA211I "OBTAIN FAILED FOR SYS1.PARMLIB DATA SET". Message IEA208I "fff FUNCTION INOPERATIVE" will follow either of these messages. The fff parameter -- RAM, BLDL, RSVC, or RERP -- shows which of the functions cannot be implemented. Processing will continue; however, any resident functions dependent on parameter lists contained in the parameter library will be omitted from the system nuclues.

Except for LNKLST00, the input format (to IEBUPDTE) for the lists is the same for all three options, consisting of library identification followed by the load module names. Use eighty character records, with the initial or only record containing the library idenitification. Continuation is indicated by placing a comma after the last name in a record and a nonblank character in column 72. Subsequent records must start in column 16.

The initial record format, with continuation, is:

```
1                                                          72
             SYS.LINKLIB
[b...]       SYS1.SVCLIB    b...name1, name2,name3,...x
```

The topic "The Link Library List" explains the input format for LNNKLST00.

List Specification

The names and content of the parameter library lists are:

List Name	List Content
IEARSV00 – standard list IEARSVxx[1] – alternative list(s)	Names of type 3 and 4 SVC routine load modules.
IEAIGE00 – standard list IEAIGExx[1] – alternate list	Names of error recovery procedure load modules specified after system generation via IEBUPDTE
IEAIGG00 – standard list IEAIGGxx[1] – alternative list(s)	Names of reenterable load modules in the SVC and Link libraries.
IEABLD00 – standard list IEABLDxx[1] – alternative list(s)	Names of Link library load modules whose directory entries are to be entered in the BLDL table.
LNKLST00 – standard list	SYS1.LINKLIB – Additional data sets may be concatenated after system generation via IEBUPDTE.

[1]xx can be any two alphameric characters.

SVC Load Module Lists: Only one standard SVC load module list -- IEARSV00 -- may be present in the parameter library. As many alternative lists can be created as needed. To use alternative lists, specify the operator communication facility at system generation. The standard list is the only list referred to by the nucleus initialization program at IPL time if the operator communication facility is not installed in the system. A suggested standard list, supplied by IBM, is shown under the resident SVC routines option description in this section. The **Storage Estimates** publication provides a list (with storage requirements) of IBM originated type 3 and 4 SVC load modules that are eligible for inclusion in the link pack area.

Error Recovery Procedure Load Module Lists: Only one standard list of error recovery procedures -- IEAIGE00 -- may be present in the parameter library. As many alternative lists can be created as needed. The standard and/or alternative lists is used as discussed under SVC Load Module Lists. The standard list supplied by IBM has no error recovery procedure entries; so you must supply these via IEBUPDTE after system generation. The **Storage Estimates** publication provides a list (with storage requirements) of IBM error recovery procedures that are eligible for inclusion in the link pack area.

Reenterable Load Module Lists: Only one standard list of reenterable load modules -- IEAIGG00 -- may be present in the parameter library. As many alternative lists can be created as needed. Load modules from the SVC library and the linkage library cannot be incorported in one list. Use of the standard and/or alternative lists is as discussed under SVC LOAD MODULE LISTS. A suggested standard list, supplied by IBM, is shown under the resident access method modules option description in section 1 of this chapter. The **Storage Estimates** publication provides a list (with storage requirements) of IBM originated reenterable load modules (other than SVC modules) that are eligible for inclusion in the link pack area.

BLDL Table Lists: Only one standard list of SVC library or linkage library modules -- IEABLD00 -- may be present in the parameter library. You may create as many alternative lists as your needs require. Use of the standard and/or alternate lists is as discussed under the topic SVC Load Module Lists. An initial list for the linkage library is shown under the BLDL table description in this section. **For the linkage library, 56 bytes per entry are required. For the SVC library, 32 bytes per entry are required. To determine the storage requirements for a list, multiply the number of modules in the list by the length of an entry.**

Note: Library load modules names must be arranged in lists(s) in the same order as they appear in the library directory. All load modules in the linkage or SVC libraries are eligible to have their directory entries placed in the BLDL table.

Operational Characteristics

Specifications at system generation time determine the types of load modules that are placed in the link pack area and whether a BLDL table is constructed in the link pack area. In response to specifications, the nucleus initialization program (at IPL time) refers to the parameter library lists to determine the specific load modules to be placed in the link pack area and/or the specific library directory entries to be placed in the BLDL tables. In the absence of the operator communication facility only the standard lists are referred to. If the operator communication facility is present the operator must specify the list or lists to be used. The operator may:

- Specify use of the standard list for each category, i.e., SVC load modules, other reenterable load modules, the BLDL table content.
- Specify alternative lists for each category, or a combination of the standard list and alternative lists. Up to four lists may be specified for each load module category.

In MVT, two lists may be specified; one for SYS1.SVCLIB and one for SYS1.LINLKIB.

- Specify that (for the current IPL) the loading of modules and/or construction of a BLDL table be suppressed. Each category is treated independently.

At each IPL, operator communication can specify the content of the link pack area extension. The number and type of load modules selected for inclusion in the link pack area, and the content of the BLDL table, can thus be altered to reflect the type of workload to be presented to the system after the IPL. If the system includes 2361 Core Storage and main storage hierarchy support, a secondary link pack area for hierarchy 1 may be created and its contents specified at this time.

The **Messages and Code** publication describes the operator message and responses associated with use of the link pack area.

The Resident BLDL Table Option

When the system issues ATTACH, LINK, LOAD, or XCTL macro instructions requesting load modules from partitioned data sets, the BLDL table operation searches the data set directory for the location of the requested module and fetches the module. The resident BLDL table option eliminates the directory search required during execution of these macro instructions when a load module (whose directory entry is resident) is requested from the linkage or SVC libraries.

This option builds lists of directory entries for use by ATTACH, LINK, LOAD, or XCTL macro instructions when they request linkage or SVC library load modules. The BLDL operation in the macro instruction routines searches the library directory only when the directory entry for the requested load module is not present in the resident BLDL table.

List, in a member of SYS1.PARMLIB, the names either of linkage or SVC library load modules whose directory entries are to be made resident. The member name for the standard list is IEABLD00. The load module names must be listed in the same order as they appear in the directory; that is, they must be in ascending collating sequence. Creation of parameter library lists is discussed later in this chapter. The next section provides guidelines for choosing the content of the list.

Note: Directory entries in the resident table are not updated as a result of updating the load module in the library. The old version of the load module is used until an IPL operation takes place and the new directory entry for the module is made resident.

Selecting Entries for the Resident BLDL Table

Any load module in the linkage or SVC library may have its directory entry placed in the resident BLDL table. Other items to consider are:

1. Table Size.
   Linkage library - Each entry requires 56 bytes.
   SVC library - Each entry requires 32 bytes.
2. Frequency of use of the load module.

Table Size: The resident BLDL table is incorporated in the link pack area. The additional storage required is governed by the number of table entries and is acquired by reducing the amount of dynamic storage area available; therefore, the system nucleus expands. Each installation using the resident BLDL table option must determine the amount of storage it can afford for the resident BLDL table.

Frequency of Use: Since resident routines reduce the amount of main storage available to problem programs, select modules used frequently. Your installation's workload should be considered.

For Link Library Lists: The scheduler, linkage editor, and language processor(s) are possible selections for link library lists.

For SVC Library Lists: In general, use any module from the SVC library you would consider for residence (RAM option). Do **not** create libraries for the following since they are not necessary:

- Load 1 of type 3 and 4 SVCs (i.e. IGC00XXX)
- Modules selected for RAM, RERP, RSVC usage

Recommended modules should be chosen from access methods and ERPs. **Always avoid** placing the following modules in the BLDL list because they have internal BLDL tables and internal directory entries: OPEN, CLOSE, TCLOSE, EOV, SCRATCH, ALLOCATE, IEHATLAS, SETPRT, STOW, machine-check handler modules.

The SVC library list can be put in SYS1.PARMLIB using the member name IEABLDnn. This nn will be picked up when the operator specifies the system parameters with the response BLDL=xx,nn.

List IEABLD00

The IBM-supplied standard list IEABLD00 is:

```
SYS1.LINKLIB   IEBCOMPR,IEBGENER,IEBPTPCH,IEBUPDTE,IEHLIST,IEHMOVE,      X
               IEHPROGRM,LINKEDIT,SORT
```

Suggested Starter List for MVT

The following SVC library list includes selected modules for BDAM, BPAM, RLSE, CATALOG, and OPEN.

```
SYS1.SVCLIB    IGG0CLC1,IGG0CLC2,IGG0CLC3,IGG0CLC4,IGG0CLC5,IGG0CLC6,    X
               IGG0CLC7,                                                 X
               IGG019AV,                                                 X
               IGG019BH,IGG019BI,IGG019BK,IGG019BM,                      X
               IGG019CG,IGG019C3,                                        X
               IGG019KA,IGG019KE,IGG019KK,IGG019KQ,IGG019KU,IGG019LI,    X
               IGG020D1,IGG020P1,IGG020P2,IGG020P3
```

Suggested Starter List for Time Sharing

The following list is recommended for improved system accesses to SVCLIB with the time sharing option. It includes the starter list for MVT plus modules of SVC 99.

```
SYS1.SVCLIB    IGC0109I,IGC0209I,IGC0309I,IGC0409I,IGC0509I,IGC0609I,   X
               IGC0709I,IGC0809I,IGC0909I,IGC1009I,IGC1109I,IGC1209I,   X
               IGC1309I,                                                X
               IGC1409I,IGC1509I,IGC1609I,IGC1709I,IGC1809I,IGC1909I,   X
               IGC2009I,IGC2109I,IGC2309I,IGC2509I,IGC2609I,            X
               IGC2709I,IGC2809I,IGC2909I,IGC3509I,                     X
               IGG0CLC1,IGG0CLC2,IGG0CLC3,IGG0CLC4,IGG0CLC5,IGG0CLC6,   X
               IGG0CLC7,                                                X
               IGG019AV,                                                X
               IGG019BH,IGG019BI,IGG019BK,IGG019BM,                     X
               IGG019CG,IGG019C3,                                       X
               IGG019KA,IGG019KE,IGG019KK,IGG019KQ,IGG019KU,IGG019LI,   X
               IGG019TX,IGG019T4,IGG019T8,                              X
               IGG020D1,IGG020P1,IGG020P2,IGG020P3
```

SEC IV

The Resident Access Method Modules Option

This option places access method load modules in the link pack area, and creates a resident list of these modules. If the system includes IBM 2361 Core Storage and main storage hierarchy support, modules may also be placed in the secondary link pack area in hierarchy 1 using the "HRAM=" reply to "SPECIFY SYSTEM PARAMETERS." A LOAD macro instruction requesting any access method module first scans the resident list. If the module is listed, no fetch operation is required.

List, in a member of SYS1.PARMLIB, the load module names of access method load modules to be made resident. The member name ofr the standard list is IEAIGG00. A standard list of most frequently used access method modules is supplied by IBM, and is in SYS1.PARMLIB of the starter system under the standard member name. The content of the list is tabulated at the end of this description.

The creation of parameter library lists is discussed in this section. The following paragraphs discuss some considerations pertaining to the use of the access method option.

Considerations for Use

The storage space required for each access method module consists of the byte requirements of the module and its associated load request block (LRB). The **Storage Estimates** publication provides the byte requirements for access method modules eligible to be made resident. The byte requirement of the code supporting the option is also provided.

All access method modules placed in the system nucleus are "only loadable". ATTACH, LINK, and XCTL macro instructions cannot refer to the resident modules.

Alter the standard access method list (or create alternative lists) to include access method modules supporting program controlled interrupt scheduling (PCI), exchange buffering, track overflow, and the UPDAT function of the OPEN macro instruction.

For example, if checkpoint/restart is used, the following access method routines must be main storage resident, whether the checkpoint data set is on tape or on DASD (direct access storage device):

IGG019BA, IGG019BB, IGG019CC

If the checkpoint data set is on DASD these additional modules must be resident:

IGG019CD, IGG019CH, IGG019BC

If chained scheduling is used to write the checkpoint data set,

IGG019CU and IGG019CW

also must be resident. If the data set is on DASD and chained scheduling is used,

IGG019CV and IGG019CZ

must be resident together with the earlier two routines. If track overflow is used to write the data set,

IGG019C1, IGG019C2, and IGG019C3

must be resident.

When a composite console is used, an alternative list should include BSAM modules for card readers and printers.

If you specify either the 3330 of the 2305 I/O devices in your system, add the following modules to the standaed RAM list (IEAIGG00):

IGG019C4, IGG019FN, IGG019FP, and IGG019EK

IGG019C4 must also be resident and is on the standard RAM list.

When using use the SAM "search direct" option, it is advisable to make IGG019FN, IGG019FP, and IGG019C4 resident through the sta standard list. Performance is improved and required region size is decreased if these modules are resident.

To be eligible for use with the resident access method option, access method load modules must be reenterable. The module name must be of the form IGG019xx, where xx can be any two alphanumeric characters.

List IEAIGG00

The content of the IBM-supplied standard list IEAIGG00 is:

Module Name	Access Method	Function
IGG019AV	QSAM(SB)	PUT Locate for Dummy Data Set
IGG019AN	QSAM(SB)	Backward Move - Format F, FB, U Records
IGG019AM	QSAM(SB)	Backward Locate - Format F, FB, U Records
IGG019BE	BSAM	Magnetic Tape Forward Space or Backspace
IGG019AG	QSAM(SB)	GET Move with CNTL - Format V Records (Card Reader)
IGG019CB	SAM	Space or Skip Printer
IGG019CA	SAM	Stacker Select (Card Reader)
IGG019AK	QSAM(SB)	PUT Move, Format F, FB, U Records
IGG019AJ	QSAM(SB)	PUT Locate, Format V, VB Records
IGG019AI	QSAM(SB)	PUT Locate, Format F, FB, U Records
IGG019AC	QSAM(SB)	GET Move, Format F, FB, U Records
IGG019AB	QSAM(SB)	GET Locate, Format V, VB Records
IGG019AA	QSAM(SB)	GET Locate, Format F, FB, U Records
IGG019AR	QSAM(SB)	PUT Synchronization Routine
IGG019AQ	QSAM(SB)	GET Synchronization Routine
IGG019AL	QSAM(SB)	PUT Move, Format V, VB Records
IGG019AD	QSAM(SB)	GET Move, Format V, VB Records
IGG019BD	BSAM	NOTE/POINT Tape
IGG019BC	BSAM	NOTE/POINT Disk
IGG019BB	BSAM	CHECK (all devices)
IGG019BA	BSAM	READ/WRITE (all devices)
IGG019CK	SAM	SYSIN Delimiter Check (Appendage)
IGG019CJ	SAM	Read Length Check, Format V Records (Appendage)
IGG019CI	SAM	Length Check, Format FB Records (Appendage)
IGG019CH	SAM	End-of-Extent Check (Data Extent Block) (Appendage)
IGG019CL	SAM	Printer Test Channels 9,12 (Appendage)
IGG019CF	SAM	ASA Character to Command Code (Printer-Punch)
IGG019CE	SAM	End-of-Block (Printer-Punch)
IGG019CD	SAM	Schedules I/O for Direct Access Output
IGG019CC	SAM	Schedules I/O for Tape, Direct Access Input, Card Reader, Paper Tape Reader
IGG019C0	SAM	Channel end (Format U).
IGG019C4	SAM	Search Direct (SD) or Rotational Position Sensing. (RPS) Fixed Standard End-of-Extent Appendage.
IGG019FN	SAM	Checks RPS values (P0). Start I/O for Search Direct (SD).
IGG019FP	SAM	Channel end appendage for Search Direct (SD).

SB=simple buffering
SAM=common sequential access method routines

Note: If the system generation statements specify the use of both MCS and of an IBM 2740 Communications Terminal as an operator's console, the RAM option list (module IEAIGG00) is effectively extended by the following character constants in the nucleus initiation program module IEAANIP:

```
DC  C'IGG019MA'     BTAM read/write module
DC  C'IGG019MB'     BTAM Appendage
DC  C'IGG019M0'     BTAM 2740 module
```

The effect of these DCs is that the named modules are loaded whether or not the RAM option is specified in the system generation statements. In MVT, the modules are always loaded into the link pack area.

The Resident SVC Routines Option

This option places any of the type 3 and 4 SVC routine load modules in the link pack area. If the system includes IBM 2361 Core Storage and main storage hierarchy support, modules may also be placed in the secondary link pack area in hierarchy 1 using the "HSVC=" reply to "SPECIFY SYSTEM PARAMETERS." Some, or all, of the modules associated with a SVC service routine may be made resident. Placing the most frequently used SVC load modules of a system service routine, such as OPEN, in main storage improves system performance. For type 3 SVC load modules and initial type 4 SVC load modules, the SVC table entries associated with these modules are adjusted to reflect an entry point address rather than a relative track address. A resident SVC load list is used by the XCTL macro instruction for transfer of control between resident type 4 SVC load modules.

List in a member of SYS1.PARMLIB, the type 3 and 4 SVC load modules to be made resident. The member name for the standard list is IEARSV00. Such a standard list (shown below) is provided by IBM in SYS1.PARMLIB of the starter system. The creation of parameter library lists is discussed later in this chapter.

If the system includes the multiple console support (MCS) function, to improve MCS performance add to the standard list (or include in a list of your own) IGC0007B, the name of the first load module of the SVC 72 routine.

The **Storage Estimates** publication provides the byte requirements of type 3 and 4 SVC routines eligible to be made resident. The byte requirement of the code supporting the option is also provided.

List IEARSV00

The content of the IBM-supplied standard list IEARSV00 is:

Module Name	Function
IEG0193A	Open - Volume Serial Function
IEG0194E	Open - Unit Selection and DSCB Read
IEG0195A	Open - DSCB and JFCB Merge
IFG0195J	Open - DSCB to JFCB Merge
IFG0196J	Open - JFCB to DCB Merge
IFG0196L	Open - Merge and DCB Exit Routine
IFG0196M	Open - Merge DCB to JFCB
IFG0196V	Open - Access Method Determination
IFG0196W	Open - Access Method Executor
IFG0198N	Open - Rewrite JFCB and Final Load
IFG0200V	Close - Initialization and Read JFCB and DSCB
IFG0200W	Close - Access Method Interface
IFG0200Y	Close - Access Method Interface and Write JFCB
IFG0202E	Close - Write File Mark
IFG0202J	Close - Restore System Function
IFG0202K	Close - Restore User Function
IFG0202L	Close - Final Load
IFG0551B	EOV - Synad Executor
IGC0001F	Purge Routine
IGC0001I	Open - Initial Load
IGC0002_ *	Close - Initial Load
IGC0002B	Open/Close Type=J - Alternate Initial Load for Open
IGC0005E	EOV - Initial Load
IGG0191A	Open - DEB Construction (First Load)
IGG0191B	Open - Main Executor (First Load)
IGG0191D	Open - Direct Access Executor
IGG01910	Open - Load Executor (First Load)
IGG01911	Open - IOB and Buffer Construction
IGG01917	Open - Load Executor (Second Load)
IGG0196A	Open - DEB Construction (Second Load)
IGG0196B	Open - Main Executor (Second Load)
IGG0201Y	Close - Release Work Areas and Buffers
IGG0201Z	Close - SAM Executor

*The last (eighth) character is a 12 and 0 punch. This character has no assigned graphic in EBCDIC. In BCD, the graphic is ? (the question mark).

The Resident Error Recovery Procedure Option

This option places error recovery procedures in main storage. Some, or all, of the modules associated with the handling of an I/O error may be made resident. If an I/O device frequently requires ERP processing, system performance improves if the error recovery procedures are made resident. The list of those error recovery procedures that may be made resident in main storage is contained in the **Storage Estimates** publication. An I/O supervisor request for an error recovery procedure will result in a search of the resident error recovery procedure list. If the error recovery procedure is resident, no fetch operation is required.

List in a member of SYS1.PARMLIB, the module names of error recovery procedures to be made resident. The member name for the standard list is IEAIGE00. After system generation, there remains the option of indicating which error recovery procedures are to be made resident. The error recovery procedures should be listed by expected frequency of use; the least used module is first in the list. **Note**: The format of the IBM-supplied IEAIGE00 list contains the required library name, SYS1.SVCLIB, and **no** error recovery procedure names. After system generation, IEAIGE00 can be updated to indicate which error recovery procedures are to be made resident or an alternate list can be created. Until this update is performed, no error recovery procedures will be made resident during the IPL process. The creation of parameter library lists is discussed later in this chapter.

The **Storage Estimates** publication provides the byte requirements of error recovery procedures that may be made resident. The byte requirement of the code supporting the option is also provided.

Programming Notes

A list of the load modules **always** placed in the link pack area by the system is contained in the **Storage Estimates** publication. The main storage space requirements of these modules determines the basic (minimum) size of the link pack area. The area is extended by the number of storage bytes needed to accommodate the load modules and BLDL table content specified at IPL time.

Placing the initiator/terminater load module IEFSD061 in the link pack area enables the system to make more efficient use of the dynamic area of storage. The operating system allocates to each job a part of a region not less than the size required to accommodate the initiator-terminator. This allocation is from processor storage (hierarchy 0) and occurs even when the REGION parameter requests less than the required space or no space. After initiation, the part of the region in heirarchy 0 is reduced by as many as 40,000 bytes when the job terminator is resident in the link pack area.

Example of Link Pack Area Specification

The following example illustrates the extension of the link pack area to contain SVC load modules, other reenterable load modules, and a BLDL table. The RESIDNT field of the system generation SUPRVSOR macro instruction would look like:

```
          .
          .
          .
SUPRVSOR  RESIDNT=TRSVC,RENTCODE,BLDLTAB...
          .
          .
          .
```

When altering the content of the link pack area, specify: OPTIONS=COMM,... in the SUPRVSOR macro instruction.

Five possibles lists that can be placed on SYS1.PARMLIB are:

1. IEARSV00, which contains names of modules of the Open SVC routine used for direct access devices.
2. IEARSV20, which contains names of modules of the Close SVC routine.
3. IEAIGG01, which contains names of modules of the basic sequential access method (BSAM).

4. IEABLD00, which contains names of modules of the initiator portion of the job scheduler.
5. IEABLDF0, which contains names of modules of both the FORTRAN compiler and the initiator.

Note that there is no standard list for reenterable modules from the linkage or SVC library (IEAIGG00). This implies that you don't want modules of this type loaded unless a list is explicitly specified.

To place these lists in SYS1.PARMLIB, use the IEBUPDTE utility program as shown:

```
//ADDLISTS     JOB  61938,R.L.WILSON
//STEP         EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT     DD   SYSOUT=A
//SYSUT2       DD   DSNAME=SYS1.PARMLIB,DISP=OLD
//SYSIN        DD   DATA
./             ADD  NAME=IEARSV00,LIST=ALL
./             NUMBER NEW1=01,INCR=02
SYS1.SVCLIB    IGG0190I,IGG0190L,IGG0190M,                              C
                    IGG0190S,IGG0190Z
./             ADD  NAME=IEARSV20,LIST=ALL
./             NUMBER NEW1=01,INCR=02
SYS1.SVCLIB    IGC00020,IGG0200A,IGG0200B,IGG0200C,IGG0200F,            C
                    IGG0200G,IGG0200Y
./             ADD  NAME=IEAIGG01,LIST=ALL
./             NUMBER  NEW1=01,INCR=02
SYS1.SVCLIB    IGG019BA,IGG019BB,IGG019BC,IGG019BD,                     C
                    IGG019BE,IGG019BF,IGG019BG,                       C
                    IGG019BH,IGG019BI,IGG019BK,IGG019BL
./             ADD  NAME=IEABLD00,LIST=ALL
./             NUMBER    NEW1=01,INCR=02
SYS1.LINKLIB   IEFSD061,IEFSD062,IEFSD064,IEFSD104,                     C
                    IEFVM1,IEFWC000,IEFWD000,                         C
                    IEFW21SD,IEFW41SD,IEFW42SD,IEFXJ000
./             ADD  NAME=IEABLDF0,LIST=ALL
./             NUMBER  NEW1=01,INCR=02
SYS1.LINKLIB   IEFSD061,IEFSD062,IEFSD064,IEFSD104,                     C
                    IEFVM1,IEFWC000,IEFWD000,IEFW21SD,                C
                    IEFW41SD,IEFW42SD,IEFXJ000,IEJAAA0,               C
                    IEJEAA0,IEJFAA0,IEJGAA0,IEJJAA0,                  C
                    IEJLAA0,IEJNAA0,IEJPAA0,IEJRAA0,                  C
                    IEJVAA0,IEJXAA0,                                  C
                    IEFWD000,IEFW41SD                                 C
/              ENDUP
/*
```

Without operator communicaton only the standard lists IEARSV00 and IEABLD00 would be referred to at IPL time. With operator communication use of all the lists or any combination could be specified at IPL time.

To extensively use the FORTRAN compiler after a given IPL and BSAM with direct access devices, it is possible to use all of these lists -- except IEABLD00 -- to specify the content of the extended link pack area. To do this, the operator would specify the following in response to the SPECIFY SYSTEM PARAMETERS operator's message:

```
REPLY id, "RSVC=00,20,RAM=01,BLDL=F0"
```

To perform general processing without extensive use of any particular compiler or access method after an IPL it is possible to put just the linkage library directory entries of initiator modules in a BLDL table. In this case, the operator's reply at IPL would be:

```
REPLY id, "RSVC=,RAM=,"
```

Since the list of initiator modules is the standard list, it need not be specified. "RSVC=," must be specified to prevent the use of the standard list of SVC modules. Although there is no standard list of reenterable modules, "RAM=," should be specified to prevent NIP from performing unnecessary processing.

The Link Library List

The link library list (LNKLST00) enables concatenating up to 16 data sets, on multiple volumes, to form SYS1.LINKLIB. LNKLST00 is inlcuded in the system when it is generated as a required member of SYS1.PARMLIB. If SYS1.PARMLIB does not include the member LNKLST00, SYS1.LINKLIB will be used as the system link library and a warning message will be provided.

Note: The amount of space required for SYS1.PARMLIB is discussed in the **Storage Estimates** publications.

LNKLST00 contains one member, SYS1.LINKLIB. After system generation there is the option of adding members via the IEBUPDTE utility program. Each member may have up to 16 extents. After making additions to SYS1.SVCLIB, SYS1.LINKLIB, or data sets concatenated to LINKLIB via LNKLST00, and before using the additions, IPL should be performed to update the description of the link and/or SVC library in main storage.

The input format (to IEBUPDTE) consists of eighty-character records. Continuation is indicated by placing a comma after the last name in a record and a nonblank character in column 72. Subsequent records must start in column 16. The initial format is:

```
[b...] SYS1.LINKLIB
```

To add member names to LNKLST00, replace the initial record with:

```
[b...] SYS1.LINKLIB, name1,name2,name3,...
```

The **Messages and Codes** publication describes the NIP messages associated with LNKLST00.

Job Queue Format

The job queue format is specified when the system is generated and may be altered during subsequent system start procedures. Formatting consists of specifying the number of queue records in a job queue logical track, reserving queue records for initiators, the write-to-programmer routine, and reader/interpreters, and reserving queue records for job cancellation.

The basic element of the system job queue (the data set SYS1.SYSJOBQE) is a 176-byte record -- the queue record. The total number of queue records available is fixed by the space allocated to the SYS1.SYSJOBQE data set. Queue records contain the tables, control blocks, and system messages developed by the reader/interpreter, write-to-programmer, and initiator control program routines -- the information used to run a job.

Lack of queue records to work with is not critical for a reader/interpreter routine. Processing of the input job stream assigned to a reader/interpreter is suspended until queue records become available, at which time processing is resumed. **An initiator, however, must have sufficient queue records available to complete the initiation and running of a job or the job is canceled.** Because one or more reader/interpreter and one or more initiators may be concurrently active, steps must be taken to ensure that queue records are available to each initiator started, so that it may complete its operations. In addition queue records must be reserved for use by initiators in the event job cancellation does take place. The main function of job queue formatting is to reserve queue records for initiator use.

To format the job queue:

1. Designate the number of queue records to be contained in a job queue **logical track**. A logical track consists of a header record (20 bytes) plus the designated number of queue records. Reader/interpreters and initiators are assigned queue records in terms of logical tracks.

2. Designate the number of queue records to be reserved for use by an initiator. Each initiator is allocated this number of records. If the allocation is insufficient for the job currently being processed by the initiator, the job is canceled in MVT.

3. Designate the number of queue records to be reserved for use in case of job cancellation. All initiators that cancel use these queue records. If the allocation is insufficient, the initiator is placed in a WAIT state and a message issued.

4. Designate the number of queue records to be reserved for write-to-programmer routine use for each job that may be started by an initiator.

The balance of the queue (total queue records less the reservations in items 2, 3, and 4) is available for use by the reader/interpreters.

Specify initial values for logical track size, queue record reservation for initators, and queue record reservation for job cancellation, in the SCHEDULR macro instruction parameters JOBQFMT, JOBQLMT, JOBQTMT, and JOBQWTP respectively. The **System Generation** publication describes the procedure.

The service aids program IMCJQDMP provides a formatted dump of the entire job queue, or selected portions of it. The formatted dump includes the master queue control record (QCR) which contains the physical parameters of the job queue. For a complete description of IMCJQDMP, see the publication **Service Aids**.

There are no comprehensive, foolproof formulas for calculating values of JOBQFMT, JOBQLMT, JOBQTMT, and JOBQWTP. The values to be estimated are dependent upon the requirements and structure of the jobs to be presented to the system, the number of job steps, the number of I/O devices required, the number and type of data sets, the number of volumes, and most upredictable, the number of system messages issued during the initiation and running of a job. The rest of this topic provides some basic guidelines for your use in determining these values.

Logical Track Size -- JOBQFMT

Logical track size -- the number of queue records in a logical track -- affects the efficient use of queue records. Reader/interpreters and initiators are allocated queue records in terms of logical tracks. Unused queue records in a logical track are **not available** for use by other reader/interpreters or initiators. Therefore, an over-generous logical track size specification results in wasted queue records and reduction of job queue capacity, i.e., the unused queue records, if available, could contain the required information for another job.

Logical track size affects performance to some extent. Specification of a logical track size of 10 queue records or less can result in excessive execution of the track assignment routines, etc., i.e., the "overhead" required to use very small logical track sizes impairs performance.

As a starting point, use the default value for JOBQFMT (12 queue records).

Logical track size (or multiples of it) may correspond to the physical track capacity of the device on which the job queue is resident. For example, if the IBM 2301 Drum Storage unit is to be used, 66 queue records may be contained in one physical track. Specify, in this case, a logical track size of 22 queue records, thereby allocating 3 logical tracks to one physical track (3 x 22 = 66 queue records). The 3 logical track header records (20 bytes each) use up the remaining record.

Logical tracks can contain the same number of queue records as are reserved for initiator use.

Reserving Initiator Queue Records -- JOBQLMT

The value specified for JOBQLMT must be large enough for the queue entries of any job that enters the system. The following list shows the factors that affect the value of JOBQLMT:

- Number of entire generation data groups in a job
- Number of passed data sets in a job
- Number of devices required for passed data sets
- Number of volumes containing the data sets in a step
- Number of system messages issued during initiation of a step
- Use of automatic restart

The sum of the queue records required for each of these items provides a JOBQLMT value.

When a START initiator command is issued, a check is made to see if enough free logical tracks are available to provide the required number of queue records for the initiator. If not, the command is rejected.

Each time an initiator is started, the number of records reserved for an intiator is added to the total number of records reserved for active initiators. For example, if the number of records reserved for each initiator is 60, the number of records reserved for termination is 40, and 4 initiators have been started, then the number of records reserved is 340. This total includes 60 records reserved for each initiator, 40 records reserved for termination, and 60 records reserved as a basic threshold.

Number of Generation Data Groups

Each entire generation data group (GDG) used during a job increases the number of queue records needed by an initiator. Two queue records should be reserved for every generation in excess of the first in a GDG. One queue record should be reserved for every four GDGs used in a job.

Thus, if a job uses two entire GDGs, one having 5 data sets (generations), and the other having 24 data sets, 55 queue records must be reserved -- (4+23)x2+1.

Number of Passed Data Sets

Two queue records are needed by an initiator for every three data sets passed during a job. If the number of data sets passed is not a multiple of three, queue records must be allocated as if the number of data sets passed was a multiple of three. Thus if one, two, or three data sets are passed, two queue records are allocated; if four, five, or six data sets are passed, 4 queue records are allocated, and so on.

Number of I/O Devices for Passed Data Sets

When a data set being passed requires more than ten I/O devices, one queue record is required by an initiator. This queue record accommodates 43 devices. If the number of required devices exceeds 53, a second queue record is needed. Separate calculations must be made for each data set.

Number of Volumes

An initiator requires queue records for each data set that occupies more than five volumes, and is located by a search of the catalog. (If a data set's location is specified in a DD statement, the reader routines acquire the necessary records.) One queue record is needed if the data set occupies between 6 and 20 volumes; two queue records if 21 to 35 volumes; three if 36 to 50 volumes; and so on. Separate calculations must be made for each data set.

Number of System Messages

An initiator requires queue records for system messages it issues. If you assume that each message is 80 characters in length, each queue record holds two messages. Messages from initiators are primarily device allocation, allocation recovery, data set disposition, SMF or accounting messages, and the keep messages for tapes used in each step.

To cover most device allocation messages, allow one queue record for every three DD statements. To cover data set disposition messages, allow one queue record for each DD statement. As part of the data set disposition messages, count the SMF or accounting messages as two lines per queue record. Also count two lines per queue record for tape messages.

Allocation recovery messages apply to devices that are offline. To cover most situations, allocate queue records as follows:

- Determine the largest number of devices of a given class that will be offline at any given time.
- Divide by seven.
- Add two.

Since this calculation is for a job step, multiply the result by the number of steps in a large job.

System messages are the least predictable of all the variables used in calculating initiator queue record needs. The number of messages depends on the number of devices offline, the number not available, and the number required at any given time.

The initiator needs queue space for a TIOT (task input/output table) for each step. The space needed can be approximated by:

- Determining the number of DD statements in the largest step in a job
- Multipling the number of DD statements by 20
- Adding 24
- Dividing by 172 and rounding the dividend up
- Adding 1

This gives the largest amount of queue records required for a job.

Under certain conditions, the initiator may need additional space. Two specific conditions are:

- VOLT (volume table) -- The initiator builds a VOLT, if one does not exist, for all non-specific device requests. One queue record will hold 28 volume serial numbers.
- Mount CVOL (control volume) -- Five records will be created for each CVOL not mounted. The initiator builds a JCT (job control table), a SCT (step control table), a SIOT (step input/output table), a JFCB (job file control block) and a VOLT if a CVOL is not mounted. The initiator writes these queue records into the jobqueue.

Use of Automatic Restart

To use automatic restart in the system, the number or records specified for the JOBQLMT parameter must be substantially increased. In general, this is due to the fact that, while the first job is going through the restart process, a second job is initiated, and that before the system can restart the first job, it must reread and reinterpret the job deck and then reinitiate the job. More specifically:

- The **initiator** needs its normal set of queue records (described by the JOBQLMT parameter) to initiate the job for the first time; it needs an additional set of records to start a second job while the first job is going through the restart process.
- Since the restart process involves rereading, reinterpreting, and reinitiating the first job, an additional set of **reader/interpreter** records is needed, together with a third set of **initiator** records.

Finally, when checkpoint restart is being performed, a set or two of restart housekeeping records are needed. Altogether, the number of records to be specified for JOBQLMT when automatic restart is being used is:

JOBQLMT + (3 x L) + R + (a x 12)

L - Number of records normally specified for JOBQLMT (that is, when automatic restart is not being used).

R - Number of records normally needed by the reader/interpreter. (See the **Storage Estimates** publication for guidance on how this number is established.)

a=1 - If jobs may be automatically restarted only once.

a=2 - If jobs may be automatically restarted more than once.

12 - Number of records needed for restart housekeeping.

If jobs with automatic restart may be held for operator restart, the initiator queue record requirement is further increased, because the system must keep both the queue records for the held jobs and their associated housekeeping records until the job is restarted. The formula then becomes:

JOBQLMT = (3 x L) + R + (a x 12) + H (L + (a x12))

H - Number of jobs that may be held.

Other terms

As explained previously.

Reserving Write-To-Programmer Queue Records - JOBQWTP

Unless specified otherwise, the system allocates two job queue records to the write-to-programmer (WTP) function. Out of the 176 bytes in each of these records, 161 are available for WTP messages. A record can hold as many messages as will fit into the available space, each message occupying 1 byte per character plus 1 byte per message for an initiator assigned serial number.

To change the number of records available for this function, specify the number either with the JOBQWTP operand of the SCHEDULR macro instruction in the system generation statements or during initialization in reply to message IEA101A (but only if you used Q-F with your set command). However, since both system and application tasks contend for the space available to an initiator in the system job queue, and since WTP message may be created faster than the writer may be writing them out, caution should be exercised in raising the JOBQWTP value above 2.

Reserving Queue Records for Cancellation -- JOBQTMT

If an intiator's queue record requirements exceed the number of queue records reserved for it, the job associated with that initiator is canceled. Queue records must be reserved for this purpose. Enough queue records must be reserved to accommodate two (or more) initiators that may be cancelling concurrently. The JOBQTMT value (like the value JOBQLMT) is unpredictable because of factors such as the installation's configuration, the size of the job being canceled, and the number of jobs that can be multiprogrammed.

The following guidelines should be used in calculating JOBQTMT:

- Number of devices used during a job.
- Number of jobs that might be concurrently canceled because of insufficient initiator queue records.
- For any system task to be started, combined JCL from its associated catalogued procedure and the START command must first be interpreted. This requires queue records, and the system allows assignment of records for this purpose whenever any logical track are available. During normal use of the queues, this space is always available. However, in order to insure availability of queue records for system tasks when the reserves approach the critical state, the value of JOBQTMT should be increased over the above amount by the number of records necessary to get tasks started. (This is especially true for writer and initiator tasks, since they return queue records to the system.) This amount may be estimated in a manner similar to calculating JOBQLMT, taking into consideration that each valid START command generates one input and one output queue entry. Formulas for estimating queue entry sizes are given in the **Storage Estimates** publications.

Number of Devices

The devices currently assigned to a job are released when the job is canceled. Since messages are issued when devices are released, you should reserve a number of queue records equal to the largest number of devices assigned at any one time to a job, multiplied by two. Thus if the largest job (in terms of devices) has three steps requiring 4, 11, and 8 devices respectively, 22 queue records should be reserved.

Number of Jobs

The number of queue records reserved for cancellation must be large enough to fill the requirements of all jobs being canceled at any one time because of insufficient initiator queue records. If your estimate of initiator queue records was accurate, it is unlikely that you will have more than one job (if any) cancelling at any one time.

An initiator that runs out of queue records for cancellation is placed in the wait state and an operator message -- IEF426I QUEUE CRITICAL -- is issued. This can result in the interlocking of all reader/interpreters, initiators, and sysout writers functioning at the moment.

Output Separation

The system output writer can use the output separator facility to write separation records prior to writing the output of each job. These separation records make it easy to identify and separate the various job outputs that are written contiguously on the same printer or card punch device.

Characteristics of an Output Separator

Both the system output writer and the direct SYSOUT (DSO) writer may be used by a problem program to channel its output eventually to a printer or punch. When this is done, however, the system output stream goes uninterruptedly from one job to another, making it difficult to separate the output of one job from that of another, unless output separation is provided for.

The output separator facility of the operating system provides a means of identifying and separating the output of various jobs processed by the same output unit. To do this, the separator writes separation records to the system output data set prior to the writing of each job's output.

The IBM output separator or your own output separator can be used.

The output separator function operates under control of both the system output writer and the direct SYSOUT writer. The separator program must reside in the link library (SYS1.LINKLIB). Its name, IEFSD094, must be included as a parameter in either of the output writer procedures -- the second part of the PARM field in the EXEC statement -- to separate job output. (Cataloged procedures for both writers are fully described in this section). The type of separation provided by the separator depends on whether the output is punch-destined or printer-destined.

Punch-Destined Output: The IBM-supplied separator provides three specially punched cards (deposited in stacker 1) prior to the punch card output of each job. Each of these separator cards is punched in the following format:

Columns 1 to 35 -- blanks
Columns 36 to 43 -- jobname
Columns 44 to 45 -- blanks
Column 46 -- output classname
Columns 47 to 80 -- blanks

Printer-destined Output: The IBM-supplied separator provides three specially printed pages prior to printing the output of each job. Each of these three separator pages is printed in the following format:

- Beginning at the channel 1 location (normally near the top of the page), the jobname is printed in block character format over 12 consecutive lines. The first block character of the 8-character jobname begins in column 11. Each block character is separated by 2 blank columns.
- The next 2 lines are blank.
- The output classname is printed in block character format covering the next 12 lines. This is a 1-character name, and the block character begins in column 55.

- The remaining lines to the bottom of the page are blank.

In addition to the above, a full line of asterisks(*) is printed twice (overprinted) across the folds of the paper. These lines are printed on the fold preceding each of the three separator pages, and on the fold following the third page. This feature provides easy separation of job output in a stack of printed pages.

For printer-destined output with the IBM-supplied separator, include a channel 9 punch in addition to the channel 1 punch on the carriage control tape or in the forms control buffer (FCB). The channel 9 punch controls the location of the line of asterisks and should correspond to the bottom of the page. To print the line of asterisks on the fold of the pages, offset the printer registration.

Programming Conventions

When using the (asynchronous) system output writer, the separator program, if specified in the output writer cataloged procedure, is brought in by a LINK macro instruction issued from module IEFSD078 of the output writer. The separator program can be any size, but a program over 8K may affect the region requirement of the output writer. If the job falls into a job call using the (synchronous) direct SYSOUT writer, the separator program (if specified in the procedure) is brought into main storage by use of a LOAD macro instruction. After performing separation on all devices required for the SYSOUT data sets in that step the program is released by means of a DELETE macro instruction.

Caution: Since the separator program operates with the supervisor protection key, but in the program mode, your separator program must insure data protection during its execution.

When writing a separator program, you must observe the following programming conventions:

- The program must conform to the standard linkage coventions. This includes saving and restoring the contents of registers 0 through 12, and 14. These registers can be preserved with the SAVE and RETURN macro instructions. When the program receives control, the address of a standard save area is in register 13.
- The program must use the PUT macro instruction in the locate mode to write separation records on the output data set. (This method is required by the QSAM DCB that is open for the output data set.)
- The program must establish its own synchronous error exit routine, and the address of this routine must be placed into the DCBSYNAD field of the output DCB. This gives control to the error exit routine in case an uncorrectable I/O error occurs while writing your program's output.
- The program should use the RETURN macro instruction to return control to the output writer. Before returning, the program must free any main storage it obtained during its operation, and the program must place a return code (binary) in register 15. The return codes signify:

  0 -- Successful operation.

  8 -- Unrecoverable output error (should be set if your error exit routine is entered).

Output From the Separator Program

The separator program can write any kind of separation identification. The jobname and the output classname for each job are available through the parameter list for inclusion in the output, if desired. You can use an IBM-supplied routine that constructs block characters (explained later). As many separator cards can be punched or as many separator pages can be printed as necessary.

The output from the separator program must conform to the attributes of the output data set. These attributes, which can be determined from the open output DCB pointed to by the parameter list, are:

- Record format (fixed, variable, or undefined length).
- Record length.
- Type of carriage control characters (machine, USASI, or none).

For printer-destined output, begin your separation records on the same page as the previous job output, or skip to any subsequent page. However, the separator program should skip at least one line before writing any records, because in some cases the printer is still positioned on the line last printed.

After completing the output of the separation records, the separator program should write sufficient blank records to force out the last separation record. This also allows the error exit routine to obtain control if an uncorrectable output error occurs while writing the last record. The requirements are:

- One blank record for printer-destined output.
- Three blank records for punch-destined output.

Using the Block Character Routine

For printer-destined output, the separator program can use an IBM-supplied routine to construct separation records in a block character format. This routine is a reenterable module named IEFSD095, and resides in the module library (SYS1.CI505).

The block character routine constructs block letters (A to Z), block numbers (0 to 9), and a blank. The program furnishes the desired character string and the construction area. The block characters are constructed one line position at a time. Each complete character is contained in 12 lines and 12 columns; therefore, a block character area consists of 144 print positions. For each position, the routine provides either a space or the character itself.

The routine spaces 2 columns between each block character in the string. However, the routine does not enter blanks between or within the block characters. The program must prepare the construction area with blanks or other desired background before entering the block character routine.

To use the IBM-supplied block character routine, the separator program executes the CALL macro instruction with the entry point name of IEFSD095. Since the block characters are constructed one line position at a time, complete construction of a block character string requires 12 entries to the routine. Each time, provide the address of a 4-word parameter list in register 1. The parameter list must contain the following:

Bytes 0-3 -- This word is the address of a field containing the desired character string in EBCDIC format.

Bytes 4-7 -- This word is the address of a full word field containing the line count as a binary integer from 1 to 12. This represents the line position to be constructed on this call.

Bytes 8-11 --This word is the address of a construction area in main storage where the routine will construct a line of the block character string. The required length in bytes of this construction area is 14n-2, where n represents the number of characters in the string.

Bytes 12-15 --This word is the address of a fullword field containing, in binary, the number of characters in the string.

Writing an Output Separator Program

Write the output separator program by using the information provided by either output writer and by conforming to the requirements explained below. The separator program, when added to the link library (SYS1.LINKLIB), is invoked by specifying its name as a parameter in the EXEC statement of the output writer cataloged procedure.

Parameter List

Either output writer provides the separator program with a 4-word parameter list of needed information. When the program receives control, register 1 contains the address of a 4-word parameter list, and the parameter list contains the following:

Bytes 0-3-- In this word, byte 0 contains switches that indicate the type of output unit, and bytes 1-3 are reserved for future use.

Bytes 4-7-- This word is the address of the output DCB (data control block).

Bytes 8-11-- This word is the address of an 8-character field containing the jobname.

Bytes 12-15--This word is the address of a 1-character field containing the output classname.

In the parameter list, the three high-order bits of byte 0 are switches that your separator program uses to determine the type of output unit. The first bit to the left is set to 1 if the output unit is a 1442 punch device. The second bit is set to 1 if the output unit is a punch device or a tape device with punch-destined output. The third bit is set to 1 if the output unit is a printer or punch device. The resulting bit combinations indicate the following:

111. .... 1442 punch device
011. .... 2520 or 2540 punch device
001. .... 1403, 1404, 1443, or 3211 printer device
010. .... tape device with punch-destined output
000. .... tape device with printer-destined output

The parameter list also points to the DCB for the output data set. This DCB is established for the queued sequential access method (QSAM), and is already open when the separator program receives control.

The address of the jobname and the address of the output classname are provided in the parameter list so that this information may be used in the separation records written by the separator program.

Writing System Output Writer Routines

When a job is executing, system messages and data sets specifying the SYSOUT parameter (e.g., in the DD statement) are recorded on direct access devices, unless the job falls into a job class assigned to a direct SYSOUT writer. In that case, both messages and data addressed to a SYSOUT data set are written directly to the device for the direct SYSOUT writer for that job class. (Messages for jobs canceled on the input queue and jobs failed by the reader/interpreter, and data produced by system tasks cannot be processed by direct system output writers.)

When the job completes (assuming it does not use a direct SYSOUT writer), entries are made in system output class queues that represent the data sets and messages directed to the output classes. Later system output writers remove these entries from the queues and process the data they represent. Processing consists of writing system messages to the output device and calling a data set writer routine for each data set encountered. The data set writer routine used for a data set may be specified by name in a DD statement, otherwise, a standard IBM-supplied writer routine is used. The standard routine transcribes the data set to the specified output device, making only those data format and control character transformation required to conform to the attributes specified for the output data set.

The following material describes how to write a nonstandard data set writer routine.

Characteristics of the Output Writer

Before writing or modifying an output writer routine, be familiar with the functions performed by the standard data set writer for Operating System/360. (For the remainder of this chapter, the Operating System/360 data set writer is referred to as the standard writer.) In general, these functions include opening the data set (referred to as an input data set) that contains the processed information, obtaining the records of the data set, making any necessary transformations in record format or control character attributes, and placing these (possibly transformed) records in the output data set, which appears on a specified output device. The standard writer also must close the input data set and restore system conditions to the state they were in before the writer routine was invoked.

Programming Conventions

To use the output writer routine, specify the name of the routine as a parameter in the SYSOUT operand of a DD statement (see the **Job Control Language** publication). (This parameter is ignored if the job falls into a jobclass assigned to a direct SYSOUT writer.) The routine must be in the system library (SYS1.LINKLIB). A writer routine is not limited in size except that size may influence the region requirements of the system output writer (see the **Storage Estimates** publication).

In MVT, the routine is attached (via the ATTACH macro instruction) when a data set requiring the routine is to be processed. The standard linkage conventions for attaching are used. Any storage required for work areas and tables should be obtained by the GETMAIN macro instruction and released by the FREEMAIN macro instruction. The output writer routines must be reenterable.

When the routine is finished, it must return control to the standard writer by using the RETURN macro instruction.

After job management routines perform initialization requirements and open the output data set into which the writer routine will put records, control is given to the routine via the

ATTACH macro instruction. At this time, general registers 1 and 13 contain information that the program must use. Register 1 contains the storage address of a 12-byte list. Figure 9 describes the information in this parameter list.

Byte 0	**Output Device**	**Indicator.**
	Bit 0	(High-order bit): If this bit is on (set to 1), the output unit is a 1442 punch.
	Bit 1	If this bit is on, the output unit is either a punch or a tape with a punch as the final destination.
	Bit 2	If this bit is on, the output unit is either a printer or a punch.
	Bits 3-7	No significant information.
Bytes 1-3	Not used, but must be present	
Bytes 4-7	This word contains the address of the data control block (DCB) for the opened output data set to be referred to by the writer.	
Bytes 8-11	This word contains the DCB address for the input data set from which your writer will obtain logical records. (At the time this 12-byte parameter list is given to your writer, the input data set is not open.)	

Figure 9. Parameter List Referred to by Register 1

The switches indicated by the three high-order bit settings in byte 0 should be used to translate control character information from the input data set records to the form required by the output data set records. Based on the indications given in Figure 9, the high-order three bits of byte 0 signify the type of output device as follows:

111.....	1442 punch unit
011.....	2520 punch unit or 2540 punch unit
001.....	1403 printer, 1404 printer, 1443 printer, or 3211 printer unit
010.....	tape unit with final punch destination
000.....	tape unit with final printer destination

When the writer gets control, it must preserve the contents of register 0 through 12, and 14. Register 13 contains the address of a standard register save area that saves the contents of these registers. Save the contents of register 13 by using the SAVE macro instruction.

An output writer routine must issue an OPEN macro instruction to open the desired input data set residing on a direct access device as a result of the previous execution of a processing program. (Note: The output data set used by a writer is opened by a job management routine before control is given to the writer. This output data set must be given records by a PUT macro instruction operating in the "locate" mode. The **Data Management Macro Instructions** publication describes this macro instruction.)

If the processing program that produces a given data set (to be used as an input data set by a writer) did not open the data set, the data set contains no records, and the DCBBLKSI and DCBBUFL fields of the input DCB contains zero. The DCBBLKSI field may also be zero even if the data set does contain records -- if the processing program did not put the block size value for the input data set in the DCB. If both these DCB fields are zero, a value (the standard writer uses the decimal value 18) is inserted in the DCBBLKSI field to permit the open routine to continue. The standard writer does this via a routine pointed to by an entry in the EXLIST parameter of the DCB. Since there is no data set, nothing is put on the output device. The data set writer must provide a SYNAD routine to process errors associated with the output as well as the input data set.

The Standard Data Set Writer also includes accounting support for the SMF Output Writer Record (record type 6). If you require SYSOUT accounting information, refer to the publication **System Management Facilities**, for details.

Before the OPEN macro instruction is issued, the DCBD macro instruction can be used to symbolically define the fields of the DCB, and the EXLIST and/or SYNAD routine addresses can be inserted. Other than SYNAD, no modifications can be made to the output DCB.

After the routine finishes writing the output data set, it must close the input data set and return using the RETURN macro instruction. A return code must be placed in register 15. This code should indicate that an unrecoverable output error either has occurred (code of 8) or has not occurred (code of 0).

3525 Note -- Interpret Punch: The programming support for the 3525 includes an INTERPRET PUNCH feature which is supported by BSAM and QSAM. The support for this feature includes the punching and printing of graphically printable punched characters on print lines one and three of the card. Line one includes the first 64 characters and line three includes the last 16 characters (right justified). Extraneous characters are printed for non-graphic eight-bit codes.

If the INTREPRET PUNCH function is designated via the new FUNC parameter in either a DCB or DD statement, an existing output data set will be interpreted as well as punched.

Note : The output must be 80 bytes, or 81 bytes if first character control is being used.

Writing an Output Writer

This provides a general description of the procedures followed by the standard writer. (See Figure 10.) When writing a writer routine, you may wish to delete, modify, or add items to some of these procedures, depending on the characterisitics of the data set(s). However, the procedures must be consistent with operating system conventions.

Saving Register Contents: Upon entering the writer program, the program must save the contents of the general registers, as previously discussed.

Obtaining Main Storage for Work Areas: In this work area, switches are established, record lengths and control characters are saved, and space is reserved for other uses. Obtain storage by a GETMAIN macro instruction.

Processing Input Data Set(s): To process a data set, the writer must get each record individually from the input data set, transform (if necessary) the record format and the control characters associated with the the record in accordance with the output data set requirements, and put the record in the output data set. Data set processing by the standard writer can be considered in three aspects.

1. The first consideration is what must be done before actually obtaining records from an input data set. If the output device is a printer, provision must be made to handle the two forms of record control character that may accompany a record in an output data set. The printer is designed so that if the output data set records contain machine control characters, a record (line) is printed before the effect of its control character is considered. However, if USASI control characters are used in the output data set records, the control character effect is considered before the printer prints a record. See Appendix B.

   Thus, if all the input data sets do not have the same type of control characters, it may be desirable to avoid overprinting of the last line of one data set with the first line of the following data set. If the records of the input data set have machine control characters

(mcc) and the output data set records are to have USASI control characters (acc), the standard writer produces a control character that indicates one line should be skipped before printing the first line of output data.

If the input data set records have acc and the output data set records are to be written with mcc, the standard writer prints a line of blanks before printing the first actual output data set record. Following this line of blanks, a one-line space is generated before the first output record is printed. The preceding "printer initialization" procedure (or a similar one based on the characteristics of your data sets) is recommeded.

2. After an input data set is properly opened and any necessary printer initialization completed, the writer obtains records from the input data set. The locate mode of the GET macro instruction is used. As each record is obtained, its format and control character must be adjusted, if necessary, to agree with that required for output.

   Note: Check the MACRF field of the input data set DCB to see if GET in locate mode can be used. If not the MACRF field must be overridden.

   Since the output data set is previously opened by another routine (job management), a writer routine must adhere to the established conventions. The data set is opened to receive records from the PUT macro instruction operating in the locate mode. For fixed-length record output, the length of the records in the output data set is obtained from the DCBLRECL field of the DCB. If an input record length is greater than the length specified for the records of the output data set, the standard writer truncates the necessary right-hand bytes of the input record. If the input record length is smaller than the output record length, the standard writer left-justifies the input record and adds blanks on the right end to give the correct length.

   When the output record length is variable and the input record length is fixed, the standard writer constructs each output record by adding control character information (if necessary) and variable record control information to the output record. The record control information is four bytes long and the control character information is one byte long. Both additions are made to the left end of the record. If the output record is not at least 18 bytes long, it is further modified by padding bytes (blanks) added to the right end of the record. If the output record length does not agrees with the length of the output buffer, the standard writer makes the proper adjustment.

3. The third aspect is an end-of-input data set routine. The standard writer handles output to either a card punch unit or a printer unit, as required. Output to an intermediate device such as a tape unit is considered in light of the ultimate destination (e.g., punch or printer). If proper consideration is not given, all records from a given data set may not be available on the output device until the output of records from the next data set is started or until the output data set is closed. When the output data set is closed, the standard writer automatically puts out the last record of its last input data set.

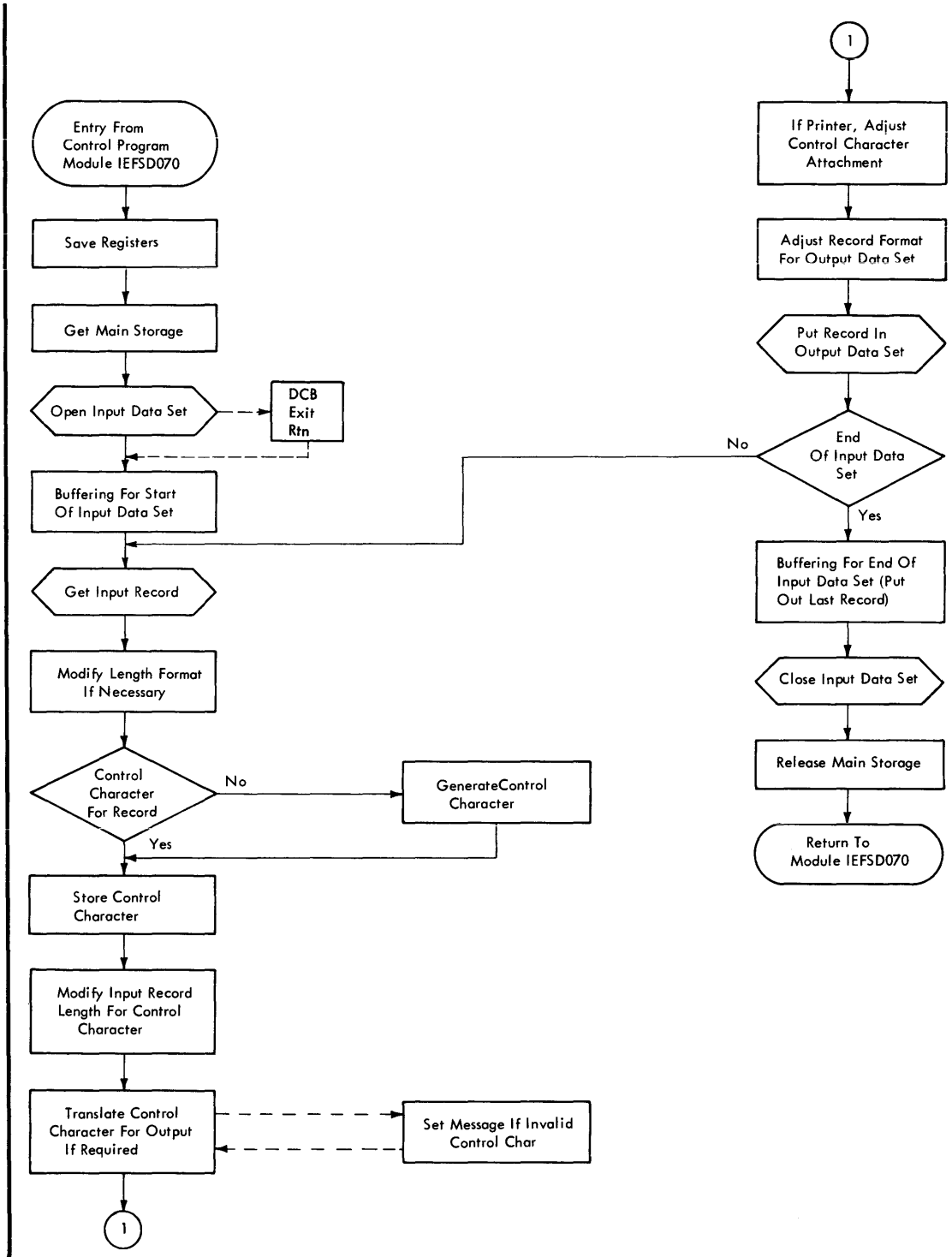


Figure 10. General Logic of Standard Output Writer

Punch Output: Normally, when the standard writer is using a card punch as the output device, the last three output records are not in the collection pockets of the punch when the input data set is closed. To put out these three records with the rest of the data set and with no intervening pauses, the writer provides for three blank records following the actual data set records.

Printer Output: When the standard writer uses a printer as an output device, the last record of the input data set is not normally put in the output data set when the input data is closed. To force out this last record, the writer generates a blank record that follows the last record of the actual data set.

The problem of overprinting the last line of one data set by the first line of the following data set must also be considered. Depending on the combination of input record control character and required output record control character, a line of blanks and a spacing control character may be used either individually or in combination to preclude overprinting. (**Note:** If overprinting is desired for some reason, control characters in the data set records themselves may be used to override the effect (but not the action) of the previously described solutions to overprinting.)

Closing Input Data Set(s): After the standard writer finishes putting out the records of an input data set, it closes the data set before returning control to the system output writer. All input data sets must be closed.

Releasing Main Storage: The storage and buffer areas obtained for the writer must be released to the system before the writer relinquishes control. The FREEMAIN macro instruction should be used for this.

Restoring Register Contents: The original contents of general registers 0 through 12, and 14 must be restored. The RETURN macro instruction is used for this. To inform the operating system of the results of the processing done by the writer, a return code is placed in general register 15 before control is returned. If the writer routine terminates because of an unrecoverable error on the output data set, the return code is 8; otherwise, the return code is 0. Unrecoverable input errors must be handled by the data set writer.

Adding SVC Routines to the Control Program

This chapter provides detailed information on how to write an SVC routine and insert it into the control program portion of the System/360 Operating System.

Characteristics of SVC Routines

All SVC routines operate in the supervisor state. Keep the following characteristics in mind when deciding what type of SVC routine to write:

- **Location of the routine** - The SVC routine can be either in main storage at all times as part of the resident control program, or on a direct access device as part of the SVC library. Type 1 and 2 SVC routines are part of the resident control program, and types 3 and 4 are in the SVC library.
- **Size of the routine** - Types 1, 2, and 4 SVC routines are not limited in size. However, a type 4 SVC routine must be divided into load modules of 1024 bytes or less. The size of a type 3 SVC routine must not exceed 1024 bytes.
- **Design of the routine** - Type 1 SVC routines must be reenterable or serially reusable; all other types must be reenterable. To aid system facilities in recovering from machine malfunctions, the SVC routines should be refreshable.
- **Interruption of the routine** - When the SVC routine receives control, the CPU is masked for all maskable interruptions but the machine check interruption. All type 1 SVC routines must execute in this masked state. To allow interruptions to occur during the execution of a type 2, 3, or 4 SVC routine, change the appropriate masks. When a type 2, 3, or 4 SVC routine will run for an extended period of time, it is recommended to allow interruptions to be processed where possible.

Programming Conventions for SVC Routines

The programming conventions for the four types of SVC routines are summarized in Figure 11. Details about many of the conventions are in the reference notes that follow the table. The notes are referred to by the numbers in the last column of the table. If a reference note for a convention does not pertain to all types of SVC routines, and asterisk indicates the types to which the note refers.

Conventions	Type 1	Type 2	Type 3	Type 4	Reference Code
Part of resident control program	Yes	Yes	No	No	
Size of routine	Any	Any	≤ 1024 bytes	Each load module ≤ 1024 bytes	
Reenterable routine	Optional, but must be serially reusable	Yes	Yes	Yes	1
May allow interruptions	No	Yes	Yes	Yes	2
Entry point	Must be the first byte of the routine or load module, and must be on a doubleword boundary				
Number of routine	Numbers assigned to your SVC routines should be in descending order from 255 through 200				
Name of routine	IGCnnn	IGCnnn	IGC00nnn	IGCssnnn	3
Register contents at entry time	Registers 3, 4, 5, and 14 contain communication pointers; registers 0, 1, and 15 are parameter registers				
May contain relocatable data	Yes	Yes	No*	No*	5
Can supervisor request block (SVRB) be extended	Not applicable	Yes*	Yes*	Yes*	6
May issue WAIT macro instruction	No	Yes*	Yes*	Yes*	7
May issue XCTL macro instruction	No	No	No	Yes*	8
May pass control to what other types of SVC routines	None	Any	Any	Any	
Type of linkage with other SVC routines	Not applicable	Issue supervisor call (SVC) instruction			
Exit from SVC Routine	Branch using return register 14				
Method of abnormal termination	Use resident abnormal termination routine	Use ABEND macro instruction or resident termination routine			9

Figure 11. Programming Conventions for SVC Routines

Reference Code	SVC Routine Types	Reference Notes
1	all	If the SVC routine is to be reenterable, macro instructions whose expansions store information into an inline parameter list cannot be used.
2	all	Write SVC routines so that program interruptions cannot occur. If a program interruption does occur during execution of an SVC routine, the routine loses control and the task that called the routine terminates. If a program interruption occurs and you are modifying a serially reusable SVC routine, a system queue, control blocks, etc., the modification will never complete; the next time the partially modified code is used, the results will be unpredictable.
3	all	Follow these conventions when naming SVC routines: • **Types 1 and 2** must be named IGCnnn; nnn is the decimal number of the SVC routine. You must specify this name in an ENTRY, CSECT, or START instruction. • **Type 3** must be named IGC00nnn; nnn is the signed decimal number of the SVC routine. This name must be the name of a member of a partitioned data set. • **Type 4** must be named IGCssnnn; nnn is the signed decimal number of the SVC routine, and ss is the number of the load module minus one, e.g., ss is 01 for the second load module of the routine. This name must be the name of a member of a partitioned data set.
4	all	Before the SVC routine receives control, the contents of all registers are saved. For type 4 routines, this applies only to the first load module of the routine. In general, the location of the register save area is unknown to the routine that is called. When the SVC routine receives control, the status of the registers is as follows: • **Register 0 and 1** contain the same information as when the SVC routine was called. • **Register 2** contains unpredictable information. • **Register 3** contains the starting address of the communication vector table. • **Register 4** contains the address of the task control block (TCB) of the task that called the SVC routine. • **Register 5** contains the address of the supervisor request block (SVRB), if a type 2, 3, or 4 SVC routine is in control. If a type 1 SVC routine is in control, register 5 contains the address of the last active request block. • **Register 6 through 12** contain unpredictable information. • **Register 13** contains the same information as when the SVC routine was called. • **Register 14** contains the return address. • **Register 15** contains the same information as when the SVC routine was called.

Use registers 0, 1, and 15 to pass information to the calling program. The contents of registers 2 through 14 are restored when control is returned to the calling program.

5 3,4 Because relocatable address constants are not relocated when a type 3 or 4 SVC routine is loaded into main storage, do not use them in coding these routines. Do not use macro instructions whose expansions contain relocatable address constants. Types 1 and 2 are not affected by this restriction since they are part of the resident control program.

6 2,3,4 Users can extend the SVRB, in 8-byte increments, from 96 bytes up to 144 bytes. The extended area is available as a work area during execution of the routine only by specifying the extension during the system generation process. When the SVC routine receives control, register 5 contains the address of the SVRB to which the extended save area is appended.

7 2,3,4 Do not issue the WAIT macro instruction unless you have changed the system mask to allow I/O and external interruptions. If you have allowed these interruptions, you can issue WAIT macro instructions that await either single or multiple events. The event control block (ECB) for single-event waits or the ECB list and ECBs for multiple-event waits must be in dynamic main storage.

8 4 When you issue an XCTL macro instruction in a routine under control of a type 4 SVRB, the new load module is brought into a transient area.

Then contents of registers 2 through 13 are unchanged when control is passed to the load module; register 15 contains the entry point of the called load module.

9 all Type 1 SVC routines must use the resident abnormal termination routine to terminate any task. The entry point to the abnormal termination routine is in the communication vector table (CVT). The symbolic name of the entry point is CVTBTERM.

Type 2, 3, and 4 SVC routine must use the ABEND macro instruction to terminate the current task, and must use the resident abnormal termination routine to terminate a task other than the current task.

Before the resident abnormal termination routine is entered, the CPU must be masked for all maskable interruptions but the machine check interruption, and registers 0, 1, and 14 must contain the following:

- **Register 0** contains the address of the TCB of the task to be terminated.
- **Register 1** contains the following information:

  Bit 0 is a 1 if you want a dump taken.

  Bit 1 is a 1 if you want to terminate a job step.

  Bits 2-7 are zero.

  Bits 8-19 contain the error code.

  Bits 20-31 are zero.

- **Register 14** contains the return address. The resident abnormal termination routine exits by branching to the address contained in register 14.

The contents of register 15 are destroyed by the abnormal termination routine.

Writing SVC Routines

Because the SVC routine will be a part of the control program, follow the same programming conventions used in SVC routines supplied with System/360 Operating System.

Four types of SVC routines are supplied with System/360 Operating System, and the programming conventions for each type differ. The general characteristics of the four types were described in the preceding text, and the programming conventions for all types were shown in tabular form.

Adding SVC Routines Into the Control Program

Insert SVC routines into the control program during the system generation process.

Before the SVC routine can be inserted into the control program, the routine must be a member of a cataloged partitioned data set. Name this data set SYS1.name.

The following text gives a description of the necessary information for the system generation process. The publication **System Generation**, describes the system generation macro instruction.

Specifying SVC Routines

Use the SVCTABLE macro instruction to specify the SVC number, the type of SVC routine, and, for type 2, 3, or 4 routines, the number of double words in the extended save area.

Inserting SVC Routines During the System Generation Process

To insert a type 1 or 2 SVC routine into the resident control program, you use the RESMODS macro instruction. Specify the name of the partitioned data set and the names of the members to be inserted into the control program. Each member can contain more than one SVC routine.

To insert a type 3 or 4 SVC routine into the SVC library, you use the SVCLIB macro instruction. Specify the name of the partitioned data set and the names of members to be included in the SVC library. The member names must conform to the conventions for naming type 3 and 4 routines, i.e., IGC00nnn and IGCssnnn.

Message Routing Exit Routines

This topic provides detailed information on how to write user exit routines that modify the routing and descriptor codes of WTO or WTOR messages for any MVT operating system that has the Multiple Console Support Option (MCS). Information is provided on inserting this exit routine into the resident portion of the control program. In addition, a description of the characteristics and configuration of MCS is supplied.

Characteristics of MCS

The multiple console support (MCS) option of the IBM System/360 Operating System routes messages to different functional areas according to the type of information that the message contains. In MCS, a functional area is defined as one or more operator's consoles that are doing the same type of work. (Some examples of functional areas are: (1) the tape pool area, (2) the disk pool area, and (3) the unit record pool area.) Each WTO and WTOR macro instruction is assigned one or more routing codes which are used to determine the destination of the message. There are sixteen routing codes that can be used. When the message is ready to be routed, the routing codes assigned to the message are compared to the routing codes assigned to each console. If any of the routing codes match, the message is sent to that console. (For descriptions and definitions of the routing codes, see the publication **Supervisor Services and Macro Instructions.**)

If the standard routing codes provided on application and system messages do not cover special situation at an installation, the routing codes used on the message can be modified by coding a user exit routine. The exit routine receives control prior to the routing of message so users can examine the message text and modify the message's routing and descriptor codes. The system will use the modified routing codes to route the message. Descriptor codes provide a mechanism for message presentation and deletion and are explained later in this chapter.

Automatic console switching occurs when permanent hardware errors are detected. Command initiated console switching is provided to permit restructuring of the system console configuration and the hard copy log by system operators. Consoles can be moved into or out of functional areas at any time during system operation.

A hard copy log option records messages, operator and system commands, and operator and system responses to commands. The hard copy log can be a console device or it can be the system log (SYSLOG). The number and type of messages recorded on the log is also optional. The installation may wish to record a selected group of messages, or it may wish to record all messages. If commands are recorded, the system automatically records command responses.

Programming Conventions For WTO/WTOR Routines

The programming conventions for the WTO/WTOR exit routine are summarized in Figure 12. Details about many of the conventions are in the reference notes that follow that table. The notes are referred to by the numbers in the last column of the table.

Conventions	Requirements	Reference Code
Part of resident control program	Yes	
Size of routine	Any size	
Reenterable routine	Optional, but must be serially reusable	1
May allow interruptions	Yes	2
Name of routine	Must be IEECVXIT	
Disposition of general registers	Registers must be saved at entry and restored prior to returning	
Format of text and codes	Provided through the DSECT IEECUCM	3
May issue WAIT, XCTL, WTO or WTOR macro instructions	No	
Method of abnormal termination	None	4
Exit from routine	RETURN macro instruction	

Figure 12. Programming Conventions for WTO/WTOR Exit Routine

Reference Code	Reference Notes
1	If the exit routine is to be reenterable, do not use macro instructions whose expansions store information into an inline parameter list.
2	Write the exit routine so that program interruptions cannot occur. If a program interruption occurs during execution of the exit routiine, the routine loses control and the communications task is terminated.
3	DSECT IEECUCM provides the format of the message text, routing codes and descriptor codes. The pointer in register 1 points to the first word of the message text, UCMMSTXT. The format is: UCMMSTXT Message Text (128 Characters-padded with blanks) UCMROUTC Routing codes (4 bytes) UCMDESCD Descriptor codes (4 bytes) DSECT IEECUCM is contained in SYS1.MODGEN System messages have a message code as the first seven characters of the message text. This code may be examined to aid in identifying system messages, but it must not be modified. The UCMROUTC field contains the routing codes. A bit setting of "1" indicates that the WTO or WTOR was assigned that particular routing code. Bit assignments and their meanings are:

Bit	Assignment	Meaning
Byte 0		
Bit 0	Routing code 1	Master Console
Bit 1	Routing code 2	Master Console Informational
Bit 2	Routing code 3	Tape Pool
Bit 3	Routing code 4	Direct Access Pool
Bit 4	Routing code 5	Tape Library
Bit 5	Routing code 6	Disk Library
Bit 6	Routing code 7	Unit Record Pool
Bit 7	Routing code 8	Teleprocessing Control
Byte 1		
Bit 0	Routing code 9	System Security
Bit 1	Routing code 10	System Error/Maintenance
Bit 2	Routing code 11	Programmer Information
Bit 3	Routing code 12	Emulator Program (under OS)
Bit 4	Routing code 13	Available for Customer Usage
Bit 5	Routing code 14	Available for Customer Usage
Bit 6	Routing code 15	Available for Customer Usage
Bit 7	Routing code 16	Reserved
Byte 2		Reserved
Byte 3		Reserved

3 The UCMDESCD field contains the descriptor codes. A bit setting of "1" indicates that the WTO or WTOR was assigned that particular descriptor code. Bit assignments and their meanings are:

Bit	Assignment	Meaning
Byte 0		
Bit 0	Descriptor code 1	System Failure
Bit 1	Descriptor code 2	Immediate Action Required
Bit 2	Descriptor code 3	Eventual Action Required
Bit 3	Descriptor code 4	System Status
Bit 4	Descriptor code 5	Immediate Command Response
Bit 5	Descriptor code 6	Job Status
Bit 6	Descriptor code 7	Application Program/Processor
Bit 7	Descriptor code 8	Out-of-line Message
Byte 1		
Bit 0	Descriptor code 9	DISPLAY or MONITOR command response
	Descriptor codes 10 through 16	Reserved
Byte 2		Reserved
Byte 3		Reserved

4 The exit routine is part of the communications task. Abnormal termination of the exit routine causes the operating system to terminate abnormally (code of F03).

Messages Not Using Routing Codes

There are certain messages that the exit routine does not see. These are messages that have the MSGTYP operand in the WTO or WTOR macro instruction coded with the JOBNAMES, STATUS, ACTIVE or Y parameter, multiple-line WTOs (inculding status displays) and messages that are being returned to the requesting console, i.e., a response to a DISPLAY A command. Routing of these messages is on criteria other than the routing codes, therefore, the system bypasses the exit routine.

Writing a WTO/WTOR Exit Routine

To modify the standard routing codes and descriptor codes write a WTO/WTOR Exit Routine. This routine will be part of the control program. If a message's routing code field is used by the operating system to route the message, the routine will receive control prior to the routing of the message. When the routine receives control, register 1 contains a pointer to the first word of the message text. The message text field is 128 bytes long; followed by a four-byte routing code field and a four-byte descriptor code field. The exit routine may examine but not modify the message text.

A message will be sent to only those locations specified in the modified routing codes. All messages with modified routing codes are sent to the hard copy log when the log is included in the operating system. When the log is not included, the exit routine must not suppress messages that contain a routing code of 1, 2, 3, 4, 7, 8, or 10 since messages with these codes are necessaty for system maintenance. Message suppression is turning off all routing codes of a message, causing the message to be discarded. WTO messages can be suppressed. If a WTOR message is suppressed, it will be sent to the master console by the operating system.

Adding a WTO/WTOR Exit Routine to the Control Program

A system generation option is available to enable you to include a resident, user-written exit routine into the communications task.

The CONOPTS operand of the SCHEDULR system generation macro instruction controls the inclusion of the exit routine. A description of SCHEDULR is found in the publication **System Generation**.

Task supervision must be performed for the exit routine when the routine is requested at system generation. This supervision is performed every time a message is routed by its routing codes, even if the exit routine is not present. To maintain optimum throughput, the exit routine should not be specified at system generation unless it will be used.

Inserting the WTO/WTOR Exit Routine

To enter the exit routine into the control program before system generation, use the Linkage Editor to replace the dummy WTO/WTOR exit routine IEECVCTE in SYS1.CI505 with the WTO/WOTR exit routine.

To enter the exit routine into the control program after system generation, use the Linkage Editor to replace the dummy WTO/WTOR exit routine IEECVCTE in the SYS1.NUCLEUS with your WTO/WTOR exit routine.

Handling Accounting Routines

Accounting routines can be added to the control program. This topic describes the input available to an accounting routine; the characteristics and requirements of an IBM-supplied data set writer that may be used to log accounting information generated by an accounting routine; and how to insert an accounting routine into the control program. Conventions to be followed in preparing an accounting routine are also noted.

Programming Conventions for Accounting Routines

User-written accounting routines can consist of more than one control section.

Attributes: A user-written accounting routine must be reenterable.

CSECT Name and Entry Point: The control section containing the entry point of the accounting routine, and the entry point, must be named IEFACCTRT.

Register Saving and Restoring: The content of registers 0 through 14 must be saved upon entry to the accounting routine and restored prior to exiting.

Entrances: Control is given to the accounting routine at the following times:

- Step Initiation
- Step Termination
- Job Termination

Exit: The RETURN macro instruction restores the content of the general registers and returns control to the operating system.

Input Available to Accounting Routines

Register 0 contains an entrance code, indicating the time that the accounting routine gained control.

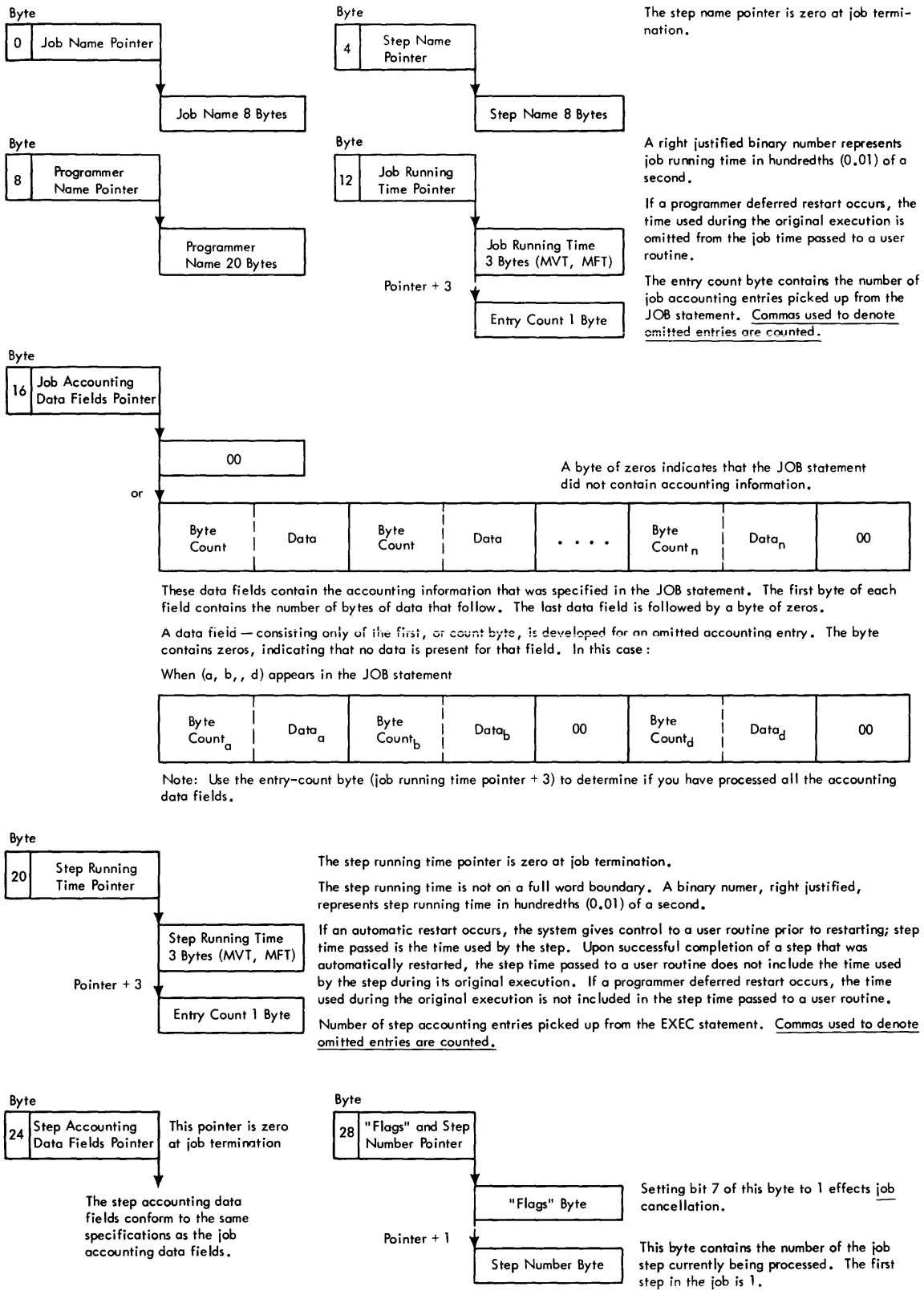
Register 0 = 8: Step Initiation

= 12: Step Termination

= 16: Job Termination

Register 1 contains the starting address of a list of pointer to items of accounting information. Each pointer is on a fullword boundary. The sequence of pointers in the list and the items of information provided are described in the following diagram.

User accounting routines should only use pointers that are in the list addressed by register 1. Other pointers are subject to change in subsequent releases.



Note: You can use the flag byte to cancel the execution of a job whose accounting information does not conform to your installation's standards. You can equate step initiation for the first step in a job to job initiation, i.e., the step number byte contains 1.

Figure 13. Accounting Information Available to User

Adding An Accounting Routine

Accounting routines can be added to the control program in two ways. First, by placing the routine on the SYS1.CI505 data set used in system generation. Second, by placing the routine in the appropriate load module of the control program after system generation. The effect of either is to replace the dummy accounting routine with the user-written routine.

At system generation, specify that an accounting routine is to be supplied. This is done through the ACCTRTN=paramenter of the system generation SCHEDULR macro instruction. This specification causes the linkage to the accounting routine to be installed in the scheduler component of the system being generated, and makes usable the accounting data set writer routine. When not installing accounting routines until after the system is generated, a dummy accounting routine, named IEFACTRT, is placed in the system at this time.

Add the size of the IEFACTRT routine to the estimate of the minimum amount of storage required to initiate a job. This storage requirement should be specified in the MINPART parameter of the SCHEDULR macro instruction.

Insertion at System Generation Time

To insert the accounting routine into the control program during system generation, use the linkage editor to place the routine in the SYS1.CI505 data set prior to the start of the system generation. The SYS1.CI505 data set (furnished with the starter operating system) contains load modules which are combined during the system generation process to form the load modules composing the control program. In response to the specification made in the system generation SCHEDULR macro instruction, the accounting routine is incorporated in the apporproate load modules for the system being generated.

Place the accounting routine in the SYS1.CI505 data set under the name IEFACTRT. This replaces the dummy accounting routine -- also named IEFACTRT.

Insertion after System Generation

To insert the accounting routine into the control program after system generation, place the routine in load modules of the scheduler component of the generated control program, using the linkage editor. **The scheduler load modules are in the linkage library (SYS1.LINKLIB data set) of the generated system.** The affected load modules of the MVT scheduler follows:

MVT Configuration

MVT Scheduler:

- load module IEFSD061 -- step and job termination
- load module IEFW21SD -- step initiation

An example of the input for a linkage editor to insert the accounting routine into any of the job schedulers follows:

```
//jobname     JOB  (parameters)
//stepname    ECEC PGM=IEWL, (parameters)
//SYSPRINT    DD   SYSOUT=A
//SYSUT1      DD   UNIT=SYSDA,SPACE=(parameters)
//SYSLMOD     DD   DSNAME=SYS1.LINKLIB,DSIP=OLD
//SYSLIN      DD   *
                .
                .                  This sequence must be repeated
                .                  for each scheduler load module
                (object code)      which contains an inserted
                .                  accounting routine.
                .
                .
                INCLUDE   SYSLMOD(load module name)
                ALAIS     alias names
                ENTRY     entry point name
                NAME      load module name (R)
```

In this example, "load module name" represents the appropriate scheduler load module as identified in the preceding text. To ensure accuracy in identifying the correct alias names and entry point names for the load modules, obtain these names from the system generation listing produced during generation of the system. These names are specified in the system generation Stage II linkage editor output execution that produced the load module.

Output From Accounting Routines

Output can be written in three ways:

- By issuing console messages
- By using standard system output writers
- By using an IBM-supplied accounting data set writer

Console Messages: Use the Write to Operator (WTO) or Writer to Operator with Reply (WTRO) macro instruction. Write-to programmer (WTO with a routing code of 11) must not be issued from accounting routines.

System Output: Assemble the following calling sequence in the accounting routine. The contents of register 12 must be the same as when the accounting routine was entered, and register 13 must contain the address of an area of 32 fullwords.

When writing an accounting routine for inclusion in the job scheduler, be aware that register saving conventions within the control program are different from those for problem programs. In the job scheduler, registers are saved in the sequence 0-14 in a 15-word save area. There is no place provided to save register 13; it can be saved in another register or in another save area not known to the control program. This can be done by adding a word to the end of the save area that is provided and is addressed as SAVE + 60.

```
Name      Operation Operand
          MVC       36(4,12),MSGADDR     MOVE MESSAGE ADDRESS AND
          MVC       42(2,12),MSGLEN      LENGTH TO SYSTEM TABLE
          L         REG15,VCONYS         BRANCH AND LINK TO MESSAGE
          BALR      REG14,REG15          ROUTINE
          .
          .
MSGADDR   DC        A(MSG)
MSG       DC        C'text of message'
MSGLEN    DC        H'two character lingth of message'
VCONYS    DC        V(IEFYS)
```

Accounting Data Set writer: This writer places accounting records in the accounting routine in a data set named SYS1.ACCT. The data set must reside on a permanently resident direct access device. The accounting routine must provide linkage to the writer. Pass the beginning address of the record to be written to the writer.

A sample accounting routine, showing the use of console messages, output to the system output writer, and the data set writer is stored under the name SAMACTRT in the SYS1.SAMPLIB data set furnished with the starter operating system.

SEC IV

Adding the Accounting Data Set Writer

The accounting data set writer (module IEFWAD) is generated in the appropriate scheduler load modules during system generation after specifing the accounting routine in the SCHEDULR macro. These are the same modules that contain the user-written accounting routine. Scheduler storage requirements are increased by the amount of storage needed by the accounting routine plus 2600 bytes. The writer places accounting records developed by the routine in a data set named SYS1.ACCT.

Linkage

The accounting routines link to the writer via the following code:

```
          L     R15,VCON
          BALR  14,15
          .
          .
          .
VCON      DC    V(IEFWAD)
```

Input

The accounting routine passes in register 1 the address of the accounting record to be written.

The record format is:

DS3 H -- space used by the data set writer

DC '___'--contains the number of bytes of data being passed. This number cannot exceed the capacity of 1 track on the direct access volume being written on.

DC ____-- the data to be written in SYS1.ACCT.

Registers 13,14, and 15 are used as specified by operating system conventions (14 and 15 are used for linkage as above; 13 must point to an 18-word save area).

Specifying the SYS1.ACCT Data Set

The SYS1.ACCT data set must be pre-allocated on a direct access volume that will be permanently resident. The data set must by named SYS1.ACCT, have no secondary extents, and be allocated contiguous space. **Do not catalog the data set.**

If the installation has two permamently resident volumes available for accounting routines, create two SYS1.ACCT data sets and utilize the console messages and replies to notify the system as to which data set is to be written to.

Output

If the IEFWAD routine successfully writes the record in the SYS1.ACCT data set, the routine returns control to the accounting routine immediately. If the routine fails to write the record, it uses message IEF507D to bring the error condition to the attention of the operator. (See the **Messages and Codes** publication (GC28-6631) for the text of, and answers to, the message.) Depending upon the answer, the routine may try again to write your record in the SYS1.ACCT data set.

In any case, a code is returned to the routine indicating either that the record was written successfully, or, if it was not written successfully, the cause of the failure. The return codes are described in the follwing table.

Contents	Type*	Meaning
		Register 15
0	D	The record was written to the data set.
4	D	The record was not written to the data set because the record exceeds the length of one track.
8	D	The record was not written to the data set because there is no more space in the data set.
12	D	The record was not written to the data set because no space had been allocated to the data set.
16	D	The record was not written to the data set because a permanent I/O error was encountered while trying to write it.
20	D	The record was not written to the data set because the previously last record could not be found.
24	D	Operator gave invalid device address.
		Register 0
n	B	Number of tracks still available in the data set. (Valid only if register 15 is zero.)

*Type -- Type of number. D -- Decimal, B -- Binary

Use of ENQ/DEQ

IEFWAD enqueues on the major Q name SYSIEFAR and the minor Q name WD.

Writing Rollout/Rollin Installation Appendages

This topic explains how to write rollout/rollin appendages for MVT configurations of the operating system and how to insert them into the operating system before or after system generation. The four exits to user-written appendages and their functions are explained. The chapter also presents sample coding for an appendage.

Characteristics of Rollout/Rollin Installation Appendages

The rollout/rollin feature of IBM System/360 Operating System is used with MVT configurations as an aid to main storage management. Rollout/rollin allows the temporary, dynamic expansion of the job step beyond its originally specified region. When the job step needs more space, rollout/rollin attempts to obtain unassigned storage for its use. If there is no such unassigned storage, another job step is rolled out -- transferred to auxiliary storage (IBM 2301, 2311, 2314 or 2321 -- so that its region may be used by your job step. When released by the job step, this additional storage is again available, either as unassigned storage, if that was its source, or to receive the job step to be transferred back into main storage (rolled in). (**Note:** Teleprocessing jobs which use the Autopoll option should not be marked eligible for rollout. A rolled-out job which is using the Autopoll option cannot be restarted properly.)

During the course of normal rollout processing, exits are taken to installation-written routines, so users can dynamically control various aspects of the rollout function. The routines must be serially reusable; they will reside as part of the resident nucleus and will be entered by a branch entry. IBM has supplied a dummy module which resolves the appendage exits during system generation.

To replace the dummy module before system generation, the object module which results from the assembly of the updated appendage routine should be link edited into the SYS1.CI535 data set. To replace the assembled dummy appendage module after system generation, you should link edit the new appendage module as a CSECT replacement in IEANUC01.

It may be necessary for the appendages to address the jobname; however, unless the job has issued an ATTACH, SYSINIT will appear in the jobname, and the actual jobname will appear in the stepname. Therefore, an appendage checking for a specific jobname should also check for SYSINIT; if it is encountered, the appendage should further check the stepname for the actual jobname.

There are four installation exits; their functions and the linkage to them are discussed in the following paragraphs.

Linkage to User Appendages

Follow these conventions for user-appendages:

1. Register 15 contains the base address of the routine.
2. Register 14 contains the return address.
3. Register 13 contains the address of an 18-word save area in which you must save any registers that you will use. The registers must be restored before exiting.

4. Register 1 contains the address of the TCB for the task that invoked rollout. (**Exception:** on entry to Appendage IV, register 1 contains the address of the PQE for the region selected for rollout.)

5. Register 0 contains the address of a three-fullword area. The first two bytes of the first word contain the number of rollouts now in effect. The third and fourth bytes of the first word contain the number of requestors now queued for rollout. The rollout queue is ordered according to dispatching priority. The second word contains the address of the queue origin for queued rollout (IEAROQUE). The third word is the address of the parameter list for the task that invoked rollout. The first word of the two-word parameter list contains the address of the TCB for the task that invoked rollout, and the second word contains a hexadecimal number which represents the length, in bytes, of the originally requested main storage area.

Appendage I: IEAQAPG1

The exit to Appendage I is taken when the current request for additional storage has tried to cause rollout, and at least one other job step has already invoked rollout. Users can determine, using their own criteria, whether to override the normal rollout procedure of allowing only one job step to invoke rollout at any given time. If users allow multiple (successive) rollouts, they are responsible for preventing system interlocks such as occur if each of two job steps needed two-thirds of main storage at the same time. (The obvious escape from this situation would be to arbitrarily cancel one of the steps.) If multiple rollouts are not allowed, the requesting task is placed upon the queue of tasks and are waiting for rollout. From the linkage information passed in the registers, decide whether or not to make an immediate attempt at rollout for the requesting step. If you desire an immediate attempt at rollout, return the TCB address passed in register 1 without change. If you do not desire an immediate attempt at rollout, return the address of the requesting task in complement form. When using the IBM-supplied Appendage I, the request will be queued and no multiple rollout will occur.

Appendage II: IEAQAPG2

The Appendage II exit is taken when there is not enough free space or there is no rolloutable job step of lower dispatching priority than the job step that invoked rollout. No attempt is made to find a higher dispatching priority step to roll out. Users have the option of requesting that the rollout function attempt to find a job step of higher dispatching priority that can be rolled out.

Users who do not want to look for a higher dispatching priority step to roll out should return the address of the requesting task without change. Users who do desire the higher dispatching pass should return the address in complement form.

Appendage III: IEAQAPG3

The exit to Appendage III is taken after the rollout function has determined, through the use of both its own and (optionally) criteria, that a job step suitable for rollout does not exist. Through this appendage users can select either the step which requested the unavailable storage or any other job step in the system for abnormal termination (ABEND). A job step not selected for ABEND (or if the IBM-supplied Appendage III is used), the requestor is placed on the rollout queue. If a job step other than the requestor is selected by the appendage, ABEND of the selected job step is initialized, and the requestor is queued for rollout.

Users who do not desire to initiate an ABEND should set register 1 to zero before exiting. The requestor is then queued for rollout. Users who do desire an ABEND must return in register 1

the address of the job step TCB to ABEND the task. (The address returned will be checked to ensure that it is a job step TCB. If it is not, it is ignored and the requestor is queued for rollout.) If the address is valid and is not the address of the requesting step, ABEND is initiated and the requestor is queued for rollout. If it is the address of the requesting step, ABEND is initiated and the requestor's IQE is returned to the available queue. If the IBM-supplied Appendage III is used, no ABEND occurs.

Appendage IV: IEAQAPG4

The Appendage IV exit is taken each time a job step has been selected as a candidate for rollout. This appendage gives you the opportunity to apply your criteria to each job step that the rollout function has found to be eligible for rollout. Job steps are considered for rollout eligibility beginning with the job step of lowest dispatching priority, and continuing upward until all eligible job steps with a lower dispatching priority (than that of the requesting job step) have been presented to your appendage. If you have supplied an appendage which permits job steps of higher dispatching priority to be eligible for rollout, these will also be presented to your appendage beginning with the job step of next highest dispatching priority (than that of the requesting step), and continuing upward until all eligible job steps with a higher dispatching priority have been presented.

The process of presenting job steps to your appendage for approval continues either until a job step is approved for rollout by the appendage, or until all eligible job steps have been examined and disapproved by the appendage.

Sample Coding of Appendages

The following pages contain sample coding illustrating the linkage to the appendage. In the example given, an Appendage II which approves the rollout of job steps with a higher priority that the requesting job step is used to illustrate appendage coding.

General Flow of Rollout Processing

The flowchart in Figure 14 depicts the overall flow of control through the various user appendages and the rollout module.

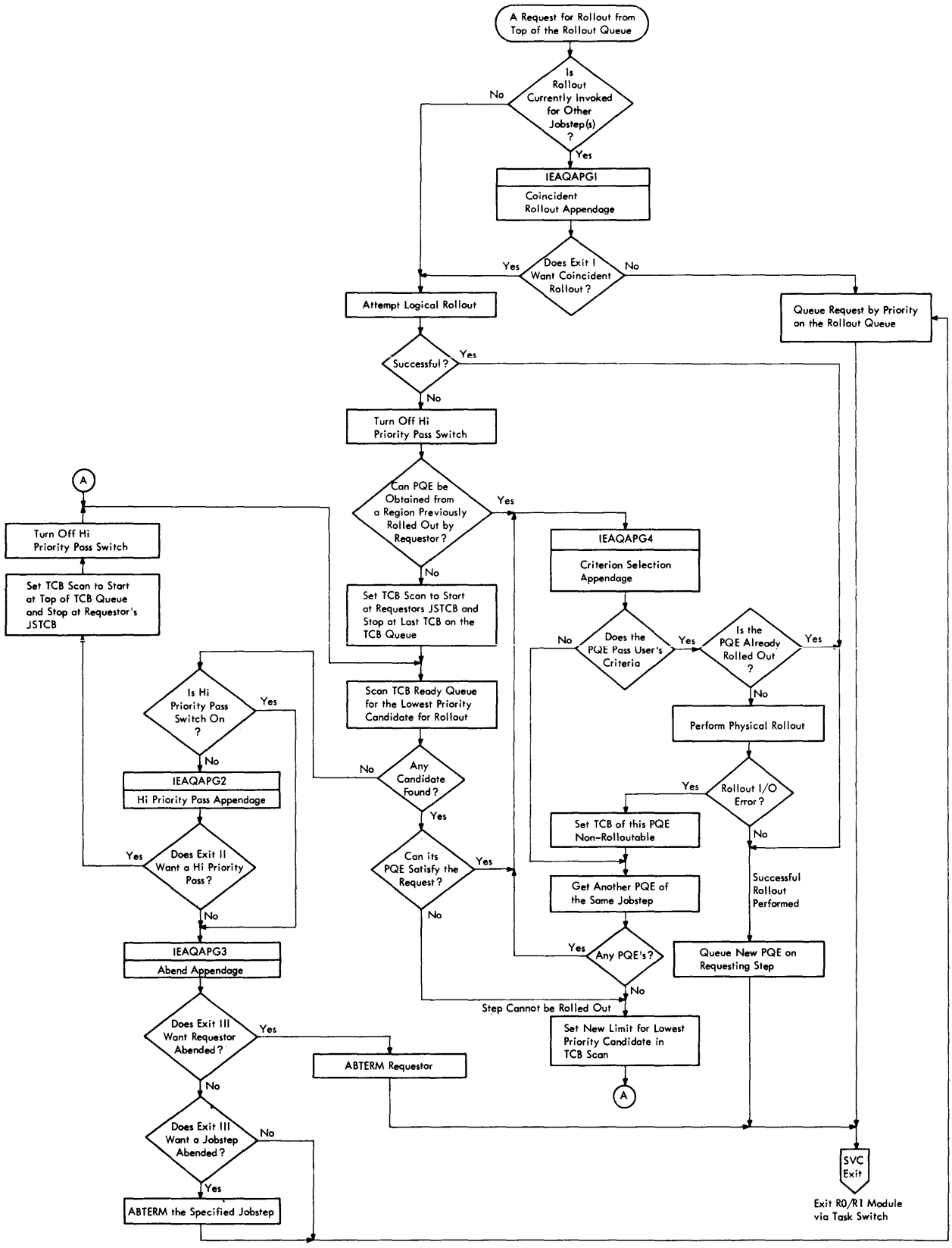


Figure 14. General Flow of Rollout/Rollin Processing

Source Statement

```
IEAPAPG2 CSECT
THIS ROUTINE WILL APPROVE THE ROLLOUT OF JOBSTEPS WITH A HIGHER
PRIORITY THAN THE REQUESTING JOBSTEP.  IT IS ENTERED FROM USER
APPENDAGE - IEAQAPG2 - WHICH IS RESIDENT IN THE NUCLEUS AS PART OF
THE ROLLOUT/ROLLIN CODE.
IT WILL WRITE TO THE OPERATOR INDICATING THE FOLLOWING:
    •    ROLLOUT STATUS (NUMBER OF ROLLOUTS IN EFFECT AND THE NUMBER OF
         ROLLOUT REQUESTS QUEUED.)
    •    THE NAME OF THE JOB REQUESTING ROLLOUT.
    •    APPROVAL OF THE REQUEST.
R1       EQU       1
R2       EQU       2
R3       EQU       3
R4       EQU       4
R5       EQU       5
R8       EQU       8
R12      EQU       12
R13      EQU       13
R14      EQU       14
         STM       R14,R12,12,(R13)
         BALR      R12,0
         USING     *,R12
         LR        R14,R13
         ST        R13,SAVEAREA+4
         LA        R13,SAVEAREA
         ST        R13,8(R14)
         LR        R2,0
         LR        R3,R1
         USING     TCB,R3
         L         R4,TIOTA            GET ADDRESS OF TASK I/O TABLE
         USING     TIOT,R4
         MVC       WTLENTER+27(8),JOBNAME
WTLENTER WTO       'IEAQAPG2 ENTERED             REQUESTS ROLLOUT'
         USING     ROSTATUS,R2
         LH        R8,INEFFECT         GET NBR OF ROLLOUTS IN EFFECT
         CVD       R8,WORK
         UNPK      WTLEXIT+29(2),WORK
         LH        R8,QUEUED      GET NBR OF ROLLOUT REQUESTS QUEUED
         CVD       R8,WORK
         UNPK      WTLEXIT+51(2),WORK
         MVC       WTLEXIT+74(3),YES
WTLEXIT  WTO   'ROLLOUTS IN EFFECT -          ROLLOUTS QUEUED -   REQUEST
               APPROVED -    '
         L         R13,SAVEAREA+4
         LM        R14,R12,12(R13)
         LCR       R1,R1
         BR        R14
         DS        0D
SAVEAREA DS        18F
WORK     DS        FL8
YES      DC        C'YES'
ROSTATUS DSECT
INEFFECT DS        H
QUEUED   DS        H
TCB      DSECT
         ORG       *+12
TIOTA    DS        F
TIOT     DSECT
JOBNAME  DS        FL8
         END
```

SEC IV

The Must Complete Function

System routines (routines operating under a storage key of zero) often engage in updating and/or manipulation of system resources such as system data sets, control blocks, queues, etc. These resources contain information critical to continued operation of the system and they must complete their operations on the resource. Otherwise, the resource may be left incomplete or may contain erroneous information -- either condition leads to unpredictable results.

The ENQ service routine provides the must complete function and ensures that a routine queued on a critical resource(s) can complete processing of the resource(s) without interruptions leading to termination. The must complete function places other routines (tasks) in a wait state until the requesting task -- the task (routine) issuing a ENQ macro instruction with the set-must-complete (SMC) operand -- has completed its operations on the resource. The requesting task releases the resource and terminates the must complete condition through issuance of a DEQ macro instruction with the reset-must-complete (RMC) operand.

Realize that, for the time it is in effect, the must complete function serializes operations to some extent in the computing system. Therefore, its use should be minimized -- use the function only in a routine that processes system data whose validity must be ensured.

As an example, in multitask environments, the integrity of the volume table of contents (VTOC) must be preserved during an updating process so that all future users may have access to the latest, correct, version of the VTOC. Thus, in this case, enqueue on the VTOC and use the must complete function (to suspend processing of other tasks) when updating a VTOC.

Just as the ENQ function serializes use of a resource requested by many different tasks, the must complete function serialize execution of tasks.

Characteristics of the Must Complete Function

When the must complete function is requested the requesting task is marked as being in the must complete mode and all asynchronous exits from the requesting task are deferred. Other tasks in the system (except the allowed tasks at the system level) or associated with the requesting task in a job step (step level) are placed in a wait state. Thus, tasks external to the requesting task are prevented from initiating procedures that will cause termination of the requesting task. Other external events, such as a CANCEL command issued by an operator, or a job step timer expiration are also prevented from terminating the requesting task.

The must complete mode of operation is not entered until the resource(s) queued upon are available.

At the system or step level, the requesting task can cause its own abnormal termination. If the requesting task does come to an abnormal termination before a reset condition has been effected, the operating system is stopped at the point of error to permit investigation of the trouble. It is then necessary to restart the system with the initial-program-load (IPL) procedure.

Levels of Use of the Must Complete Function

The must complete function can be applied at two levels:

The System Level: Only the requesting task, and system tasks included during system generation, are allowed to execute. All other tasks in the system are placed in a wait state.

The Step Level: In a region, only the requesting task is allowed to execute. All other tasks in the region, including the initiator task, are placed in a wait state.

CAUTION: Use of the must complete function at the system level should not be attempted until all aternatives have been exhausted. Except for extremely unusual conditions the system level of must complete should never be used.

Requesting the Must Complete Function

Request the must complete function by coding the set-must-complete (SM(GC) operand in an ENQ macro insturction. The format is:

```
Name      Operation Operand
[symbol] ENQ     ...,SMC= SYSTEM
                          STEP
```

Two parameters, SYSTEM and STEP, indicate the level to which the must complete function is to apply. The **Supervisor Services and Macro Instructions** publication describes the other operands of the ENQ macro.

Because of the properties of the TEST and USE parameters of the RET operand of the ENQ macro insturction, the SMC operand should be used only if the RET operand is to use the parameters HAVE, or NONE (in the E-form of ENQ), or if the RET operand is not used at all.

Request the must complete function only in routines operating under a protection key of zero. If the protect key is not zero, the task using the routine requesting "must complete" is abnormally ended.

Programming Notes

1. All data used by a routine that is to operate in the must complete mode should be checked for validity to ensure against a program-check interruption.

2. A routine that is already in the must complete mode should avoid calling another routine which also operates in the must complete mode. However, one level of nesting is permitted, when necessary, with the following cautions:

   a. A task may set the must complete mode for both the system and the step. If multiple settings are made for either the system or the step, only the first setting of each is effective -- the others are treated as no operation.

   b. The same is true for reset-must-complete. The first RMC for the system will reset the status of the system, the first RMC for the step will reset the status of the step, and all others will be treated as no operation.

3. Interlock conditions that can arise with the use of the ENQ function are discussed in the **Supervisor Services and Macro Instructions** publication.

   Additionally, an interlock may occur if a routine issues an ENQ macro instruction while in the must complete mode. The wanted resource may already be queued on by a task placed in the wait state due to the must complete request already made. Since the resource cannot be released, all tasks wait.

4. The macro instructions ATTACH, LINK, LOAD, and XCTL should not be used, **unless extreme care is taken**, by a routine operating in the must complete mode. An interlock

condition will result if a serially-reusable routine requested by one of these macro instructions has been requested by one of the tasks made non-dispatchable by the use of the SMC operand or was requested by another task and has been only partially fetched.

For example, suppose routine "b" in task B has requested and is using subroutine "c". Subsequently routine "a" in task A (of a higher priority than task B) receives control of the processing before routine "b" finishes with subroutine "c". If routine "a" issues an ENQ macro instruction with the SMC operand and puts task B (and, thus, routine "b") in a non-dispatchable condition, subroutine "c" remains assigned to routine "b". Now, if routine "a" issues a request (via a LINK, LOAD, etc. macro instruction) for subroutine "c", an interlock will occur between tasks A and B: task A cannot continue since subroutine "c" is still assigned to task B, and task B cannot continue (and thus release subroutine "c") because task A in the must complete mode has made task B nondispatchable.

5. The time a routine is in the must complete mode should be kept as short as possible -- enter at the last moment and leave as soon as possible. One suggested way is to:

   a. ENQ (on desired resource(s))

   b. ENQ (on same resource(s)),RET=HAVE,SMC= SYSTEM

                                STEP

   Item a gets the resource(s) without putting the routine into the must complete mode.

   Later, when appropriate, issue the ENQ with the must complete request (Item b). Issue a DEQ macro instruction to terminate the must complete mode as soon as processing is finished.

Terminating the Must Complete Function

Terminate the must complete function and release the resource queued upon by coding the reset-must-complete (RMC) operand in a DEQ macro instruction. The format is:

```
Name     Operation Operand
[symbol] DEQ     ...,RMC=  SYSTEM
                           STEP
```

The parameter (SYSTEM or STEP) **must agree** with the parameter specified in the SMC operand of the corresponding ENQ macro instruction.

Tasks placed in the wait state by the corresponding ENQ macro instruction are made dispatchable and asynchronous exits from the requesting task are enabled.

The PRESRES Volume Characteristics List

This chapter describes the creation and use of a direct access volume characteristics list that is placed in the system parameter library under the member name PRESRES (permanently resident and reserved).

Characteristics of the PRESRES Volume Characteristics List

The PRESRES volume characteristics list defines the mount and allocation characteristics of direct access device volumes used at an installation. Using the list predefines mount characteristics (permanently resident, reserved) and allocation characteristics (storage, public, private) for any, or all, direct access device volumes used by the installation. The **Job Control Language** publication describes volume characteristcs and the operating system's response to the various designations.

The scheduler compares the volume serial numbers in the PRESRES characteristics list with those of currently mounted direct access volumes after receiving control from the nucleus initialization program (NIP). Each equal comparison results in the assignment to the mounted volume of the characteristics noted in the PRESRES entry. (Fields in the unit control block for the device on which the volume is mounted are set to reflect the desired characteristics.) If the volume is: the IPL volume; the volume containing the data sets SYS1.LINKLIB, SYS1.PROCLIB, SYS1.SYSJOBQE; or a physically nondemountable volume (such as a 2301 Drum Storage Unit) the mount characteristic (permanently resident) has already been assigned and only the allocation characteristic is set.

A mounting list is issued for the volumes in the PRESRES characteristics list that are not currently mounted (except those for which mounting messages have been suppressed) and the operator is given the option of mounting none, some, or all of the volumes listed. The mount and allocation characterisitics for the volumes mounted by the operator are set according to the PRESRES list entry for the volume. The operator mounts the unit on the volume he selects.

The **Messages and Codes** publication describes the operator messages and responses associated with the use of the PRESRES volume characteristics list.

After the scheduler has finished PRESRES processing, reading of the job input stream begins, and the PRESRES list is not referred to again until the next IPL.

Note:

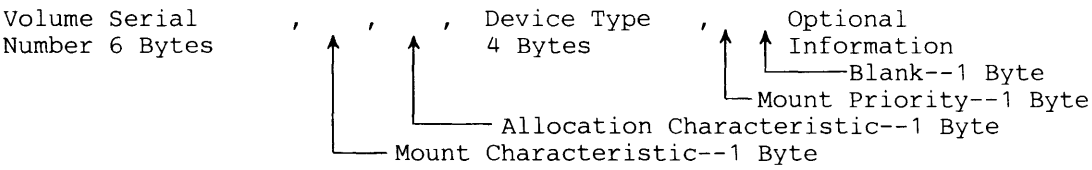
1. A PRESRES entry identifying a physically nondemountable volume will appear in the mount list issued to the operator if the volume (device) is OFFLINE or is not present in the system.
2. Use of the PRESRES list can only be suppressed by deleting the member from the parameter library (SYS1.PARMLIB).
3. Only the first 102 volumes on the PRESRES list can be placed on the mount list.

Users can use a PRESRES characteristic list entry or refer to the volume in the input stream to assign an allocation characteristic other than "public" to volumes whose mount characteristic is "permanently resident".

Selection of the volumes for which PRESRES entries are to be created should be done so that critical volumes are protected. Since the combination of mount and allocation characteristics assigned to a specific volume determine the types of data sets that can be placed on the volume and its usage, you can exercise effective control over the volume through a PRESRES list entry.

Writing the PRESRES Entry Format

Each PRESRES entry is an 80-byte record, consisting of a 6-byte volume serial number field, a 1-byte mount characteristic field, a 1-byte allocation characteristics field, a 4-byte device type field, a 1-byte mount-priority field, and an optional information field. Commas are used to delimit the fields, except the optional information field is **always** preceded by a blank. All character representation is EBCDIC. This format is shown below.



The volume serial number consists of up to six characters, left justified.

Mount characteristics are defined by:

0 to denote permanently resident

1 to denote reserved

The default characteristic is "permanently resident" and is assigned if any character other than 0 or 1 is present in the field.

Allocation characteristics are defined by:

0 to denote storage

1 to denote public

2 to denote private

The default characteristics is "public" and is assigned if any character other than 0, 1, or 2 is present in the field.

The device type is defined by: A four-digit number designating the type of direct access device on which the volume resides, e.g. the IBM 2311 Disk Storage Drive is indicated by the notation 2311. Note that is field only indicates the basic device type for the associated volume. **Advise the operator if the device requires special features (such as track overflow) to process the data on the designated volume.**

The mount priority field is used to suppress mount messages at IPL time for a volume; the alphabetic character N should be inserted in this field to suppress the mount message. This field allows the user to list seldom used volumes in the PRESRES list without having a mount message issued at each IPL. When these volumes are required, they may be mounted and attributes will be set from the PRESRES list entry. If the user does not wish to have the mount message suppressed, he may omit the mount priority field and the preceding comma.

The optional information field contains: Any descriptive information about the volume. This information is not used by the system, but will be available to the user on a printout of the list. If necessary, comments may start in the second byte after the mount priority field or if the mount priority field is omitted, in the second byte following the comma after the device type field.

Embedded blanks are not permitted in the volume serial, mount, allocation, or device type fields.

Adding the List

The IEBUPDTE utility program places the list (under the member name PRESRES) in the system parameter library, SYS1.PARMLIB. This utility is also used to maintain the list.

The following pages contain sample coding illustrating the linkage to the appendage. In the example given, an Appendage II which approves the rollout of job steps with a higher priority that the requesting job step is used to illustrate appendage coding.

Task Directory for Section IV: Modifying the System

For information about:

- Reader cataloged procedures, see
  Reader Procedures
  RA - Automatic SYSIN Batch Reader Procedure
  RB - Reader for Blocked SYSIN data
  RU - Reader Procedure for Unblocked SYSIN

- Writer cataolged procedures, see
  System Output Writer Procedures
  Command Chaining
  Direct SYSOUT Writer Procedures
  Choosing Direct SYSOUT Writers
  WC - Writer with Command Chaining
  WU - Writer for Special Chains

- Initiator cataloged procedures, see
  Initiator Procedures
  Mounting Control Volumes
  Initiator Cataloged Procedures

- Dedicated data set cataloged procedures, see
  Procedure INITD

- Using dedicated data sets, see
  How to Dedicate a Data Set
  How to Use a Dedicated Data Set
  Use of Dedicated Data Sets by Processor Programs for Utility Data Sets
  System Library Data Sets as Dedicated Data Sets
  Disposition of Temporary Data Sets

- Writing JCL for user-written reader cataloged procedures, see
  The EXEC Statement
  DD Statement for the Input Stream
  DD Statement for the Procedure Library
  DD Statement for the Spooling Data Set

- Writing JCL for user-written initiator procedures, see
  The EXEC Statement
  DD Statement Formats

- Writing JCL for dedicated data sets, see
  INITD Procedure Statements
  The EXEC Statement
  DD Statement for the Dedicated Utility Data Set
  DD Statement for the Loadset Data Set

- Writing JCL for user-written system output cataloged procedures, see
  The EXEC Statement
  DD Statement for the Output Data Set

- Writing JCL for user-written direct SYSOUT cataloged procedures, see
  The EXEC Statement
  The DD Statement

- Writing SVC routines, see
  The Resident SVC Routines Option
  Characteristics of SVC Routines
  Programming Conventions for SVC Routines
  Writing SVC Routines
  Adding SVC Routines Into the Control Program
  Specifying SVC Routines
  Inserting SVC Routines During the System Generation Process

- Communicating with the control program, see
  Programming Conventions for WTO/WTOR Routines
  Writing a WTO/WTOR Routine

- Increasing system response, see
  Procedures for Using the Link Pack Area
  The Resident BLDL Table Option
  Selecting Entries for the Resident BLDL Table
  The Resident Access Method Modules Option
  The Resident SVC Routines Iption
  The Resident Error Recovery Procedure Option
  Characteristics of SVC Routines

- Increasing user convenience, see
  Characteristics of an Output Separator
  Writing an output Separator Program
  Characteristics of the Output Writer
  Writing an Output Writer
  Characteristics of Rollout/Rollin Installation Appendages
  Characteristics of the Must Complete Function
  Levels of Use of the Must Complete Function
  Requesting the Must Complete Function
  Terminating the Must Complete Function

- Using library lists, see
  Procedure for Using the Link Pack Area
  List IEABLD00
  List IEAIGG00
  List IEARSV00
  Example of Link Pack Area Specification
  The Link Library List
  Characteristics of the PRESRES Volume Characteristic List
  Writing the PRESRES Entry Format
  Adding the List

- Increasing system efficiency, see
  Logical Track Size -- JOBQFMT
  Reserving Initiator Queue Records -- JOBQLMT
  Number of Generation Data Groups
  Number of Passed Data Sets
  Number of I/O Devices for Passed Data Sets
  Number of Volumes

Number of System Messages
Use of Automatic Restart
Reserving Write-to-Programmer Queue Records -- JOBQWTP
Reserving Queue Records for Cancellation -- JOBQTMT
Number of Devices
Number of Jobs

- Writing accounting routines and gathering accounting information, see
  Programming Conventions for Accounting Routines
  Input Available to Accounting Routines
  Adding an Accounting Routine
  Output From Accounting Routines
  Adding the Accounting Data Set Writer

Section V: Logic Summary

This section contains detailed descriptions of the five major routines of the MVT control program. The five routines described are:

- Job management
- Task management
- Data management
- Volume management
- Recovery management

Job Management Routines

This topic describes the command processing and the job processing routines of job management.

Command Processing

Processing of commands has three phases:

- Reading the command
- Scheduling the command
- Executing the command

These three phases are performed under various system tasks. For some commands, all three phases are performed as part of one task. However, processing of most commands requires several system tasks. Figure 15 shows the relationship between the system tasks and phases of command processing.

Reading the Command

Operator commands are entered into the system through either a console device or an input job stream. Reading of commands entered via a console device is performed by routines operating under the console communications task; reading of commands entered via an input job stream is performed by routines operating under the reading task associated with that input job stream.

Console Communications Task

The console communications task is created at system generation (SYSGEN) time when its task control block (TCB) is assembled. The console communications task does not have a region of its own, its storage requirements are filled from the region of another command processing task -- the master scheduler task.

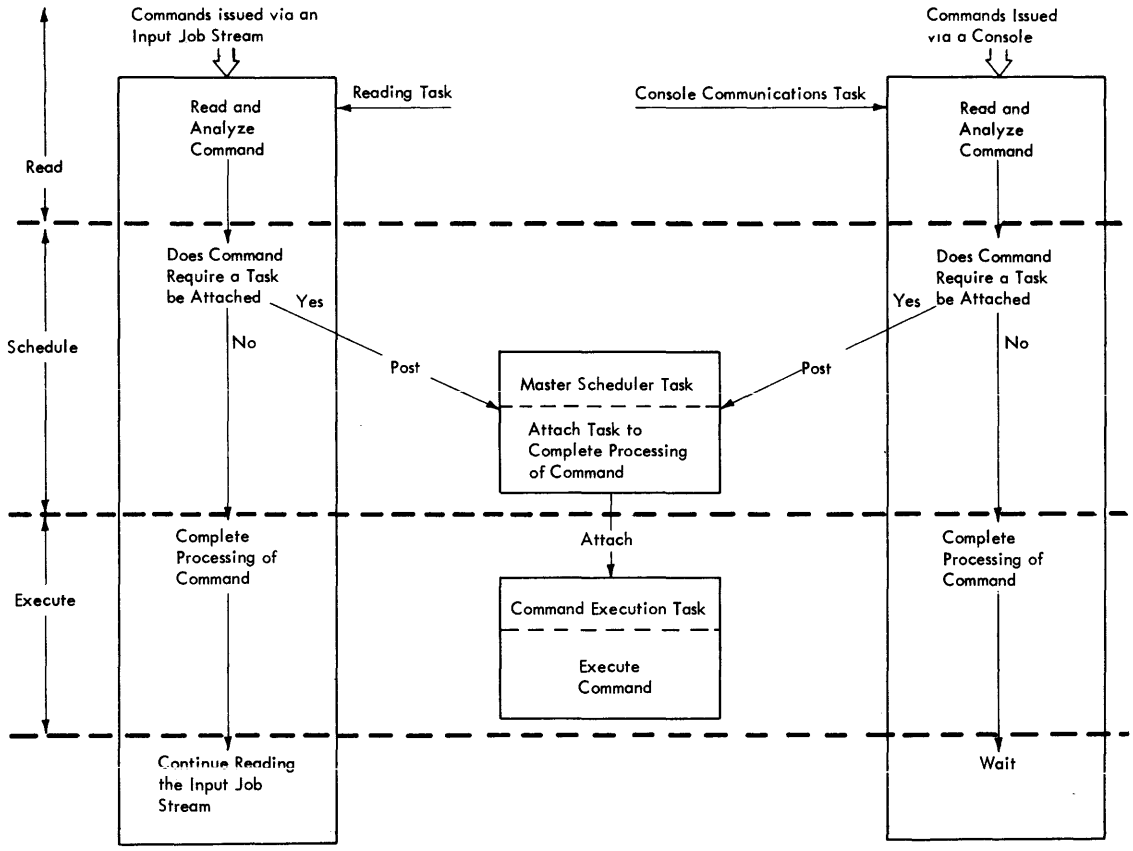
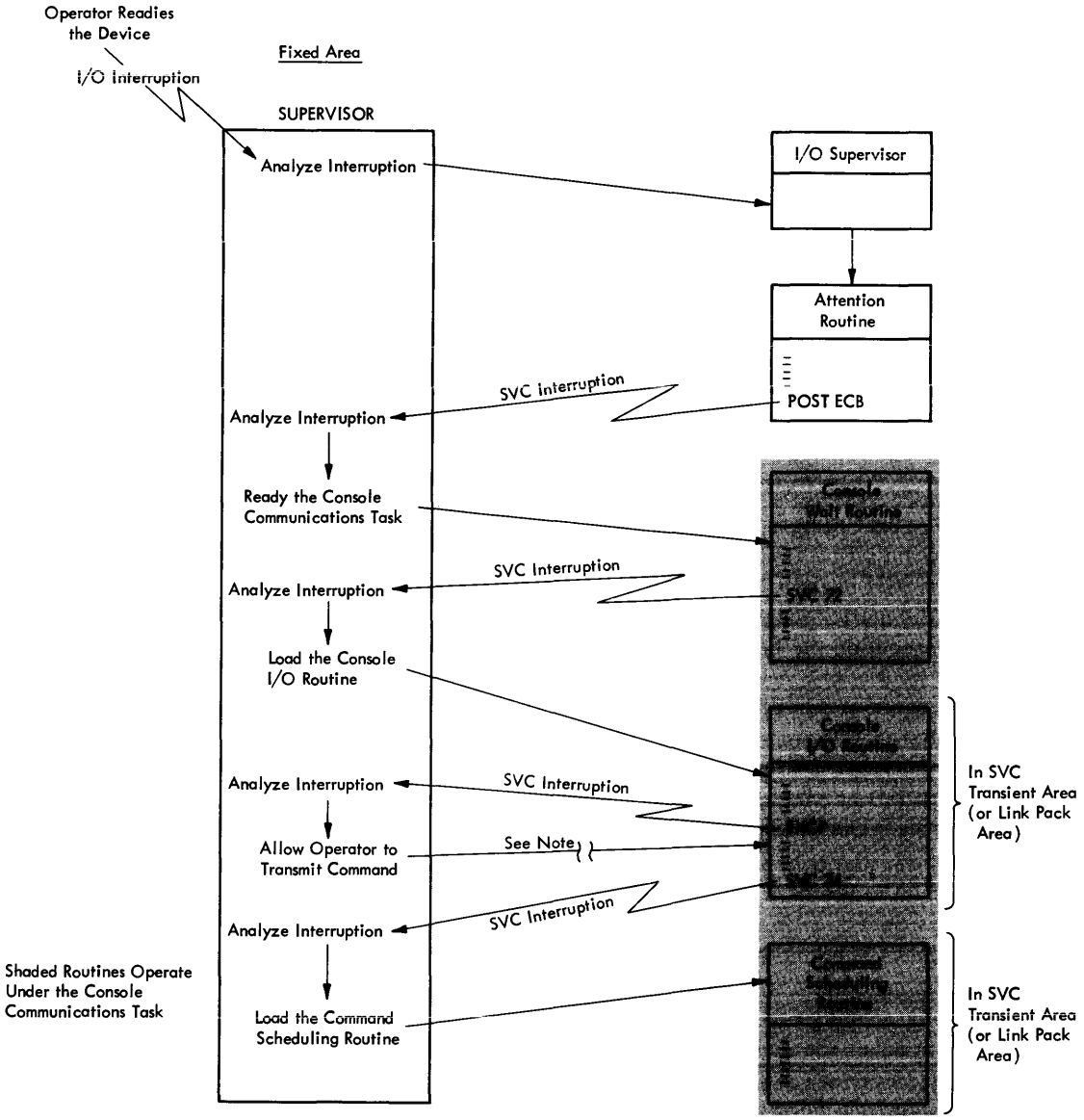


Figure 15. Relationship Between Tasks and Phases of Command Processing

Figure 16 shows the flow of control when a command is issued via a console device. The operator pushes the "Attention" button or readies that device, causing an I/O interruption to occur; CPU control is passed first to the supervisor and then to the I/O supervisor. After the I/O Supervisor determines the cause of this interruption, it passes control to an attention routine which issues a POST macro instruction for an event control block (ECB) being awaited by the console communications task. The supervisor's processing of this "posting" includes readying the console communications task, and passing CPU control to a routine of this task.

The console wait routine resides in the nucleus, and is the controlling routine in the console communications task. It is the first routine that receives control for this task, and is the routine that issues the WAIT macro instruction to return the task to the wait state when it is complete. The console wait routine receives CPU control from the supervisor, and passes control to the appropriate processing routine.

There is a console I/O routine (device support routine) for each type of device used as a console. Each routine performs the same function, but one routine varies slightly from another because each type of device operates differently. The routine requests main storage to receive the command and issues an EXCP macro instruction that enables the operator to transmit the command. When the command has been transmitted, the command scheduling routine schedules commands that require another system task, and executes commands that do not.



Note: While Command is being transmitted, other processing is performed. Control returns to the Console I/O routine after the command is transmitted.

Figure 16. Flow of Control When a Command is Issued via a Console

Reading Tasks

When a command is issued via an input job stream, the initial processing and analysis of the command is part of the reading task associated with that input job stream (see Figure 15). The interpreter control routine of the reading task detects the command and invokes the command scheduling routine. This routine analyzes and schedules the command for the reading task in the same way as it does for the console communications task. After completing its operation for a reading task, the command scheduling routine returns CPU control to the interpreter control routine which continues reading the input job stream.

Scheduling the Command

Scheduling a command is the storing of the command, and the readying of another system task to continue the command's processing. The command scheduling routine (a type 4 SVC routine) operates under either the console communications task (when the command was issued via a console device), or a reading task (when the command was issued via an input job stream).

If execution of the command does not require additional tasks, the command scheduling routine executes the command, and returns control to the routine that called it. In the console communications task, a console I/O routine receives control and it passes control to the console wait routine; the console wait routine places the task in the wait state. In the reading task, control is returned to the interpreter control routine which continues reading the input job stream.

However, processing of the command usually requires additional system tasks for execution. If the task for executing the command does not yet exist, the command scheduling routine schedules execution of a command by creating a command scheduling control block (CSCB), placing the CSCB in the CSCB queue, and posting an event control block (ECB) being awaited by the master scheduler task. The routines of the master scheduler task attach the task needed for executing the command.

If the system task for executing the command is already in the system, an ECB in the CSCB for that task is posted.

The CSCB queue contains a CSCB for each command that has a task created for its execution. Command processing routines use this queue to associate commands with the tasks or functions that the commands affect. When creation of a new task is required, a CSCB is created, and an indicator is set in it (i.e., the CSCB is made pending.)

The routines of the master scheduler task check the CSCB queue and attach a task for each CSCB that is pending. The attach routine of the master scheduler task scans the CSCB queue until it finds a pending CSCB. When one is found, the attach routine removes it from pending status, and attaches the appropriate task. The attach routine then continues scanning the CSCB queue, attaching a task for each pending CSCB. When no more pending CSCBs are found, the wait routine of the master scheduler task causes the task to be placed in the wait state.

Executing the Command

A given command is executed under a system task that is in one of three categories:

- The task is the same task under which the command was read (i.e., the console communications task or a reading task).
- The task is already in the system and has functions other than execution of the command.
- The task is created especially for execution of this command.

A discussion of which commands are in each of the three categories, and the tasks required for processing of these commands is given in the **MVT Job Management PLM**. Relatively few commands are executed under the task that read them. When the command scheduling routine detects such a command, it performs all the required processing and returns control to the routine that called it.

Some tasks can include the processing of more than one type of command. For example, the routines that process a MODIFY command can operate under control of a task created in response to an earlier START command.

The job processing tasks (i.e., reading, writing and initiating tasks) are created especially for execution of a given command. Each time a START command is issued, a new job processing tasks is created for the requested function.

Job Processing

Job processing is made up of three types of tasks:

- Reading tasks, which control the reading of input job streams and the interpreting of the control statements in these input streams.
- Initiating tasks, which control the initiating of job steps whose control statements have been read and interpreted. Termination procedures are also part of initiating tasks.
- Writing tasks, which control the transferring of system messages and user data sets from direct access volumes on which they were initially written to some other external storage medium (usually a tape, printer, or punch).

When system management facilities are included in the system, another type of task, the system management facilities (SMF) task, is created and placed into a wait state during master scheduler initialization. Then, while tasks are being performed, system management data is accumulated in a buffer. Every time the buffer becomes filled, the ECB for the SMF task is posted and SMF routines write the contents of the buffer into the system management data set.

Reading, initiating, and writing tasks are created in response to START commands for readers, initiators, and writers, respectively. Figure 17 shows the processing performed to attach a job processing task for a START command.

Whenever a START command is issued, the resultant command processing includes the attaching of a "START-command" task by a routine of the master scheduler task. A routine (the system task control routine) of the START-command task determines what is to be started, and then attaches either a reading, a writing, or an initiating task.

When the START-command task was attached, it was assigned a region large enough for the requirements of the system task control routine. This region is released by a system task control routine module in the link pack area unless the reading, writing, or initiating task to be attached is also going to use this region (instead of being assigned a new region by the supervisor). This module in the link pack area then places the START-command task in the wait state.

When its associated reading, writing, or initiating task is terminated, a module of the START-command task detached the terminated task, and performs some termination processing. The system task control routine then passes control to the supervisor which terminates the START-command task.

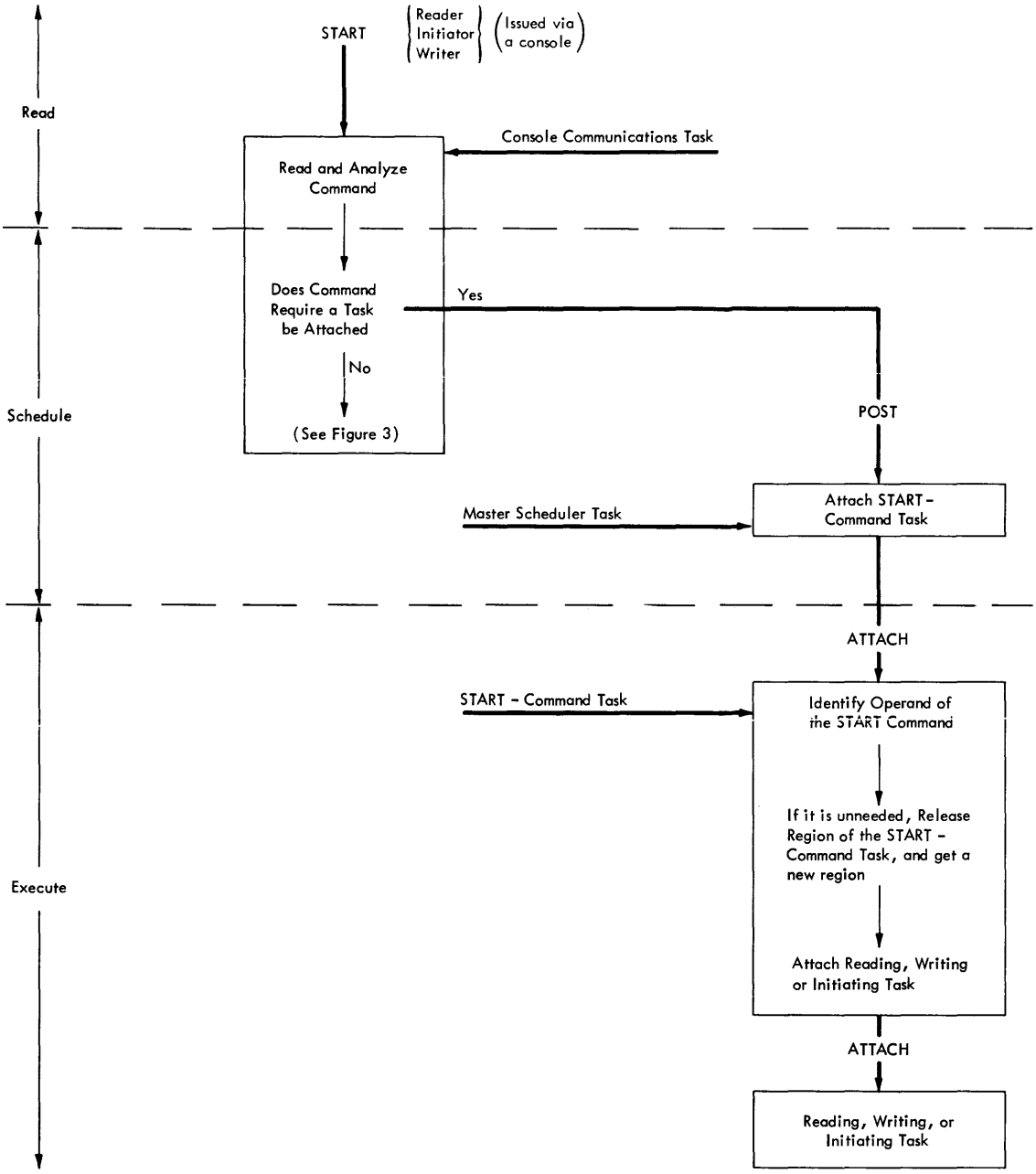


Figure 17. Processing to Create a Task for START Command

There may be more than one of each of the job processing (reading, writing, initiating) tasks. You may have input job streams read from several input devices by issuing a START reader command for each input stream. Each command results in a reading task being attached for the associated device. You may have system messages or data sets being written on several output devices by issuing a START writer or DSO command for each device; a writing task results from each command. An initiating task is created in response to a START initiator command. Up to 15 initiating tasks (in addition to other system tasks) can exist concurrently.

The routines that perform the three job processing functions are sometimes called the **job scheduler**.

Reading Tasks

The major functions of a reading task are the reading of an input job stream, and the building of control blocks and tables from the job control statements in that job stream. These blocks and tables contain information required during the performance of the initiating tasks in the system. Routines of reading tasks also process commands and store data encountered in the input streams.

The primary routine of a reading task -- the interpreter control routine -- reads the input stream, builds the blocks and tables in the region of its reading task, and invokes a queue manager routine to write these blocks and tables on the SYS1.SYSJOBQE data set. This routine resides on SYS1.LINKLIB and operates from the region of its reading task unless it (the routine) is already in the link pack area.

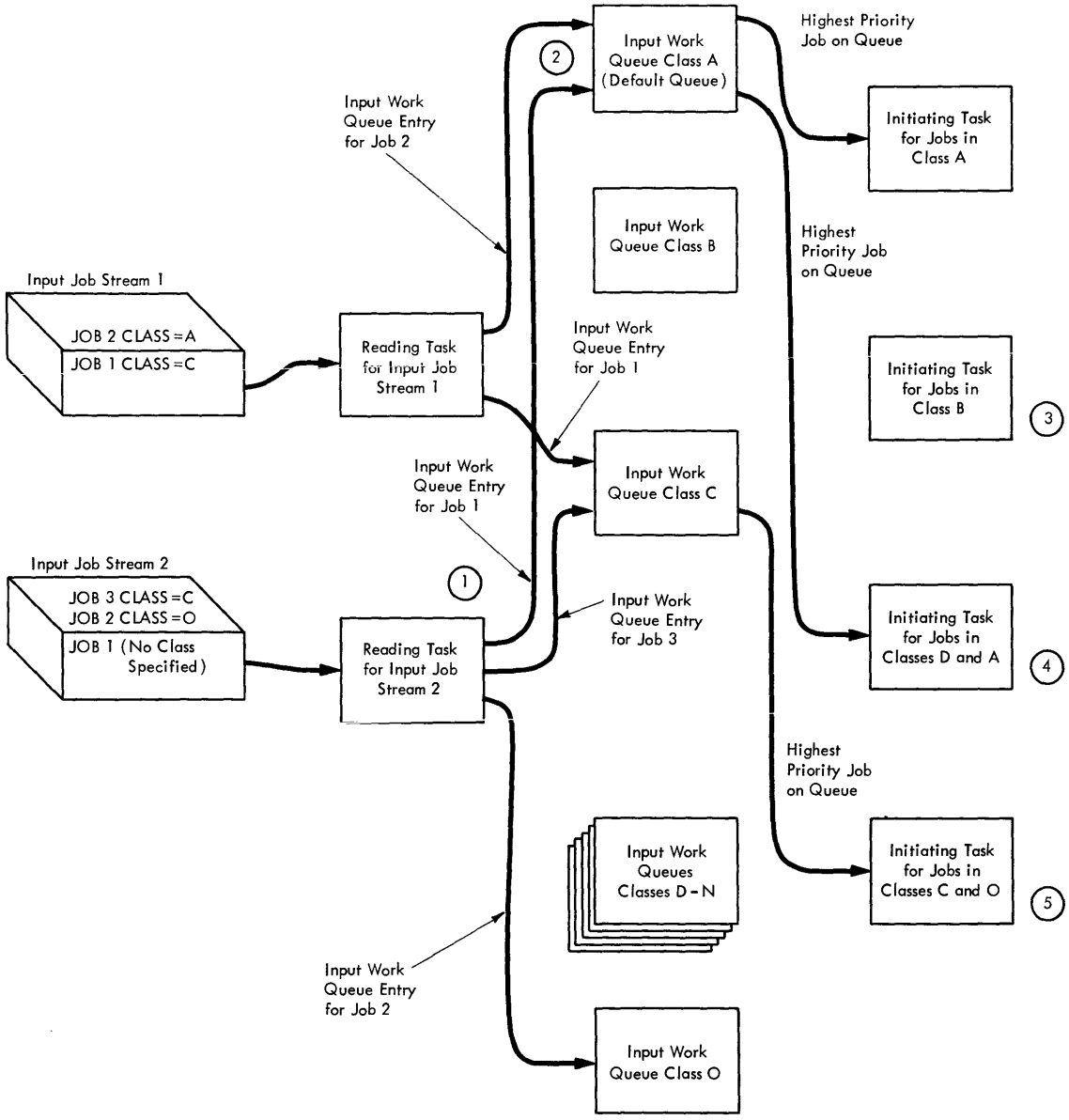
If system management facilities are included with the system, the interpreter control routine can invoke a user-written routine before processing each job control statement. The user-written routine can check and change the contents of the job control statements and can also indicate whether or not the job should be canceled.

When construction of the control blocks and tables for a given job is complete, the interpreter control routine invokes a queue manager routine which enqueues them in an input work queue. The enqueued blocks and tables of a given job are referred to as a work queue entry. Figure 18 shows an example of the relationship between the input work queues and the contents of input job streams.

Input Work Queues

The input work queues are made up of control blocks and tables containing information needed to initiate job steps. The blocks and tables of a given queue are records chained together in the work queue data set (SYS1.SYSJOBQE).

There are 15 input work queues -- one for each of the job classes that can be specified in the JOB statement. These queues are identified by the 15 alphabetic characters -- "A" through "O". The interpreter control routine has a given work queue entry placed in the queue that corresponds to the CLASS parameter in the JOB statement of that entry. Work queue entries in a given queue are ordered according to the priorities of the jobs and the times of submission to the system. They remain in the queue until they are dequeued by an initiator routine that is seeking a new job to initiate.



1. The queue for class A is used because no class was specified in the JOB statement.
2. Entries in the same input work queue are enqueued according to priority.
3. This task is in the wait state if there are not entries in the queue for class B.
4. Queue for class D is checked for entries before queue for class A.
5. Entries in queue for class C are always processed first regardless of priorities of entries in queue for class O.

Figure 18. Example of Processing of Input Work Queues

Contents of a Work Queue Entry

The major control blocks and tables in a work queue entry are:

- Job control table (JCT), which is built for each job from information in the JOB statement. This table contains job and job step attributes.

- Step control table (SCT), which is built from information in the EXEC statement. This table contains job step attributes.
- Step input/output table (SIOT), which is built for each DD statement and contains information needed to assign devices to the data set defined in the statement.
- Job file control block (JFCB), which is built for each DD statement and contains data set attributes. Construction of a JFCB is completed when its associated data control block is opened.

Figure 19 shows the flow of information between the control statements and these blocks and tables.

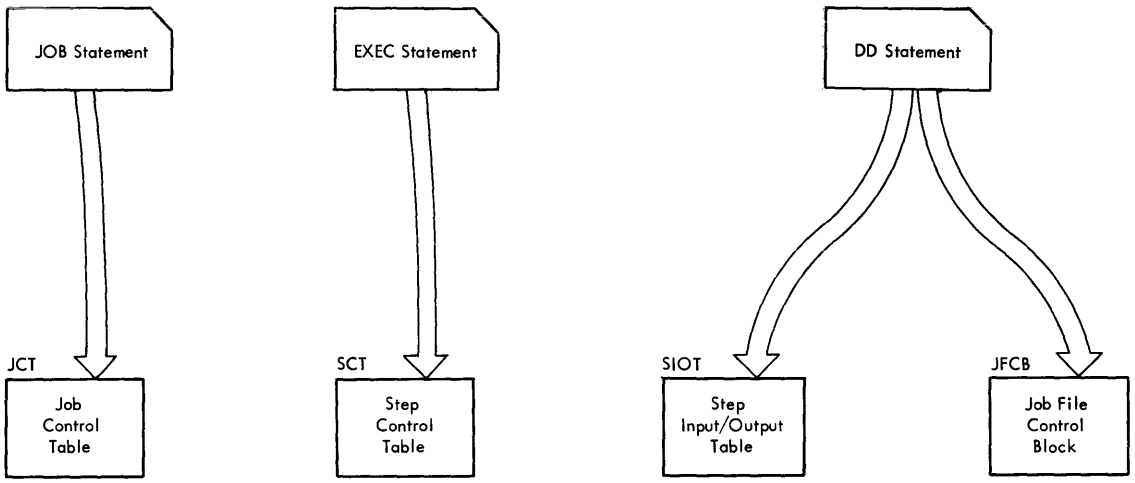


Figure 19. Information Flow Between Control Statements and Blocks of a Reading Task

Commands and Data Sets

When the interpreter control routine encounters a command in an input job stream, it invokes the command scheduling routine, and processing of this command becomes part of the reading task associated with the input device (see Figure 15). When a data set is encountered in an input job stream, it is written on a direct access volume.

Termination of a Reading Task

A reading task is terminated either by a STOP reader command, or when an end-of-data condition is detected on the associated reader device.

Initiating Tasks

An initiating task has two parts, the preparing of job steps for execution, and the performing of termination processing when job steps are complete.

Preparing of Job Step for Execution

Preparing a job step for execution consists of:

- Acquiring a region of main storage
- Locating data sets that are input to the job step

- Assigning I/O devices required for the job step
- Reserving auxiliary storage for data sets created during the step
- Attaching a task for the job step.

An initiator routine invokes a queue manager routine to dequeue the first entry on one of the input work queues. (This is the entry of the highest priority job on that queue that is not already being initiated under some other initiating task.) Figure 18 shows an example of relationships between the input work queues and initiators.

Each initiating task is assigned to one or more job classes via the parameter field of the associated START initiator command. The classes specified determine the queues from which this initiating task selects jobs. When an initiating task is assigned to more than one job class, the order in which the queues were specified in the START command determines the order in which the queues are checked.

Once a work queue entry (i.e., a job) has been assigned to an initiating task, initiation, execution, and termination of the steps of that job are performed sequentially under control of that initiating task. Multiprogramming is obtained when several initiating tasks are concurrently controlling the initiation, execution, and termination of the steps of different jobs.

In a multiprogramming environment, one data set may be required for two or more jobs that are operating concurrently. Before starting to initiate the first step of the job, the initiator invokes the ENQ/DEQ routine of the supervisor to determine if any of the data sets required during this job are currently being used in another job, and cannot be shared. If a data set required by any step in the job is not currently available, initiation of the first job step is suspended until the data set becomes available. Multiple use of a data set is allowed only when all users have specified that the data set can be shared.

If system management facilities are included with the system, the initiator can invoke and provide information to a user-written routine before starting to initiate a job or any of its steps. When it returns control to the initiator, the user-written routine can indicate whether or not the job should be canceled.

Acquiring a Region: The initiator does not obtain a region for a step until the main storage requirements of that step are determined. When an initiator is first started, its initial main storage requirements are filled from the region assigned when this initiating task was attached. After an initiator has processed at least one job step, it uses the region of the last step terminated until the current step's requirements are determined from the SCT.

After the SCT is read, the initiator determines the main storage requirements of the step, releases the region currently being used, and requests a new region that meets the storage requirement of either the job step or the initiating task -- whichever is larger. If there is not enough contiguous storage in the dynamic area to fill the request for the region, or if there is not enough space in the supervisor queue area for the tables and queues needed for the job, the initiating task is put in the wait state until enough storage is available. The device allocation routine is then brought into the region to continue initiating the step.

Locating Input Data Sets: The device allocation routine determines which volumes contain input data sets from either the DD statement, or a search of the catalog. (A catalog management SVC routine is invoked to perform this search.) Once the volumes that contain the data sets are determined, the device allocation routine determines if any I/O devices are available for these volumes.

Assigning Input/Output Devices: A job step cannot be initiated unless there are enough I/O devices to fill its needs. The device allocation routine determines whether the required devices are available. If there are sufficient devices available, they are assigned to the step; if there are

not, a message is issued to the operator. If the operator cannot make the required number of devices available, he may have the initiating task put in the wait state until sufficient devices are available, or he may cancel the job.

After devices are allocated, the device allocation routine builds a task input/output table (TIOT) from information in the JCT, SCT, and SIOTs of the job step. The TIOT has one entry for each data set used during the job step; this entry contains pointers to other control blocks needed in the processing of the data set. The TIOT is in the system queue area, and exists for the life of the job step. Figure 20 shows the relationship of the TIOT to other job processing control blocks. The shaded portion of the figure shows the major blocks and tables built during a reading task (see Figure 19). The nonshaded portion of Figure 20 shows the major blocks and tables created during initiating tasks.

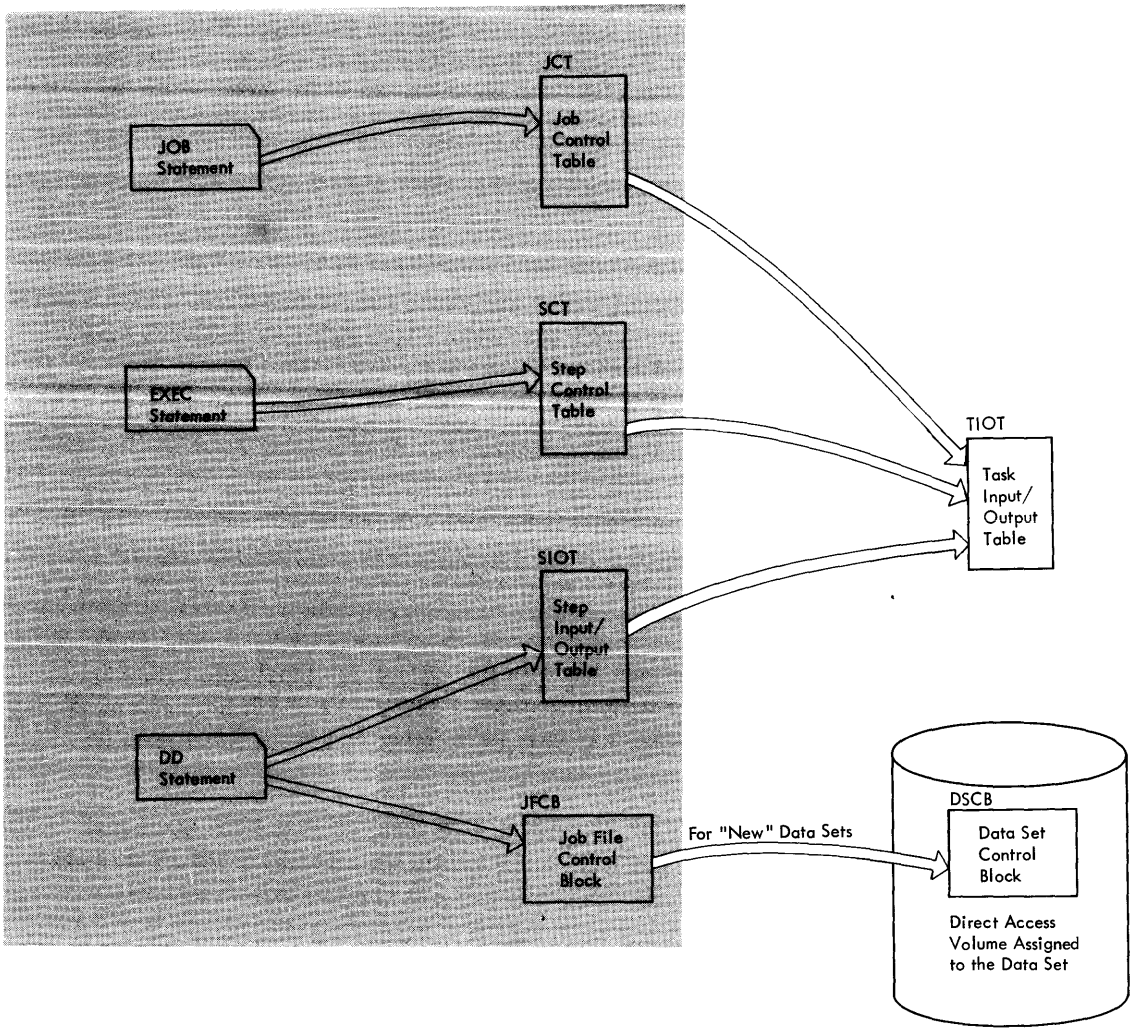


Figure 20. Relationship of Block of Initiating Task to Blocks of Reading Task

Reserving Auxiliary Storage Space: Any direct access volume space required by the output data sets of a job step, if not being written by DSO, is acquired at the request of the device allocation routine by a direct access device space management (DADSM) routine of data management. If the volumes specified by the device allocation routine do not have the required space, the DADSM routine returns control to the device allocation routine which issues an

operator message. The operator must then either make a new volume available, cancel the job, or indicate that the initiating task be put into the wait state until space becomes available.

When direct access volume space is being assigned, a DADSM routine partially builds a data set control block (DSCB) for the output data set that will occupy the space. The DSCB is the label for that data set and contains data set characteristics obtained from the JFCB, and the track addresses assigned to the data set. Construction of DSCBs for output data sets is completed when the associated data control block (DCB) is opened.

Attaching a Task for the Job Step: The final operation in the initiation of a job step is the creating (attaching) of a task for that job step. At this point, if the requested region is less than MINPART and the termination modules were loaded in the link pack area at IPL time, then the system may reduce the current initiator region size by as much as 40K bytes for problem program execution to conserve main storage. The step attach routine issues an ATTACH macro instruction causing CPU control to be given to the attach SVC routine. This routine builds a task control block (TCB) which the control program uses to control the job step processing; this TCB is the job step TCB. It is placed in the TCB queue according to the priority of the related job.

Since initiation of the step is complete, the initiating task is placed in the wait state until the step is to be terminated.

Terminating a Job Step

A job step is terminated either when it is complete, when a specified time interval expires, when an error prevents any more processing, or when the job is canceled by the operator. Any of these conditions cause the supervisor to make ready the initiating task that controlled initiation of that step. Termination processing for the step is performed under control of this initiating task.

The termination routine of job management is brought into the region assigned to the step. This routine disposes of data sets created or used during the job step, and releases the I/O devices assigned to the job step. The initiator routine issues a DETACH macro instruction to remove the job step TCB from the system.

If system management facilities are included with the system, the termination routine can invoke and supply information to user-written routines during step and job termination. During step termination, the user-written routine may indicate whether or not the job should be canceled.

After terminating the last step of a job, the initiator routine performs some additional processing. It deletes the work queue entry for the job from the SYS1.SYSJOBQE data set, and makes entries in output work queues for system messages and SYSOUT data sets.

The output work queues reside in SYS1.SYSJOBQE. They are used by writing tasks to indicate the SYSOUT data sets and system messages that are to be written. There are 36 output work queues, one for each SYSOUT class. A SYSOUT class is an alphabetic or numeric designation by which the user can group his messages and SYSOUT data sets.

An output work queue is made up of entries containing data set blocks (DSBs) and system message blocks (SMBs) for data sets and system messages in a given class. During step termination, the initiator routine builds a DSB for each SYSOUT data set created during that step. SMBs are created for system messages as they are generated. Messages are blocked so one SMB can contain several messages.

All the DSBs and SMBs in a given class for a given job are referred to as an output work queue entry for that class. After the last step of a job is terminated, the initiator routine invokes a queue manager routine to enqueue the output work queue entries from that job into the appropriate output work queues.

Restarting a Job Step

If a job step is terminated before successful completion, checkpoint/restart routines can make it possible to resume execution from the beginning of the step or from a place within the step. Either way, the restart can be made to occur after resubmissions of the job by the programmer or it can be made to occur automatically when the failure occurs.

Checkpoint/restart routines include a checkpoint routine and several restart routines.

The checkpoint routine is an SVC routine that is invoked when an SVC 63 instruction is encountered in a processing program. It gathers and records on a checkpoint data set enough information about the status of the job step and its related control blocks to allow a restart to be made from the place where the checkpoint is taken. If a failure occurs, the checkpoint data set is used by restart routines to return the job step to main storage so that execution can be resumed.

The restart routines can be invoked when a job step is resubmitted for restart, or they can be invoked automatically when a failure occurs. The functions performed by restart routines depend upon the type of restart that is requested.

If the restart is to be made from the beginning of a job step, there is no need for a checkpoint data set and one is not used. Also, unless the step is to be restarted automatically, no restart functions are performed until the step is resubmitted. At that time, the RESTART parameter of the JOB statement contains the name of the step to be restarted, and routines of the reading task simply bypass preceding steps and begin processing with the named step.

If a step is to be restarted from the beginning, automatically, then restart processing begins during step termination. The step termination routine of job management invokes restart preparation and activation routines, which verify that a restart can be performed and request the operator to authorize the restart. These routines also dispose of data sets created during execution of the step, and cause the master scheduler to create a restart reader task. Routines of the restart reader task reinterpret the job (job control statements are found in SMBs), and create a new work queue entry for the step to be restarted. The step is then selected and initiated via normal initiator processing.

If a step is to be restarted from a place where a checkpoint was taken and the job is resubmitted, the RESTART parameter of the JOB statement will identify the step, while a SYSCHK DD statement will describe the checkpoint data set. When routines of the reading task encounter the SYSCHK DD statement, they create a job step to be initiated before the one that is to be restarted. The purpose of this step is to ensure that I/O device allocation for the restarting step will be equivalent to that which existed at the time the checkpoint was taken. It also causes the restart SVC routine to be invoked. The restart SVC routine opens the checkpoint data set, returns the restarting job step and its related control blocks to main storage, and ensures that volumes needed by the step have been mounted and repositioned. Then, to cause a restart, it places a pointer to the instruction at which execution is to begin into the RB of the restarting step.

If a step is to be restarted automatically from a place where a checkpoint was taken, the processing is as follows:

- The step termination routine invokes the restart preparation and activation routines and ensures that all data sets for the step are kept.
- The master scheduler creates a restart reader task.
- The job is reinterpreted.
- Compatible allocation is performed.
- The restart SVC routine restores the job step to main storage.

Writing Tasks

A writing task controls the writing of system messages and SYSOUT data sets in a specified class (or classes) from the direct access volume on which they were initially placed to a specified SYSOUT device. A writing task is created as a result of a START writer command that specifies a class or classes of messages and data sets to be processed. When all outstanding SYSOUT data sets and messages in that class are written, the writing task is placed in the wait state, until more data of that class becomes available or until a STOP writer command is issued.

The writer routines indicate to the queue manager routines which message class is to be written. The queue manager routine passes the first output work queue entry from the appropriate output work queue to the writer routines which write the associated messages and data sets. After all messages and data sets from this queue entry are written, the writer routine requests the next entry from the queue, and upon receiving it, writes the messages and data sets. When all the messages and data sets in all the classes associated with this writing task are written, the writing task is placed in the wait state. A writing task is again made ready when an output work queue entry is placed in one of the queues associated with this task.

Job Management in a Multiprocessing Environment

Basic job management functions are essentially unchanged from MVT in a multiprocessing environment. In response to an extended VARY command, command processing routines can logically place offline or online a CPU, an area of storage, or channels. With the Model 65 multiprocessing System, the operator uses the VARY command to logically reconfigure the system. In response to a QUIESCE command, processing routines can allow activity on I/O devices to be completed to permit the physical removal of an element from the system. Job processing routines continue to maintain a single SYS1.SYSJOBQE data set. Those special functions that are performed by job management routines for a multiprocessing system are described in the **MVT Job Management PLM.**

Task Management Routines

This chapter describes the five routines used by Task Management. These routines are:

- Interruption Supervision
- Task Supervision
- Main Storage Supervision
- Contents Supervision
- Timer Supervision

Interruption Supervision

CPU control is passed to the supervisor via an interruption. The supervisor analyzes the interruption to determine which control program routine is to process it. An interruption can occur either because the interrupted program has requested some control program service, or because some event requires supervisory processing. There are five types of interruptions:

- SVC interruption, which occurs when an SVC instruction is executed.
- Input/output interruption, which occurs when an I/O operation terminates, or an I/O device is readied.
- Timer-external interruption, which occurs when an event (e.g., a timer or external signal) indicates the need for control program processing.
- Program interruption, which occurs when either a program attempts an invalid operation (e.g., execution of a privileged instruction by a program in the problem state), or a data error (e.g., overflow) is detected.
- Machine-check interruption, which occurs when the CPU detects a hardware malfunction.

Any interruption except a machine-check interruption causes CPU control to be taken from the interrupted program and given to an interruption handling routine of the supervisor. There are four interruption handling routines, one for each type of interruption except machine-check. A machine-check interruption causes control to be passed directly to a recovery management routine (either SER0, SER1, or MCH), if one is in the operating system. If none of these recovery management routines are provided, the system is placed in the wait state.

The interruption handler does not process the interruption, but analyzes it and passes control to the proper interruption processing routine. The SVC interruption handler loads (if necessary) the SVC routine indicated in the SVC instruction and passes control to it. The I/O interruption handler passes control to the I/O supervisor. The timer/external interruption handler determines what caused the particular interruption and passes control to the appropriate routine. The program interruption handler determines whether the user has provided a routine to process this particular type of program interruption. If such a routine is provided, it is given control; if not, the task being performed by the interrupted program is terminated. If the interrupted program is in supervisor state (e.g., an SVC routine), the associated task is always terminated.

On the Model 91, a program interruption occurs whenever a decimal arithmetic instruction is encountered during job step execution. If the optional decimal simulator routine is included in the system, the program interruption handler gives control to it; otherwise, the program interruption handler performs the same functions it performs following other types of program interruptions.

The interruption handlers are disabled for all interruptions except machine-check so that they are not interrupted before they can save critical information about the interrupted program. This critical information comprises the registers' contents and PSW information necessary to return control to the interrupted program after the interruption is processed.

Task Supervision

Task supervision routines enter tasks into the system, supervise their performance, perform task termination processing, and pass control to programs performing tasks after supervisory processing is complete. Tasks are represented to the control program by task control blocks (TCBs); task supervision consists primarily of modifying the TCB queue.

When the initiation of a job step is complete, an initiating routine issues an ATTACH macro instruction causing the supervisor to create a task for the step. The supervisor uses this job step task, its TCB, and other associated control blocks and tables to control the processing originally specified as the job step.

Expansion of an ATTACH macro instruction includes an SVC instruction which causes an SVC interruption. CPU control is passed through the SVC interruption handler to the attach SVC routine. This routine builds a TCB for the task, and a request block (RB) for the first program of that task. These blocks are placed in the TCB and RB queues respectively.

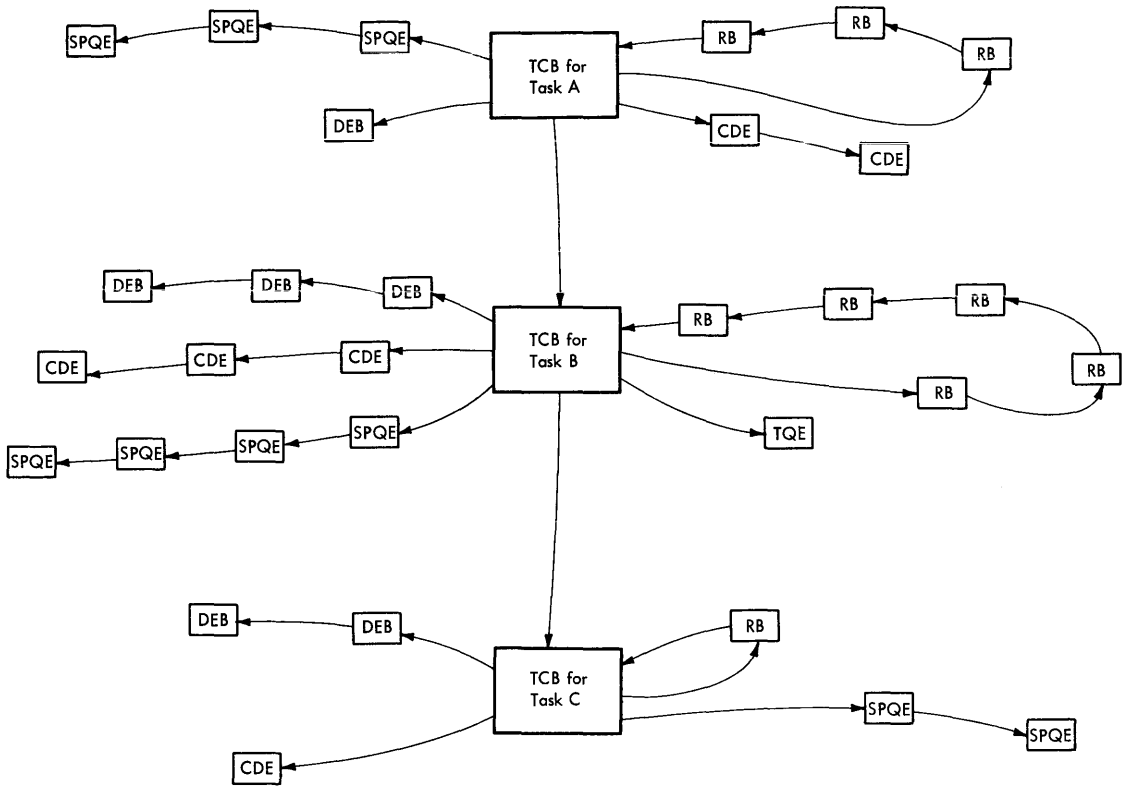
Task Control Block Queue

Whenever the control program needs any information about tasks, its starting point is the TCB queue. There is one TCB for each task currently scheduled in the system. A TCB indicates the status and characteristics of the task it represents. A major part of this status information is pointers to other control blocks and queues, and control information about resources needed during the task. Figure 21 shows the concept of the TCB queue and queues that originate from a TCB.

A TCB is placed in the queue according to the priority of its task, and is pointed to by the next higher TCB in the queue. The first TCB is pointed to by the communications vector table (CVT), which is part of the nucleus.

Request Block Queue

The supervisor builds a request block (RB) for each program of a task that is entered via a supervisor-assisted linkage (LINK, XCTL, or ATTACH), and for types 2, 3, and 4 SVC routines invoked by programs of a task. The RB is placed in the RB queue which originates at the associated TCB. The RB queue indicates, to the supervisor, the programs that perform a given task.



SPQE -- Element of a queue used in main storage supervision
RB -- Element of a queue used in contents supervision
CDE -- Element of a queue used in contents supervision
DEB -- Element of a queue used in data management
TQE -- Element of a queue used in timer supervision

Figure 21. Concept of the TCB Queue

During a task, the addition of RBs to the queue as new programs are invoked, and the deletion of RBs as these programs are completed, allow the supervisor to determine which program (for a particular task) is to receive control at any given time. When a program terminates, the RB queue indicates whether the associated task has been completed, or whether execution of another program is required. If the RB for the terminated program is the only one on the RB queue, the task is complete. If other RBs are on the queue, the programs represented by these RBs must be performed before the task is complete.

Figure 22 shows how the RB queue is modified during a task that is performed by three programs. The first (program A) was specified in the ATTACH macro instruction. The second (program B) was invoked by program A via a LINK macro instruction. The third (program C) was given control by program B via an XCTL macro instruction. When program A is executing, the RB queue for the associated task is shown in Figure 22A. After program B is invoked, the RB queue is shown in Figure 22B. After program C has received control via the XCTL macro instruction, the RB queue is as shown in Figure 22C. When program C

completes and issues a RETURN macro instruction, its RB is deleted, and the RB queue is again as shown in Figure 22A.

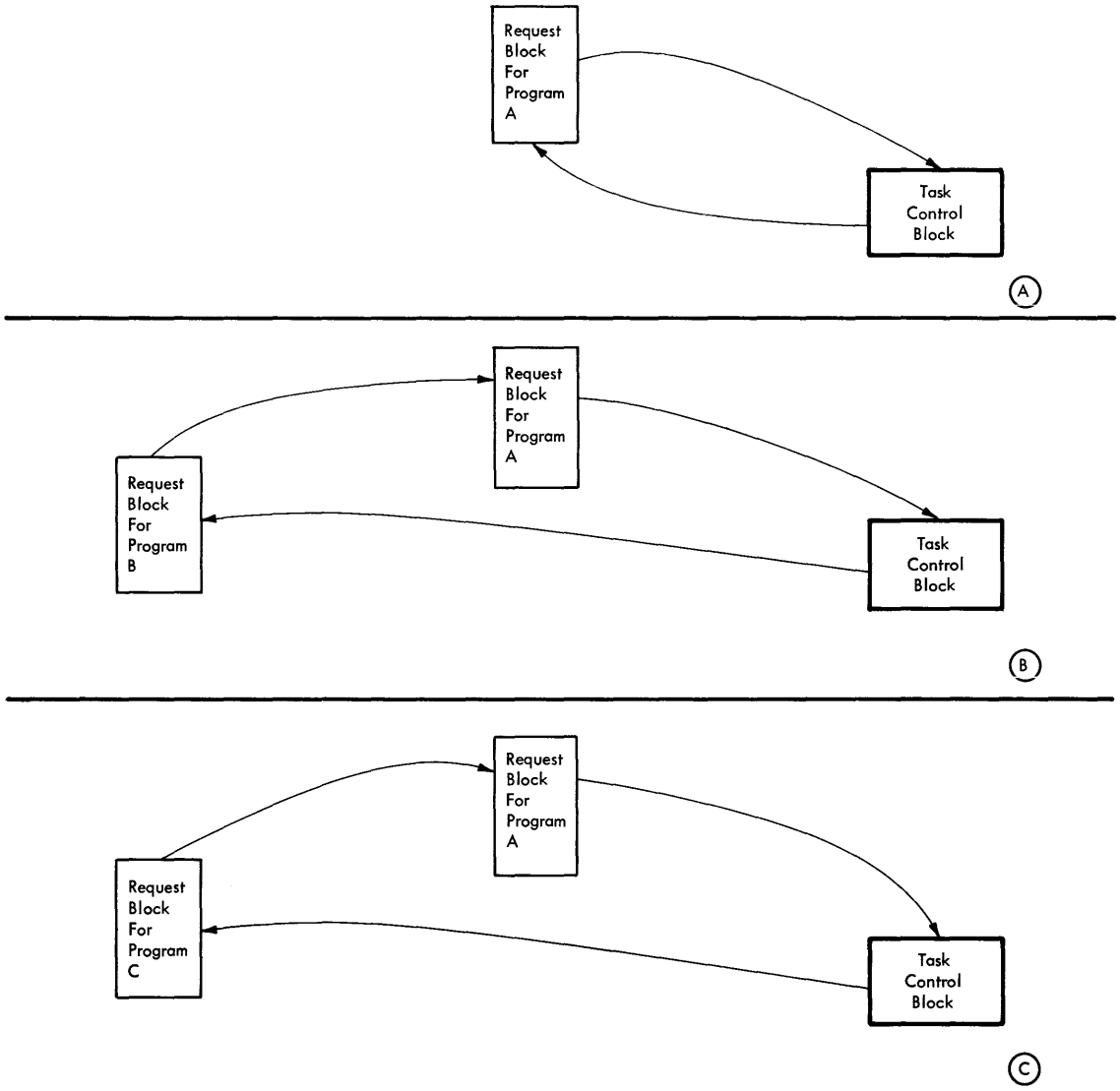


Figure 22. Example of the Modification of the RB Queue During a Task

When the last program of a task has completed, the task supervision routines usually delete the TCB from the TCB queue. If, however, the completed task had an ECB or ETXR parameter specified when it was attached, the TCB is not deleted from the TCB queue. The ECB parameter indicates that the attaching task needs information in the TCB of the completed task. The ETXR parameter indicates that an End-of-task Exit routine (ETXR) is to be executed. The ETXR routine has an RB enqueued to the TCB of the completed task.

If the completed task is a job step task, its associated initiating task is made ready. When this initiating task is dispatched, its routines complete termination processing for the job step.

Passing Control to a Program of a Task

Once the supervisor has completed its interruption processing it passes CPU control to a program operating under control of a TCB. If the program to receive control is the program last interrupted, CPU control is passed directly to it. (This occurs when certain type 1 SVC routines processed the interruption.) If, however, it is possible that another program (other than the one last interrupted) is to receive control, control is passed to the dispatcher routine.

Dispatcher Routine

The dispatcher routine checks the TCB queue for the highest priority ready task, and passes CPU control to the pprgram currently indicated to perform that task. This passing of control is referred to as dispatching a task.

The task that is dispatched, therefore is not necessarily the same task that was last interrupted. The interruption processing could have created or made ready several tasks of higher priority. Any such tasks are dispatched before the task that was last interrupted.

Normally, when several tasks of the same priority are ready, the tasks that are lower on the TCB queue are not dispatched until the higher tasks are completed or waiting. CPU control can be more equitably distributed among such tasks by using the time-slicing feature.

Time Slicing

Time slicing is a feature that causes each of the tasks of a specified priority (a time-slice group) to relinquish control of the CPU after a specified time interval, if that task had not already relinquished control for some other reason. Normally a task retains control either until it is complete, until a higher-priority task becomes ready, or until it must wait for some event (such as an I/O operation). A time-sliced task keeps control until one of these three conditions occurs or until its time-slicing interval expires. If the interval expires, a timer interruption occurs, and control is given to the next ready task in this time-slice group. Control is passed to each ready task in the group for the specified interval according to the positions of the TCBs on the TCB queue.

When a higher-priority task becomes ready, or when all the tasks in the time-slice group are complete or waiting, time slicing ceases (unless the next task dispatched is part of another time-slice group).

When a time-sliced task loses control prior to the expiration of its interval (either because it must wait or because a higher-priority task becomes ready), the remainder of the interval is not saved. When control is returned to this group the next task is dispatched, not the task that lost control.

At system generation, you specify the priorities to be time sliced, and the associated intervals. When the nucleus is initialized, a time-slice control element (TSCE) for each group is intialized in the nucleus. A TSCE contains the timer interval for the group, and pointers to the first, last, and next-to-be-dispatched TCB.

The TCB indicates whether a given task is a member of a time-slice group. When the highest-priority ready task belongs to a time-slice group, the dispatcher checks the TSCE to determine the next task of the group to be dispatched.

Main Storage Supervision

Main storage supervision routines control and allocate main storage in the dynamic and system queue areas. Storage in the dynamic area is assigned to job steps and system tasks. Storage in the system queue area is assigned to contain control blocks that must have a supervisor storage protection key.

Storage Allocation in the Dynamic Area

Initially, the dynamic area is a large number of unassigned, contiguous blocks of main storage, each 2048 (2K) bytes in length. As a system task or job step is initiated, an initiating routine requests the main storage required for the task or step. A main storage supervision routine (the GETMAIN routine) assigns enough contiguous 2K blocks to this step or task from the higher end of the dynamic area. this group or 2K blocks is a **region**. If there is insufficient free, contiguous storage for the step or task, the initiating task is put in the wait state until sufficient storage becomes available.

Storage Allocation in a Region

Storage within a region is assigned in response to requests from programs performing the step or system associated with that region. Storage from a region is required not only for work areas, but also for programs not already in the link pack area.

Rollout/Rollin

Normally all storage requested by programs of a given step or task is assigned from its region. However, if the rollout/rollin feature has been included in the system, an additional region (or regions) may be temporarily assigned to a job step to meet storage requirements that cannot be filled from the step's region.

When such additional storage is required, the rollout/rollin routines try to assign a new region to a step from unassigned storage in the dynamic area. If there is not enough unassigned storage, the contents of a region already assigned to another job step is written on a direct access volume (rolled-out), and that region is used by the step requiring the additional space. Later when the additional region is no longer required, the step that was rolled out is read back into the region (rolled-in) and allowed to continue its operation.

You indicate which steps can cause rollout, and which steps can be rolled out. In any given instance, the relative priorities of these steps determines whether or not rollout will occur, and if so, the order in which steps are rolled out.

Determining Available Storage

Main storage supervision routines determine what storage in a region is available via control blocks called the partition queue element (PQE) and free block queue elements (FBQEs). The PQE of a given region is created when the region is assigned, resides in the system queue area, and is pointed to by the TCBs of all the tasks associated with that region. The PQE is, in turn, the origin of a queue made up of FBQEs. Each FBQE points to a group of contiguous, available 2K blocks of storage within the region, and resides in the lower end of the lowest 2K block of the group.

When all the storage in a region is available, or when all the available storage is contiguous, only one FBQE exists. As storage is assigned, used, and subsequently released, unassigned sections of the region become interspersed with the assigned sections. As this fragmentation occurs, an FBQE is created and enqueued for each unassigned section. When the releasing of storage in a region causes a larger available section to be formed from several smaller sections, FBQEs are deleted from the queue so that this new section has only one FBQE.

When storage is requested, the FBQEs are scanned for an available section large enough to fill the request. If none is available and rollout is not used, the task is terminated. If the request can be filled, a sufficient number of 2K blocks from the unassigned storage is made a part of a **subpool**. The subpool to which this storage is assigned depends on how the storage is to be used.

Subpools

A subpool is, generally, all the 2K blocks of main storage allocated for a particular task under one label called the subpool number. (The exceptions to this definition are shared subpools and subpools in the system queue area.) Initially all storage in a region is unassigned, is not part of any subpool, and has a storage protection key of zero (see Figure 23). When storage within a region is required, unassigned storage in that region is made part of a subpool. The request for this storage specifies (either explicitly or by default) the subpool number of the subpool to which the storage is to be assigned.

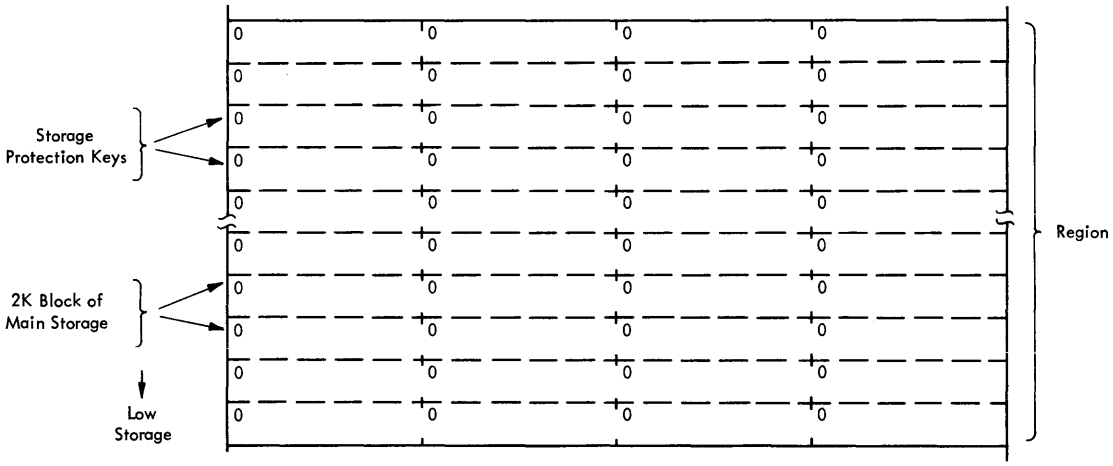


Figure 23. Initial Format of a Region

When storage is requested in a subpool that does not exist (i.e., this is the first request specifying that subpool for a particular task), the subpool is created by allocating a sufficient number of contiguous 2K blocks from the unassigned storage in the region. When the specified subpool already exists (i.e., there were previous requests), the storage currently part of that subpool is checked to determine if a contiguous area, large enough to fill the request, is available. If such an area is available, it is used to fill the request; if not, a sufficient number of contiguous 2K blocks are assigned to the subpool from the remaining free storage in the region. Insufficient free storage in the region results in termination of the task, unless rollout/rollin can occur.

Storage made part of a subpool for any one request must be contiguous. The storage that makes up a complete subpool (i.e., for all requests specifying that subpool) can be noncontiguous.

Requests for storage in subpools of a region are made either by programs performing the task (for working storage), or by the control program (for storage to load a program or for working storage). The subpool specified in a given request depends on what the storage is to be used for, and on what type of routine made the request.

Assigning Storage to Subpools: Subpools are usually assigned to a specific task. When one region is being used for several tasks (i.e., a job step task has one or more subtasks), each task can have separate subpools between 0 and 127, or the tasks can share subpools between 0 and 127. In any one region, there is only one subpool 251 and one subpool 252.

When a control program routine in supervisor state needs storage in a region, it usually requests this storage in subpool 251 or 252. Subpool 252 is specified either for storage needed to contain reenterable routines from SYS1.SVCLIB or SYS1.LINKLIB, or for storage to contain control program data that must be protected. Storage assigned to subpool 252 is given a storage protection key of zero to prevent programs of the job step from writing in the subpool. Storage is assigned to subpool 252 from the highest available storage in the region.

Subpool 251 is specified for storage needed to contain all serially reusable and nonreusable programs, and reenterable programs from private libraries. Storage in subpool 251 is assigned from the lowest available storage in the region. It has the same storage protection key as the programs using the region, and therefore its contents can be modified by these programs. Subpools 251 and 252 of a given region are sometimes collectively referred to as the **job pack area.**

Subpool Queue: Each time a new subpool is created for a task, a subpool queue element (SPQE) for that subpool is placed in the subpool queue of that task. The subpool queue originates at the TCB of the task, and is made up of a series of SPQEs -- one for each subpool of the task. The main storage supervision routines use the subpool queue to determine what subpools are being used in the task, and what storage is assigned to each of the subpools.

Each SPQE has, in turn, a queue of elements that indicates what storage in the region is assigned to that subpool. These queues are called descriptor queues. Each element in a descriptor queue represents one group of contiguous 2K blocks assigned to the subpool.

Figure 24 is an example showing how the main storage supervision routines modify the region, the subpool queue, and the descriptor queues of a task to obtain storage for the three programs previously discussed in the "Task Supervision" section and illustrated in Figure 22. The shaded portions on the left side of Figure 24 show the request block queue as the three programs are used (see Figure 22). The unshaded portions show how the region, and the subpool queue is modified as storage is assigned for the programs. This example assumes that the programs are not initially in storage; programs A and B are reenterable and reside on SYS1.LINKLIB, and program C is not reusable.

Figure 24A shows how the region and the subpool queue are set up for program A. When the task was first dispatched, the Link routine of the supervisor, not program A (which is not yet in storage), receives control. The link routine requests storage for program A from subpool 252. If we assume that program A requires 3.5K bytes of storage, the main storage supervision routines assign three 2K blocks of storage from the highest available part of the region to subpool 252, builds an SPQE, and enqueues it to the TCB. The first 2K block is a work area for Program Fetch (only one per region is required). The other two 2K blocks are for program A. A descriptor queue element (DQE) is built and enqueued to the SPQE. The single DQE indicates that, at the moment, subpool 252 is made up of only one contiguous area of storage.

If program A issues a GETMAIN macro instruction for 2K bytes of working storage in subpool 3, the region and subpool queue are modified as shown in Figure 24B. Since there is no subpool 3, the highest available 2K block of storage in the region is assigned to subpool 3, and the SPQE and DQE are placed in the subpool queue (assume that the storage protection key for this job step is 5).

After the LINK macro instruction for program B is issued, the Link routine requests storage from subpool 252 for program B. If subpool 252 currently has a free section large enough for program B, this program is brought into that area and the subpool queue remains as shown in Figure 24B. If however, subpool 252 must be enlarged for program B (assume that program B is 4000 bytes long), the main storage supervision routines assign two 2K blocks from the highest available part of the region. Since this addition to subpool 252 is not contiguous to the already-existing part of subpool 252, a new DQE is added to the queue. After program B receives control, the queues are as shown in Figure 24C.

Figure 24D shows the region, and the subpool and RB queues after program C receives control. When the XCTL macro instruction for program C is issued, the XCTL routine requests storage (assume 6000 bytes) for program C from subpool 251. Since there is no subpool 251, three 2K blocks of storage from the lowest available part of the region is assigned to subpool 251 and the SPQE and DQE are placed in the subpool queue.

Storage Allocation in the System Queue Area

Storage in the system queue area is used to build control blocks that can be modified by only the control program. The system queue area has storage protection keys of zero, and only programs that operate under a protection key of zero can write into it.

Space in the system queue area is allocated to six subpools, subpools 243, 244, 245, 253, 254, and 255. Subpools 243, 244, and 245 are used in requests for system queue area space from a swapped region; subpools 253, 254, 255 are used in requests for system queue area space for nonswapped regions. System queue space assigned to subpool 245 or 255 is released only when the program that requested it issues a FREEMAIN macro instruction. Subpool 244 or 254 is used for system queue space that must be retained until a step is completed. Space in subpool 244 or 254 can be released by the requestor, or is released by the supervisor (exit or ABEND routine) when the associated job step is terminated. Subpool 243 or 253 is used for space that must be retained until a task is completed. Space in subpool 243 or 253 can be released by the requestor or is released by the supervisor when the associated task is terminated.

Subpools 243, 244, 253, and 254 are used to ensure that system queue space is released, when a job step or task is terminated before the requestor of that space can release it. (This situation frequently occurs when ABEND is invoked.)

Unlike the subpools in the regions, subpools 243, 244, 245, 253, 254, and 255 can share a 2K block of storage. A request for storage for any of these six subpools will be filled from the highest available area of sufficient size even if this area is in a 2K block that is partially assigned to another subpool.

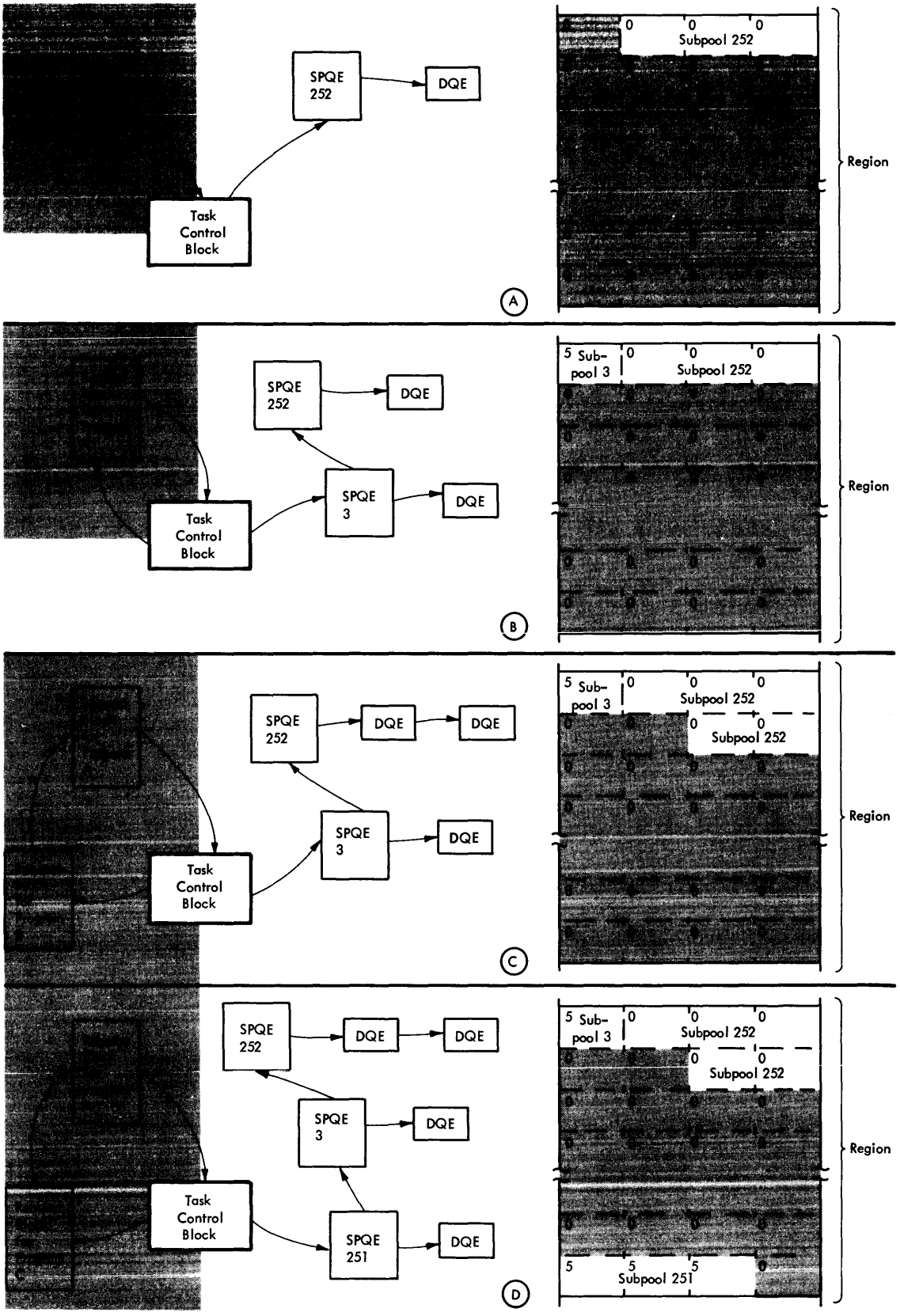


Figure 24. Example of Main Storage Allocation

If the request for system queue space cannot be filled, the system queue area can be expanded by assigning to it 2K blocks of adjacent free dynamic area storage. Before such an expansion is attempted however, the control program attempts to release enough system queue space by purging routines that are currently in the regions but are not being used (e.g., reenterable or serially-reusable routines that have completed their operation). This purge results in the deletion of contents supervision control blocks from the system queue area. If this deletion of control blocks releases enough system queue space to fill the pending request, the system queue area is not expanded.

Once expanded, the system queue area will not be reduced to its original size until the IPL procedure is repeated. If the portion of the dynamic area, adjacent to the system queue area, is already assigned as a region, any attempt to expand the system queue area results in the system's being placed in the wait state.

If a request for space in subpool 253, 254, or 255 is made for a swapped region, space is allocated from an area known as the Local System Queue Area. This space is swapped and is located adjacent to the requesting region. A request for storage in the Local System Queue Area that cannot be fulfilled causes the issuing task to be abnormally terminated.

Contents Supervision

Contents supervision routines bring non-resident routines into main storage, and record what routines are in the dynamic and link pack areas. Routines are brought in as a result of a LINK, ATTACH, XCTL, or LOAD macro instruction, if a usable copy of the desired routine is not already in main storage. If main storage hierarchy support is included in the system, these macro instructions may be used to bring non-resident routines into specified hierarchies of main storage.

The characteristics of a called routine, and its location in main storage determine whether that routine is usable to the calling routine. Routines in the link pack area (all of which are reenterable) can be used by any routine that calls them, and, in fact, the one copy in that area can be used concurrently by several calling routines. Reenterable and serially-reusable routines in subpools 252 and 251 of a given region can be used only by other routines performing the task or tasks associated with that region.

Contents supervision routines determine whether a routine is in main storage by checking the contents directory. Each time a routine is brought into main storage, an entry for it is made into the contents directory. If the routine is brought in via a LOAD macro instruction, an entry is also made to a load list.

Contents Directory

The contents directory is a group of queues indicating the routines in the link pack and dynamic areas.

There are two contents directory queues, one for the link pack area, and one for subpools 251 and 252 of each region. The contents directory resides in the system queue area. The pointer to the contents directory queues for the link pack area is in the communications vector table. The pointer to the contents directory queue for a region is in the first TCB created for that region (job step TCB).

A contents directory queue contains a contents directory entry (CDE) for each program in the region to which it applies (or to the link pack area). When a reenterable or serially reusable program has completed its operation, the copy of that routine remains in the region, and its CDE remains on the queue. When a nonreusable program has completed its operation, its CDE is deleted from the queue. (The storage in subpool 251 occupied by the program is made available; sometimes this storage remains as part of subpool 251, sometimes it is released from subpool 251 for assignment to any subpool of the region.)

Figure 25 shows how a contents directory queue is affected for the programs discussed in the sections "Task Supervision" and "Main Storage Supervision" and shown in Figures 22 and 24. We assumed that none of the programs were already in main storage. During the processing of the ATTACH macro instruction that specified program A (which was previously defined as reenterable), the contents supervision routines determine that a copy is neither in the associated region, nor (since it is reenterable) in the link pack area. After main storage is assigned, program A is brought into subpool 252 and the contents directory queue for that region is as shown in Figure 25A. If program A was already in the region a new copy is not brought in.

During the processing of the LINK macro instruction that invokes program B, the contents supervision routine determines that this program is neither in the region, nor in the link pack area. Main storage is requested and assigned in subpool 252, and program B is brought in. A contents directory entry is constructed and enqueued as shown in Figure 25B.

The XCTL macro instruction involving program C results in a copy of that program being brought into subpool 251 of the region, and a contents directory entry is added as shown in Figure 25C. When program C terminates and passes control to the supervisor, the contents directory entry from program C is deleted, and the contents directory is again as shown in Figure 25B. (This deletion occurs because program C is not reusable.)

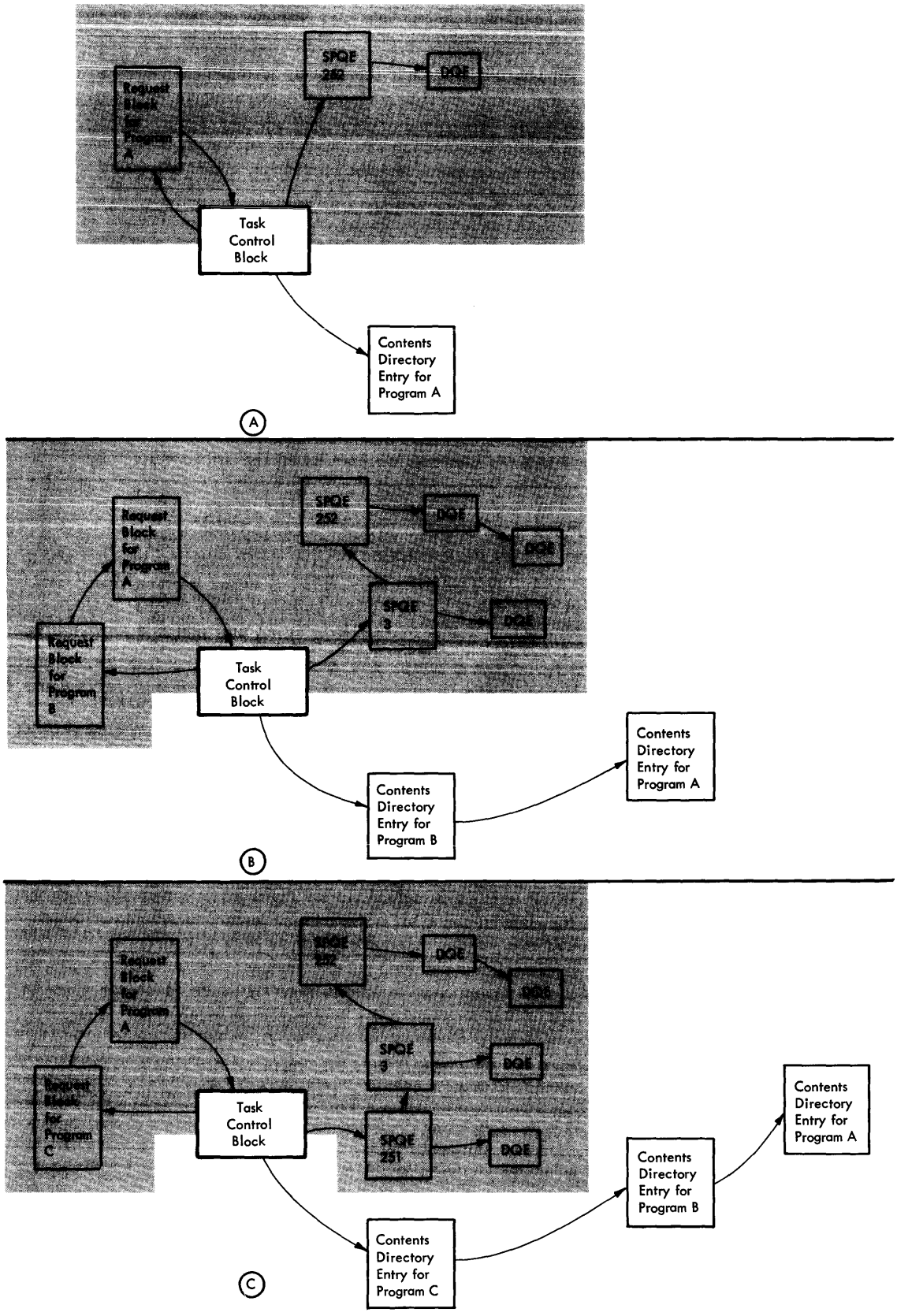


Figure 25. Example of the Modification of Contents Directory During a Task

Load List

A load list is a queue of elements for routines in either the link pack area or a given region, that were invoked via a LOAD macro instruction. Each load list element corresponds to a loaded routine, and points to the contents directory entry for that routine. Each load list element contains a count of the number of times a LOAD macro instruction is issued for the associated routine during a given task. This count is decremented each time a DELETE macro instruction is issued for the routine to reflect the number of current users of the routine.

Timer Supervision

Timer supervision routines process both timer interruptions and requests for timing services. The operating system provides the capability of obtaining the date and time of day, measuring periods of time, and scheduling certain processing for a specific time.

You specify these functions by the timer macro instructions, TIME, TIMER, and STIMER. The expansion of each of these macro instructions includes an SVC instruction which causes CPU control to be passed through the SVC interruption handler to the appropriate timer supervision SVC routine.

When the value in the interval timer goes from positive to negative (indicating the expiration of some interval), a timer/external interruption occurs. After determining that this interruption is a timer interruption, the timer/external interruption handler passes control to a timer supervision routine (the timer second-level interruption handler). This routine performs any processing specified for the completion of this particular interval, and places a new interval in the timer.

An expired time interval (JOB CPU, STEP CPU, or WAIT) can be extended if system management facilities (SMF) are included with the system. With SMF, the timer second-level interruption handler can schedule an SMF asynchronous routine that passes control to a user-written routine that may indicate how much additional time is desired.

Even if you do not specify any timer intervals of your own, the timer supervision routines set the internal timer so that a timer/external interruption occurs every 6 hours; the first such interruption occurs 6 hours after initial program loading (IPL). The timer supervision routines also cause a timer interruption to occur every midnight. The midnight interruption allows timer supervision routines to increment the date in the CVT. The 6-hour interruptions allow timer supervision routines to update two of three main storage locations called **pseudo-clocks**.

Each CPU in the Model 65 Multiprocessing System has an interval timer. However, one timer is active, and only its associated CPU receives timer interruptions. Both CPUs have access to the active timer. If it becomes necessary for the inactive (backup) timer to be made the active timer (as when the active CPU is removed from the system), the conversion is performed by the control program. The timer has the same uses under MVT with Model 65 multiprocessing as it does under MVT.

Pseudo-Clocks

There are three pseudo-clocks, the local time pseudo-clock (LTPC), the 24-hour pseudo-clock (T4PC), and the 6-hour pseudo-clock (SHPC). The LTPC contains the time of day specified in the SET command when IPL was performed. The T4PC is incremented by 6 each time a 6-hour timer interruption occurs, unless its value is already 18. The 6-hour interruption that occurs when the T4PC contains 18 causes the T4PC to be set to zero. The 6-hour

pseudo-clock (SHPC) contains the value of the next interval that will expire; this value is never greater than 6 hours.

In addition to the periodic timer interruption, other timer interruptions are required for program-requested timing of intervals. The timer supervision routines use the **timer queue** to record the lengths of intervals and the order in which these intervals expire.

Timer Queue

The timer queue is a series of elements in the system queue area. Each timer queue element refers to a particular timer interval, and indicates both the length of the interval and the processing to be performed when the interval ends. Whenever a request to set the timer is issued, a timer queue element is placed in the queue. The elements are queued in the order in which the intervals expire. Thus when a timer interruption occurs, the first element in the timer queue is the one associated with the expired interval. After the interruption is processed, this element is removed from the queue, and the next interval to be timed is obtained from the element that is now first in the queue. Timer queue elements of intervals associated with particular tasks (as opposed to intervals being timed regardless of what task is being performed) have pointers to and from their respective TCBs. A task can have only one interval being timed at any given time.

An interval may be of three types: TASK, REAL, or WAIT. A TASK interval is dequeued each time the task loses control of the CPU and is reestablished when the task is again dispatched, thus timing the task only when it is active. The REAL or WAIT intervals remain enqueued throughout the interval, thus effectively providing an elapsed time interval measurement.

Time-of-Day Clock

If the MVT configuration is generated for a System/370, the timer supervision routines use the Time-of-Day (TOD) Clock, a standard feature of System/370 CPUs. The TOD Clock has timing resolution to one microsecond, and provides increased accuracy in timing intervals greater than one hour. The TOD Clock runs continuously while power is on; it is not affected by system wait conditions.

The timer second-level interruption handler and the processing routines for the TIME, STIMER, and TTIMER macro instructions use the TOD Clock in addition to the interval timer. Details of the logic for TOD Clock processing are described in the **MVT Supervisor PLM.**

Task Management in a Multiprocessing Environment

Most basic functions performed by task management routines are unchanged for a multiprocessing environment. There continues to be a single task control block queue, and other work queues are also unchanged. The major difference is that task managment routines additionally coordinate the acitvities of the CPUs. There are two ways in which the coordination is achieved. A technique called "lockout" is used to prevent access to critical data by both CPUs at the same time, and a technique called "shoulder-tap" makes use of the WRITE DIRECT instruction (Model 65 Multisystem Feature) for communication between the two CPUs.

A lockout function is needed because when interruptions are disabled on one of the two CPUs, they are not disabled on the other. Therefore, vital table and queue manipulation operations could be interrupted, and the partially manipulated data could be used or altered by a routine being executed on the other CPU. To achieve lockout, a programmed switch called a lock byte is tested and set wherever interruptions are disabled (all first-level interruption handlers) and released where interruptions are again enabled (the dispatcher and the type 1 exit routines). When the lock byte is on, the affected routines may be executed only on the CPU which executed instructions that turned on the lock byte.

Shoulder-tap processing is performed by two routines -- a WRITE DIRECT routine and a shoulder tap receiving routine, both of which are resident. The WRITE DIRECT routine is used by supervisor routines when there is a need for one CPU to communicate with the other. It uses the WRITE DIRECT instruction and places a unique code into main storage to identify the reason for the shoulder tap. The shoulder tap receiving routine examines the code and passes control to an appropriate processing routine.

All functions performed by task management routines in a multiprocessing environment are described in the **MVT Supervisor PLM**.

Data Management Routines

This chapter describes the four routines used by Data Management. These routines are:

- Assigning Space on Volumes
- Maintaining the Catalog
- Support Processing for I/O Operations
- Processing I/O Operations

Assigning Space on Volumes

Assigning of tracks and cylinders on direct access volumes is performed by the direct access device space management (DADSM) SVC routines of data management. These routines are used primarily by job management routines during the initiating of a job step to get space for output data sets. The DADSM routines are also used by other data management routines to increase the space already assigned to a data set, and to release space no longer needed. The DADSM routines are described in the **Direct Access Device Space Management PLM.**

The DADSM routine controls allocation of space on direct access volumes through the volume table of contents (VTOC) of that volume. The VTOC is built when the volume is initialized by the direct access storage device initialization (DASDI) utility program. The VTOC indicates the current usage of the space on the volume.

The VTOC is a collection of data set control blocks (DSCBs). Each DSCB corresponds either to a data set currently residing on the volume, or to contiguous, unassigned tracks on the volume. DSCBs for data sets are the data set labels, which contain characteristics of the data set and the tracks on which it resides. DSCBs for unassigned tracks indicate the locations of unassigned, contiguous tracks.

When space is needed on a volume, the DADSM routines check the VTOC for enough contiguous, available tracks to satisfy the request. If there are not enough contiguous tracks, the request is filled using up to five noncontiguous groups of free tracks. The appropriate DSCBs are modified to reflect the assignment of the tracks.

When space is released, the DADSM routines delete the DSCB of the deleted data set from the VTOC. A DSCB is built or modified to indicate that the tracks containing the deleted data set can be reallocated.

If system management facilities (SMF) are included, the system uses SMF to collect and record the following information on the SMF data set:

- Space information about user data sets, such as the number of extents or the amount of space released.
- Information about any change in the status of a data set (e.g., the opening, closing, scratching, or renaming of a data set).
- Information about unused space on a direct access volume (e.g., the amount of unused space and whether the space is fragmented, the number of extents, if the volume table of contents (VTOC) is filled).

Maintaining the Catalog

The catalog is a collection of data sets that indicates the volumes on which cataloged data sets reside. The catalog management routines of data management maintain the catalog, and locate cataloged data sets.

To maintain the catalog, catalog management routines create and delete indexes, and catalog and uncatalog data sets. To locate a data set, catalog management routines search through the indexes, specified in the qualified name of the data set, for the index entry containing the last part of the qualified name. This index entry contains the serial number (or numbers) and device type of the volume (or volumes) on which the data set resides. The catalog management routines are described in the **Catalog Management PLM.**

The catalog management routines are used primarily by job management routines and the IEHPROGM utility program. Job management routines may invoke the catalog management routines during the initiation and termination of a job step. During initiation, a catalog management routine locates cataloged data sets. During termination, a catalog management routine may catalog or uncatalog data sets referred to during the job step and specified for the catalog. The IEHPROGM utility program invokes catalog management routines to perform any of their functions except locating a data set. Processing programs can also invoke the catalog management routines via the CATALOG, INDEX, and LOCATE macro instructions.

Support Processing for I/O Operations

Support processing for I/O operations has three subdivisions:

- Open processing which is required before I/O operations can be performed.
- Close processing which is required after I/O operations have been completed.
- End-of-volume (EOV) processing which is required when space for a sequential data set on either a direct or sequential access volume is exhausted during an I/O operation.

The routines that perform these functions are the I/O support routines, (open, close, and EOV). Their operation is discussed in the **Input/Output Support (OPEN/CLOSE/EOV) PLM.**

Open Processing

Before any information can be read from or written into a data set, initialization must be performed. This initialization is referred to as "opening" the data control block of the data set, and basically consists of:

- Ensuring that the volumes required for reading or writing the data set are mounted.
- Constructing control blocks required by the I/O supervisor to initiate the I/O operations.
- Loading the access method routines that are to process the I/O operations on the data set.

Insuring Proper Volume Mounting

The open routine determines whether the volumes required for the data set are mounted on devices assigned to the job step. If the volumes are not mounted, the open routine issues a mounting message to the operator and, after mounting has been performed, checks the volume labels to verify that the proper volumes have been mounted.

The open routine then locates the data set (or the space to receive the data set) on the volume. For tape, the volume is positioned; for direct access volumes, the DSCB is read into main storage.

Constructing Control Blocks

The open routine constructs (or completes construction of) control blocks that are used when the data set is to be read or written. These are the data control block (DCB), job file control block (JFCB), header labels or data set control block (DSCB), and the data extent block (DEB).

Completing the DCB, JFCB, and DSCB: The DCB, JFCB, and DSCB are in various stages of completion before the DCB is opened. The open routine completes them by merging information from one block to another. There are two distinct merge operations, the forward merge and the reverse merge.

The forward merge is the passing of information first from the DSCB (or standard tape label) to the JFCB, and then from the JFCB to the DCB. Information is passed only when the field of the block receiving the information is empty. The forward merge does not change any fields that already contain information.

The reverse merge is the passing of information from the DCB back to the JFCB, and then to the DSCB (or header label). This merge occurs after the Open routine has given control to any user-written DCB-exit routines. When the associated data set is for output, the reverse merge not only fills empty fields in the JFCB and DSCB, but also overrides existing fields except the DSORG field. When the data set is for input, the DCB to JFCB merge fills only empty fields; no JFCB to DSCB merge is performed. Figure 26, which is an expansion of Figures 19 and 20, shows the flow of information in the forward and reverse merges. The numbers indicate the sequence in which the flow occurs.

Constructing the DEB: A data extent block (DEB) is built for each DCB being opened. The DEB contains the volume location (or locations) of its associated data set, and the names of the access method routines that are to be used on this data set. The DEB is used by the I/O Supervisor in starting an I/O operation. The relationship of the DEB to other control blocks is shown in Figure 26.

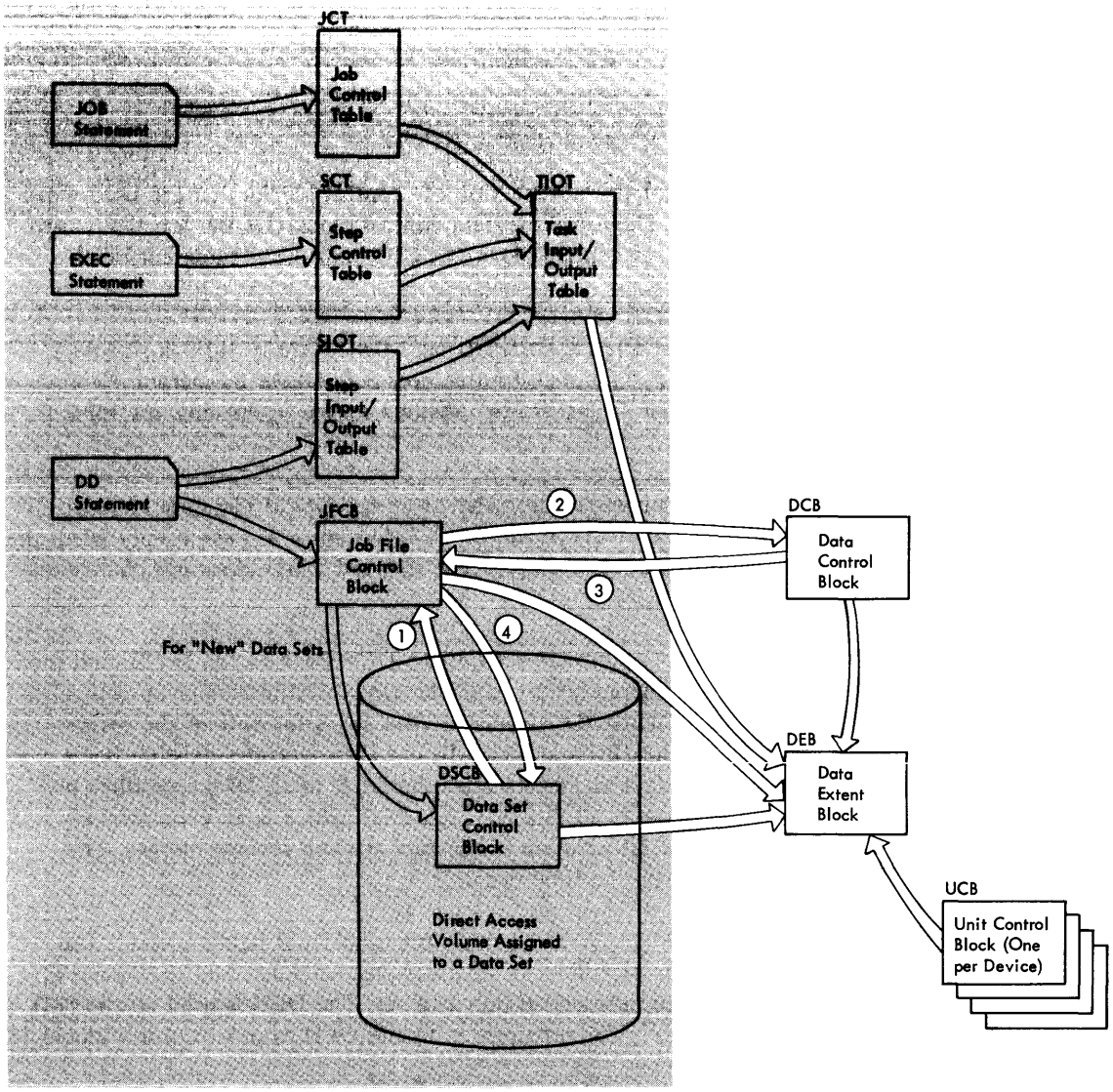


Figure 26. Flow of Information During the Merges of the Open Routine

The DEB is built from information in the DCB, JFCB, DSCB, and UCBs (unit control blocks) of the devices associated with the data set. A UCB exists for each device in the system. UCBs are built when the system is generated, and contain characteristics of the devices that they represent.

DEBs are built in the system queue area so that their contents cannot be changed by processing programs. All DEBs for a given task are placed in a queue originating at the TCB of that task.

The DEB is built by an access method executor module. These modules operate as part of the open routine but perform processing unique to the access method to which they apply. The operation of the access method executors is described in the various access method program logic manuals.

Loading Access Method Routines

The open routine uses the DCB to determine which access method is to be used on the associated data set. The executor of that access method then determines which routines of the access method are required to operate on the data set. These routines are then loaded into the appropriate region unless they are already in the link pack area.

Close Processing

After I/O operations on a data set are complete, the DCB of that data set should be closed. The close routine restores the DCB fields that were filled by the forward merge during open, processes labels, determines volume disposition, and deletes the unneeded access method routines.

Label processing includes the building of trailer labels for output data sets on tape, and the updating of DSCBs of data sets with OUTPUT, OUTIN, or INOUT dispositions.

Volume disposition includes not only dismounting instructions to the operator, but also the writing of tape marks and the positioning of tape reels.

If the access method routines associated with this close operation are not in the link pack area and are not required for any more I/O operations in the region, the storage that these routines occupy is released.

End-of-Volume Processing

End-of-volume (EOV) processing is performed when end-of-data set or end-of-volume conditions occur during an I/O operation on a sequentially organized data set. When a routine of a sequential access method encounters a tape or file mark or an end-of-volume condition, the routine issues an SVC instruction to pass control to the EOV routine.

EOV processing consists primarily of verifying and constructing labels. If the data set for which the condition occurred is continued on another volume, the EOV routine issues mounting instructions for the next volume and checks the mounting.

If the EOV condition occurred because direct access volume space assigned to an output data set is used, the EOV routine invokes a DADSM routine to obtain more space for the data set.

Processing I/O Operations

The processing of I/O operations is performed in two distinct parts: processing required to start the operation, and processing required when the operation is terminated.

Starting an I/O Operation

- Access method routines, which organize the information required to initiate the I/O operation.
- The EXCP routine of the I/O supervisor, which initiates and supervises the I/O operation.

Figure 27 shows the relationship that exists between a processing program, an access method, and the I/O supervisor.

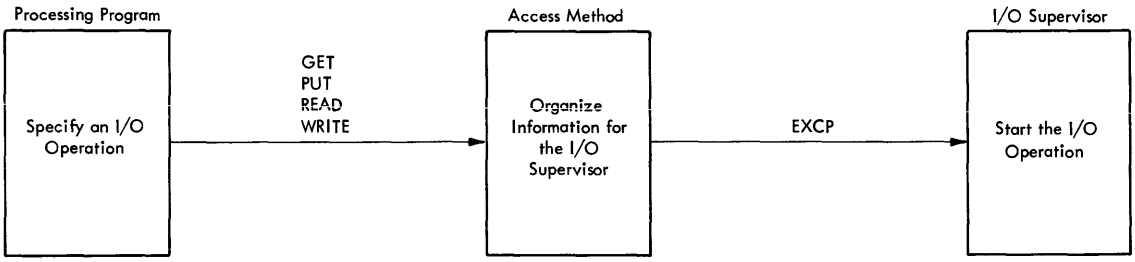


Figure 27. Relationship Between a Processing Program, an Access Method, and the I/O Supervisor

The expansion of an I/O macro instruction specified in the processing program results in a branch to the access method routine. This routine gathers information used to initiate the I/O operation and places this information in control blocks. The routine then issues an EXCP macro instruction causing an SVC interruption. The SVC interruption handler gives CPU control to the I/O supervisor, which either starts or queues the I/O operation.

After the EXCP routine has completed its operation, it passes control to the Type 1 SVC exit routine which returns control to the access method routine. This routine finishes its processing before passing control to the processing program that issued the I/O macro instruction. Figure 28 shows the flow of control for an I/O operation.

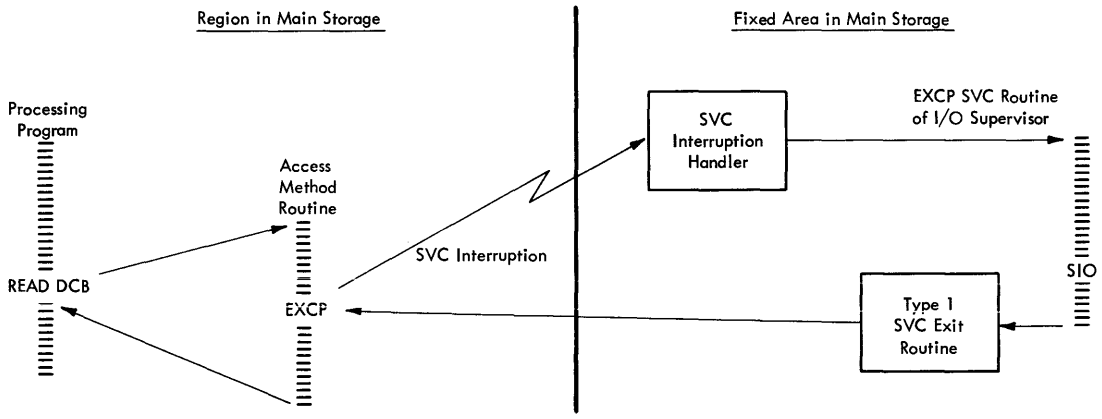


Figure 28. Flow of Control for an I/O Operation

Access Methods

Access method routines prepare information required by the I/O supervisor to start an I/O operation. Routines of certain access methods also perform services that are not directly associated with the actual I/O operation. These services include allocating and controlling buffer areas, moving data to and from the buffer areas, and blocking and deblocking records. When you assemble your program, you indicate an access method in the DCB of the data set. When the DCB is opened, the access method routines to be used on the data set are brought into the appropriate region, unless these routines are already there or in the link pack area.

Routines of every access method construct a number of input/output blocks (IOBs) and channel programs for the I/O supervisor. The number of IOBs and channel programs built for a given data set depends on the number of main storage areas used by the data set, and the number of I/O operations to be performed on the data set.

An IOB contains information required by the I/O supervisor to start an I/O operation. Each IOB points to a channel program that is to be executed. When the operations for that channel program terminate, the channel status word (CSW) is stored in the IOB associated with the channel program.

A channel program is a group of one or more channel command words (CCWs) that specify I/O operations, and indicate the main storage areas for the data involved in these operations. The CCW and CSW formats are described in the **Principles of Operation** publication.

For some access methods, the IOBs and CCWs are partially built by the open executor routine of that access method when the DCB is opened; these IOBs and CCWs are completed by the access method routine, when the I/O operation is being processed. For other access methods, the IOBs and CCWs are completely built during the processing of the I/O operation.

EXCP Routine

The EXCP SVC routine is the portion of the I/O supervisor that initiates I/O operations. This routine receives control from the SVC interruption handler and builds a request element for the requested I/O operations.

Whenever an I/O operation is in process, the UCB for the device points to the request element for that operation. When an operation cannot be started, the request element for that operation is placed in a queue for the device. This queue is the request element table.

The EXCP routine determines whether the I/O device associated with the operation is free, and if so, whether any channel associated with the device is free.

When both the device and an associated channel are free, the EXCP routine issues a START I/O (SIO) instruction to initiate the operation. If the device or all associated channels are busy, the request element is placed in the queue for that device. The elements that make up this queue are contained in the request element table.

The EXCP routine is described in the **Input/Output Supervisor PLM**.

After the I/O operation is started, an indicator is set in the UCB to show that the device is now busy, and a pointer to the request element is placed in the UCB. When an I/O interruption occurs, the I/O supervisor uses the request element in the UCB to determine the request that has been executed.

Terminating an I/O Operation

I/O operations terminate either normally because the operation is completed, or abnormally because an error is detected. When an I/O operation terminates, an I/O interruption occurs, causing CPU control to be passed first to the I/O interruption handler and then to the I/O interruption supervisor portion of the I/O supervisor to process the interruption.

The I/O interruption supervisor posts the completion of the I/O operation, schedules error routines (i.e., places a request block for the routine in the request block queue) when the operation terminated abnormally, and, if possible, starts another I/O operation on the channel. The I/O interruption supervisor returns CPU control to the interruption handler. The I/O interruption supervisor is described in the **Input/Output Supervisor PLM**.

Sharing Direct Access Devices

One or more 2301, 2303, 2305, 2311, 2314, 2321, or 3330 direct access devices can be shared among two or more CPUs when the drives are connected to a control unit which has a path to each CPU. This feature allows access to the devices through separate channels connected to separate CPUs.

The I/O supervisor and ENQ/DEQ routine of the control program control the use of a shared device so that data being used in one CPU is protected from modification by a program in another CPU, and so that access-arm contention between CPUs is minimized. Figure 29 shows the flow of CPU control when either the catalog management or DADSM routines use a shared direct access device. These routines are the control program routines that must reserve a shared device before using it; all other reserves are user-initiated.

Data Protection

If a user of a shared device requires data protection while he works on that data, he must issue a RESERVE macro instruction. This macro instruction expands into an SVC instruction which, when executed, causes CPU control to be passed through the SVC interruption handler to the ENQ/DEQ routine (see Figure 29). This routine increments a counter in the UCB of the device. The I/O supervisor performs the actual reserving of the device later when the I/O operation is started. The UCB counter tells the I/O supervisor whether to release the device after the I/O operation has completed. The device is not released until the user has decremented the counter to zero by issuing a DEQ macro instruction for each RESERVE macro instruction. Thus a user can read, modify, and write multiple records from a shared device without fear that a program in another CPU has had access to his data between his own I/O operations.

Control of Access Arm Movement

When the user of the shared device executes his I/O macro instruction, CPU control is passed through the SVC interruption handler to the EXCP routine of the I/O supervisor. This routine performs a "seek" to position the arm for the data transfer. The seek channel program contains a Reserve command (not to be confused with the RESERVE macro instruction) which reserves the device. This reservation is necessary (even if the user did not specify a RESERVE macro instruction) to ensure that the arm is not moved by a channel program in another CPU between the seek and the actual data transfer.

Before the EXCP routine starts the data transfer, it checks the UCB counter. If the counter is zero, the EXCP routine places a Release command in the channel program for the data transfer. This, in effect, causes the device to be released after the data transfer is complete. If the counter is not zero (indicating that a user of the device wants it held), the Release command is not issued.

When the user no longer needs the shared device, he issues a DEQ macro instruction. The DEQ routine decrements the UCB counter and if it is zero, issues an EXCP macro instruction. The EXCP routine starts a channel program that releases the device, and returns control to the DEQ routine which, in turn, passes control to the user routine.

When a task is terminated with outstanding device reservations, the termination is abnormal. The ABEND routine invokes the DEQ purge routine to cancel the device reservations.

The ENQ/DEQ routines and the EXCP routine are described in the **MVT Supervisor PLM** and the **Input/Output Supervisor PLM** respectively.

Data Management in a Multiprocessing Environment

Except for I/O supervisor functions,data management in a multiprocessing encironment is the same as it is in a single-CPU environment. With two CPUs, the I/O supervisor routines may have to keep track of up to twice the maximum number of channels available with one CPU.

Each CPU has a channel table, located in its prefixed storage area (PSA), which continually reflects the availablility of each channel on that CPU (either CPU may refer to the other CPU's channel availability table). Also, extensions to the UCBs identify whether a CPU has a path through a control unit to a device.

When a CPU executes a request for an I/O operation and the requested device is unavailable because the channel or control unit is busy, an I/O supervisor routine scans the other CPU's channel availability table and UCBs to determine if a path from the other CPU to the device is open. If it is, a shoulder tap is used and the I/O operation is started via the other CPU.

The role of the I/O supervisor in a multiprocessing environment is described on the **Input/Output Supervisor PLM.**

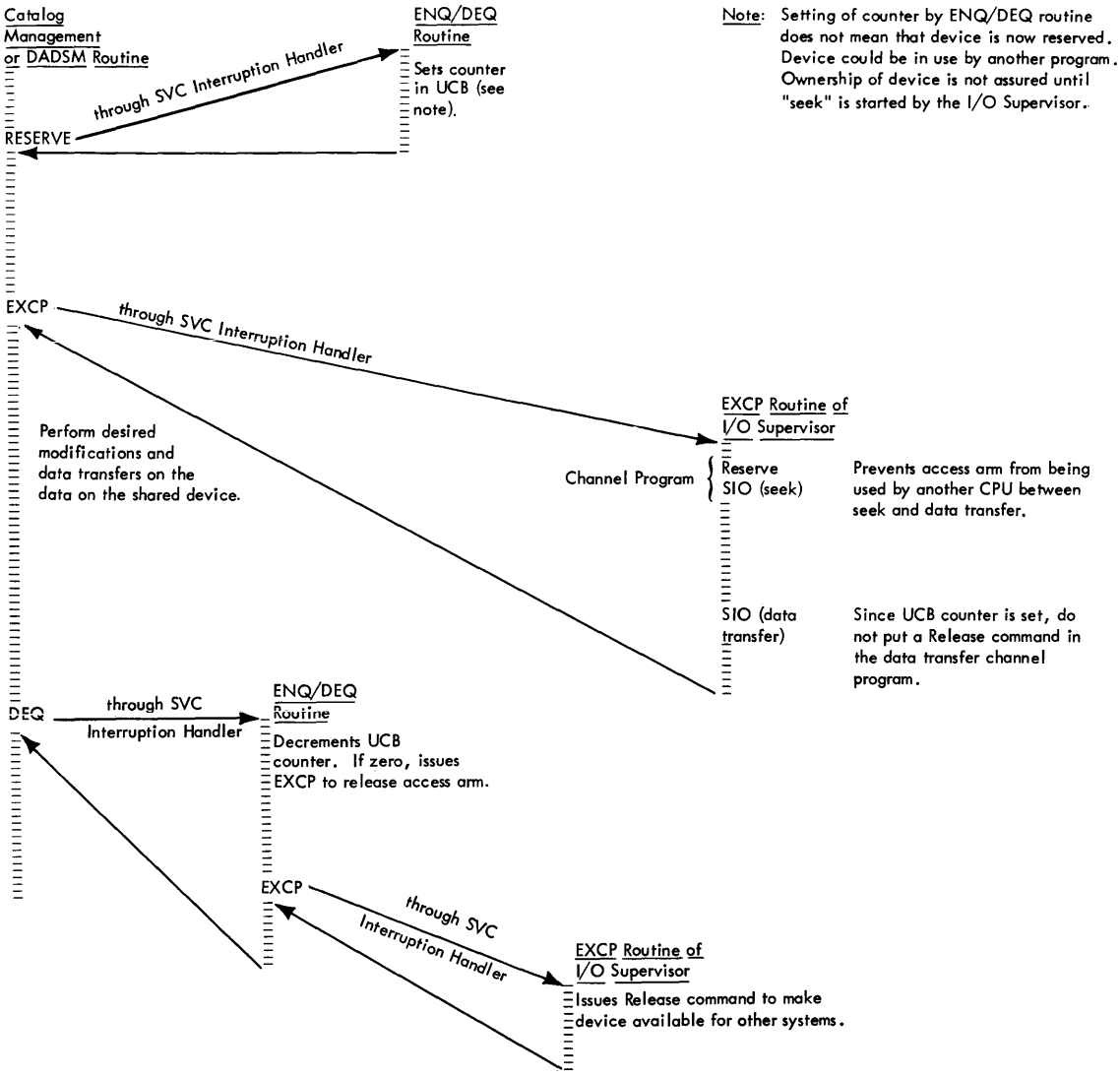


Figure 29. Flow of CPU Control When Using a Shared Device

Volume Management Routines

This topic describes the error statistics by volume (ESV) and the error volume analysis (EVA) routines used by Volume Management.

Error Statistics by Volume (ESV)

ESV routines can cause the system to collect the following set of statistics for each tape volume during any interval that the volume is open (an abridged set of these statistics is collected if the ESV records are to be printed at the console rather than on a line printer):

- The volume serial number.
- The CPU serial number.
- The system model number.
- The date that this set of statistics was collected.
- The time of day that this set of statistics was collected.
- The address of the unit on which this volume was mounted, and the channel through which it was operating.
- The number of temporary read errors that occurred.
- The number of temporary write errors that occurred.
- The number of permanent read errors that occurred.
- The number of permanent write errors that occurred.
- The number of noise blocks encountered.
- The number of erase gaps written while trying to correct write errors.
- The number of Start I/O operations encountered.
- The bit density of the volume (in bits per inch).
- The physical record length of the volume (for fixed length records; the length is forced to zero for undefined or for variable length records and when EXCP access method is used).

The EVA option requires the system operator to specify two minimum values, one for the number of temporary read errors and one for the number of temporary write errors. If the number of read or write errors for a volume currently being accessed exceeds the values specified by the system operator, the system will immediately type a message to this effect at the console. EVA can be used for both labeled and unlabeled volumes (if SER parameter is used in the DD card for the volume), but its primary purpose is to monitor errors on unlabeled volumes, inasmuch as the ESV option can only provide data for labeled volumes.

An I/O error calls an error recovery procedure (ERP) which accumulates statistical information and records an entry in a table called the volume statistics table. This table is located by a pointer in the device unit control block (UCB). When end of volume is reached, or when the volume is closed, the VOLSTAT routine collects the ESV records if ESV is specified; if SMF is also specified, the VOLSTAT routine constructs the records in the extended save area of the supervisor request block (SVRB) and transfers the records to data sets SYS1.MANX and SYS1.MANY (disk) or to data set SYS1.MANX only (tape) as shown, depending on an option selected at system generation. If the records are written to disk, they will be written first to SYS1.MANX. When this data set is full, a warning message will be printed at the console and records will be diverted to SYS1.MANY until it becomes full, after which records are again written to SYS1.MANX.

It is possible to specify that the ESV records be printed out at the console. In that case they will be printed when the volume is demounted and will not be written to either of the SYS1.MAN data sets.

If the ESV records collected by the VOLSTAT routine are written to disk, the operator can dump the records to tape by running an SMF dump utility called IEBSMFDP. This will be unnecessary if the records are already written to tape instead of to disk. The operator may then run the IFHSTATR utility, which will format and print the volume statistics report.

The error recovery procedures, the volume statistics table, and the counter update routines are described in the **Input/Output Supervisor PLM**. The format of the records as they appear on the ESV tape is shown in Figure 30. These records are called type 21 records.

Error Volume Analysis (EVA)

If EVA is specified, and the threshold values for read or write errors are greater than zero, the tape ERP will compare the threshold values with the temporary read and write error counters every time it updates the counters. If either of the threshold values is equal to its corresponding read or write error value, the tape ERP prepares a message and issues a write-to-operator (WTO) macro instruction to print the message at the console. A message is printed only when the read or write error count and its threshold are equal. Thus, during the time that the tape volume is open, there can be no more than two messages, one for attaining the read error threshold, and one of attaining the write error threshold. When the volume is demounted the temporary read and write error counters are zeroed.

<table>
<tr><td colspan="2">Total Record Length</td><td colspan="2">'00'</td></tr>
<tr><td></td><td>Record Type</td><td colspan="2">Time Of Day</td></tr>
<tr><td colspan="2">Time Of Day (continued)</td><td colspan="2">Current Date</td></tr>
<tr><td colspan="2">Current Date (continued)</td><td colspan="2">CPU ID</td></tr>
<tr><td colspan="2">Model No.</td><td colspan="2">Data Length</td></tr>
<tr><td colspan="2" rowspan="2">Volume Serial No.</td><td colspan="2"></td></tr>
<tr><td colspan="2">Channel/Unit Address</td></tr>
<tr><td colspan="4">UCB Type</td></tr>
<tr><td>Temporary Read Errors</td><td>Temporary Write Errors</td><td colspan="2">Start I/Os</td></tr>
<tr><td>Permanent Read Errors</td><td>Permanent Write Errors</td><td>Noise Blocks</td><td>Erase Gaps</td></tr>
<tr><td>Erase Gaps (continued)</td><td colspan="2">Cleaner Actions</td><td>Tape Density</td></tr>
<tr><td colspan="2">Block Size</td><td colspan="2"></td></tr>
</table>

Figure 30. ESV Record Format for Tape Volumes

Recovery Management Routines

This topic describes the error recovery routines used by Recovery management.

Recovery management facilities fall into two general categories: facilities for CPU error recovery, and facilities for I/O error recovery.

CPU Recovery Facilities

Three facilities provide recovery from CPU and main storage errors:

- System Environment Recording, Option 0 (SER0)
- System Environment Recording, Option 1 (SER1)
- Machine-Check Handler (MCH)

These facilities are model-dependent and mutually exclusive. One of them, and only one, must be included in every operating system with MVT.

System Environment Recording, Option 0 (SER0)

SER0 is the least complex of the CPU recovery management facilities. When a machine check occurs, SER0 determines the type of malfunction, collects data about the error, and writes the data as a record in SYS1.LOGREC.

Model-dependent versions of SER0 are provided for System/360 Models 40, 50, 65, and 75. One or more versions of SER0 can be included in SYS1.LINKLIB during system generation. During nucleus initialization, resident routines of the appropriate version are loaded as part of the nucleus.

When a machine check occurs, resident SER0 routines save machine and program data, and halt all I/O operations. They then load other SER0 routines into the dynamic area of main storage. These routines save additional data, and write all saved data as a record in SYS1.LOGREC. SER0 then asks the operator to reload the operating system, and places the computing system in the wait state.

If I/O errors prevent the writing of a record in SYS1.LOGREC, SER0 asks the operator to run SEREP (the stand-alone system environment recording, editing, and printing program). It then places the system in the wait state. After running SEREP, the operator must run the IPL program to reload the system.

SER0 is described in more detail in the **MVT Supervisor PLM**.

System Environment Recording, Option 1 (SER1)

SER1 performs the same data collection functions as SER0. In addition, it analyzes each machine error, and permits system operation to continue when the error affects only a single noncritical task.

Model-dependent versions of SER1 are provided for System/360 Models 40, 50, 65, 75, 91, 95, and 195, and the System/370 Model 195. One or more versions can be included in SYS1.LINKLIB during system generation. During nucleus initialization, the appropriate version is loaded as part of the nucleus.

When a machine check occurs, SER1 determines the type of malfunction, collects data about the error, and writes the data as a record in SYS1.LOGREC. If only one task is affected, that task is abnormally terminated, and system operation is allowed to continue. If more than one task is affected, SER1 asks the operator to reload the operating system, then places the computing system in the wait state.

If I/O errors prevent the writing of a record in SYS1.LOGREC, SER1 asks the operator to run SEREP; it then places the computing system in the wait state. After running SEREP, the operator must run the IPL program to reload the operating system.

SER1 is described in more detail in the **MVT Supervisor PLM**.

Machine-Check Handler (MCH)

MCH is the most complex of the CPU recovery management facilities. The goal of MCH is total recovery from machine errors: when it achieves this goal, MCH permits a program interrupted by a machine-check to continue processing. When total recovery is not possible, MCH performs essentially the same functions as SER1.

Model-dependent versions of MCH are provided for System/360 Model 65, Model 65 Multiprocessor, and Model 85, and for System/370 Models 145, 155, and 165.

MCH for Model 65 and Model 65 Multiprocessor

When a machine check occurs, MCH determines the type of malfunction, collects data about the error, and writes the data as a record in SYS1.LOGREC. MCH retries the interrupted instruction (that is, tries to re-execute the instruction), provided the error has not made retry impossible. Retry normally is impossible if the error involves damage to the interrupted program; however, if the program is refreshable, MCH tries to repair the damage by loading a fresh copy of the program.

When instruction retry is successful, MCH permits the interrupted program to continue processing. When retry is not successful (or not possible), MCH analyzes the error and tries to associate it with a specific task. If the error affects a problem program task or noncritical system task, that task is abnormally terminated and system operation is allowed to continue. If the error affects a critical component of the control program, MCH informs the operator and places the computing system in the wait state.

A permanent storage failure normally requires a halt for system repair. In a Model 65 Multiprocessing system, however, 2048-byte blocks of main storage can be logically removed from the system through storage reconfiguration. When the failure does not affect a critical task, critical storage, or a TSO task, the block where the failure occurred can be removed from the system, permitting system operation to continue.

MCH is described in more detail in the **Machine-Check Handler for System/360 Model 65 PLM**.

MCH for Models 85, 145, 155, and 165

MCH programs for Models 85, 145, 155, and 165 differ from MCH for Model 65 in that machine recovery facilities handle instruction retry. If the interrupted instruction is retried successfully, MCH is entered only to collect and analyze data about the error, and to write the data as a record in SYS1.LOGREC.

If the instruction is not retried successfully, MCH performs essentially the same functions as in the case of the Model 65: error identification and analysis, program repair or task termination, and error recording in SYS1.LOGREC. (Exception: For Model 145, MCH does not attempt to repair program damage; the task is always terminated. For Models 155 and 165, MCH repairs damage to the control program only; in most cases, it does this by a checksumming technique rather than by loading a fresh copy of a load module.)

Through the MODE command, the operator can control the method of error recording and certain other aspects of recovery management. The exact function of the MODE command depends on the CPU model, and is described in the **Operator's Reference** publication.

MCH is described in more detail in the following program logic manuals:

- **Machine-Check Handler for System/360 Model 85 PLM**
- **Machine-Check Handler for System/370 Model 145 PLM**
- **Machine-Check Handler for System/370 Models 155 and 165 PLM**

Input/Output Recovery Facilities

Four facilities aid recovery from I/O errors:

- Channel-Check Handler (CCH)
- Error Recovery Procedures (ERPs)
- Alternate Path Retry (APR)
- Dynamic Device Reconfiguration (DDR)

Any or all of these facilities can be included in the same system. ERPs are required for all systems, while CCH, APR, and DDR are required or optional, depending on the CPU model.

When a system does not include CCH, channel checks are handled by the system's CPU recovery management routine (SER0, SER1, or MCH). This routine writes an error record in SYS1.LOGREC, informs the operator of the error, and places the computing system in the wait state.

Channel-Check Handler (CCH)

When a channel-check occurs, CCH prepares for a retry of the unsuccessful I/O operation by an error recovery procedure (ERP). It also collects data about the error, and places this data in a record to be written in SYS1.LOGREC.

CCH supports IBM 2860, 2870, and 2880 channels, and the integrated channels of System/370 Models 145 and 155. CCH is required for Model 65 Multiprocessing, for System/360 Models 85 and 195, and for System/370 Models 145, 155, 165, and 195. It is optional for System/360 Models 65, 75, 91, and 95; however, if the operating system includes APR, it must also include CCH for APR to function properly.

CCH consists of a central module that is channel-independent, and separate error-analysis modules for the IBM 2860, 2870, 2880, Model 145, and Model 155 channels. During nucleus initialization, the central module is made part of the nucleus, along with the error analysis modules for all online channels.

When a channel-check occurs, CCH receives control from the I/O supervisor. It collects information about the error, and formats a record for SYS1.LOGREC. If the error does not impair system integrity, CCH constructs an error recovery procedure interface block (ERPIB), and returns control to the I/O supervisor.

The I/O supervisor writes the channel error record into SYS1.LOGREC, and informs the operator that an error has occurred. It then passes the ERPIB to the appropriate error recovery procedure (device-dependent ERP), which uses the ERPIB to retry the unsuccessful I/O operation. (Exception: CCH does not produce an ERPIB for a channel data check: the ERP is able to retry the operation without it.)

If CCH finds that a channel error affects system integrity, it passes control to the system's CPU recovery management routine (SER0, SER1, or MCH). This routine informs the operator of the error, and places the computing system in the wait state. If the routine is MCH, it also writes the channel error record in SYS1.LOGREC.

For a more detailed description of CCH, refer to the **Input/Output Supervisor PLM**.

Error Recovery Procedures (ERPs)

ERPs are standard procedures performed by routines that attempt recovery from errors on I.O devices. They ensure that the routines, which are device-dependent, provide a uniform type and quality of information. For convenience, the routines themselves are generally referred to as ERPs.

When an error occurs during a read, write, or control operation, the appropriate ERP determines the type of error, and (when possible) retries the unsuccessful operation. The routine also determines the number of retries to be performed before the error is considered permanent. At completion of error processing, the routine causes termination of the I/O request, and notifies the user of completion (successful or unsuccessful).

IBM supplies ERPs for all IBM devices. At system generation, the user selects (or provides his own) ERPs for devices included in his system. The selected ERPs are placed in SYS1.SVCLIB; they are loaded into main storage when they are needed, except for a portion of the direct access ERP, which is permanently resident in main storage. The resident direct access ERP handles errors on the system residence device, and exceptional conditions such as end-of-cylinder, head-switching, and alternate track procedures.

For a more detailed description of ERPs, refer to the **Input/Output Supervisor PLM**.

Alternate Path Retry (APR)

When an I/O operation is to be retried because of a channel check, APR ensures that an alternate path (channel or selector subchannel) is used whenever possible. In addition, APR enables the operator to vary a path online or offline.

APR is required for Model 65 Multiprocessing, optional for other systems. For APR to be effective, CCH must be included in the same system; CCH is not available for System/360

Models 40 and 50. APR consists of two routines: the selective retry routine (part of the I/O supervisor) and the vary path processor (part of SVC 34). The selective retry function of APR is optional and the VARY PATH function of APR is standard.

Selective Retry Routine: After a channel-check, the unsuccessful I/O operation is retried by an error recovery procedure (ERP). The ERP retries the I/O operation some standard number of times before the error is considered permanent. Before each retry, APR ensures the use of an alternate path to the device by marking the previously used path offline. When only one path remains, APR restores all of the original paths, and the process is repeated until the I/O operation is successful or the standard number of retries is performed. (The standard number of retries is determined by the ERP, which is provided either by IBM or by the installation.)

By placing failing paths offline, APR ensures the use of an alternate path whenever one is available. The chance of a successful retry is thereby increased. When retry succeeds, APR restores all of the original channel paths, and normal system operation is resumed.

Vary Path Processor: The vary path processor enables the operator to vary a channel path online or offline. For example, the operator can vary a path offline when intermittent channel errors begin to degrade system performance.

For a more detailed description of APR, refer to the **Input/Output Supervisor PLM.**

Dynamic Device Reconfiguration (DDR)

DDR enables the operator to swap I/O devices that are allocated and in use. The operator can substitute one device for another, or simply interrupt processing on a device to carry out cleaning procedures. A device swap can be requested by the system (after a permanent I/O error) or by the operator (through the SWAP command).

In its basic form, DDR supports unit record devices, magnetic tape units (for standard-label or no-label tapes), and direct access devices with demountable storage volumes (except devices used for system residence). Options of DDR support tapes with nonstandard labels and direct access devices used for system residence.

The basic DDR facility is required for Model 65 Multiprocessing, optional for other systems. DDR consists of the SWAP command processor, which is part of SVC 34, and other routines, which are part of the I/O supervisor. Basic DDR is partly resident in main storage, while DDR with the system residence option is entirely resident (except for the command processor).

System-Requested DDR: When a permanent I/O error occurs on a device with a demountable storage volume (tape or direct access), the system requests a device swap to permit retry of the unsuccessful I/O operation. The operator can demount the volume and move it to another device (which may be on a different channel), or he can carry out cleaning procedures and remount the volume on the same device. (If the device is a shared DASD, the volume must be remounted on the same device.) If the volume is a tape reel, DDR repositions the volume after it has been remounted.

For system residence devices, DDR receives control from transient error fetch, the error fetch sequence, or the resident DASD error recovery procedure. For other devices (tape and direct access), DDR receives control from the outboard recorder (OBR) routine of the I/O supervisor.

Operator-Requested DDR: The operator can request a device swap at any time by entering a SWAP command at the console. Before entering the SWAP command, however, the operator must complete (or cancel) any swap requested by the system. As an example, the operator can request a device swap when a unit record device requires intervention, but cannot be made ready due to a permanent error condition. (The system does not request a device swap after an error on a unit record device.)

For a more detailed description of DDR, refer to the **Input/Output Supervisor PLM**.

Recovery Management in a Multiprocessing Environment

All functions performed by the machine-chack handler (MCH) routine, the channel-check handler (CCH) routine, the alternate path retry (APR) routine, and the dynamic device reconfiguration (DDR) routine for a single-CPU environment are performed in the multiprocessing environment. When a failure occurs, the non-failing CPU is placed into a timed wait loop pending succussful completion of recovery management operations.

An additional feature of the MCH routine in the multiprogramming environment may permit system operation to continue even though machine failures have occurred that would otherwise cause system operation to stop. When normal retry and repair procedures are unsuccessful but damage is confimed to an area of main storage in which there are no critical system routines, (e.g., master scheduler, SQS, link pack area, nuclues), the MCH routine automatically prevents further usage of the failing area. (System operation can then be resumed in the remaining operative main storage areas.) Where possible, the MCH routine also frees for reassignment the resources assigned to the routines contained in the failing area.

MCH, CCH, APR, and DDR are automatically included in every Model 65 Multiprocessing System (except DDR options that support system residence devices and nonstandard label tape). MCH is described in the **Machine-Check Handler for System/360 Model 65 PLM**; CCH, APR, and DDR are described in the **Input/Output Supervisor PLM**.

Task Directory for Section V: Logic Summary

For information about:

- Processing operator commands, see
  Reading the Command
  Scheduling the Command
  Executing the Command

- Commands entered through the console, see
  Console Communications Task

- Commands entered through the input stream, see
  Reading Tasks
  Commands and Data Sets

- The job stream, see
  Reading Tasks
  Initiating Tasks
  Writing Tasks

- Work Queues, see
  Input Work Queues
  Contents of a Work Queue Entry
  Task Control Block Queue
  Request Block Queue
  Storage Allocation in the System Queue Area
  Timer Queue

- Processing of job steps, see
  Preparing of Job Step for Execution
  Terminating a Job Step
  Restarting a Job Step

- Processing of tasks, see
  Reading Tasks
  Initiating Tasks
  Writing Tasks
  Task control Block Queue
  Passing Control to a Program of a Task
  Dispatcher Routine
  Time Slicing

- Allocating storage, see
  Storage Allocation in the Dynamic Area
  Storage Allocation in a Region
  Rollout/Rollin
  Determining Available Storage
  Subpools
  Storage Allocation in the System Queue Area
  Contents Directory
  Load List

- Timing, see
  Time Slicing
  Pseudo-Clocks
  Timer Queue
  Time-of-Day Clock

- Preparing data before processing, see
  Open Processing
  Insuring Proper Volume Mounting
  Constructing Control Blocks
  Loading Access Method Routines

- Transferring data during processing, see
  Starting an I/O Operation
  Access Methods
  EXCP Routine
  Terminating an I/O Operation

- Closing data sets after processing, see
  Close Processing
  End-of-Volume Processing

- Using shared direct access storage devices, see
  Sharing Direct Access Devices
  Data Protection
  Control of Access Arm Movement

- Recovering from CPU malfunctions, see
  System Environment Recording, Option 0 (SER0)
  System Environment Recording, Option 1 (SER1)
  Machine-Check Handler (MCH)
  MCH for Model 65 and Model 65 Multiprocessor
  MCH for Models 85, 145, 155, 165, and 195

- Recovering from channel malfunctions, see
  Channel-Check Handler (CCH)
  Error Recovery Procedures (ERPs)
  Alternate Path Retry (APR)
  Dynamic Device Reconfiguration (DDR)

Appendix A: System Macro Instructions

This appendix contains the description and formats of macro instructions that allow users to modify or to obtain information from control blocks.

CIRB - Create IRB for Asynchronous Exit Processing

The CIRB macro instruction is included in SYS1.MACLIB and must be included in a system at system generation time to be used. The issuing of this macro instruction causes a supervisor routine (called the exit effector routine) to create an interruption request block (IRB). In addition, other operands of this macro instruction may specify the building of a register save area and/or a work area to contain interruption queue elements, which are used by supervisor routines in the scheduling of the execution of user exit routines.

```
Name      Operation Operand
[symbol]  CIRB      {EP=addrx}, KEY={PP  }, MODE={PP  }, [STAB=code,]
                    {         } {SUPR}      {SUPR}
                    {SVAREA= NO } , [WKAREA=value]
                    {        YES}
```

EP
specifies the entry point address of the user's asynchronous exit routine.

KEY
specifies whether the user's asynchronous routine will operate with a CPU protection key established by the supervisory program (SUPR) or with a protection key obtained from the task control block of the task for which the macro instruction is issued (PP).

MODE
specifies whether the user asynchronous routine will be executed in the problem program (PP) state or in a supervisory (SUPR) state.

STAB
indicates the status condition of the interruption request block. The "code" parameter may be either of the following:

(RE) to indicate that the IRB is reusable in it current form.

(DYN) to indicate that the storage area assigned to the IRB is to be made available (i.e., freed) for other uses when the asynchronous exit routine is completed.

SVAREA
specifies whether a register save area (of 72 bytes) is to be obtained from the main storage assigned to the problem program. If it is, the address of this save area is placed in the IRB. The asynchronous exit routine then follows the system register saving convention of using the SAVE and RETURN macro instructions. In this manner, a generalized subroutine can be used as an asynchronous exit routine.

WKAREA
specifies the number of doublewords (given as a decimal value) required for an area in which the routine issuing the macro instruction can construct interruption queue elements.

SYNCH - Synchronous Exits to Processing Program

The SYNCH macro instruction is a system macro instruction that permits control program supervisor call (SVC) routines to make synchronous exits to a processing program.

```
Name      Operation Operand
[symbol]  SYNCH     {entry-point}
                    {    (15)   }
```

entry-point
specifies the address of the entry point for the processing program that is to be given control.
If (15) is specified, the entry-point address of the processing program must have been pre-loaded into parameter register 15 before execution of this macro instruction.

SYNCH Macro Definition

```
          MACRO
&NAME     SYNCH     &EP
          AIF       ('&EP' EQ '').E1
          AIF       ('&EP'(1,1) EQ '(').REG
&NAME     LA        15,&EP                    LOAD ENTRY POINT ADDRESS.
          AGO       .SVC
.REG      AIF       ('&EP' EQ '(15').NAMEIT
&NAME     LR        15,&EP(1)                 LOAD ENTRY POINT ADDRESS.
.SVC      SVC       12                        ISSUE SYNCH SVC
          MEXIT
.NAMEIT   ANOP
&NAME     SVC       12                        ISSUE SYNCH SVC
          MEXIT
.E1       IHBERMAC  27,405
          MEND
```

Programming Notes: In general, use the SYNCH macro instruction when a control program in the supervisor state is to give temporary control to a processing program routine, and when expecting the processing program to return control to the supervisor state. The program to which control is given must be in main storage when the macro instruction is issued. The use of this macro instruction is similar to that of the BALR instruction in that register 15 is used for the entry point address. When the processing program returns control, the supervisor state bit, the storage protection key bits, the system mask bits and the program mask bits of the program status word are restored to the settings they had before execution of the SYNCH macro instruction.

Example: As a result of an OPEN macro instruction, label processing may be carried out to a point at which a user's processing program indicates that private processing is desired (or necessary). The control program's open routine then will issue a SYNCH macro instruction giving the entry point of the subroutine required for the user's private label processing.

STAE - Specify Task Asynchronous Exit

The STAE macro instruction permits control to be returned to a user exit routine when a task is scheduled for ABEND. When issuing the STAE macro instruction, a STAE control block (SCB) is created and initialized with the address of your exit routine. When issuing multiple

STAE requests within the same program, the SCB associated with the last issued STAE request becomes the active SCB: it will be the first to gain control when an ABEND is scheduled. If the active SCB is canceled, the preceding SCB, if there is one, will become the active SCB.

Notes:

- Do not cancel or overlay an SCB not created by a user program.
- The execution of a LINK macro instruction does not cancel the active SCB for the program in control.

Execute and Standard Form of STAE

```
Name      Operation Operand
                    {    0          }          {OV}
[symbol]  STAE      {exit address   } , [ {CT} ]   ,PARAM=list address

                  [,XCTL={YES}]  [,PURGE= {QUIESCE}]  [,ASYNCH= {NO }]
                         {NO }           {HALT   }              {YES}
                                         {NONE   }
                                ,MF=(E, remote list address )
                                                      (1)
```

exit address
specifies the address of a STAE exit routine to be entered if the task issuing this macro instruction terminates abnormally. If 0 is specified, the last SCB created is canceled and the previously created SCB becomes current. The address may be loaded into one of the general registers (r_1) 2 through 12.

Note: If you use the Execute form of the macro and specify a zero, the exit address in the parameter list will be zeroed.

OV
indicates that the parameters passed in this STAE macro instruction are to overlay the data currently in the SCB.

CT
indicates the creation of a new active SCB.

PARAM=
specifies the address of a parameter list containing data to be used by the STAE exit routine when it is scheduled for execution. The address may be loaded into one of the general registers (r_2) 2 through 12.

XCTL=YES
indicates that the STAE macro instruction will not be canceled if an XCTL macro instruction is issued.

XCTL=NO
indicates that the STAE macro instruction will be canceled if an XCTL is issued.

PURGE=QUIESCE
indicates that all active input/output operations will be purged with the quiesce option. If this fails, active input/output operations will be purged with the halt option.

Note: If you use the execute form of the STAE macro instruction and omit the PURGE parameter, QUIESCE will not be the default; the option specified for the preceding use of STAE will be used.

PURGE=HALT
indicates that all active input/output operation will be purged with the halt option.

PURGE=NONE
indicates that all active input/output operations will not be purged.

ASYNCH=NO
indicates that asynchronous exit processing will be prohibited while STAE exit processing is being done.

ASYNCH=YES
indicates that asynchronous esit processing will be allowed while STAE exit processing is being done.

MF=(E,[remote list adress][(1)])
Indicates the execute from of the STAE macro instruction using a remote parameter list. The address of the remote parameter list can be loaded into register 1, in which case MF=(E,(1)) should be coded.

Note: When using the Execute form of the STAE macro instruction and omitting the ASYNCH parameter, the option specified for the preceding use of STAE will be used.

List Form of STAE

Use the List form of the STAE macro instruction to construct program parameter lists. The description of the Standard and Execute forms describes the List form with the following exceptions:

```
Name      Operation Operand
[symbol] STAE       exit address     [,PARAM=list address]
                                     [,PURGE= {QUIESCE}]  [,ASYNCH= {NO }]
                                              {HALT   }           {YES}
                                              {NONE   }
                                      ,MF=L
```

exit address
any address that may be written in an A-type address constant.

MF=L
indicates the List form of the STAE macro instruction.

There are several conditions that you should be aware of when you use the PURGE and ASYNCH parameters of the STAE macro instruction.

- If the user exit routine requests a supervisor service that requires asynchronous interruptions to complete its normal processing, you must specify ASYNCH=YES.

- Specify ASYNCH=YES if you use an access method that requires asynchronous interruptions to complete its normal processing and you have specified PURGE=QUIESCE.
- When using the Indexed Sequential Access Method (ISAM) and specifying PURGE=HALT, only the I/O event for which the PURGE is done will be posted. Subsequent ECBs will not be posted; this causes the ISAM CHECK routine to treat purged input/output operations as waiting input/output operations and you will never get past the CHECK in your program.
- Specify ASYNCH=YES when you have the following combination of conditions: an access method that requires asynchronous interruptions to complete its normal processing, a specifications of PURGE=NONE, and a request of CHECK in your user exit routine.
- When specifying PURGE=HALT and an ISAM data set is being updated when a failure occurs, part of the data set may be destroyed.
- If quiesced input/output operations are not restored when using ISAM, the ISAM CHECK routine will treat purged input/output operations as waiting input/output operations and part of the ISAM data set may be destroyed if it is being updated when a failure occurs.
- If input/output operations are allowed to complete while the exit routine is in progress and there is a failure in the I/O processing, an ABEND recursion will be encountered when the I/O interrupt occurs. This can be misleading because it will appear that your exit routine failed while the actual cause of the failure was in the I/O processing.

Programming Notes

When control is returned to the user after the STAE macro instruction has be issued, register 15 contains one of the following return codes:

Code	Meaning
00	An SCB is successfully created, overlaid, or cancelled.
04	Storage for an SCB is not available.
08	The user is attempting to cancel or overlay a non-existent SCB, or is issuing a STAE in his STAE exit routine.
0C	The exit routine or parameter list address is invalid.
10	The user is attempting to cancel or overlay an SCB not associated with his level of control.

When a program with an active STAE environment encounters and ABEND situation, control is returned to the user through the ABEND/STAE interface routine at the STAE exit routine address. The register contents are as follows:

- Register 0:

Code	Indication
0	Active I/O at time of ABEND was quiesced and is restorable.
4	Active I/O at time of ABEND was halted and is not restorable.
8	No I/O was active at the time of the ABEND.

- Register 1:Address of a 104-byte work area:

0	STAE exit routine parameter list addr or 0		ABEND completion code
8	PSW at time of ABEND		
16	Last P/P PSW before ABEND		
24	Registers 0-15 at time of ABEND (64 bytes)		

If problem program issued STAE:

88	Name of ABENDing program or 0		
96	Entry point addr of ABENDing program		0

If supervisor program issued STAE:

88	Request Block addr of ABENDing program		0
96	0		

- Register 2-12: Unpredictable.
- Register 13: Address of a supervisor-provided register save area.
- Register 14: Address of an SVC 3 instruction.
- Register 15: Address of the STAE exit routine.

Register 13 and 14, if used by the STAE exit routine, must be saved and restored prior to returning to the calling program. Standard subroutine conventions are employed.

If storage was not available for the work area, the register contents upon entry to the STAE exit routine are as follows:

- Register 0: 12 (decimal).
- Register 1: Flags and completion code.
- Register 2: Address of STAE exit parameter list.
- Register 3-13: Unpredictable.
- Register 14: Return address.
- Register 15: Exit routine address.

The STAE exit routine may contain an ABEND, but must not contain either a STAE or an ATTACH macro instruction. At the time the ABEND is scheduled, the STAE exit routine must be resident as part of the program issuing STAE, or brought into storage via the LOAD macro instruction.

Scheduling of STAE and STAI Exit and Retry Routines

Each STAE exit routine is represented by one or more STAE control blocks (SCBs). Each STAE control block is queued in a last-in, first-out order to the TCB (TCBNSTAE field) of the task within which they were created. STAI control blocks also represent exit routines, but are created when the STAI operand is specified in an ATTACH macro instruction. STAI control blocks are always placed at the top of the queue (ahead of the STAE control blocks) in a last-in, first-out order and are propagated (a duplicate STAI control block is created and queued) to all lower-level subtasks of the subtask created with the STAI operand. Thus, if task A attached subtask B specifying the STAI operand, and subtask B attached subtask C which, in turn, attached subtask D, a STAI control block would be created and queued to the TCB for subtask B, and could be propagated to the queues originating at the TCBs for subtask C and subtask D. If a STAI control block were created for subtask C (the ATTACH macro instruction issued by subtask B specified the STAI operand), this STAI control block would be placed at the top of subtask C's SCB queue ahead of the STAI control block created for subtask B. In this case, both STAI control blocks would be propagated to the TCB for subtask D. All STAI control blocks precede all STAE control blocks on the SCB queue.

If a task is scheduled for abnormal termination, the exit routine specified by the most recently issued STAE macro instruction (represented by the highest STAE control block on the queue) is given control and executes under a program request block created by the SYNCH service routine. The STAE exit routine must specify, by a return code in register 15, whether a retry routine is to be scheduled. If no retry routine is to be scheduled (return code=0) and this is a subtask with a STAI control block on the SCB queue, the exit routine specified in the STAI control block is given control. If there is no STAI control block on the queue, abnormal termination continues.

If the STAE exit routine indicates that a retry routine has been provided (return code=4), register 0 must contain the address of the retry routine and register 1 must contain the address of the same work area passed to the exit routine. (The first word of the work area may be modified by the exit routine to point to another parameter list in his region.) The STAE control block is freed and the request block terminated up to, but not including, the RB of the program that issued the STAE macro instruction. This is done by placing an SVC 3 instruction in the old PSW field of each RB to be purged. In addition, open DCBs which can be associated with the purged RBs are closed and queued I/O requests associated with these DCBs being closed are deleted from the I/O restore chain..

The RB purge is an attempt to cancel the effects of partially executed programs that are at a lower level in the program hierarchy than the program under which the retry will occur. However, certain effects on the system will not be canceled by this RB purge. Examples of these effects are as follows:

- Subtasks created by a program to be purged.
- Resources allocated by the ENQ macro instructions.
- DCBs that exist in dynamically acquired main storage.

When your STAE exit routine gains control, it can examine the code in register 0 to determine if there were active input/output operations at the time of the ABEND and if the input/output operations are restorable. If there are quiesced restorable input/output operations, you can restore them, in the STAE retry routine, by using word 26 in the work area. Word 26 contains the link field passed as a parameter to SVC Restore. SVC Restore is used to have the system restore all I/O requests on the I/O restore chain.

Users can selectively restore specific I/O requests on the I/O restore chain by using word 2 in the work area. Word 2 contains the address the first I/O block on the I/O restore chain. This address can be used as a starting point for issuing EXCP for the I/O requests that you want to restore.

In supervisor mode, users may want the failing task to remain in its present status and not be reestablished. A retry routine may be scheduled without a purge of the RB chain by returning to the ABEND/STAE interface routine with an 8 in register 15, and registers 0 and 1 initialized as described above. If the STAE retry routine is scheduled, the system automatically cancels the active SCB and the preceding SCB, if there is one, will become the active SCB. Users wanting to maintain within the retry routine must reestablish an active SCB within the retry routine, or must issue multiple STAE requests prior to the time that the retry routine gains control. Also, if a STAI had been issued for this task, it must be reissued by the retry routine to be made effective again.

A STAI exit routine, if specified in a previous ATTACH macro instruction, will receive control if a STAE exit routine is not specified, if a STAE exit routine is specified but indicates that a retry routine is not provided, if a STAE exit routine terminates abnormally, or if a STAE or a STAI retry routine abnormally terminates. The STAI exit routine must specify by a return code in register 15 one of the following:

Return Code	Action to be Taken
0	No retry provided. The next STAI exit routine is to be given control or, if there is not another STAI exit routine, abnormal termination is to continue.
16	No further STAI processing is to occur Abnormal termination processing is to continue.
4 or 12	A retry routine is to be scheduled and the request block queue is to be purged.
8	A retry routine is to be scheduled but the request block queue is not to be purged (if the user is not in supervisor mode, this return code will be ignored and abnormal termination processing continues).

When the RB queue is not to be purged, a new PRB is created for the retry routine and placed on the RB queue immediately after the SVRB for the ABEND routine, so that when the ABEND routine returns via an AVC 3 instruction the retry routine will receive control.

If the RB queue is to be purged, the STAI retry routine is executed under the PRB for the last STAE or STAI exit routine or, if no PRB for an exit routine exists on the queue, under the most recently created PRB that is pointed to by the oldest (first created) non-PRB on the queue (the oldest non-PRB will be the last RB purged).

Like the STAE/STAI exit routine, the STAE/STAI retry routine must be in storage when the exit routine determines that retry is to be attempted. If not already resident within your program, the retry routine may be brought into storage via the LOAD macro instruction by either the user's program or exit routine.

Upon entry to the STAE/STAI retry routine, register contents are as follows:

- Register 0: 0
- Register 1: Address of the work area, as previously described, except that word 2 now contains the address of the first I/O Block and word 26 now contains the address of the I/O restore chain.
- Register 2-13: Unpredictable.
- Register 14: Address of an SVC 3 instruction.
- Register 15: Address of the STAE/STAI retry routine.

The retry routine should use the FREEMAIN macro instruction to free the 104 bytes of storage occupied by the work area when the storage is no longer needed. This storage should be freed from subpool 0 which is the defualt subpool for the FREEMAIN macro instruction.

Again, if the ABEND/STAE interface routine was not able to obtain storage for the work area, register 0 contains a 12; register 1, the ABEND completion code upon entry to the STAE retry routine; and register 2, the address of the first I/O Block on the restore chain, or 0 if I/O is not restorable.

Note: If the program using the STAE macro instruction terminates via the EXIT macro instruction, the EXIT routine cancels all SCBs related to the terminating program. If the program terminates via the XCTL macro instruction, the EXIT routine cancels all SCBs related to the terminating program except those SCBs that were created with the XCTL=YES option. If the program terminates by any other means, the terminating program must reinstate the previous SCB by cancelling all SCBs related to the terminating program.

ATTACH--Create a New Task

This explicit form of ATTACH permits greater flexibility in both the use and the result of use of the ATTACH macro instruction. This form of the macro instruction differs from the implicit form by the addition of six keyword parameters to those described for the implicit form in the **Supervisor Services and Macro Instructions** publication. Only the added six parameters are shown and explained in this description.

These six parameters can be used only with tasks whose protection key is zero. If they are used with other tasks, the default values are used.

```
Name      Operation                 Operand
[symbol]  ATTACH    ... ,JSTCB= {YES}    ,SM= {SUPV }   ,SVAREA= {YES}
                                {NO }         {PROB}             {NO }
                    ,KEY= {ZERO}  ,GIVEJPQ= {YES}   ,JSCB=jscbaddr
                          {PROP}           {NO }
```

For ordinary ATTACH macro instruction parameters, see the description in the **Supervisor Services and Macro Instructions** publication.

JSTCB
Address to be placed in the TCBJSTCB field of the TCB of the newly created task. The address determines whether the attached task is a new job step or a task in the present job step. A new job step is required if the ownership of programs is to pass from the attaching to the attached task, that is, if you are coding GIVEJPQ=YES in the macro instruction. (Also, see note below.)

YES - Address of the TCB of the newly created task, that is, this TCB points to itself, thus creating a new job step. A new job step is required if ownership of programs is being transferred from the attaching to the attached task, that is, if you are coding GIVEJPQ=YES in the macro instruction.

NO- Address of the TCB of the task using the ATTACH, that is, the attached task is to be a task in the present job step.

,SM
Operating state of the machine when executing the attached task.

SUPV -Supervisor mode.

PROB -Problem program mode.

,SVAREA
Need for save area.

YES - A save area is needed for the attaching task. The ATTACH routine will obtain a 72 byte save area. If both attaching and attached task share subpool zero, the save area is obtained there, otherwise it is obtained from a new 2K byte block.

NO- No save area is needed.

,KEY
Protection/Key of the newly created (attached) task.

ZERO - Zero.

PROP - Copy the key from the TCBPKF field of the TCB for the task using the ATTACH.

,GIVEJPQ
Ownership of programs used by the attaching task. If ownership is to pass to the attached task, the attached task must be a new job step, that is, you must use JSTCB=YES. (Also see note below.)

YES- Pass ownership to the newly created task. On completion of the new task all programs, both those passed to the new task by the old and those acquired by it, are freed.

NO- Ownership of programs used by the attaching task remain with that task; programs acquired by the attached task remain with it. The attached task shares use of the programs of the attaching task during their common existence. At the conclusion of the attached task, the programs it acquired are freed; when the attaching task terminates, its programs are freed.

,JSCB
Job step control block address.
If specified, that job step control block is used for the new task. If not specified, the job step control block of the attaching task is also used for the new task.

Note: If the task to be attached is to be a separate step (JSTCB=YES), ownership of programs may be passed (GIVEJPQ=YES) or retained (GIVEJPQ=NO). If the newly attached task is not to be a separate step (JSTCB=NO), ownership of programs cannot be passed but must be retained (GIVEJPQ=NO). The following table summarizes these combinations.

		JSTCB=	
		YES	**NO**
GIVEPJQ=	YES	Valid	Invalid
	NO	Valid	Valid

IMGLIB -- Open or Close SYS1.IMAGELIB

The IMGLIB macro instruction is used to open or close SYS1.IMAGELIB. When issued to open the Image Library, it is usually followed by a BLDL macro instruction and a LOAD macro instruction which, respectively, search the library for the image and load it into storage.

```
Name      Operation Operand
[symbol] IMGLIB    OPEN,dcb addr
                   CLOSE
```

OPEN
specifies that SYS1.IMAGELIB is to be opened and the address of the DCB returned in register one.

CLOSE
specifies that IMAGELIB is to be closed.

dcb addr
is either the address of the IMAGELIB DCB or is a register containing the IMAGELIB DCB address.

QEDIT -- Linkage to SVC 34

The QEDIT macro instruction generates the required entry parameters and the linkage to SVC 34 for the following uses:

- Dechaining and freeing of a CIB (command input buffer) from the CIB chain for a task.
- Setting a limit for the number of CIBs that may be simultaneously chained for a task.

The format of the QEDIT macro instruction and an explanation of the operands are as follows:

```
Name      Operation Operand
[symbol] QEDIT     ORIGIN=address [,BLOCK=address]
                                  [,CIBCTR=number]
```

ORIGIN
The address of the pointer to the first CIB on the CIB chain for the task. This address is obtained using the EXTRACT macro instruction. If ORIGIN is the only parameter specified, the entire CIB chain will be freed.

,BLOCK
The address of the CIB that is to be freed from the CIB chain for a task.

,CIBCTR
An integer (from 0 to 255) to be used as a limit for the number of CIBs to be chained at any time for a task.

address
Any address valid in an RX instruction or one of the general registers (2-12) previously loaded with the indicated address. The register must be designated by a number or symbol added within the parentheses.

WTO/WTOR -- Write-to-Operator and Write-to-Operator with Reply

The write-to-operator (WTO) and write-to-operator with reply (WTOR) macro instructions have two special operands, MSGTYP and MCSFLAG. Only operators familiar with the multiple console support (MCS) communications task should use these operands, since using them improperly could impede the entire message routing scheme. These operands set flags to indicate that certain system functions must be performed, or that a certain type of information is being presented by the WTO or WTOR macro instruction.

The MSGTYP and MCSFLAG operands may be specified in either the standard or list form of the WTO and WTOR macro instruction. The standard form of the WTO macro instruction is shown below.

```
Name      Operation Operand
[symbol] WTO        'message' [ROUTCDE=(number [number],...)]
                    [,DESC=number]
                                 {N        }
                                 {Y        }
                    [,MSGTYP=    {JOBNAMES}   ]
                                 {STATUS   }
                    [,MCSFLAG=(name [, name],...)]
```

'message'
specifies that the message text is to be placed between the first and second apostrophes.

ROUTCODE=
specifies routing codes assigned to the message.

DESC=
specifies the descriptor codes assigned to the message.

MSGTYPE=JOBNAMES or MSGTYP=STATUS
specifes routing the message to the console which issued the DISPLAY JOBNAMES or DISPLAY STATUS command, respecitvely. When the operating system identifies the message type, the message will be routed only to those consoles that requested the information. If ommitted, messages routed as specified in the ROUTCDE parameter.

MSGTYP=Y or MSGTYP=N
specifies that two bytes are to be reserved in the WTO or WTOR macro expansion so flags can be set to describe the MSGTYP functions desired. Y specifies that two bytes of zeros are to included in the macro expansion at displacement WTO+4 plus the total length of the message text, descriptor code, and routing code fields. N, or ommission of the MSGTYP parameter, specifies that the two bytes are not needed, and that the message is to be routed as specified in the ROUTCDE parameter. If an invalid MSGTYP value is encountered, a value of N is assumed, and a diagnostic message is produced with a severity code of 8.

The bit definitions for MSGTYP=Y follow:

Bit 0: DISPLAY JOBNAMES

Bit 1: DISPLAY STATUS

Bits 2-15: Reserved for future system use. Must be zeros.

When specifying MSGTYP=Y, set the appropriate message identifier bit in the MSGTYP field of the macro expansion. Prior to executing the WTO or WTOR SVC (SVC35), also set byte 0 of the MSGFLAG field in the macro expansion t hexadecimal 10. This value indicates that the message routing criteria will use the MSGTYP field. When the system identifies the message type, the message will be routed to all the consoles that requested that particular type of information. Routing codes, if present, will be ignored.

MCSFLAG
specifies that the macro instruction should set bits in the MSCFLAG filed as indicated by each name coded.

Appendix B: Control Character Transformations

To help determine what can be done with a writer routine, this appendix describes the control character transformation features of the standard writer.

Effectively there are nine control character combinations that can occur between input data set records and output data set records. Both data sets may have records whose control characters are either USASI type (acc) or machine type (mcc), or the records may not contain any control characters. However, within any given data set, the records all must contain the same form of control character. The standard writer has procedures to handle control character transformations for both an output to a card punch unit and an output to a printer unit.

Card Punch Unit

If an input data set record does not have a control character, the standard writer produces one that indicates output into pocket 1 of the punch. If the output unit is a tape unit and the ultimate destination is a punch unit, the standard writer assumes that the punch unit is either a 2540, 3525, or a 2520 unit and sets a control character accordingly. The standard writer translation of punch-type control characters is given in Figure 31. In this table, the first three columns of figures are machine control character codes, and the right hand column of figures represent USASI control character codes. Each record that requires a control character has one of these 8-bit codes attached to it. Input records whose control characters are mcc and are shown in horizontal rows 1, 2, 5, and 6 are given the acc code of "V" if they are placed in an output data set that has acc. An mcc given in rows 3 or 4 is changed to an acc code of "W" However, if translation is from an acc input to an mcc output, the standard writer translates the control character into the appropriate mcc on the same horizontal row.

	Machine Control Characters			
Stacker Unit	3525/ 2540 Punch	2520 Punch	1442 Punch	USASI Control Characters
1. P1	00000001	00000001	10000001	11100101 (V)
2. P1 Column Binary	00100001	00100001	10100001	
3. P2	01000001	01000001	11000001	11100110 (W)
4. P2 Column Binary	01100001	01100001	11100001	
5. RP3	10000001			
6. RP3 Column Binary	10100001			

Figure 31. Control Character Translation for Punch Unit Output

Printer Unit

When the output unit is a printer, the standard writer prevents overprinting between data sets. If the successive data sets contain records with the same type of control character, there is no overprinting problem. If the control characters vary from one data set to the next, the standard writer solutions are applications of the technique illustrated by Figure 32. In this figure, the possible forms of the input record control characters are given in the left hand column. The three right hand columns (containing cases 1-9) represent the possible forms of the output record control characters. Within each of the 12 main sections of the figure is shown a symbolic representation of a data set whose records possess the indicated form of control character. Each record consists of a print line representation and a control character representation (where appropriate). For records with acc, the control character is shown preceding the print line, since the effect of the control character occurs before the line is printed. For records with mcc, the converse is shown. An input record with no control character is treated as if it had an acc. Because of this variance in the printer's mechanical action, whenever there is a control character transformation, the standard writer places a transformed control character with an output data set record other than the record to which the character was attached in the input data set.

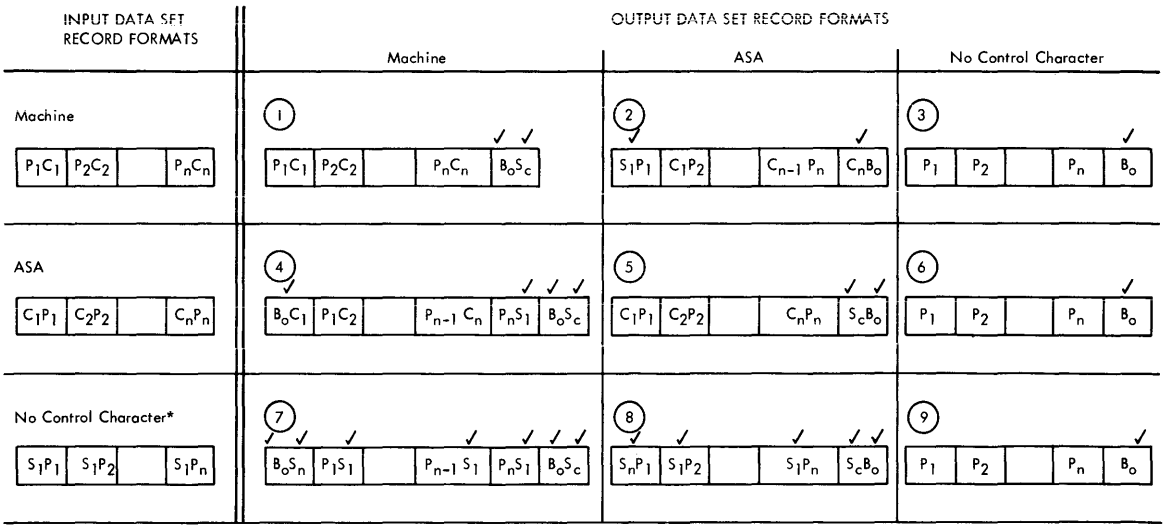
In Figure 32, case 1 and 5 represent situations in which there is the same type of control character in the output as there is in the input. Thus, for records 1 through n, there is no change in the record format. However, there is a provision to allow for the possibility that two consecutive input data sets may have different control characters. In this case, a minimum separation between the data sets as they appear in the output data set is provided as indicated by the printing of blanks and suppressing the spacing of the printer to allow another control character to take effect. The 'extra' record (S B or B S) provides the more important function of forcing out the last record of the current data set before the writer's processing of that data set is done.

In cases 2 and 4 of Figure 33, the output data sets records have different control characters than the input data set records. Case 2 shows that the standard writer generates a 1-line space control character to precede the first print line of the output. When the output is written, each control character of an input record is then attached to the next record. The last input record control character (C_n) is attached to a line of blanks (B). In case 4, the first input record control character is attached to a line of blanks, and each of the other control characters is attached to a preceding record, as indicated. The last input record (P_n) has a writer-generated space 1-line control character attached to it before the buffering and forcing record (B S) generated by the writer is put out.

Cases 7 and 8 show that the standard writer first generates a 'skip to channel 1' control character and then generates '1 line space' and then generates '1 line space ' control characters for all but the last control character. The last control character is the space suppression character shown as part of the buffering or forcing record generated.

Cases 3, 6, and 9 show that if no control characters are required in the output data set, the records are printed consecutively and a line of blanks generated by the writer is printed after the final record in a data set. Any control character appearing in the input data set are dropped in the output data set.

Notice that in all cases involving control characters in the output data set, the standard writer allows for (1) an output record to force the printing of the last record of an input data set and (2) a means of minimum buffering between data sets by using generated control characters and print lines in conjunction with the actual data set control characters.



= Writer generated.
* = No control character on input causes the standard writer to generate an ASA control character as indicated.
B_o = A print line of blanks.
C_1-C_n = Control characters of records 1-N of a given data set.
P_1-P_n = Print lines of a given data set.
S_1 = A control character causing a 1-line space.
S_c = A control character causing spacing to be suppressed.
S_n = A control character causing a skip to channel 1.

Figure 32. Symbolic Representation of Record Formats

APP B

The standard writer translation of printer-type control characters is given in Figure 33. In this figure, the type of action indicated is given in the left-hand column. The middle column and the right-hand column show, respectively, the bit settings of the control character byte for machine type and USASI type control characters corresponding to the entries in the left-hand column. A control character transformation is effected by changing the bit-configuration of the control character byte as indicated in the figure.

Action Desired	**Machine Type Control (1403, 1404, 1443, 3211** Printers)	USASI Type Control
Write space 0	00000001	01001110
Write space 1	00001001	01000000
Write space 2	00010001	11110000
Write space 3	00011001	01100000
Write skip to channel 1	10001001	11110001
Write skip to channel 2	10010001	11110010
Write skip to channel 3	10011001	11110011
Write skip to channel 4	10100001	11110100
Write skip to channel 5	10101001	11110101
Write skip to channel 6	10110001	11110110
Write skip to channel 7	10111001	11110111
Write skip to channel 8	11000001	11111000
Write skip to channel 9	11001001	11111001
Write skip to channel 10	11010001	11000001
Write skip to channel 11	11011001	11000010
Write skip to channel 12	11100001	11000011

Figure 33. Control Character Translation for Printer Unit Output

When machine control characters are used which indicate spacing or skipping without writing (bit 6 set to 1, e.g., write and space 0-00000011) the standard writer generates the indicated USASI control character and also generates a blank record of the proper type and length.

Appendix C: RESERVE Macro Instruction Used with Shared DASD Option

The RESERVE macro instruction is used to reserve a device for use by a particular system; it must be issued by each task needing device reservation. The RESERVE macro instruction protects the issuing task from interference by other tasks in the system. Each task issuing the RESERVE macro instruction must also use the DEQ macro instruction to release the device; two RESERVE instructions for the same resource without an intervening DEQ will result in an abnormal termination unless the second one specifies the keyword parameter RET=. (If a restart occurs when a RESERVE is in effect for devices, the system will not restore the RESERVE; the user's program must reissue the RESERVE.) Even if a DEQ is not issued for a particular device, termination routines will release devices reserved by a terminating task.

RESERVE Macro Instruction

The use of the RESERVE macro instruction is explained below:

```
Name      Operation Operand
[symbol] RESERVE   (qname address,rname address,[E]
                                                [S]
          [rname length],SYSTEMS) [,RET={TEST}] ,UCB=pointer address
                                  [     {USE }]
                                  [     {HAVE}]
```

qname
: the address in main storage of an eight-character name. Every task (within the system) issuing RESERVE against the same resource (data and device) must use the same qname-rname combination to represent the resource. The qname should not start with SYS.

rname address
: the address in main storage of a name used in conjunction with the qname to represent the resource. The rname can be qualified, and may be 1 to 255 bytes in length.

[E or S]
: specify either exclusive control of the resource (E); or shared control with other tasks in the system (S). Default to E.

rname length
: the length, in bytes, of rname. If omitted, the assembled length of rname is used. If zero (0) specified, the length of rname must be contained in the first byte of the field designated by the rname address.

SYSTEMS
: specifies that the resource represented by qname-rname is known across systems as will as within the system whose task is issuing RESERVE, i. e., the resource is shared between systems.

APP C

RET
specifies a conditional request for all of the resources named in the RESERVE macro instruction. If the operand is omitted, the request is unconditional. The types of conditional requests are as follows:

TEST -- tests the availability status of the resources but does not request control of the resources.

USE -- specifies that control of the resources be assigned to the active task only if the resources are immediately available. If any of the resources are not avialable, the active task is not placed in a wait condition.

HAVE -- specifies that control of the resources is requested only if a request has not been made previously for the same task.

Return codes are provided by the control program only is RET=TEST, RET=USE, or RET=HAVE is designated; otherwise, return of the task to the active condition indicates that control of the resource has been assigned to the task. Return codes are identical to those supplied by the ENQ macro instruction (see the **Data Management Macro Instructions** publication).

UCB=pointer address
the keyword specifies either:

1. The address of a fullword that contains the address of the Unit control Block (UCB) for the device to be reserved.

2. A general register (2-12) that points to a fullword containing the address of the unit contol block for the device to be reserved.

To use the Shared DASD option in higher level languages, write an assembler language subroutine to issue the RESERVE macro instruction. Pass the following information to this routine: ddname, qname address, rname address, rname length, and RET parameter.

The EXTRACT Macro Instruction

The EXTRACT macro instruction is used to obtain the address of the task input/output table (TIOT) from which the UCB address can be obtained. The topic "Finding the UCB Address" explains this procedure.

Releasing Devices

The DEQ macro instruction is used in conjunction with RESERVE just as it is used with ENQ. It must describe the same resource and its scope must be stated as SYSTEMS; however, the UCB=pointer address parameter is not required. If the DEQ macro instruction is not issued by a task which has previously reserved a device, the system will free the device when the task is terminated.

Preventing Interlocks

Certain precautions must be taken to avoid system interlocks when the RESERVE macro instruction is used. The more often device reservations occur in each sharing system, the greater the chance of interlocks occurring. Allowing each task to reserve only one device minimizes the exposure to interlock. The system cannot detect interlocks caused by program use of the RESERVE macro instruction and enabled wait states will occur on the system or systems.

Volume Assignment

Since exclusive control is by device, not by data set, consider which data sets reside on the same volume. In this environment it is quite possible for two tasks in two different systems -- processing four different data sets on two shared volumes -- to become interlocked. For example, data sets A and B reside on device C, and data sets D and E reside on device F. Task X in system X reserves device C in order to use data set A; task Y in system Y tries to reserve device F in order to use data set D. Now task X in system X tries to reserve device F in order to use data set E and task Y in system Y tries to reserve device C in order to use data set B. Neither can ever regain control, and neither will complete normally. In MVT without job step timing, the job or jobs, will be canceled. When the system has job step time limits, the task, or tasks, in the interlock would be abnormally terminated when the time limit expires. Moreover, an interlock could mushroom, encompassing new tasks as these tasks try to reserve the devices involved in the existing interlock.

Program Libraries

When assigning program libraries to shared volumes, precaution must be taken to avoid interlock. For example, SVCLIB for system A resides on volume X, while SVCLIB for system B resides on volume Y. Task A in system A invokes a direct access device space management function for volume Y, resulting in that device being reserved. Task B in system B invokes a similar function for volume X. reserving that device. However, since the DADSM functions are transient SVCs, each load module transfers to another load module via XCTL. Since the SVCLIB for each system resides on a volume reserved by the other system, the XCTL marco instruction cannot complete the operation, therefore an interlock occurs in this particular case, since no access to SVCLIB is possible, both systems will eventually enter an enabled wait state.

Finding the UCB Address

This explains procedures for finding the UCB address for use the RESERVE macro instruction; it also shows a sample assembler language subroutine which issues the RESERVE and DEQ macro instructions and can be called by higher level languages.

Providing the Unit Control Block Address to RESERVE

The EXTRACT macro instruction is used to obtain information from the Task Control Block (TCB). The address of the TIOT can be obtained from the TCB in response to an EXTRACT. Prior to issuing an EXTRACT macro instruction, the user sets up an answer area in main storage which is to receive the requested information. One full word is required for each item to be provided by the control program. If the user wishes to obtain the TIOT address, he must issue the following form of the macro instruction:

```
EXTRACT   answer-area address, FIELDS=TIOT
```

The address of the TIOT is then returned by the control program, right adjusted, in the full work answer area.

The TIOT is constructed by job management routines and resides in main storage during step execution. The TIOT consists of one or more DD entries, each of which represents a data set defined by a DD statement for the jobstep. Each entry includes the DD name. Associated with each DD entry is the UCB address of the associated device. In order to find the UCB address, the user must locate the DD entry in the TIOT corresponding to the DD name of the data set for which he intends to issue the RESERVE macro instruction.

The UCB address can be obtained via the DEB and the DCB. The Data Control Block (DCB) is the block within which data pertinent to the current use of the data set is stored. The address of the Data Extent Block (DEB) is contained at offset 44 decimal after the DCB has been opened. The DEB contains an extension of the information in the DCB. Each DEB is associated with a DCB, and the two point to each other.

The DEB contains information concerning the physical characteristics of the data set and other information that is used by the control program. A device dependent section for each extent is included as part of the DEB. Each such extent entry contains the UCB address of the device to which that portion of the data set has been allocated. In order to find the UCB address, the user must locate the extent entry in the DEB for which he intends to issue the RESERVE macro instruction. (In disk addresses of the form MBBCCHHR, the M indicates the extent number starting with 0).

Procedures for Finding the UCB Address of a Reserved Device

If the data set is a multivolume sequential data set, it must be assumed that all jobs will process that data set in a sequential manner starting with the first volume of the data set. In this case, by issuing a RESERVE for the first volume only, the user effectively reserves all the volumes of the data set.

For data sets using the queued access methods in the update mode or for unopened data sets:

1. Extract the TIOT from the TCB.
2. Search the TIOT for the DD name associated with the shared data set.
3. Add 16 to the address of the DD entry found in step 2. This results in a pointer to the UCB address obtained in step the TIOT.
4. Issue the RESERVE macro specifying the address obtained in step 3 as the operand of the UCB keyword.

For opened data sets:

1. Load the DEB address from the DCB field labeled DCBDEBAD.
2. Load the address of the the field labeled DEBDVMOD in the DEB obtained in step 1. The result is a pointer to the UCB address in the DEB.
3. Issue the RESERVE macro specifying the address obtained in step 2 as the operand of the UCB keyword.

For BDAM data sets the user may reserve the device at any point in the processing in the following manner:

1. Open the data set successfully.
2. Convert the block address used in the READ/WRITE macro to an actual device address of the form MBBCCHHR. The publication **Data Management for System Programmers** shows how to convert addresses to the form MBBCCHHR.
3. Load the DEB address from the DCB field labeled DCBDEBAD.
4. Load the address of the field labeled DEBDVMOD in the DEB.
5. Multiply the "M" of the direct access address by 16.
6. The sum of steps 4 and 5 is the address of the correct extent entry in the DEB for the next READ/WRITE operation. The sum is also a pointer to the UCB address for this extent.
7. Issue the RESERVE macro specifying the address obtained in step 6 as the operand of the UCB keyword.

If the data set is an ISAM data set, QISAM in the load mode should by used only at system update time. Further, if it is a multivolume ISAM data set, it must be assumed that all jobs will access the data set through the highest level index. The indexes should never reside in main storage when the data set is being shared. In this case, by issuing a RESERVE macro for the volume on which the highest level index resides, the user effectively reserves the volumes on which the prome data and independent overflow areas reside. The following procedures can by used to achieve this:

1. Open the data set successfully.
2. Locate the actual device address (MBBCCHHR) of the highest level index. This address can be obtained from the DCB.
3. Load the DEB address from the DCB field labeled DCBDEBAD.
4. Load the address of the field labeled DEBDVMOD in the DEB.
5. Multiply the "M" of the actual device address located in step 2 by 16.
6. The sum of steps 4 and 5 is the address of the correct extent entry in the DEB for the next READ/WRITE operation. The sum is also a pointer to the UCB address for this extent.
7. Issue the RESERVE macro specifying the address obtained in step 6 as the operand of the UCB keyword.

RES and DEQ Subroutines

The following assembler language subroutine can be used by FORTRAN, COBOL, or assembler language programs to issue the RESERVE and DEQ macro instructions. Parameters that must by passed to the RESDEQ routine, if the RESERVE macro instruction is to be issued, are:

DDNAME - the eight character name of the DDCARD for the device to be reserved.

QNAME - an eight character name.

RNAME LENGTH - one byte (a binary integer) that contains the RNAME length value.

RNAME - a name from 1 to 255 charcters in length.

The DEQ macro instruction does not require the UCB=pointer address as a parameter. If the DEQ macro is to be issued, a fullword of binary zeros must be placed in the DDNAME field before control is passed.

```
RESDEQ   CSECT
         SAVE  (14,12),T     SAVE REGISTERS
         BALR  2,0           SET UP ADDRESSABILITY
         USING *,2
         ST    13,SAVE+4
         LA    11,SAVE       ADDRESS OF MY SAVE AREA IS STORED
         ST    11,8(13)      IN THIRD WORD OF CALLER'S SAVE AREA
         LR    13,11         ADDRESS OF MY SAVE AREA
         LR    9,1           ADDRESS OF PARAMETER LIST
         L     3,0(9)        DDNAME PARAMETER OR WORD OF ZEROS
         CLC   0(4,3),=F'0'  WORD OF ZEROS IF DEQ IS REQUESTED
         BE    WANTDEQ
*PROCESS FOR DETERMINING THE UCB ADDRESS USING THE TIOT
         XR    11,11         REGISTER USED FOR DD ENTRY
         EXTRACT ADDRTIOT,FIELDS=TIOT
         L     7,ADDRTIOT    ADDRESS OF TASK I/O TABLE
         LA    7,24(7)       ADDRESS OF FIRST DD ENTRY
NEXTDD   CLC   0(8,3),4(7)   COMPARE DDNAMES
         BE    FINDUCB
         IC    11,0(7)       LENGTH OF DD ENTRY
         LA    7,0(7,11)     ADDRESS OF NEXT DD ENTRY
         CLC   0(4,7),=F'0'  CHECK FOR END OF TIOT
         BNE   NEXTDD
         ABEND 200,DUMP      DDNAME IS NOT IN TIOT, ERROR
FINDUCB  LA    8,16(7)       ADDRESS OF WORD IN TIOT THAT
*                            CONTAINS ADDRESS OF UCB
*PROCESS FOR DETERMINING THE QNAME REQUESTED
WANTDEQ  L     7,4(9)        ADDRESS OF QNAME LENGTH
         MVC   QNAME(8),0(7) MOVE IN QNAME
*PROCESS FOR DETERMINING THE RNAME AND THE LENGTH OF RNAME
         L     7,8(9)        ADDRESS OF RNAME LENGTH
         MVC   RNLEN+3(1),0(7) MOVE BYTE CONTAINING LENGTH
         L     7,RNLEN
         STC   7,RNAME       STORE LENGTH OF RNAME IN THE
*                            FIRST BYTE OF RNAME PARAMETER
*                            FOR RES/DEQ MACROS
         L     6,12(9)       ADDRESS OF RNAME REQUESTED
         BCTR  7,0           SUBTRACT ONE FROM RNAME LENGTH
         EX    7,MOVERNAM    MOVE IN RNAME
         CLC   0(4,3),=F'0'
         BE    ISSUEDEQ
         RESERVE (QNAME,RNAME,E,0,SYSTEMS),UCB=(8)
         B     RETURN
ISSUEDEQ DEQ   (QNAME,RNAME,0,SYSTEMS)
RETURN   L     13,SAVE+4     RESTORE REGISTERS AND RETURN
         RETURN (14,12),T
         BCR   15,14
MOVERNAM MVC   RNAME+1(0),0(6)
ADDRTIOT DC    F'0'
SAVE     DS    18F
QNAME    DS    2F
RNAME    DS    CL256
RNLEN    DC    F'0'
         END
```

APP C

Appendix D: List of Acronyms and Abbreviations

This list contains the names associated with the acronyms used in this publication:

Acronym	Name	Acronym	Name
APR	alternate path retry	INIT	initiator
CCH	channel-check handler	I/O	input/output
CCW	channel command word	IOB	input/output block
CDE	contents directory entry	IPL	initial program loading
CPU	central processing unit	JCT	job control table
CSCB	command scheduling control block	JFCB	job file control block
CSW	channel status word	LTPC	local time pseudo-clock
CVT	communications vector table	MCH	machine-check handler
DADSM	direct access device space management	MCS	multiple console support
DASDI	direct access storage device initialization	MVT	multiprogramming with a variable number of tasks
DCB	data control block	M65MP	Model 65 Multiprocessing
DD	data definition	NIP	nucleus initialization program
DDR	dynamic device reconfiguration	OBR	outboard recorder
DEB	data extent block	OLTEP	online test executive program
DEQ	dequeue	PQE	partition queue element
DQE	descriptor queue element	PSA	prefixed storage area
DSB	data set block	PSW	program status word
DSCB	data set control block	RB	request block
ECB	event control block	RDR	reader
ENQ	enqueue	RMS	recovery management support
EOV	end-of-volume	SCT	step control table
ERP	error recovery procedure	SDR	statistical data recorder
ETXR	end-of-task exit routine	SER	system environment recording
EXCP	execute channel program	SHPC	six hour pseudo-clock
EXEC	execute	SIO	start I/O
FBQE	free block queue element	SIOT	step input/output table
FCB	forms control buffer	SMB	system message block

Acronym	Name	Acronym	Name
SMF	system management facilities	TSCE	time-slice control element
SPQE	subpool queue element	TSO	time sharing option
SVC	supervisor call	T4PC	twenty-four hour pseudo-clock
SYSGEN	system generation	UCB	unit control block
SYSOUT	system output	UCS	universal character set
SYSRES	system residence volume		
TCB	task control block	VTOC	volume table of contents
TIOT	task input/output table	WTR	writer
TQE	timer queue element	XCTL	transfer control

Appendix E: MVT Control Program Logic Manuals

The following list contains the names and abstracts of control program logic manuals that discuss the MVT configuration.

Basic Direct Access Method GY28-6771

This publication describes the internal logic of the IBM System/360 Operating System basic direct access method.

Basic Telecommunications Access Method GY28-6617

This publication explains the method and logic of the channel program generation for the basic telecommunications access method (BTAM).

Catalog Management GY28-6606

This manual provides detailed information on catalog management routines. These routines record identification of volumes used by data sets by maintaining information in logical records called indexes. The functions and structures of the routines are described, as are their relationships to other portions of IBM System/360 Operating System. This manual also describes the structure of catalog data sets that contain the indexes processed by catalog management routines.

Direct Access Device Space Management GY28-6607

This manual provides detailed information on direct access device space management (DADSM) routines. These routines control the user of external direct access storage by maintaining the information in data set control blocks. The functions and structures of the routines are described, as are their relationships to other portions of IBM System/360 Operating System. This manual also describes the structure of volume tables of contents which are processed by DADSM routines.

Graphic Access Method GY27-7113

This publication describes the operation of the graphics access method (GAM) for the IBM 2250 Display Unit, Models 1, 2, and 3, the IBM 2260 Display station (Local Attachment), and the IBM 2280/82 File Units.

Graphic Job Processor Support GY27-7159

This publication describes the internal logic of the graphic job processor (GJP) and the graphics interface task (GFX), which are features of the IBM System/360 Operating System that permits jobs to be defined and initiated by responding to frames displayed on an ibm 2250 Display.

Indexed Sequential Access Methods GY28-6618

This publication describes the program logic of the two indexed sequential access methods: the queued indexed sequential access method (QISAM) and the basic indexed sequential access method (BISAM). It also discusses the relationship of indexed sequential access method routines to other parts of the control program.

Initial Program Loading/Nucleus Initialization Program GY28-6661

This publication describes the internal logic of the initial program loader (IPL) program and the nucleus initialization program (NIP). The initial program loader prepares main storage to receive the nucleus and then loads the nucleus. The nucleus initialization program initializes the resident part of the control program and prepares main storage for control program operation.

Input/Output Supervisor GY28-6616

This publication describes the operation of the I/O supervisor within the IBM System/360 Operating System control program. The I/O supervisor's components, the EXCP supervisor and the I/O interruption supervisor, are discussed in detail to show the internal structure and logic involved in the control of I/O devices and channels.

Input/Output Support (OPEN/CLOSE/EOV) GY28-6609

This publication describes the internal logic of IBM System/360 Operating System input/output support. The discussion includes the relation of I/O support routines to other portions of the control program. Detailed descriptions of the open, close, and EOV routines provide the basis for the discussions of the other I/O support routines openJ, RDJFCB, Tclose, and FEOV.

Machine-Check Handler for IBM System/360 Model 65 GY27-7155

The machine-check handler is designed to reduce the number, and minimize the impact of unscheduled system interruptions resulting from machine-check interruptions in multiprogramming environments of the IBM System/360 Operating System. The program is designed for use with the IBM System/360 Model 65 only. This publication describes the program logic associated with the machine-check handler error recovery procedures.

Machine-Check Handler for IBM/System 360 Model 85 GY27-7184

This publication describes the internal logic of one of the recovery management programs for the Model 85, the machine-check handler (MCH/85). The MCH/85 program identifies and analyzes machine malfunction, attempts to repair damage or retries failing instructions, and, if successful, either terminates the affected task or puts the system into the disabled wait state. MCH/85 also records error statistics which assist the user in machine maintenance.

Machine-Check Handler for IBM System/370 Model 145 GY27-7237
Machine-Check Handler for IBM System/370 Models 155 and 165 GY27-7198

These publications describe the machine-check handler programs for the Models 145, 155, and 165, and what they do to prevent or minimize down time. The machine-check handler is an IBM System/360 Operating System recovery management facility which (1) reduces the number of times that system processing is halted because of machine-check interruptions, and (2) minimizes the impact of machine malfunctions and the resulting down time from machine-check interruptions when system processing cannot be continued. The program also records error statistics on the SYS1.LOGREC data set to assist the user in machine maintenance.

MVT Job Management GY28-6660

This publication describes the internal logic of the job management routines for the MVT control program of the IBM System/360 Operating System. Included are discussions of input stream processing, work queue management, job initiation and termination, I/O device allocation, system output processing, and the scheduling and execution of operator commands.

MVT Supervisor GY28-6659

This publication describes the internal logic of the MVT supervisor. The MVT supervisor is one part of the control program of the IBM System/360 Operating System. The supervisor controls the basic computing system and programming resources needed to perform several data processing tasks concurrently. Specifically, it was designed to:

- Handle interruptions
- Supervise tasks
- Control programs in main storage
- Control main storage itself
- Supervise the timer
- Supervise console communications and the system
- Supervisor exiting procedures
- Supervise termination procedures

Online Test Execute Program GY28-6651

The online test executive program (OLTEP) functions as an interface between the IBM System/360 Operating System and unit test programs performing online testing of input/output devices. This publication describes the internal logic of the program, including its theory and organization.

Queued Telecommunications Access Method GY30-2002

This publication describes the internal logic of the queued telecommunications access method (QTAM). It includes discussions on the physical and logical organization of QTAM and the function of BTAM within QTAM. It also contains a summary of the internal logic at the routine level and flowcharts of each routine.

Sequential Access Methods GY28-6604

This publication describes the internal logic of the routines of the queued sequential access method, the basic sequential access method, and the basic partitioned access method of IBM System/360 Operating System.

Time Sharing Option Command Processors

Volume 1: ACCOUNT, ACCOUNT ADD, ACCOUNT CHANGE, ACCOUNT DELETE, ACCOUNT LIST, ACCOUNT BROADCAST, ACCOUNT SUBROUTINES GY28-6771

Volume 2: ALLOCATE, CALL, CANCEL/STATUS, DELETE GY28-6772

Volume 3: EDIT GY28-6773

Volume 4: EXEC, FREE, HELP, LINK, LISTALC, LISTBC GY28-6774

Volume 5: LISTCAT, LISTDS, LOADGO, OPERATOR, OUTPUT GY28-6775

Volume 6: PROFILE, PROTECT, RENAME, RUN, SEND, SUBMIT, TERMINAL, TIME, WHEN/END GY28-6776

Volume 7: TEST GY28-6777

This series of publications describes the internal logic of TSO command processors. Command processors are programs invoked by the TSO Terminal Monitor Program in response to commands entered at the terminal.

Time Sharing Option (TSO) Control Program GY27-7199

This publication describes the internal logic of the System/360 Operating System Time Sharing Option (TSO). The main body of the manual describes the three main components of the control program -- the supervisory program, the Terminal Input/Output Coordinator (TIOC), and the Logon/Logoff Scheduler. Described in separate appendixes are the TSO Trace Writer and TSO Trace Data Set Processor, the TSO Background Reader, and the TSO/RMS Interface.

Utilities GY28-6614

This publication describes the internal logic of the utility programs provided for the IBM System/360 Operating System. Included are discussions of:

- System utilities, which are executed under the operating system to manipulate system data sets such as catalogs, and to dump, restore, and initialize direct access volumes.
- Data set utilities, which are executed under the operating system to work with data sets at the logical-record level.
- Independent utilities, which are executed outside of the operating system to dump, restore, and recover data, and to initilize and assign alternate tracks on direct access devices.

Appendix F: SVC Routines

This appendix contains a list of the names of SVC routines, arranged in order of SVC number. Included with each name are the hexadecimal form of the SVC number, the name of the macro instruction (or instructions) by which the routine is invoked, the type of the routine, and the name of the PLM containing additional information about the routine. The 'TYPE' field indicates which type of SVC routine the routine is:

- Type 1 SVC routines, which are part of the nucleus and are disabled for I/O and external interruptions. These routines do not issue SVC instructions because they cannot be restarted following interruption.

- Type 2 SVC routines, which are part of the nucleus but may be enabled for I/O and external interruptions during part of their operation. These routines may issue SVC instructions.

- Type 3 SVC routines, which are nonresident (reside in SYS1.SVCLIB), may be enabled for I/O and external interruptions, and are not larger than 1024 bytes. These routines may issue SVC instructions.

- Type 4 SVC routines, which are nonresident (reside in SYS1.SVCLIB), may be enabled for I/O and external interruptions, and are larger than 1024 bytes. They are brought into a transient area in storage, one load module at a time. Each load module is 1024 bytes or less. These routines may issue SVC instructions.

SVC Number	HEX SVC Number	Macro Instruction	Routine Name	Routine Type	PLM
00	00	EXCP/XDAP	EXCP Supervisor	1	Input/Output Supervisor
01	01	WAIT/WAITR/PRTOV	Wait	1	MVT Supervisor
02	02	POST	Post	1	MVT Supervisor
03	03	-none-	Exit	1	MVT Supervisor
04	04	GETMAIN	GETMAIN	1	MVT Supervisor
05	05	FREEMAIN	FREEMAIN	1	MVT Supervisor
06	06	LINK	Link	2	MVT Supervisor
07	07	XCTL	XCTL	2	MVT Supervisor
08	08	LOAD	Load	2	MVT Supervisor
09	09	DELETE	Delete	2	MVT Supervisor
10	0A	GETMAIN/ FREEMAIN/FREEPOOL	GETMAIN/FREEMAIN	1	MVT Supervisor
11	0B	TIME	Time	1	MVT Supervisor
12	0C	SYNCH	SYNCH	2	MVT Supervisor
13	0D	ABEND	ABEND	4	MVT Supervisor
14	0E	SPIE	SPIE	2	MVT Supervisor
15	0F	-none-	Error EXCP	1	Input/Output Supervisor
16	10	PURGE	Purge	3	Input/Output Supervisor
17	11	RESTORE	Restore	3	Input/Output Supervisor
18	12	BLDL/FIND	BLDL/Find	2	Sequential Access Methods
19	13	OPEN	Open	4	Input/Output Support (OPEN/CLOSE/EOV)
20	14	CLOSE	Close	4	Input/Output Support (OPEN/CLOSE/EOV)
21	15	STOW	Stow	3	Sequential Access Methods
22	16	OPEN	OPENJ	4	Input/Output Support (OPEN/CLOSE/EOV)
23	17	CLOSE	TCLOSE	4	Input/Output Support (OPEN/CLOSE/EOV)
24	18	DEVTYPE	DEVTYPE	3	Input/Output Supervisor
25	19	-none-	Track Balance	3	Sequential Access Methods
26	1A	CATALOG/ INDEX/LOCATE	Catalog	4	Catalog Management
27	1B	OBTAIN	Obtain	3	Direct Access Device Space Management
28	1C	-none-	CVOL	4	Catalog Management
29	1D	SCRATCH	Scratch	4	Direct Access Device Space Management
30	1E	RENAME	Rename	4	Direct Access Device Space Management
31	1F	FEOV	FEOV	4	Input/Output Support (OPEN/CLOSE/EOV)
32	20	-none-	Allocate	4	Direct Access Device Space Management
33	21	IOHALT	IOHALT	3	Input/Output Supervisor
34	22	MGCR	Command Scheduling	4	MVT Job Management
35	23	WTO/WTOR	Write to Operator	4	MVT Supervisor

SVC Number	HEX SVC Number	Macro Instruction	Routine Name	Routine Type	PLM
36	24	WTL	Write to Log	4	MVT Supervisor
37	25	SEGLD/SEGWT	Overlay Supervisor	2	MVT Supervisor
38	26	-none-	Resident SVC	2	TESTRAN
39	27	LABEL	Label	3	Utilities
40	28	EXTRACT	Extract	1	MVT Supervisor
41	29	IDENTIFY	Identify	2	MVT Supervisor
42	2A	ATTACH	Attach	2	MVT Supervisor
43	2B	CIRB	Stage 1 Exit Effector	2	MVT Supervisor
44	2C	CHAP	CHAP	1	MVT Supervisor
45	2D	-none-	Overlay Supervisor	2	MVT Supervisor
46	2E	TTIMER	TTIMER	1	MVT Supervisor
47	2F	STIMER	STIMER	2	MVT Supervisor
48	30	DEQ	Dequeue	2	MVT Supervisor
49	31	TEST	TTOPEN	3	TESTRAN
50	32	Unassigned			
51	33	SNAP	ABDUMP	4	MVT Supervisor
52	34	-none-	Restart	4	MVT Supervisor
53	35	RELEX	RELEX	3	Basic Direct Access Method
54	36	-none-	Disable	2	Indexed Sequential Access Methods
55	37	EOV	EOV	4	Input/Output Support (OPEN/CLOSE/EOV)
56	38	ENQ/RESERVE	Enqueue	2	MVT Supervisor
57	39	FREEDBUF	Dynamic Buffer	3	Basic Direct Access Method
58	3A	REQBUF/RELBUF	REQBUF	1	Basic Telecommunications Access Method
59	3B	-none-	OLTEP	3	Online Test Executive Program
60	3C	STAE	STAE	3	MVT Supervisor
61	3D	-none-	Save; SVC 61	3	TESTRAN; TSO Command Processor, Volume VII, TEST
62	3E	DETACH	Detach	2	MVT Supervisor
63	3F	CHKPT	Checkpoint	4	MVT Supervisor
64	40	RDJFCB	RDJFCB	3	Input/Output Support (OPEN/CLOSE/EOV)
65	41	-none-	QWAIT	2	Queued Telecommunications Access Method
66	42	-none-	BTAM Terminal Test	4	Basic Telecommunications Access Method
67	43	ENDREADY	QPOST	2	Queued Telecommunications Access Method
68	44	SYNADAF/SYNADRLS	SYSNADAF	4	
69	45	BSP	Backspace	3	Sequential Access Methods
70	46	GSERV	Graphic Attention Service	2	Graphics Access Method
71	47	ASGNBFR/RLSEBFR/ BUFINQ	Buffer Management	3	Graphics Access Method

SVC Number	HEX SVC Number	Macro Instruction	Routine Name	Routine Type	PLM
72	48	-none-	Communications Task Router	4	MVT Supervisor
73	49	SPAR	Specify Attention	3	Graphics Access Method
74	4A	DAR	Delete Attention	3	Graphics Access Method
75	4B	-none-	Dequeue	3	Graphics Access Method
76	4C	-none-	Statistics Update	4	Input/Output Supervisor
77	4D	-none-	QTAM Terminal Test	4	Queued Telecommunications Access Method
78	4E	-none-	L-Space	3	Direct Access Device Space Management
79	4F	STATUS	Set Status	1	MVT Supervisor
80	50	-none-	GJP SVC	3	Graphic Job Processor Support
81	51	SETPRT	SETPRT	4	Sequential Access Methods
82	52	-none-	Alternate Track Assignment	4	Utilities
83	53	SMFWTM	SMF Buffer	4	MVT Job Management
84	54	-none-	Restart Address	1	Graphics Access Method
85	55	-none-	Dynamic Device Reconfiguration	4	I/O Supervisor
86	56	ATLAS	IEHATLAS	3,4	Utilities
87	57	DOM	Delete Operator Message	3	MVT Supervisor
88	58	MOD88	MOD88	3	Emulator Program
89	59	EMSRV	EMSRV	3	Emulator Program
90	5A	XQMNGR	XQMNGR	4	Job Management
91	5B	VOLSTAT	VOLSTAT	3	I/O Supervisor
92	5C	-none-	TCB EXCP	1	Input/Output Supervisor
93	5D	TGET/TPUT	TGET/TPUT	4	TSO Control Program
94	5E	Terminal Status	Terminal Status	4	TSO Control Program
95	5F	TSEVENT	TSIP	1	TSO Control Program
96	60	STAX	STAX	3	TSO Control Program
97	61	-none-	TEST Command Processor	3	TSO Command Processor, Volume VII, TEST
98	62	PROTECT	Password Data Set Maintenance	4	Direct Access Device Space Management
99	63	-none-	Dynamic Allocation	4	MVT Job Management
100	64	FIB	Foreground Initiated Background	3	TSO Command Processor, Volume VI, SUBMIT
101	65	QTIP	QTIP Main Control	1	TSO Control Program
102	66	AQCTL	TCAM Interpartition Communications	1	Telecommunications Access Method
105	69	IMGLIB	Image Library DEB/DCB Controller	3	Sequential Access Methods
200-255	Available for assignment to user-written SVC routines. Until a number is assigned, its use in a processing program causes abnormal termination.				

Index

Indexes to systems reference library manuals are consolidated in the publication **IBM System/360 Operating System: Systems Reference Library Master Index**, GC28-6644. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

When more than one page reference is given, the major reference is first.

INDEX

INDEX

READER'S COMMENT FORM

IBM System/360 Operating System:
MVT Guide

Order No. GC28-6720-4

Please use this form to express your opinion of this publication. We are interested in your comments about its technical accuracy, organization, and completeness. All suggestions and comments become the property of IBM.

Please do not use this form to request technical information or additional copies of publications. All such requests should be directed to your IBM representative or to the IBM Branch Office serving your locality.

- Please indicate your occupation: _____
- How did you use this publication?
  - ☐ Frequently for reference in my work.
  - ☐ As an introduction to the subject.
  - ☐ As a textbook in a course.
  - ☐ For specific information on one or two subjects.
- Comments (Please include page numbers and give examples.):

- Thank you for your comments. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS, PLEASE . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY ...

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

Fold Fold

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Cut Along Line

System/360 OS MVT Guide (S360-36) Printed in U.S.A. GC28-6720-4

System/360 OS MVT Guide (S360-36) Printed in U.S.A. GC28-6720-4

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]