File No. S360-36
Order No. GC27-6939-10

OS

IBM Systems Reference Library

IBM System/360 Operating System: MFT Guide

OS Release 21.7

This publication provides information concerning Version II of Multiprogramming With a Fixed Number of Tasks (MFT) for installation personnel who are responsible for selection, evaluation, and implementation of System/360 Operating System configurations. The information is presented in five major sections:

- The MFT Control Program
- MFT Options
- Planning Considerations
- Modifying the System
- Logic Summary

The descriptive information is supplemented by examples and illustrations including a sample job scheduling sequence and sample partition configurations for systems with 128K, 256K, and 512K bytes of main storage.

Eleventh Edition (March, 1972)

This is a major revision of, and obsoletes, GC27-6939-9. Section IV of this publication contains a major portion 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 follows the contents.

This edition with Technical Newsletter GN28-2546 applies to Release 21.7 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, consult the latest **IBM System/360 and System/370 Bibliography**, GA22-6822, 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, Publications Development, Department D58, Building 706-2, PO Box 390, Poughkeepsie, N. Y. 12602. Comments become the property of IBM.

Preface

This guide describes how to use the MFT (multiprogramming with a fixed number of tasks) control program. The MFT 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 MFT control program.

It presents material in five sections and four appendices.

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

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

Section III explains device requirements, system generation macro instructions, and planning aids. 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 Reader/Interpreters and writers, and methods for modifying the control program. This section explains how to write cataloged procedures, includes examples of modified cataloged procedures, and describes methods of modifying the control program. Planning personnel need this information to tailor the control program to their installation's requirements.

Section V explains in detail the MFT 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 describes the recovery management routines available for MFT.

Appendix B contains the formats for system macro instructions that either modify system control blocks or obtain information from system control blocks.

Appendix C explains control character transformations used with user-written writer routines.

Appendix D describes the use of the RESERVE macro instruction with the Shared DASD (Direct Access Storage Device) option.

As a prerequisite, users must have read:

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

Other publications that the reader will find helpful, and that are referred to within this publication are:

IBM System/360 Operating System:

Advanced Checkpoint/Restart Planning Guide, GC28-6708

Data Management for System Programmers, GC28-6550

Data Management Macro Instructions, GC26-3794

Data Management Services, GC26-3746

Job Control Language Reference, GC28-6704

Messages and Codes, GC28-6631

Operator's Reference, GC28-6691

Storage Estimates, GC28-6551

Supervisor Services and Macro Instructions, GC28-6647

System Control Blocks, GC28-6628

System Generation, GC28-6554

Utilities, GC28-6586

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

IBM System/360 Operating System:

Input/Output Supervisor Program Logic Manual, GY28-6616

Machine-Check Handler for:

System/360 Model 65 Program Logic Manual, GY27-7155

System/360 Model 85 Program Logic Manual, GY27-7184

System/370 Models 135 and 145 Program Logic Manual, GY27-7237

System/370 Models 155 and 165 Program Logic Manual, GY27-7198

MFT Supervisor Program Logic Manual, GY27-7236

Contents

Figures

Summary of Amendments for GC27-6939-10 as Updated by GN28-2546 OS Release 21.7

Main Storage Organization Change

Describes how a fragment of the system queue area rounds the fixed area up to a 2K boundary.

Job Step Timing

Describes the correct method for determining CPU time.

EXEC Statement

Describes the default parameter from the initiator to the reader for either system tasks or problem programs started from the operator's console.

SYSIN and SYSOUT Data Blocking

Lists a new blocking factor for PL/IF on SYSPRINT.

The Resident Access Method Modules OPTION

Describes additional requirements for the use of the system log data sets on devices with rotational position sensing.

STAE - Specify Task Asynchronous Exit

Describes additional return codes encountered when a program in an active STAE environment abnormally terminates.

Inter-Partition POST - Post a Nonresident Routine

Introduces this TSO macro and tells how to use the list and execute forms of it in MFT to pass a parameter list to a nonresident control program routine.

Summary of Amendments for GC27-6939-10 OS Release 21

Generalized Trace Facility (GTF)

Describes the tracing and editing abilities of GTF.

BSAM modules for ABDUMP routine

Three BSAM modules should be resident for ABDUMP

Changed list IEARSV00 to support Open/Close/EOV

Modules in the list IEARSV00 changed.

System/370 Model 135 Support

MFT supports System/370 Model 135.
Channel Check Handler supports System/370 integrated channels.
Machine Check Handler supports System/370.

Problem Determination

Adds the new command DUMP to the list of operator commands.

Automatic Start Command

If a dedicated reader partition exists, the automatic reader starts in that partition.

Alternate Path Retry (APR)

The VARY PATH function of APR is now standard for MFT.

SVAREA=NO parameter of ATTACH Macro

The parameter SVAREA=NO in the ATTACH macro is not valid with MFT.

Disposition of Temporary Dedicated Data Sets

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

MFT Options List

Lists the options available for the MFT 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:
Resident Routines Option
Job Queue Format
Output Separation
Writing System Output Writer Routines
Adding SVC Routines to the Control Program
Message Routing Exit Routines
Handling Accounting Routines
The Must Complete Function
The PRESRES Volume Characteristic List

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

System Macro Instructions

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

Control Character Transformations (from Writing System Output Writer Routines)

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

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

Summary of Amendments for GC27-6939-9 OS Release 20.1

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, and IBM 3330 Disk Storage Drive.

Resident BLDL Table

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

Recovery Management

Describes recovery management for MFT. The discussion is parallel to that found in **IBM System/360 Operating System: MVT Guide**, GC28-6720, and includes information formerly contained in **IBM System/360 Operating System: Concepts and Facilities**, GC28-6535.

Summary of Amendments for GC27-6939-8 OS Release 20

MFT Redocumentation

Adds information formerly contained in **IBM System/360 Operating System: Control Program With MFT, Program Logic Manual**, GY27-7128.

Section I: The MFT Control Program

The basic concepts of the MFT control program are described in this section for data processing executives, planners, and system analysts. The organization of this section describes concurrent porcessing of up to 15 jobs, the functions of the MFT control progrram, and examples of how MFT processes jobs. Significant and unique features of the MFT control program are described.

The topics this section describes are:

- Concepts of MFT
- MFT Terminology
- Functions of the Control Program
- Control Program Organization
- Main Storage Organization
- Sequence of Operation
- Job Processing Under MFT
- MFT Features

Concepts of MFT

MFT is a System/360 Operating System option that provides extended multiprogramming capabilities and increased flexibility to the Operating System user whose system has 128K bytes or more of main storage. The system may reside on any of the following devices:

- IBM 2301 or 2303 Drum Storage
- IBM 2305-1 or 2305-2 Fixed Head Storage Facility
- IBM 2311 or 3330 Disk Storage Drive
- IBM 2314 or 2319 Direct Access Storage Facility

The system may not reside on an IBM 2302 Disk Storage unit or an IBM 2321 Data Cell Drive.

Systems with MFT can use the Shared Direct Access Storage Device (Shared DASD) feature. This feature allows two or more independently-operating computing systems to use common direct access storage devices. MFT can share devices with MVT (multiprogramming with a variable number of tasks), and other MFT configurations of the IBM System/360 Operating System.

As many as 15 multi-step jobs can proceed concurrently with the operation of: up to three system input readers, 36 system output writers, and as many direct system output writers as there are available devices. Each job is processed in a discrete area of main storage known as a partition. A maximum of 52 partitions is allowed in MFT. When a job must wait for completion of an event such as an input/output operation, another job of lower priority is allowed to proceed. When the higher-priority job is ready to resume, the lower-priority job's processing is suspended and control of the CPU is returned to the higher-priority job. The priority of each job is determined by the partition in which it resides. Jobs are directed to a given partition or group of partitions through the CLASS parameter of the JOB card.

By using the CLASS parameter to denote different types of jobs, the user can direct jobs to partitions consistent with the jobs' characteristics. Process-limited jobs, for instance, can be directed to low-priority partitions so that they do not interfere with efficient processing of jobs that do not require the CPU as often. Telecommunications jobs can be directed to higher-priority partitions so that system response time to the terminal user is minimal. If equal intervals of CPU time are to be allotted to certain graphics or other jobs, these jobs can be directed to time-sliced partitions (if the time-slicing feature is included in the system). If direct system output writers are used, the job class of a job must be a job class that the writer can process. Direct system output writers can handle up to eight different job classes. The CLASS parameter can be used to establish which jobs are going to be handled by direct system output writers. Additional applications of the CLASS parameter can be established based on any job characteristics meaningful to the installation.

To use MFT efficiently, both system and application programmers must understand how it operates. Because other user personnel may be interested in a summary of MFT operation. This section summarizes MFT operation in seven major topics:

- "MFT Terminology" defines basic terms used in this publication.
- "Functions of the Control Program" introduces the basic techniques of the MFT control program. This includes a discussion of the job management, task management, and data management functions of the control program.
- "Control Program Organization" describes the resident and non-resident portions of the control program. Three partitioned date sets -- SYS1.NUCLEUS, SYS1.SVCLIB, and SYS1.LINKLIB -- contain the control program.
- "Main Storage Organization" shows the difference between fixed and dynamic areas, and describes partitions.
- "Sequence of Operation" describes, on a conceptual level, the scheduling, initaiation, and termination of jobs.
- "Job Processing Under MFT" describes the principles of job processing used by the MFT control program.
- "MFT Features" introduces the reader to what the MFT control program can do.

MFT Terminology

Unique MFT terminology is explained as it is presented. Certain basic definitions, however, are essential to understanding the terms as they are introduced. These basic definitions are given in the following paragraphs.

Multiprogramming With a Fixed Number of Tasks

"Multiprogramming" refers to the concurrent execution of several units of work (tasks), any one of which would, in a single-program environment, occupy the computing system until the task is finished.

Note: Throughout this publication, "job" refers to an externally specified unit of work (a problem program specified by a JOB card), and "task" refers to any unit of work that must be performed by the Central Processing Unit (CPU), under control of a task control block (TCB).

The significance of multiprogramming is that it provides increased throughput and better utilization of resources. A typical task makes use of only a small part of the resources available in the system. In a single-program environment, this means that overall application of resources is low. In a multiprogramming environment, however, resource application is markedly improved, because the relatively limited demands of each of several tasks combine to produce a net demand that is more efficient in terms of the system's capabilities.

The phrase "a fixed number of tasks" indicates that the maximum number of tasks the system is capable of performing at one time is determined at system generation. The number of tasks that can be performed at one time can be varied during and after system initialization.

System Initialization

System initialization is the preparation for execution of those elements of the IBM System/360 Operating System that reside in the fixed area of main storage. This preparation is performed by the Nucleus Initialization Program (NIP) when the system is brought into main storage through the initial program loading (IPL) procedure, and is supplemented by operator action.

Partitions

Main storage is divided into a system area and a dynamic area. The dynamic area is further divided, by the user, into a number of discrete areas called partitions. The number of tasks that can execute in each partition depends on whether the user has selected MFT without subtasking or MFT with subtasking. The type of ATTACH facility, selected during system generation, will determine the number of tasks that can execute in each partition.

When MFT without subtasking is selected, only one task will be executed in each partition. With this version of MFT, the number of partitions determines the number of tasks that can execute concurrently.

When MFT with subtasking is selected, each task can attach any number of subtasks (up to a maximum of between 196 and 249); each subtask will then execute in the same partition.

Partitions can be defined as reader, writer, or problem program partitions. Each partition has a fixed priority within the system (partition 0 has the highest priority; partition 51, the lowest). The priority determines which partition will gain control of the CPU first when a wait condition occurs. Small partitions are problem program partitions that are too small to contain the scheduler.

Concurrent Operation

In a multiprogramming system, tasks are performed **concurrently**. This is a significant programming concept. Execution is not simultaneous, or overlapped, or alternating in a fixed pattern. Each task is contained within a partition. The determination of which task gains control is based on "waits" and "posts." Waiting for an event, such as the completion of an input/output operation, removes the task from contention for control. Posting of the task, which signals that the awaited event has been completed, causes the task to be placed in a "ready" status. The task that becomes active is the highest priority task that is ready. This high-priority task proceeds until another event causes the task to relinquish control. The relinquishing of control by one task, and the gaining of control by another task, is called a **task switch**.

Task Switching

There are two ways in which a task switch can occur:

1. The active task relinquishes control because it must wait for the completion of an event, such as an input/output operation.

2. Control is seized by a higher-priority ready task as a result of an interruption signaling an event for which it has been waiting.

The first case illustrates how multiprogramming ensures optimum utilization of resources. Whenever one unit of work cannot proceed, another (highest-priority) is advanced. In a single-program environment, no work can proceed while the single task waits for an event. The second case illustrates how an internal balance between the tasks is achieved. When a task has control, it retains control only until a task of higher priority becomes ready to proceed.

Note: Throughout this manual, the suffix "K" denotes the value 400 (hexadecimal), or 1024 (decimal).

Functions of the Control Program With MFT

The control program routines of MFT have four major functions: job management, task management, data management, and recovery management.

Job Management

Job management is the processing of communications from the programmer and operator to the control program. There are two types of communications: operator commands, which start, stop, and modify the processing of jobs in the system, and job control statements, which define work being entered into the system. Processing of these commands and statements is referred to as command processing and job processing, respectively.

Task Management

Task management routines monitor and control the entire operating system, and are used throughout the operation of both the control and processing programs. Task management routines have six major functions:

- Interruption supervision.
- Task supervision.
- Main Storage supervision.
- Contents supervision.
- Overlay supervision.
- Timer supervision.

The task management routines are collectively referred to as the "supervisor."

Data Management

Data management routines control all operations associated with input/output devices: allocating space on volumes, channel scheduling, storing, naming, and cataloging data sets, moving data between main and auxiliary storage, and handling errors that occur during input/output operations. Data management routines are used by processing programs and control program routines that require data movement. Processing programs use data management routines primarily to read and write required data, and also to locate input data sets and to reserve auxiliary storage space for output data sets of the processing program.

There are five categories of data management routines:

- Input/Output (I/O) supervisor, which supervises input/output requests and interruptions.
- Access methods, which are used to communicate with the I/O supervisor.
- Catalog management, which maintains the catalog and locates data sets on auxiliary storage.
- Direct access device space management (DADSM), which allocates auxiliary storage space.
- Open/Close/End-of-Volume, which performs required initialization for I/O operations and handles end-of-volume conditions.

Recovery Management

Recovery management routines: (1) collect and record machine and program data when a machine check or a channel check occurs; (2) ensure use of all operational online channels; and (3) attempt to recover from channel and device errors by allowing the operator to exchange volumes between two devices.

Recovery management routines are described briefly in Appendix C.

Control Program Organization

The control program is on auxiliary storage in three partitioned data sets created when the system is generated. These data sets are:

- The NUCLEUS partitioned data set (SYS1.NUCLEUS), which contains the Nucleus Initialization Program (NIP) and the resident portion of the control program.

- The SVCLIB partitioned data set (SYS1.SVCLIB), which contains nonresident SVC routines, nonresident error-handling routines, and the access methods routines.

- The LINKLIB partitioned data set (SYS1.LINKLIB), which contains other nonresident control program routines and IBM-supplied processing programs.

Resident Portion of the Control Program

The resident portion (nucleus) of the control program is in SYS1.NUCLEUS. It is made up of those routines, control blocks, and tables that are brought into main storage at initial program loading (IPL) and are never overlaid by another part of the operating system. The nucleus is loaded into the fixed area of main storage.

The resident task management routines include all of the routines that perform:

- Interrupt supervision.
- Main storage supervision.
- Timer supervision.

They also include portions of the routines that perform:

- Task supervision.
- Contents supervision.
- Overlay supervision.

The resident job management routines are those routines of the communications task that receive commands from the operator, and the master scheduler task.

Communications Task: The communications task routines process the following types of communication between the operator and the system:

- Operator commands, issued through a console.
- Write-to-operator (WTO) and write-to-operator with reply (WTOR) macro instructions.
- Interruptions caused when the INTERRUPT Key is pressed, the signal to switch from the primary console to an alternate console.

Master Scheduler Task: The master scheduler task processes job queue manipulation commands and partition definition. For example, a HOLD or DEFINE command is processed by the master scheduler task.

The resident data management routines are the input/output supervisor and, optionally, the BLDL routines of the partitioned access method, and QSAM for readers and writers.

Note: The resident QSAM routines must be generated with the MFT system, and are also available to problem programs. They may either be included in the 34K nucleus, or, if a smaller nucleus is to be generated, the routines may be used in the reader and writer partitions. However, the size of the reader and writer partitions may increase beyond the 30K/44K and 10K sizes, respectively.

Optionally, other access method routines may be made resident. The 34K MFT nucleus also includes space for the optional storage protection routines. Although the storage protection feature is not required for MFT, it should be included. Storage protection prevents the contents of each partition from being destroyed or changed by another task.

The user may also select resident reenterable routines, which are access method routines from SYS1.SVCLIB, and other reenterable routines from SYS1.LINKLIB. At system generation, the user specifies that he wants such routines resident in main storage. At IPL, he identifies the specific routines desired in the RAM=entry. The selected routines are loaded during system initialization and reside adjacent to the higher end of the system queue area unless the BLDL table is also resident (see Figure 1). (The resident BLDL table is optional; if present, it contains directory entries for selected modules in SYS1.LINKLIB and SYS1.SVCLIB.)

Normally-transient SVC routines (i.e., types 3 and 4 SVC routines) can be made resident through the RSVC option, specified by the user. NIP loads these routines adjacent to the higher end of the resident reenterable routines. If there is no resident BLDL table or resident reenterable routines, the routines are loaded adjacent to the higher end of the system queue area. (See Figure 1.)

Nonresident Portion of the Control Program

The nonresident portion of the control program comprises routines that are loaded into main storage as they are needed, and which can be overlaid after their completion. The nonresident routines operate from the partitions and from two sections of the nucleus called transient areas.

The non-resident routines which perform job management are collectively referred to as the scheduler. The scheduler for MFT is packaged in two basic configurations, 30K and 44K. (Detailed considerations of actual scheduler size are given in the **Storage Estimates** publication.) At least one partition in the system must be large enough to contain the scheduler. The choice of scheduler is determined by the amount of main storage available, the number of partitions needed, and the required sizes of those partitions. (See "Choosing the Size of the Scheduler" in the **Considerations** section.)

Main Storage Organization

In a single task environment, main storage is divided into two areas: the fixed area, and the dynamic area. In multiprogramming with a fixed number of tasks (MFT), the dynamic area is divided further into as many as fifty-two discrete areas called partitions. Figure 1 shows the division of main storage.

The fixed area, located in the lower portion of main storage, contains the resident portion of the control program, and control blocks and tables used by the system. The size of the fixed area depends on the number of partitions established by the user, and the control program options selected at system generation.

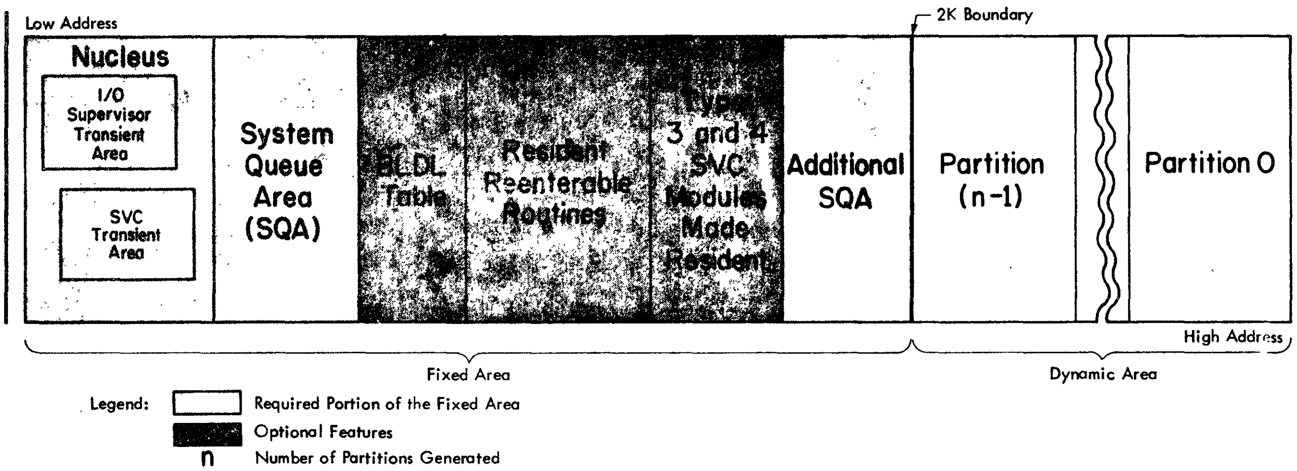


Figure 1. Main Storage Organization

The MFT system nucleus occupies a fixed area in main storage containing at least 34K bytes (systems with MCH for 135 require an additional 4K to 8K bytes).

Partitions are defined within the dynamic area, located in the upper portion of main storage, at system generation. The number of partitions may then be varied within the number specified at system generation, and the sizes and job classes of partitions may be redefined at system initialization or during operation. Each partition may be occupied by a processing program, or by control program routines that prepare job steps for execution (job management routines), or handle data for a processing program (access method routines).

Provided the total number of partitions does not exceed 52 and enough computing system resources are available, MFT provides for the concurrent execution of as many as 15 problem programs, 3 input readers, and 36 output writers, each in its own fixed partition of main storage. The MFT system provides for task switching among the tasks operating in the partitions, and between those tasks and the communications task and master scheduler task in the system area.

Fixed Area

The fixed area is that part of main storage into which the nucleus is loaded at IPL. The storage protection key of the fixed area is zero so that its contents can only be modified by the control program. The fixed area also contains two transient areas into which certain nonresident routines are loaded when needed: the SVC transient area (1024 bytes) and the I/O supervisor transient area (1024 bytes). These areas are used by nonresident SVC routines and nonresident I/O error-handling routines, respectively, which are read from SYS1.SVCLIB.

Each transient area may contain only one routine at a time. When a nonresident SVC or error-handling routine is required, it is read into the appropriate transient area. The transient area routines operate with a protection key of zero, as do other routines in the fixed area.

System Queue Area: The system queue area (SQA) is a protected area in the nucleus. It contains ENQ/DEQ control block and command scheduling control block (CSCBs). In addition, if the communications task cannot obtain WTO buffer space, SQA is used.

The size of the system queue area is initially established at system generation (via the SYSQUE parameter of the CTRLPROG macro instruction).

In a system that has MFT with subtasking, the system queue area also contains task related control blocks for each active subtask. In this case, the size of the system queue area is determined by the number of partitions and the number of subtasks that can be concurrently active. The size of the system queue area, established during system generation, should be retained.

The system queue area (SQA) is established by NIP adjacent to the fixed area and provides the main storage space required for tables and queues built by the control program. The SQA must be at least 1600 bytes for a minimum two-partition system. Its storage protection key is zero so that it can be modified by control program routines only. Data in the system queue area indicates the status of all tasks.

In a system with the storage protection feature, there is, as Figure 1 shows, additional SQA space between the resident SVC area and the next higher 2K boundary.

Dynamic Area

Figure 2 shows how the contents of each partition in the dynamic area are organized and how they are related to the rest of main storage.

Job management routines, processing programs, and routines brought into storage via a LINK, ATTACH, or XCTL macro instruction, are loaded at the lowest available address. The highest portion of the partition is occupied by the user parameter area and user save area. The next portion of the partition is occupied by the task input/output table (TIOT) which is built by a job management routine (I/O Device Allocation routine). This table is used by data management routines and contains information about DD statements.

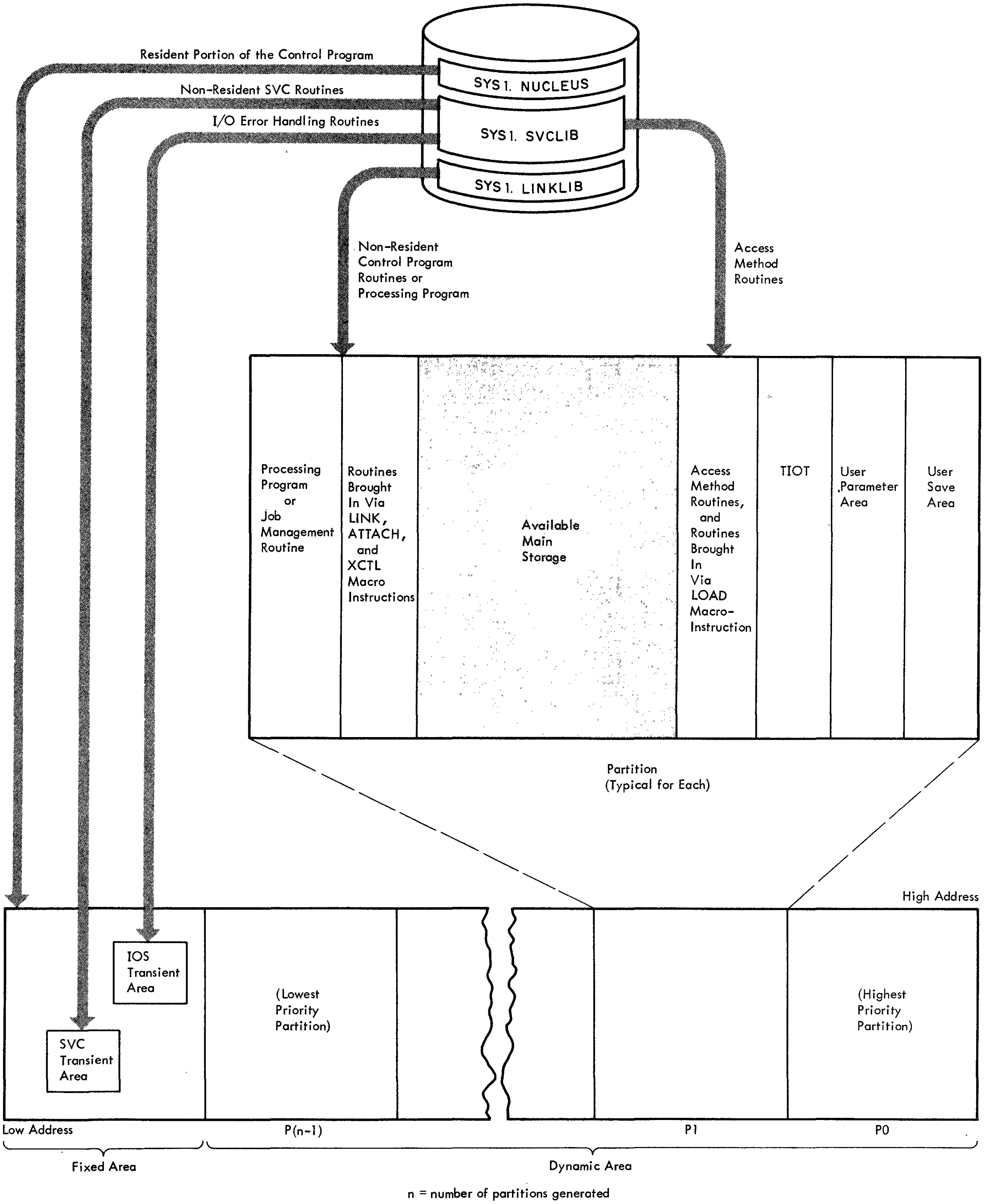

Resident Portion of the Control Program
Non-Resident SVC Routines
I/O Error Handling Routines
SYS1. NUCLEUS
SYS1. SVCLIB
SYS1. LINKLIB
Non-Resident Control Program Routines or Processing Program
Access Method Routines
Processing Program or Job Management Routine
Routines Brought In Via LINK, ATTACH, and XCTL Macro Instructions
Available Main Storage
Access Method Routines, and Routines Brought In Via LOAD Macro-Instruction
TIOT
User Parameter Area
User Save Area
Partition (Typical for Each)
High Address
IOS Transient Area
SVC Transient Area
(Lowest Priority Partition)
(Highest Priority Partition)
Low Address
P(n-1)
P1
P0
Fixed Area
Dynamic Area
n = number of partitions generated

Figure 2. Division of Main Storage

Each partition may be used for a problem program as well as for system tasks (readers, initiators, and writers). When the control program requires main storage to build control blocks or work areas, it obtains this space from the partition of the processing program that requested the space. Access method routines and routines brought into storage via a LOAD macro instruction are placed in the highest available locations below the task input/output table.

Working storage and data areas are assigned from the highest available storage in a partition.

System Input Reader Partitions

MFT allows the user to specify as many as three resident reader partitions. The size of the reader used in the system is the same size as the job scheduler specified at system generation.

A partition is designated at system generation to contain a resident reader by assigning "R" in place of the problem program job class identifiers. For example, to establish 0 a resident reader in partition the PARTITNS macro instruction would be coded as: P0(C-R,S-30K). (For a description of this macro see "PARTITNS Macro Instruction" in Section III.) To establish a resident reader after system initialization, the partition's job class identifiers are replaced by "RDR" in reply to the message IEE802A "ENTER DEFINITION". (In a listing of partitions, "RDR" will appear in place of the job class identifier(s) for a resident reader partition.) It is unnecessary to establish a resident reader partition if transient readers will be used. (See "Input Readers" in this section.)

Problem Program Partitions

MFT permits up to 15 partitions to be specified for problem programs. Each partition may have up to three job class identifiers; more than one partition may be assigned to service the same job class(es). Problem program partitions must be at least 8K in size, and at least one of the partitions should be the size of the scheduler. (If a resident reader partition is not established, at least one of the problem program partitions **must** be the size of the scheduler.) This partition should not be used for processing a long-duration or unending job, such as telecommunications or graphics, if it is the only large partition in the system.

Large problem program partitions may also be used by the scheduler (if it has work to do), by a transient input reader, or by a non-resident output writer.

Problem programs run concurrently with system readers and writers. When a problem program in a large partition is terminated, a scheduler is brought into that partition to initiate or terminate a small partition, to start a reader or writer in the partition, or to retrieve another job from the input work queue for the appropriate job class(es). Control is then given to the appropriate task; e.g., if a problem program is retrieved from an input queue, control is given to the program for execution.

For example, in Figure 3, P4 is assigned job classes E, D, and A. If a job of class E has just been terminated, and no small partition has requested scheduling, the initiator first searches the job class E input work queue. If no class E jobs exist, the initiator searches for job class D jobs; if no class D jobs exist, the job class A input queue is searched. If all three queues are empty, the partition remains dormant until it is used by the initiator to schedule small partitions, P1, P2, or P3, or until another job with class E, D, or A is read into the system and scheduled.

When a job in a small partition has finished executing, the small partition remains dormant until a large partition is free to terminate it, and to initiate a new job into it.

As many as l4 small problem program partitions may be defined. Efficient use of small partitions is achieved through associating one or more job classes with each. Thus, jobs whose main storage requirements are small can be directed to small partitions for execution. Small high-priority jobs can be accommodated by defining a small partition in the upper portion of main storage. This partition might then be assigned a primary job class reserved by the installation for critical jobs. By assigning secondary and tertiary job classes, the partition need not remain idle when there is no critical work. By defining a small partition, only the necessary amount of main storage is set aside for such work.

System Output Writer Partitions

A resident writer partition may be as small as 10K plus the size of the input and output buffers. (See "Output Writer Partition Size Requirement" in the **Considerations** section.) The size of the writer depends upon the output buffer space necessary; i.e., if blocked output is used, the size of the partition must be increased by the logical record length times the blocking factor. MFT provides the capability of running as many as 36 system output writers concurrently with problem programs and system input readers.

At system generation, a resident writer partition is established by writing "W" in place of the problem program job class identifier(s). After system initialization, it is established during partition definition by replacing the job class identifier(s) with "WTR". No other task can operate in this partition unless the partition is assigned a problem program job class through partition redefinition. Each writer can accommodate as many as eight output (SYSOUT) classes and can share output classes with other writers. (See "Output Writers" in this section.)

Direct System Output Writer Partitions

Direct system output writers can operate in small or large partitions. However, to control the writing of a job's output, the direct system output writer must be operating in the same partition as the job; also, the job's class must be an eligible job class. An eligible job class is one that has been assigned to the direct system output writer when the writer was started. Direct system output writers can handle up to eight different job classes.

Extending Main Storage

Main storage can be extended by including IBM 2361 Core Storage Units in the system. Figure 3 illustrates the IBM 2361 Core Storage Unit. There are no unique instructions used in programming IBM 2361 core storage. It is simply an extension of IBM 2050, 2065 or 2075 Processor Storage (processor storage) units; that is, the address of the first byte of core storage is one address higher than that of the last byte of processor storage.

With Main Storage Hierarchy Support for IBM 2361 Models 1 and 2, main storage is divided into two blocks called hierarchies. Main storage comprises (1) processor storage, which is referred to as hierarchy 0, and (2) IBM 2361 Core Storage, which is referred to as hierarchy 1. At system generation, a partition can be generated entirely within either hierarchy 0 or 1, or it can span hierarchies 0 and 1. The exception to this is that system tasks (readers and writers) can only be generated in **either** hierarchy 0 or 1. (In using storage hierarchies on a Model 50, however, if a reader or writer is placed in hierarchy 1 or if a program containing CCWs for direct access devices is loaded into hierarchy 1, overrun will occur, thereby degrading the performance or resulting in an uncorrectable input/output error.)

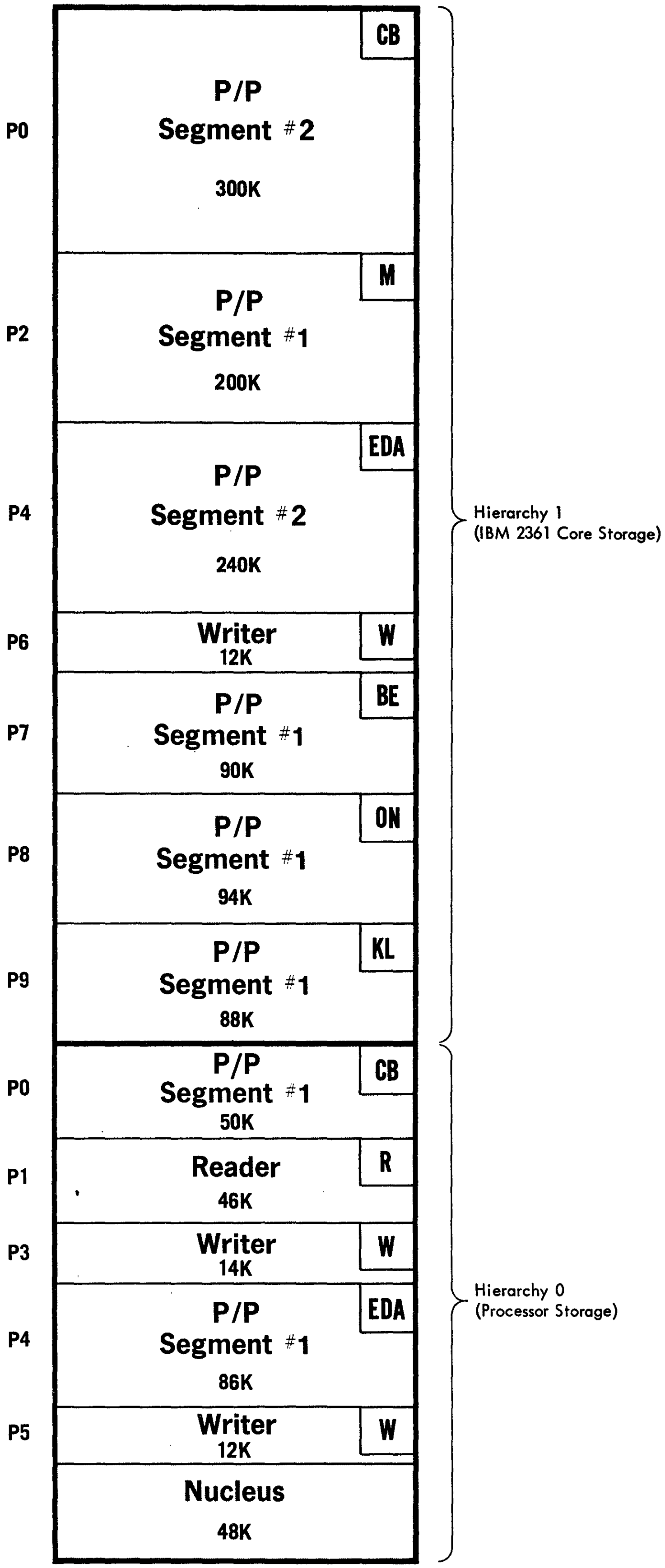


Figure 3. Sample 256K Batch System Configuration With 1024K IBM 2361 Core Storage Unit

For example, in Figure 3, partitions P1, P3, and P5 contain only one segment and are generated in processor storage. P2, P6, P7, P8, and P9 contain only one segment and are generated in IBM 2361 Core Storage. P0 and P4 are two-segment partitions with a partition segment in each hierarchy. For a complete description of partition establishment in IBM 2361 Core Storage, see the **System Generation** publication.

Each partition established during system generation is described by a boundary box. The first half of the boundary box describes the processor storage partition segment and the second half describes the core storage partition segment. Any partition segment not assigned main storage in the system has the applicable boundary box pointers set to zero; if a partition is established entirely within hierarchy 1, the processor storage pointers in the first half of the partition's boundary box are set to zero. If a partition segment is not generated in core storage, the core storage pointers in the second half of the partition's boundary box are set to zero. If core storage has been included in the system, but is offline, the second half of the boundary box will contain zeros.

Sequence of Operation

To illustrate how MFT works, a sample sequence of operation is described below. (For an overview of the processing performed by various components of the control program, refer to the first part of Section V: Logic Summary. The second part describes how the dynamic area of main storage is prepared by the master scheduler task after completion of the nucleus initialization program.) In the job stream shown in Figure 17, the following CLASS parameters appear on the JOB cards:

1. CLASS=N
2. CLASS=D
3. CLASS=L
4. CLASS=J
5. CLASS=M
6. CLASS=C
7. None
8. CLASS=J,PRTY=12
9. CLASS=C

The system is loaded by use of the normal IPL procedure and initializes itself by use of the nucleus initialization program (Figure 4). After system initialization, the contents of main storage are as shown in Figure 5.

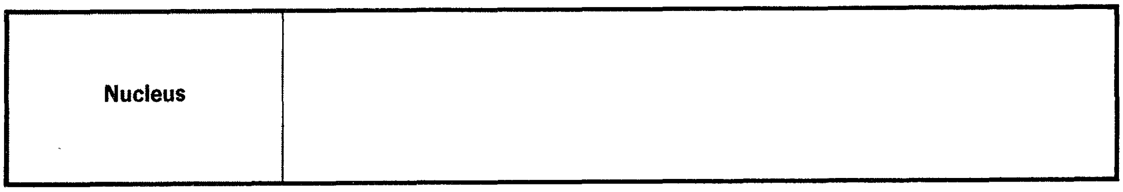


Figure 4. Contents of Main Storage After Nucleus Initialization

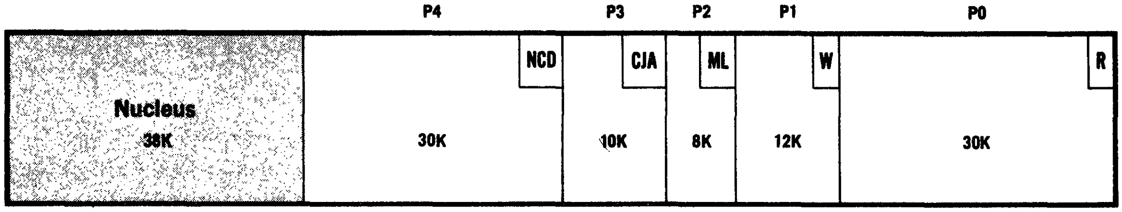


Figure 5. Contents of Main Storage After System Initialization

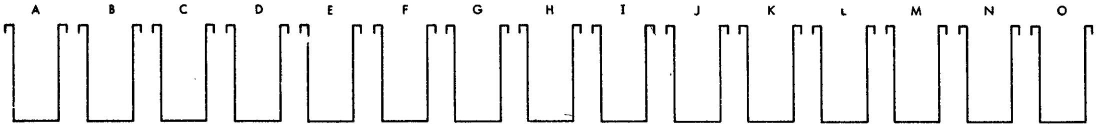


Figure 6. Input Work Queue After System Initialization

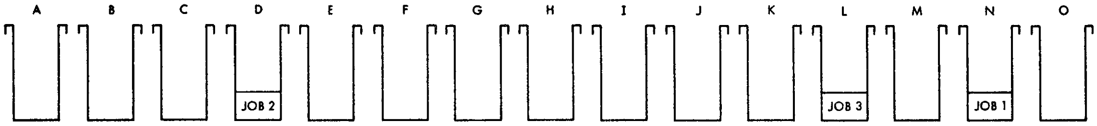


Figure 7. Input Work Queues After First Three Jobs Have Been Entered

The job class identifiers assigned to each partition at system generation are the alphabetic characters shown in the upper right corner of the partitions. Figure 4 illustrates the input work queues established at system initialization. When a START command is entered for the reader in partition 0 (PO), the reader begins reading the job stream and entering jobs into the input work queues for each CLASS (Figure 7). The START commands for the initiator and writer are also entered.

The initiator in P4 now schedules the first job (Figure 8). Because N is the primary job class assigned to P4, the initiator searches the CLASS=N queue first (job 1 has been placed on the queue by this time), and job 1 is initiated and given control (Figure 9). The reader continues reading the input stream, placing jobs in their appropriate queues. Because job 7 has no CLASS parameter, it is placed on input work queue A (the default job class). Job 8, with a PRTY parameter specifying priority 12, is placed on input queue J ahead of job 4, unless job 4 has already been scheduled. At this point jobs 1 through 9 have been read and placed on their appropriate queues (Figure 10).

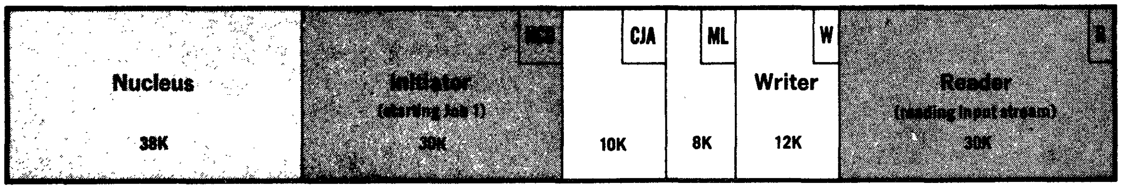


Figure 8. Contents of Main Storage After START Commands

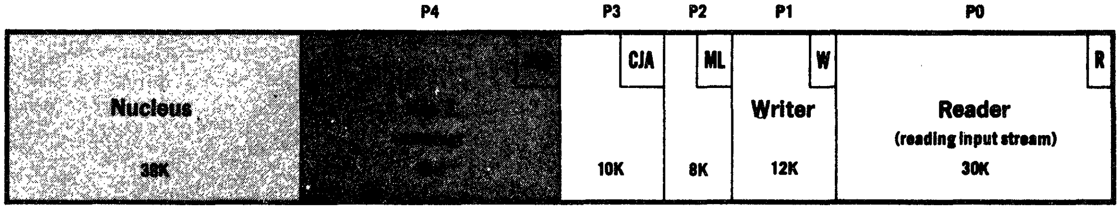


Figure 9. Contents of Main Storage After First Job Has Been Scheduled

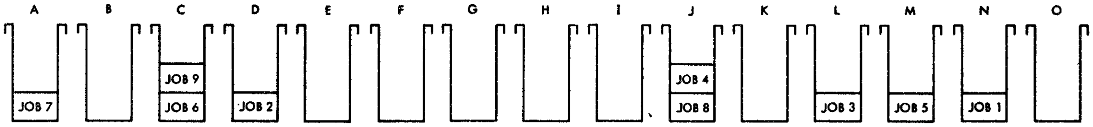


Figure 10. Input Work Queues After All Nine Jobs Have been Entered

When job 1 has finished processing, a scheduler is brought into P4 to terminate the job, and to initiate jobs in small partitions P2 and P3. Job 5 is scheduled into P2 (because M is the primary class) and job 6 into P3. The scheduler now searches input work queue N for a job for P4. Because the CLASS=N queue is empty, the CLASS=C queue is searched. Job 9 is waiting on the queue; therefore, it is scheduled into P4. At this point, the contents of main storage are as shown in Figure 11.

A scheduler enters P4 each time a job terminates in that partition. The scheduler first checks whether jobs in P2 or P3 need to be terminated. If so, they are terminated by the scheduler in P4, and new jobs are then scheduled into P2 and P3. This sequence continues until all jobs have completed processing, or until the system is shut down.

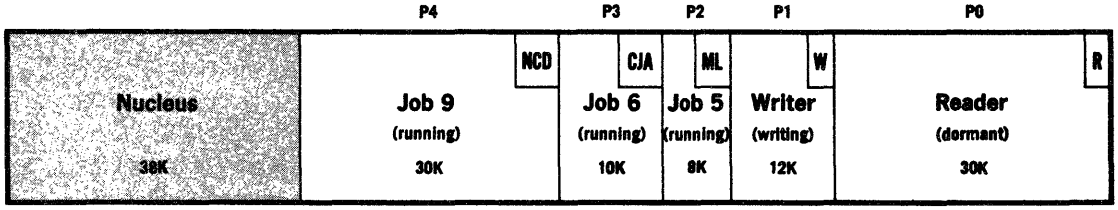


Figure 11. Contents of Main Storage With All Partitions Active

Job Processing Under MFT

This topic briefly describes the operation of the MFT control program during job processing. The description consists of:

- Partition Job Class Facility
- Partition Redefinition
- Input Readers
- Job Initiation and Termination
- Direct System Output Writers
- System Output Writers
- System Restart

The description goes from job intitiation to job completion. To better understand the description, frequent referrals should be made to Figure 12: Sample Five-Partition Configuration.

Partition Job Class Facility

The partition job class facility allows one or more partitions to be assigned to selected jobs. During system generation, a partition must be assigned to service each job class that will be used. These assignments may be modified later. (See "Partition Redefinition" in this section.) Each problem program partition may be assigned as many as three job classes designated by the alphabetic characters A through O. These job class designations have no inherent meaning; they can be used to denote any job characteristic, which would influence the choice of partitions for the job, meaningful to the installation. More than one partition may be assigned to service the same job class(es). In Figure 12, P3 is assigned to job classes C, J, and A; P4 is assigned to N, C, and D. These partition job class identifiers are used by the system to determine which input queue is searched first by an initiator servicing a specific partition. (See "Problem Program Partitions" in this section.) The sequence in which jobs are selected from each input work queue is determined by the PRTY parameter. (See "Enqueuing Jobs by CLASS and PRTY" in this section.)

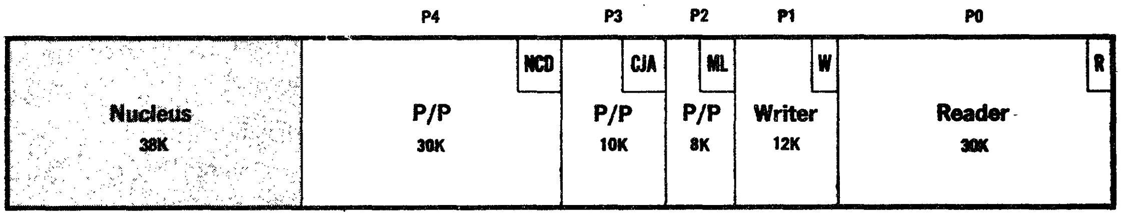


Figure 12. Sample Five-Partition Configuration

The assignment of partition(s) in which a job executes is controlled by using the CLASS parameter on the JOB statement. The format of this keyword parameter is:

CLASS=job class

where job class is the alphabetic identifier (A-O) assigned to the job. If this parameter is omitted from the JOB card, a job class of A is assigned by the system. Any of the 15 job classes may be used, provided at least one partition has been assigned to each of the classes specified. When a job class for a particular job is designated (by the CLASS parameter), the job is executed only in a partition that has been assigned to service that class. If more than one partition is assigned to service that job class, the job is executed in the first available problem program partition. A typical JOB card may be specified as follows:

```
//JOBPAY JOB 661,'JDOE',CLASS=C,PRTY=12
```

In the configuration illustrated in Figure 12, this JOB card causes the job to execute in either P3 or P4, whichever is available first.

Partition Redefinition

Partition redefinition allows the operator to change the number of partitions, their size, and their job classes at any time after initial program loading (IPL). Adjacent partitions may be combined to accommodate jobs with large storage requirements; these partitions may be reestablished subsequently (within SYSGEN limits) when the need for a large partition has passed. Job classes assigned to a partition may be changed also, to accommodate changes in the work load for one or more job classes. Reader and writer partitions may be respecified as problem program partitions and assigned to service jobs from as many as three job classes. Problem program partitions may be respecified as reader or writer partitions by assigning RDR or WTR in place of the problem program job class.

In addition, if the time-slicing feature is included in the system, the number of time-slicing partitions can be decreased or increased, within system generation limits, the range of the highest and lowest partition number to be time-sliced can be changed, or the amount of time to be allotted to each task can be modified. All time-slicing attributes may also be canceled.

Partition redefinition is invoked in either of two ways, depending on whether it is to be invoked during or after system initialization. At system initialization, the partition configuration may be changed by replying "YES" to message IEE801D "CHANGE PARTITIONS?". Alternatively, partition redefinition may be invoked after system initialization by entering the operator command, DEFINE. The format of this command is shown in the **Operator's Reference** publication.

Note: The DEFINE command cannot be entered through the input stream.

Partition Combination

Adjacent partitions may be combined as soon as their jobs have been terminated. If an unending job is being executed in a partition that is to be combined with an adjacent partition, the unending job must first be terminated with a CANCEL command. All other partitions that are to be conbined, including readers and writers, are made quiescent by the system as soon as their current tasks are completed. Any number of **adjacent** partitions may be combined. For example, in Figure 12, P2 and P3 may be combined into one larger partition of 18K. However, P2 and P4, and Pl and P3, may not be combined. When P2 and P3 are combined, the new configuration is as shown in Figure 3. P2 **or** P3 may be made the inactive partition. When P2 and P3 are combined, the job classes to be serviced by the new partition (P2) must be determined. If no change in job class(es) is specified, the classes currently being serviced by P2 remain as the job class assignments of the new partition. (See "Identity Change" below.) The inactive partition (P3) is made nondispatchable until it is recovered.

With the optional storage protection feature attached to the system, a unique protection key is available for each problem program partition. A list is kept of each available key for subsequent reassignment to combined or recovered partitions. When partitions are combined or recovered, the first available protection key on the list is assigned to them.

Note: With systems having the protection feature, storage assignment increases through partition redefinition should be made in increments of 2K bytes. If they are not, the system rounds the value to the next 2K increment.

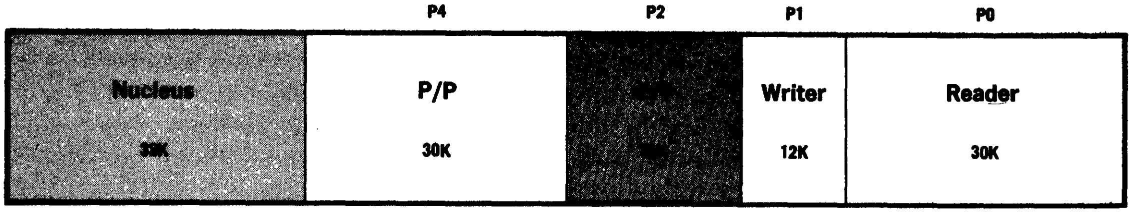


Figure 13. Partition Configuration After Combination

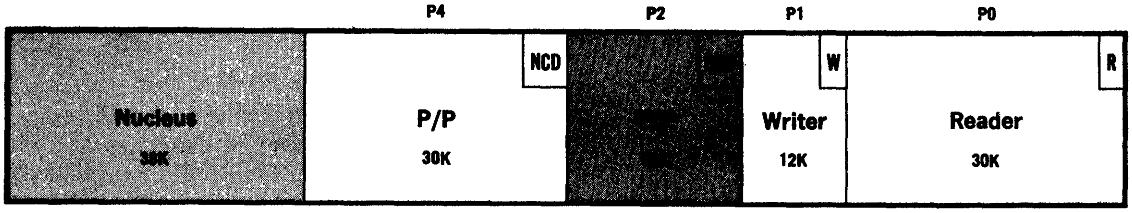


Figure 14. Partition Identification After Combination

Identity Change

Partition redefinition also allows the job classes specified at system generation or at system initialization to be changed. Problem program partitions may be redefined either as readers (by entering RDR) or as writers (by entering WTR). Reader and writer partitions may be changed to problem program partitions to service as many as three problem program job classes. When partitions are combined or recovered, the job class(es) that will be assigned to the resulting partitions must be determined. In Figure 13, P2 and P3 were combined into the larger partition P2. However, the original partitions each had three job classes. Therefore, the decision must be made whether to choose new job classes or some combination of the six old classes. For example, P2 could be assigned job classes C, A, and M, or a new job class could be specified, such as O. A new configuration is illustrated in Figure 14. If a new job class identifier(s) is not included during partition combination, the job class(es) originally assigned to the partition which remains active is unchanged. As a result, some jobs already enqueued on the input queue may not have a partition assigned to service them.

Partition Recovery

Partitions that were combined may be reestablished, or recovered. In Figure 13, P3 is now inactive, but is to be recovered. Once again the decision must be made as to which job classes will be assigned to both P2 and P3. P2 and P3 need not retain their original size, nor their previous job classes. Figure 15 shows a possible new configuration with P3 recovered.

Note: When recovering partitions, a job class(es) must be specified for the partitions being recovered, since only the currently active partition has a job class(es) assigned.

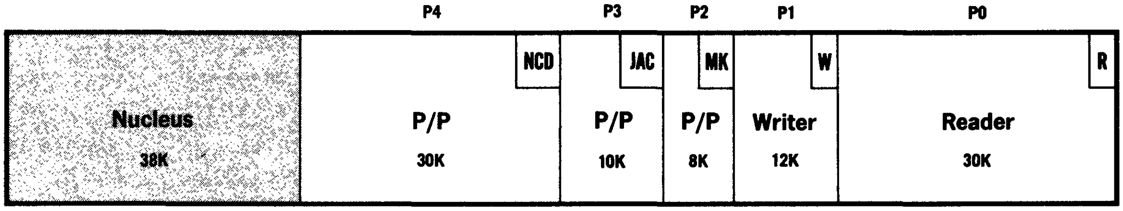


Figure 15. Partition Configuration After Recovery

Partition Definition Processing

As shown in Figure 16, when the operator enters either DEFINE or the reply "YES" to the "CHANGE PARTITIONS?" message, the system requests that the definitions be entered. If "LIST" was specified, the system lists the current partition configuration, including the time-slicing specifications, if this feature was chosen. (The operator must remember to CANCEL all affected unending jobs before redefining the system. If he does not, the new partition definitions do not take effect.) After definitions are entered, the system checks their validity and also inhibits scheduling subsequent jobs into the affected partitions. The system will stop any active direct system output writer in the affected partition. When the current jobs have been terminated, the new definitions are implemented. Section III contains operating considerations associated with partition redefinition. For a complete discussion of partition redefinition, see the **Operator's Reference** publication.

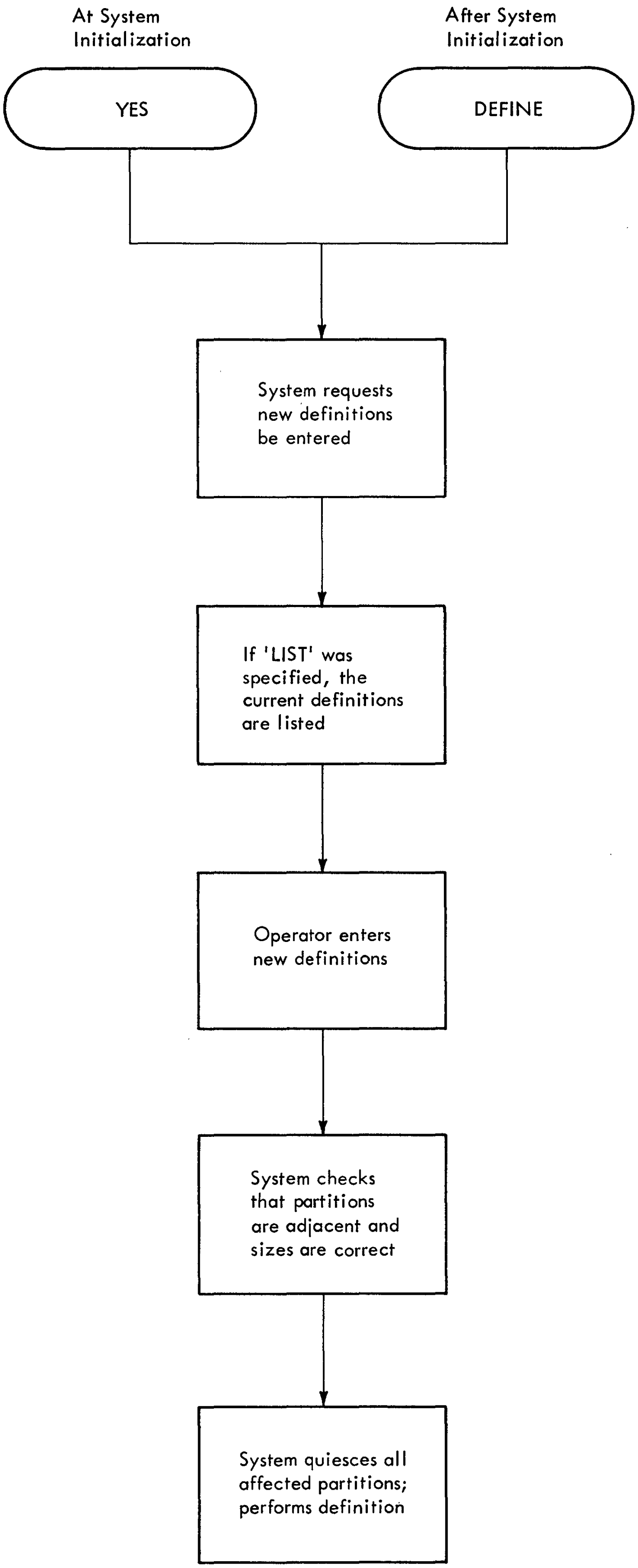

At System Initialization
YES
After System Initialization
DEFINE
System requests new definitions be entered
If 'LIST' was specified, the current definitions are listed
Operator enters new definitions
System checks that partitions are adjacent and sizes are correct
System quiesces all affected partitions; performs definition

Figure 16. Partition Definition Processing

Input Readers

MFT allows as many as three input readers to read and analyze input streams. Each reader executes concurrently with problem programs and writers in other partitions. A reader executes in any problem program partition (transient reader) or previously defined reader partition (resident reader) large enough to accommodate it. Input stream data for the step being read is transferred to direct access storage, where it is held until execution of the associated job. The problem program retrieves the data directly from the storage device. Multiple input stream data sets are permitted within job steps. A card reader, magnetic tape unit, or disk storage device unit may be specified as the input device. Figure 17 illustrates a job stream on cards.

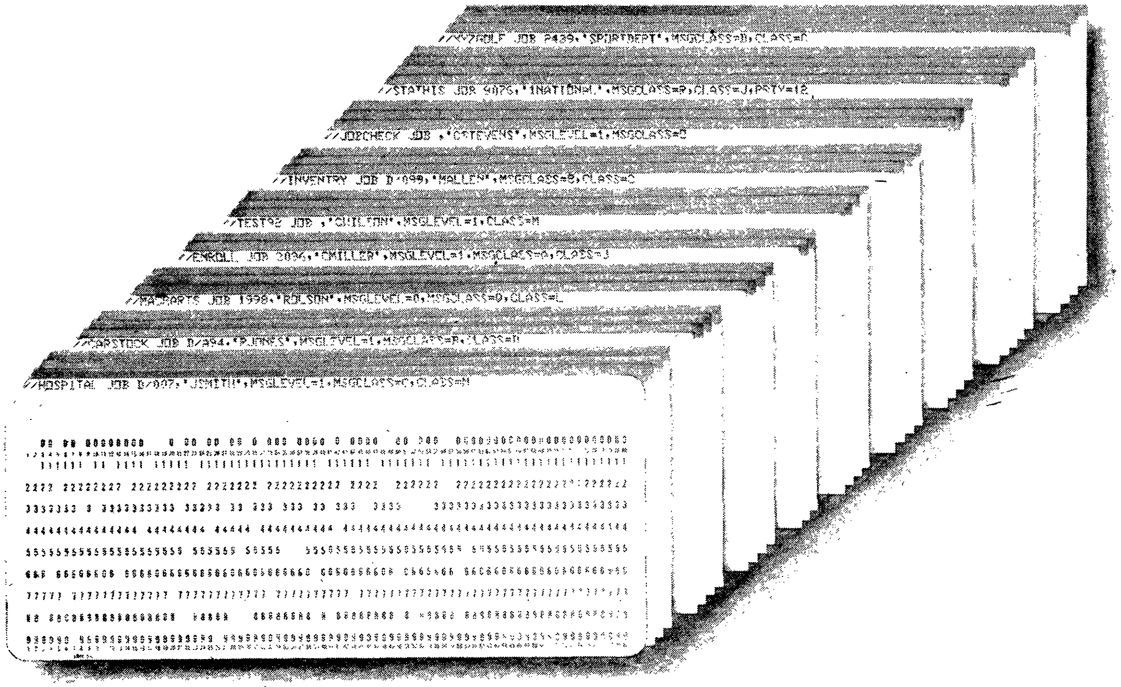

//XYZGOLF JOB P439,'SPORTDEPT',MSGCLASS=B,CLASS=C
//STATHIS JOB 9076,'INATIONAL',MSGCLASS=R,CLASS=J,PRTY=12
//JOBCHECK JOB ,'CSTEVENS',MSGLEVEL=1,MSGCLASS=C
//INVENTRY JOB D/099,'MALLEN',MSGCLASS=B,CLASS=C
//TEST92 JOB ,'CHILTON',MSGLEVEL=1,CLASS=M
//ENROLL JOB 2096,'CMILLER',MSGLEVEL=1,MSGCLASS=A,CLASS=J
//MAJPARTS JOB 1998,'ROLSON',MSGLEVEL=0,MSGCLASS=D,CLASS=L
//CARSTOCK JOB D/394,'PJONES',MSGLEVEL=1,MSGCLASS=R,CLASS=D
//HOSPITAL JOB D/007,'JSMITH',MSGLEVEL=1,MSGCLASS=C,CLASS=H

Figure 17. Input Job Stream

A START command is entered to activate the reader. As the input stream is read, statements are analyzed as shown in Figure 18. If the statement is Job Control Language (JCL), it is scanned, and keyword and positional parameters are determined. If errors are found (e.g., no programmer's name, invalid continuation card, invalid parameters), control is passed to a routine which prints error messages on a system output device. The remaining JCL statements are analyzed and any further error messages are written out. The job is then run out and must be reentered in the input stream with the correct JCL. If no error is found, JCL statements for the entire job are converted to control tables and entered in the input work queue corresponding to the CLASS and PRTY parameters specified on the JOB statement. (See "Enqueuing Jobs by CLASS and PRTY" in this section.)

If the statement is data, it is transferred to direct access storage where it is retrieved by the problem program. Command statements in the input stream are passed to the master scheduler for processing. When end-of-file (EOF) is reached, or a STOP command is issued, control is passed to the communications task to issue the READER CLOSED message.

Resident Readers

A resident reader partition is established by assigning "R" to a partition at system generation, or "RDR" during redefinition, in place of the job class identifier.

The resident reader in Figure 10 would be started by the following START command:

```
    START RDR.P0,devicename
```

This reader can be stopped only when the job stream input device reaches end-of-file, or by the STOP command:

```
    STOP RDR.P0
```

If a resident reader is used, input to the reader can be blocked. However, the size of the resident reader partition must be increased if the input is to be blocked. (See "Reader Partition Size Requirement" in the **Considerations** section for reader partition size information, and "Blocked Input" for information related to blocked and unblocked data.)

Note: "RDR" and "WTR", and "DSO" are used throughout the text as names of standard reader and writer cataloged **procedures**. See "Choosing the Reader Procedure" in Section III for information on the IBM-supplied reader procedure to use. (Cataloged procedures are explained in Section in the IV and **Job Control Language Reference** publication.)

Transient Readers

A transient reader operates in a problem program partition until it reads a job for that partition's, or a small partition's, job class(es). At that time, the reader's work areas are saved on direct access storage, and control is passed to the initiator to schedule the job into the partition. When the job has been terminated, and no further jobs require the partition, the reader's work areas are restored, and the reader resumes execution.

If a transient reader is started in a specific partition by including the partition assignment in the START command, it always resumes operation in the same partition when there are no more jobs on the queue(s) that the partition is assigned to service. This type of transient reader is referred to as **user-assigned** and is started in the same manner as the resident reader.

An "S" may be substituted for the partition assignment in the START command, allowing the system to place the reader in the first problem program partition of sufficient size that has no work enqueued on any of the queues that the partition is assigned to service. This type of transient reader is referred to as **system-assigned** and is started with the following command:

```
    START RDR.S,devicename
```

Resident and transient readers may operate in the same system, provided no more than one system-assigned transient reader is specified, and the total number of readers does not exceed three.

Note: Blocked input (from tape or disk) to a transient reader is **not** permitted. (See "Blocking Input" in Section III.)

Hierarchy Support: If main storage hierarchy support is included in the system, the transient reader can be placed in **either** hierarchy 0 or hierarchy 1, depending on the following conditions:

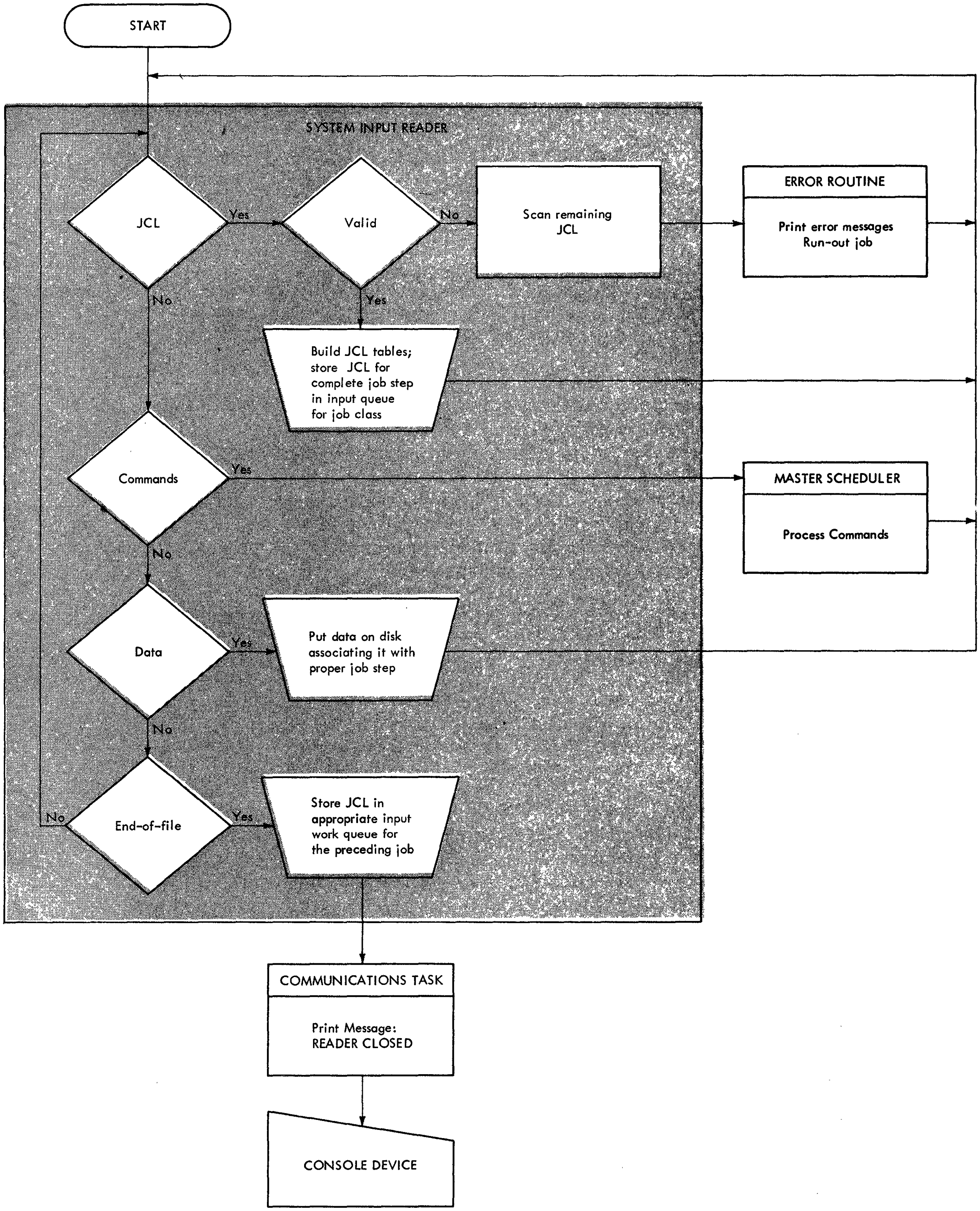

START
SYSTEM INPUT READER
JCL
Yes
Valid
No
Scan remaining JCL
ERROR ROUTINE
Print error messages
Run-out job
No
Yes
Build JCL tables; store JCL for complete job step in input queue for job class
Commands
Yes
MASTER SCHEDULER
Process Commands
No
Data
Yes
Put data on disk associating it with proper job step
No
No
End-of-file
Yes
Store JCL in appropriate input work queue for the preceding job
COMMUNICATIONS TASK
Print Message: READER CLOSED
CONSOLE DEVICE

Figure 18. System Input Reader Processing

- If a partition is defined only in one hierarchy, and is at least the size of the reader, the transient reader is placed in that hierarchy.
- If a partition is defined in both hierarchies, and hierarchy 0 is large enough to contain the reader, it is placed in hierarchy 0. However, if hierarchy 0 is not large enough for the reader, the reader is not placed in either hierarchy, regardless of the size of hierarchy 1.

  Examples:

  H0=0K,H1=50K -- the reader is placed in hierarchy 1

  H0=44K,H1=100K -- the reader is placed in hierarchy 0

  H0=10K,H1=200K -- the reader is not placed in either hierarchy

- If a Model 50 is being used, and a reader is placed in hierarchy 1, overrun will occur.

Enqueuing Jobs by CLASS and PRTY

Each job read by system input readers is converted into tables that are placed in the queue specified by the CLASS parameter. Jobs are entered into the input work queues for each class according to the PRTY parameter (PRTY values range from a low of 0 to a high of 13). One input work queue exists for each of the 15 job classes. Jobs having the same class and priority are placed in the queue first-in/first-out (FIFO). When the input work queue for a job class contains one or more jobs, termination of a job in any partition assigned to service work for that job class is followed by selection of the next highest-priority job from the input queue. Selection and initiation of the new job does not require operator intervention.

For example, if CLASS=D, PRTY=12 is specified on the JOB statement, the job is placed on the input work queue for job class D, behind any previously enqueued PRTY=12 jobs, but ahead of all other jobs of lower priority.

The PRTY parameter applies only to **initiation** priority, not to dispatching priority. Dispatching priority governs which job should be given control of the CPU. In MFT, dispatching priority is derived from the relative position of the partitions: P0 has highest priority, P51 lowest.

The dispatching priority of a task is determined from the relative position of the task control block (TCB) on the dispatching queue. (The dispatching queue is the chain of TCBs indicated by the TCBTCB fields.) If the MFT system does not have the subtasking option, all TCBs are established in the nucleus at system generation. Their order provides a dispatching priority starting with resident system task TCBs, through the job step task TCB of the highest priority partition (P0), to the successively lower priority partitions' TCBs (P1-P51). Control of the CPU is given to the program represented by the highest-priority ready TCB.

If the MFT system has the subtasking option, TCBs established at system generation in the nucleus represent the resident system tasks and the job step task of each partition. However, each job step task can attach subtasks, each of which will have a TCB located in the system queue area. The dispatching priority is initially the same as the partition's priority. The dispatching priority becomes different than the partition's priority when a job step task issues a CHAP (change priority) macro instruction to change its dispatching priority. If dispatching priorities are not changed, each partition's job step task is dispatched before its subtasks, which are then dispatched in the order in which they were attached. When all of a job step's subtasks have been dispatched, the job step task of the next lower priority partition can be dispatched.

Note: If no PRTY parameter is specified on the JOB card, the job is assigned the default priority specified in the reader procedure.

If the time-slicing feature was specified at system generation, the effective dispatching priority of a group of time-sliced partitions can be altered. Time-slicing allows the user to establish one group of consecutive partitions in which the task in each of the partitions is assigned a uniform interval of time to retain control of the CPU. When the allotted time has elapsed, the next lower-priority ready task gains control of the CPU for its allotted time. This process continues until either all tasks are waiting and completed, or until a task of higher-priority becomes ready.

Job Initiation and Termination

The job scheduler contains the initiation, allocation, and termination portions of the control program. As illustrated in Figure 19, the job initiation portion selects jobs from input work queues and determines which type of output writer to use for each output class. As each problem program is executed, it retrieves its input (SYSIN) data from the direct access device where it was previously stored by the system input reader. (Note that this retrieval takes place at direct access speeds, which is faster than reading input data directly from a card reader.) During problem program execution, output data directed to an output class is recorded on a direct access device, or written directly to the output device depending on the type of output writer selected.

Jobs are scheduled for execution according to:

1. Their job class identifier.
2. Their priority within the job class queue.
3. An available partition corresponding to the appropriate job class.

When a job is complete, the scheduler performs the required termination and informs a system output writer that the data produced by the problem program is ready to be written on the specified device.

Job Initiation

To schedule a job, the system places an initiator in an available partition. The initiator selects a job from the input work queues established by the system input reader, allocates devices for it, selects an output writer and, schedules it for execution. The initiator operates in any scheduler-size problem program partition. Small partitions are scheduled by an initiator contained in a large partition. Initiators obtain jobs for partitions based on the job classes assigned to the partitions, and the priority of the jobs within the job classes. The jobs are then scheduled for execution. An initiator can be given control during system initialization, or after a job has been terminated.

An initiator first checks whether a small partition needs scheduling. If the small partition has requested scheduling, a job requiring that partition is scheduled into it. The initiator then schedules the next available job into its partition and passes control to the first step of that job. When system output writers are used, output data sets are placed on direct access storage devices while the job is being processed. The output data sets are enqueued by output class and subsequently retrieved by system output writers.

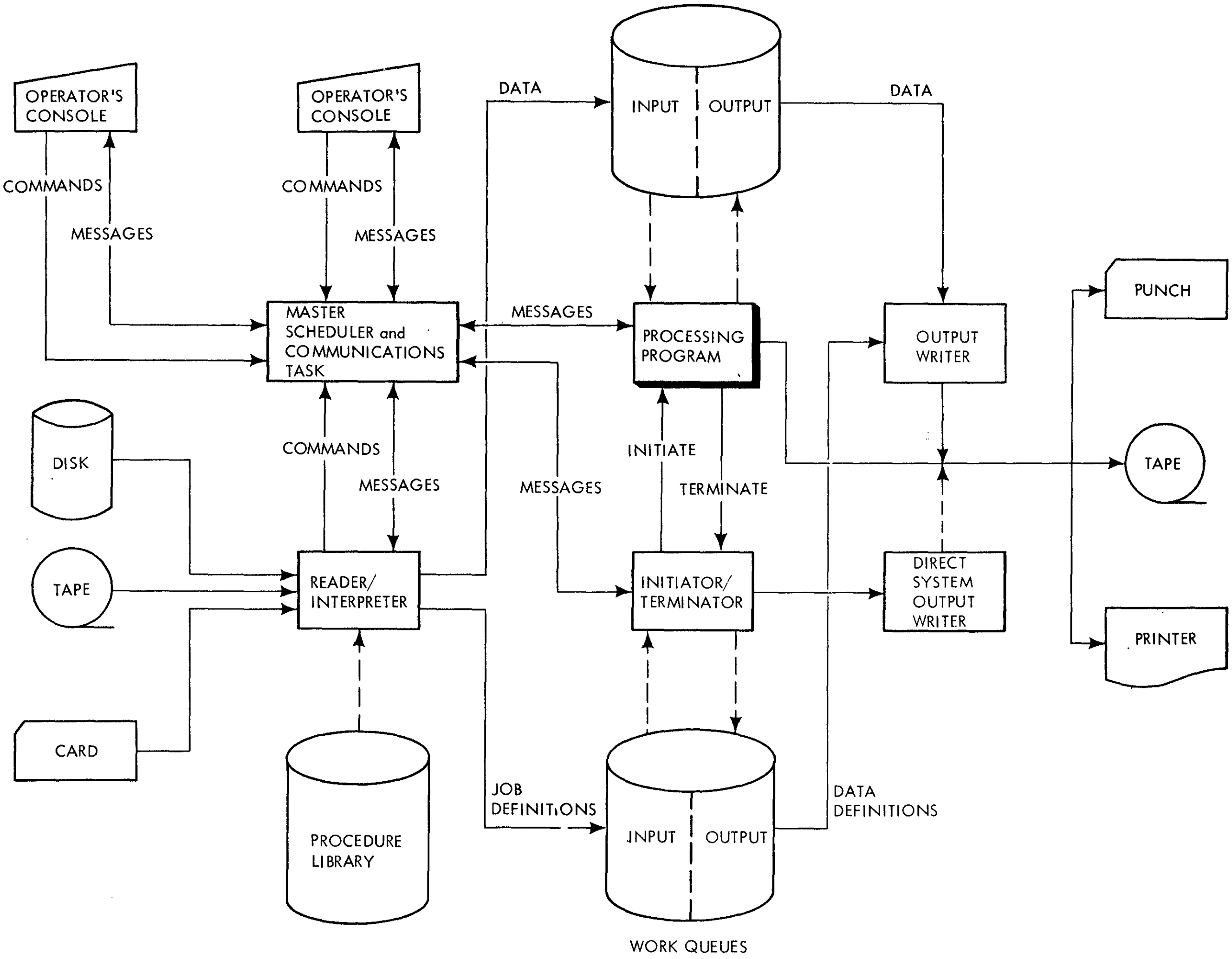


Figure 19. The MFT System

Job Termination

The termination portion of the control program determines first whether **step** termination or **job** termination is to be performed. Step termination includes disposing of data sets, deallocating input/output devices, processing condition codes, and executing an accounting routine. If the job contains additional steps, control is returned to the initiator to schedule the next job step. Job termination is performed after the last step of a job has been terminated. An accounting routine is executed and data set disposition and input/output de-allocation that could not be done at step termination are completed.

If the job used direct system output writers, its output was written directly to an output device or devices: no intermediate device had to be used and no output work is enqueued.

If the job used system output writers, its output is entered in the output work queue for processing by the system output writer. Output work queue members are enqueued within output class according to the PRTY parameter on the JOB card. Jobs having the same output class and priority are placed in the queue on a first-in-first-out (FIFO) order. For example, if a single output class is specified for system messages **and** all output for a particular problem program, the output work queue for that class includes, at job termination:

1. All system messages produced at job initiation, such as allocation messages.

2. All problem program output.

3. All system messages produced during job termination.

This output is transferred to the specified output device in the order shown. Different types of output, such as system messages and problem program data, are never intermixed. Control is then returned to the initiator for scheduling of a new job.

Data sets for a job are enqueued on the output work queue according to output class and priority, so that they can be written by an output writer. These data sets may include data sets produced during a job step, as well as control program messages. Depending on its characteristics and the way it is to be processed by the control program, a data set may be assigned to any one of 36 output classes (A-Z, 0-9) defined at an installation. A particular output class may reflect such characteristics as priority of the data, type of device to record it, or location or department to which it is to be sent. (See "Choosing Output Classes" in Section III.)

Direct System Output Writers

Direct system output writers are a job scheduler function. They enable a job's output data sets to be written directly to an output device during execution of the job. They write problem program and system messages, produced by the initiator/terminator, directly to system output devices. Valid output devices are: printer, punch, and magnetic tape.

Direct system output writers operate in large or small problem program partitions. As many writers can operate in a partition as there are devices available. Each writer must be assigned to only one device and can process one system output class. Problem program output can be handled by a direct system output writer if the job class of the problem program is an eligible job class and the problem program is executing in the same partition as the writer. 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.P3,283,,(JOBCLASS=ABC,
OUTCLASS=B)
```

any job running in partition 3 with a jobclass of A, B, or C and an output class of B will have its output written directly

The job class and output class of a direct system output writer can be changed by use of 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 user-supplied DSO procedure may be used, but it must execute the IBM-supplied direct system output writer.

System Output Writers

System output writers are IBM-supplied or user-written programs that retrieve problem program output from temporary direct access storage, and transcribe it to the device specified by the problem programmer. This device is specified on the SYSOUT parameter on the DD statement describing the output data set. The temporary data sets may be written on any direct access storage device except the IBM 2302 Disk Storage Unit; i.e., the valid devices are:

- IBM 2301 and 2303 Drum Storage.
- IBM 2305-1 and 2305-2 Fixed Head Storage Units.
- IBM 2311 and 3330 Disk Storage Drives.
- IBM 2314 and 2319 Direct Access Storage Facilities.

Up to 36 system output writers can be established. Each writer operates independently in its own partition concurrent with the execution of other system tasks and problem programs. MFT provides the same output writer services as MVT. Each system output writer can handle as many as eight different output classes; more than one writer can service the same output class. As many as 14 non-resident writers may be started if the total number of resident and non-resident writers does not exceed 36. Non-resident writers operate in the same way as resident writers, except that they are executed in problem program partitions.

When a job is terminated, system messages and output data are enqueued in the appropriate system output queue. One system output queue exists for each output class. Queues are serviced in the order specified in the START command. These classes override those in the writer cataloged procedure. The writer dequeues the first entry from the primary queue. If there are no entries in the primary queue, the writer dequeues the first entry in the secondary queue. This continues through the eighth queue, or until the writer finds work.

For example, to start a writer in P1 (see Figure 10) to process six output classes, the operator could enter the following command:

```
    START WTR.P1,00E,,ABCDEF
```

(The comma before output class A replaces the positional parameter, volumeserial.)

The writer first processes all the output from output class A. When the queue for class A is empty, the writer processes class B. When the output class B queue is empty, the writer again searches the class A queue. If no output has been enqueued for class A, the writer now searches the output class C queue. The writer continues processing in this manner until the following STOP command is entered:

```
    STOP WTR.P1
```

If no work is enqueued for any of the classes assigned to an output writer, the writer is placed in a wait condition until a job is terminated that has system messages and/or system output data sets for one of the writer's classes. When the last record in a queue entry has been processed, the writer deletes the entry before dequeuing another entry.

Writer tasks are performed concurrently with other writer tasks, readers, and problem programs. A writer task is terminated only when the operator issues a STOP command. The MODIFY command can be used to change a writer's classes. (See "Changing Output Classes" in the Section II.)

If a resident writer is established at system generation, "W" is assigned to the partition, in place of the problem program job class. No other work is performed in the partition unless its identity is changed through partition redefinition. Resident writers operating in small partitions are scheduled by a scheduler-size partition, when the large partition is available for scheduling duties. (See "Job Initiation and Termination" in this section.) After a writer is started, no other scheduling services are necessary until a STOP writer command is entered. At that time, a scheduler in a large partition terminates the writer.

Writers that operate in problem program partitions (non-resident writers) are brought into a problem program partition of sufficient size (at least 10K plus the input and output buffer sizes) by a START command which specifies that partition or contains an "S" for a system-assigned writer. A system-assigned writer does not leave its partition, as does a system-assigned reader. The writer operates in the partition until a STOP command is entered. At that time, the scheduler may initiate a problem program in the partition.

A user-written output writer procedure may also be used. This procedure may execute a user-written writer program or the IBM-supplied writer. If a user-written writer procedure is used, it must be placed in SYS1.PROCLIB and named in the START command. For example, if a user-written output writer procedure called USERWRIT is to be started in P3, the following command would be used:

```
START USERWRIT.P3,00E
```

Hierarchy Support: If main storage hierarchy support is included in the system, the non-resident writer can be placed in **either** hierarchy 0 or hierarchy 1, depending on the following conditions:

- If a partition is defined in only one hierarchy, and is at least the size of the writer, the non-resident writer is placed in that hierarchy.

- If a partition is defined in both hierarchies, and hierarchy 0 is large enough to contain the writer, it is placed in hierarchy 0. However, if hierarchy 0 is not large enough for the writer, the writer is not placed in either hierarchy, regardless of the size of hierarchy 1.

  Examples:

  H0=0K,H1=50K -- the writer is placed in hierarchy 1

  H0=44K,H1=100K -- the writer is placed in hierarchy 0

  H0=10K,H1=200K -- the writer is not placed in either hierarchy

- If a writer is placed in hierarchy 1 and a Model 50 is being used, overrun will occur.

(See "System Output Writers" in the **Considerations** section for additional information on resident and non-resident writers.)

System Restart

Because it is sometimes 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 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 (see Figure 20). When the system is restarted, the operator receives messages describing the status of each job in the system. If the job was being interpreted, the jobname is written out at the console. If the job was under control of a scheduler, the jobname **and** stepname are given. In addition, the operator is told whether allocation was being performed for the job, or whether the job was being executed or terminated.

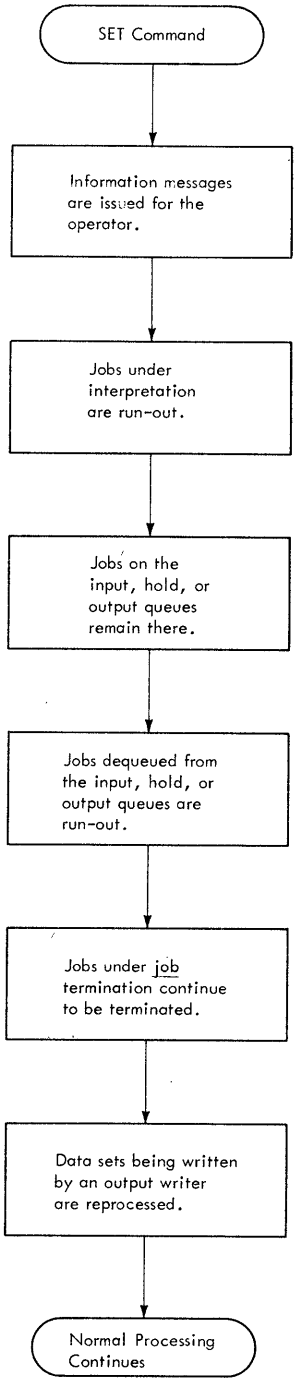

SET Command
Information messages are issued for the operator.
Jobs under interpretation are run-out.
Jobs on the input, hold, or output queues remain there.
Jobs dequeued from the input, hold, or output queues are run-out.
Jobs under job termination continue to be terminated.
Data sets being written by an output writer are reprocessed.
Normal Processing Continues

Figure 20. System Restart Processing

Invoking System Restart

After the system is reloaded, and after nucleus initialization, system restart may be invoked by omitting the "F" suffix from the Q=(unitname,"F") parameter of the SET command, or by omitting the Q= parameter entirely. This is valid only for the initial SET command, and cannot be done at any other time. This omission indicates to the system that the job queue data set (SYS1.SYSJOBQE) already exists in proper format, and requires only initialization.

Jobs That Were Being Interpreted

When the system must be restarted, jobs that were being interpreted are run-out and must be reentered in the input stream.

Jobs on Input, Hold, and Output Queues

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

Jobs that were dequeued from the input and hold queues are run-out and must be reentered in the input stream. However, if the entire job (i.e., not job step) was being terminated, the job does not have to be reentered in the input stream; system restart completes the termination.

With the checkpoint/restart facility, any job that was in step termination at restart time will be executed starting at the next step of the job, after system restart is complete. Any job that requested restart will be restarted at the current step, after system restart is complete, if the following conditions are satisfied:

- The step completed the allocation phase (that is, the phase between allocation and step termination).

- System completion code 2F3, designating system restart, was specified as eligible for restart.

- The operator replied yes to the verification request (message IEF225D) to restart the job.

System Output Processing

Output jobs that were being processed by a system output writer are requeued to reprocess any 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.

MFT Features

MFT makes possible the concurrent execution of up to 15 separate jobs within a single computing system having only one central processor, while continuing to provide all other applicable services of the IBM System/360 Operating System. Other features of MFT include:

- Independent job scheduling.
- System Management Facilities (SMF)
- Job/step CPU timing.
- Job step CPU time limiting
- WAIT time limiting
- Small partitions (smaller than the size of the scheduler).
- Checkpoint/restart.
- Reading of an input stream from an IBM 2311, 2314, 2319, or 3330.
- Concurrent execution of tasks within a partition.
- Generalized Trace Facility (GTF)

Each feature is described briefly in the following paragraphs.

Extended Multiprogramming Capabilities

MFT extends to 15 the number of jobs that may operate concurrently in the system. Jobs are scheduled into partitions through use of the CLASS parameter on the JOB statement, in conjunction with the PRTY parameter. The CLASS parameter may designate one of the 15 available job classes, A-O. With the optional storage protection feature, each of these jobs is protected from damage by other jobs, and the system areas are protected from damage by other jobs, and the system areas are protected from all problem programs.

Independent Job Scheduling

All partitions are independent with respect to job scheduling and initiation. Jobs are scheduled into the first available problem program partition that services the corresponding job class. Jobs are initiated according to the PRTY parameter on their JOB cards. The operator intervention and job sequencing requirements imposed by the WAITR macro instruction and the SHIFT command in the first version of MFT, are eliminated. The WAITR is treated as a WAIT macro instruction.

System Management Facilities (SMF)

SMF is optional with MFT. SMF collects and, optionally, records system, job management, and data management information, and provides control program exits to installation-supplied routines that can periodically monitor the operation of a job or job step.

SMF is invoked at system generation by specifying TIMER=JOBSTEP in the SUPRVSOR macro instruction and ACCTRTN=SMF in the SCHEDULR macro instruction. There are two options for SMF:

- OPT=1 specifies that only system and job information be collected.
- OPT=2 specifies that system, job, and job step information be collected.

The collected information is written onto data sets resident on either tape or direct access devices.

Job/Step CPU Timing

The job step CPU timing feature is optional (unless SMF is selected) with MFT. The amount of time that each job or job step has control of the CPU is calculated by task management routines. If SMF is selected, this value is passed to the SMF routines, and to a user-supplied accounting routine if one is provided. If SMF is not selected, the value is passed to a user-supplied accounting routine.

The job/step CPU timing feature is invoked at system generation by specifying TIMER=JOBSTEP in the SUPRVSOR macro instruction, and ACCTRTN=SUPPLIED or ACCTRTN=SMF in the SCHEDULR macro instruction. CPU timings are calculated for each job step. This value is then passed, along with an accumulated value for the **entire** job, to a user-supplied accounting routine for further processing.

Note: The CPU timings include only the **active** time that the CPU has control of the job step. It does **not** include the **wait** time or the time used by the initiator and terminator for that job.

Job Step CPU Time Limiting

This feature allows the user to specify the maximum amount of time that a job step can use the CPU. However, if the SMF option is selected and a user exit routine is provided, this routine can extend the time limit so that processing can continue.

Wait Time Limiting

This feature suspends processing of a job step if the job step remains in a wait state for more than a established time limit. If the SMF option is selected, the installation can provide a user exit routine to extend the time limit.

Small Partitions

Small problem program partitions can vary in size from 8K bytes to the size of the scheduler selected for the system. Small partition scheduling permits jobs to be scheduled into partitions that are smaller than the size of the scheduler. This scheduling service is provided by a partition of scheduler size whenever the large partition is available for scheduling purposes. This can take place at initiation and termination of jobs.

Checkpoint/Restart

The checkpoint/restart facility provides an opportunity to restart a job that terminates abnormally due to a hardware, programming, or system error. The restart is permitted either at the beginning of a job step or at a checkpoint within a job step. In either case, the restart may be automatic or may be deferred until the job is resubmitted. For a the checkpoint/restart facility, see the publication **Advanced Checkpoint/Restart.**

The checkpoint/restart facility permits a restart either at the beginning of a job step (step restart) or at a checkpoint within a job step (checkpoint restart). A checkpoint restart is requested by issuing a CHKPT macro instruction; a step restart by including special parameters in the job control statements for the job.

In a checkpoint restart, the restart must be executed in the same main storage area as was used for the original execution. The required main storage must be contained within one partition. Furthermore, the partition must be a problem program partition; that is, the partition must not be defined as a reader or writer partition. Restart may also be delayed if a DEFINE command entered by the operator changes the boundaries of the partition. In a step restart, there are no such restrictions.

In a system that has MFT with subtasking, a CHKPT macro should not be issued by a subtask or by a job step task that has active subtasks. For further information, see the **Operator's Reference** publication.

Input Stream From Disk

MFT allows the user to establish a disk storage drive (IBM 2311 2314, 2319, or 3330) as a system input device. Data in the input stream is permitted, including multiple data sets for the same job step, providing the facility for:

1. Reading input from sequentially organized data sets.
2. Deblocking of blocked input records (resident readers only).
3. Automatically switching volumes if end-of-volume is detected on a data set extending across volumes, or concatenated data sets are being processed.
4. Starting more than one reader for the same disk storage unit.

MFT With Subtasking

MFT with subtasking allows the user to have any number of tasks (up to a maximum of between 196 and 249) executed concurrently within a partition. This option provides the user with a version of MFT that has:

- An ATTACH function that can create subtasks.
- A DETACH function that removes completed tasks that were requested by the ATTACH routine.
- A CHAP function that allows a problem program to change dispatching priorities within the dynamic area of storage.

For description of MFT with subtasking, see the **Supervisor Services** publication.

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 interrrupts, both for event classes or for an individual interrupt code
- External interrupts
- 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 MFT control program occupying less than 1000 bytes when inactive and the specfied partition 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 not installed with 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 partition (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 recording -- or 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 recieving 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, order number GC28-6719, describes both the Generalized Trace function and the Edit function in detail.

Section II: MFT Options

This section describes the options that can be included in a system during system generation. The descriptions are not lengthy and other publications contain further and more detailed descriptions. The **System Generation** publication describes how to include these options.

Alternate Path Retry (APR)

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

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 at least 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.

Attach Function

Standard For: MFT systems without Subtasking
Optional For: MFT systems with Subtasking

There are two versions of the ATTACH function -- with and without the subtasking capability. One of these versions is always part of the control program.

The ATTACH function without the subtasking capability passes control to a requested load module and, when the requested load module terminates, returns control to the program that issued the ATTACH macro instruction.

The ATTACH function with the subtasking capability creates subtasks so that the issuing program and the program requested in the ATTACH macro instruction compete for system resources. This ATTACH function allows more than one task to be executed within one partition.

Attach Function Made Resident

The routines that make up the ATTACH function can be made resident in main storage as part of the nucleus. If this function is not resident, every time an ATTACH macro instruction is issued the ATTACH routines are brought into the supervisor transient area.

Basic Direct Access Method (BDAM)

The basic direct access method (BDAM) can be included in the operating system. If the user plans to use the SVC 2B (CIRB) in his programs, then BDAM, ISAM, or BTAM must be specified.

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 operatot 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 for 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 cause 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.

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 when specifying the multiple console option (MCS). (See the description in "Consoles - Multiple Console Support (MCS)".) One alternate console can be specified 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.
- 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.

The following consoles must be specified:

- One console must be specified in the SCHEDULR macro instruction as 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 exectuion 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 graphic 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 detail operating instructions see the publication **Operator's Reference.**)

- 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 console 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.

The following guidelines must be followed:

- 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.

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)

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. 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, any 2400 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 th: t has DDR.
- The direct access serial number verification routines must be in the system that has th: DDR routines.

For FETCH: When I/O errors occur while the FETCH routines are addressing the SVC .IB, 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 ma:ro 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.

Extract Function Made Resident

The routines that make up the EXTRACT function can be made resident in main storage as a part of the nucleus. If this function is not resident, every time an EXTRACT macro instruction is issued the routines are brought into the supervisor transient area.

The EXTRACT function that is included in an MFT system with the subtasking capability is the same as the EXTRACT function in an MVT system.

The EXTRACT macro instruction provides the user's program with information contained in specified fields of the task control block (TCB) of either the task that issued the macro instruction or, in a multiprogramming environment, one of its subtasks.

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.

Identify Function Made Resident

The routines that make up the IDENTIFY function can be made resident in main storage as a part of the nucleus. If these routines are not resident, every time an IDENTIFY macro instruction is issued the routines are brought into the supervisor transient area. If the IDENTIFY modules are resident, performance increases, but the amoun of fixed main storage required also increases.

The IDENTIFY macro instruction is used to inform the supervisor of an embedded entry point within a load module.

After the IDENTIFY macro instruction has been executed, the entry point can be used in an ATTACH, LINK, XCTL, or LOAD macro instruction.

Indexed Sequential Access Method (ISAM)

The indexed sequential access method can be included in the 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.

Job Step Timing

The operating system can time each job step and enforce limits on the time it may run. In systems that include job step timing, the control program passes a parameter called "CPU time" to the user accounting routine or to the SMF routines. CPU time represents the elapsed time of step execution minus the unoverlapped wait time.

Subsequent runs of the same job may have different CPU times for the following reasons:

- The frequency with which the task gets interrupted.
- The amount of code executed for each interruption before a time limit inhibits the interrupted task.
- The varying exectuion times for the SVCs that the job issues.

In addition, the job step timing option includes the following: 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 hierarchy 1 to the 2361. Program controlled interrupt (PCI) must always be specified. (See the description of "Program Controlled Interrupt (PCI)" later in this section.)

If there is no 2316 Core Storage Unit in the system, any references to hierarchy 1 will have to be defined at IPL time. Hierarchies for partitions in MFT are defined in the PARTITNS system generation macro instruction.

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 boundaries 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 is 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. MFT systems can have only user-written load modules and the loader program modules from LINKLIB made resident in the reenterable load module 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.

Resident Access Method Routines

Reenterable access method load modules can be made resident from the SYS1.SVCLIB.

The standard list, IEAIGG00, contains the names of the access method routines that are to be loaded and made resident by IPL. If the user desires to create his own list to load certain modules, then the operator communication option (OPTIONS=COMM) must be specified in the SUPRVSOR macro instruction. Then at IPL time the message (IEA101A) will print out "SPECIFY SYSTEM PARAMETERS". The operator answers with the unique identification numbers for the user's list. The access method routines that are pointed to by this list will then be loaded into main storage at IPL time.

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 may 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 Disk 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. (See Appendix D for a full discussion about using the RESERVE macro instruction with the Shared DASD Option.)

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.MANY

SYS1.SYSVLOGY SYS1.ACCT

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.

SPIE Routines Made Resident

The Set Program Interruption Element (SPIE) function can be made resident. If this function is not resident, it is brought into the supervisor transient area whenever a SPIE macro instruction is executed.

The SPIE macro instruction specifies the address of a routine to be used when specified program interrupts occur in the task that issued the macro instruction.

Storage Protection Option

When the storage protection feature is included in the central processing unit, this option may be specified. The validity check option is included as a standard feature when this option is specified.

When the storage protection option is specified, the size of all protected areas must be a multiple of 2,048 bytes.

The storage protect hardware is a standard feature for models 50 and larger. But the storage protection programming option must be specified during system generation. If the storage protection programming option is specified, the validity checking function is included and cannot be specified in the system generation coding.

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. 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 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 MFT. If CIRB (SVC 2B) is desired in the new system, BDAM, ISAM, or BTAM must be specified.

The Time Slicing Facility

The user can establish a group of tasks (called the time-slice group) or partitions 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 or partition 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 guarantee response time, the time slice group should be defined in the high priority partitions.

A group of contiguous partitions defines the time-slice group. All tasks scheduled into those partitions are time-sliced and are treated as though they had the same dispatching priority. In MFT, only one group of tasks can be specified to be to be sliced.

Time-slicing operates within the structure of the current dispatcher. 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 containing a ready task. Each partition in the group is dispatched for the specified time slice. The time slicing continues until either all partitions are waiting, or a partition of higher priority than that of the group becomes ready.

The dispatcher will recognize that a priority level is being time-sliced; it will determine which partition within the group is to be dispatched and then dispatch that partition 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 partition becomes ready), the remainder of the time is not saved. That is, when control returns to the time-slice group, the next ready partition in the group is given control, not the interrupted partition.

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 in MFT with the DEFINE command. Any partition in the system that is not defined within the time-slice group is dispatched under the current priority structure; that is, the partition is dispatched only when it is the highest priority ready partition on the TCB queue.

Time slicing is invoked through either the JOB statement or, in MFT systems with subtasking, through the use of the ATTACH and CHAP macro instructions.

If a task becomes part of the time slice-group through the use of ATTACH and CHAP (in an MFT system with subtasking), the task gains control according to the priority used with ATTACH or CHAP. The task gains control, as part of the time-slice group, when the partition with the same priority gains control (even though the task resides in a partition that is not part of the time slice group). Equally, a task that is time-sliced may use ATTACH and CHAP with a priority that does not fall within the range of priorities assigned to the time slice group. The attached or changed task is not part of the time-slice group even though it resides in a time-slice partition.

The time-slice group is composed of a group of contiguous partitions and all tasks scheduled into those partitions are time-sliced. Also, each partition in the system is assigned to at least one job class. Since a job is scheduled into a partition, according to the CLASS parameter on the JOB statement, careful consideration should be given to the job-class assignment in order to enable the user to control the use of time slicing at his installation. For example:

1. Partitions P0-P2 have been assigned as the time-slice partition.
2. The partitons have been assigned the following job classes:

   P0=G,P1=G,P2=(G,D),P3=B,P4=(B,C,D)

In this example, the user can insure that a job will be time-sliced by specifying CLASS=G on the JOB statement. This specification guarentees that the scheduler will initiate the job only into a partition assigned to CLASS G, i.e.; P0, P1, or P2. Since P0-P2 have been designated as time-slice partitions, that job will be time-sliced.

MFT systems with subtasking assign time-slicing both by partition and by dispatch priority of the job classes assigned to the time slice partitions. If a program uses the ATTACH or CHAP macro instruction, the priority used with ATTACH of CHAP determines when the attached or changed task is time sliced, not the partition in which it resides. (However, a program cannot exceed the limit priority assigned its jobclass.) For a discussion of dispatch and limit priority, see the **Supervisor Services and Macro Instructions** publication.

Modifications to time slicing are made like other partition modifications. At system initialization, changes can be indicatied by replying "YES" to the message "IEE801D CHANGE PARTITIONS". After system initialization, changes can be indicated through the DEFINE command. In both cases, changes are actually made by responding to the message: "IEE002A ENTER DEFINITIONS" of "IEE803A CONTINUE DEFINITION" with the new TMSL reply. With this reply, the operator can request a list of current time-slicing specifications, change the range of time-slicing partitions and the time interval, or cancel time-slicing specifications altogether.

Note: Non-interactive jobs should not be run concurrently and time sliced since this may significantly decrease performance.

Timing Options

The TIME Option

The INTERVAL Option

The JOBSTEP Option

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

- **Time** (TIME) provides the facilities of the TIME macro instruction, which is the date and time of day.
- **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.

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.

Transient SVC Table Made Resident

The relative track addresses (TTR) of all transient supervisor (SVC) routines are included as part of the resident table of control program SVC routines. (See the description in "Type 3 and 4 SVC Routines Made Resident" in a succeeding topic.)

If type 3 and 4 SVC routines are being made resident, this option must be specified also.

During a nucleus generation this option can be added or deleted from the options specified during a complete system generation.

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 the **Storage Estimates** publication. (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.

Validity Check Option

Extra validity checking can be added to the new system to determine whether addresses are loaded within proper boundaries. The validity checking is provided for the WAIT, POST, and GETMAIN/FREEMAIN modules. The checking for wait also checks for the number of events.

This option is specified in the SUPRVSOR system generation macro instruction.

During a nucleus generation this option can be added or deleted from the options specified during a complete system generation.

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 for MFT systems. 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 acts as a 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.

Section III: Planning Considerations

Programming planners and systems analysts preparing an MFT system need to know many factors besides how the control program works or how many options are available to the user. Planning personnel must know minimum system storage and device requirements. They should be familiar with the ways of influencing system performance by altering job priorities, changing job classes, or choosing readers and writers. Furthermore, planners and system analysts want to know how their special needs at an installation can be considered. To assist in planning an installation, this section describes:

- General Considerations
- Minimum System Storage and Device Requirements
- Common Considerations
- Special Considerations
- Typical System considerations
- Operating Considerations

General Considerations

In preparing to use MFT, data processing planners and system programmers should evaluate not only the characteristics and requirements of the jobs to be processed by the system, but the characteristics and facilities of the system that influence how a job is processed once it has been presented to the system. Some of these characteristics are general and apply equally to all types of jobs. Others are related directly to job type. An important category of characteristics, although related to job types, is exhibited primarily in system operation, and must be considered by machine room supervisors and machine operators.

In this section, the topic "General Considerations" describes items of interest primarily to planning personnel, that should be considered before generation of an MFT system. The topic "Special Considerations" describes "Batch Processing," "Telecommunications," "Graphics," and "Concurrent Peripheral Operation" -- considerations important to the systems programmer and the application programmer. These four topics are organized similarly, in "checklist" fashion, so that the reader interested in a given job type need read only "General Considerations" and the topic corresponding to the job type in which he is interested, to learn all considerations pertinent to that job type. Because partition configurations will depend on the amount of main storage available as well as on the types of jobs to be run, the topic "Typical System Configurations" describes partition arrangements for systems with 128K bytes, 256K bytes, and 512K bytes of main storage. These configurations are general, but should be helpful for planning. The seventh topic, "Operating Considerations" describes briefly characteristics of MFT that may affect operating procedures.

All of the messages, including operator responses and system actions, are explained in the **Messages and Codes** publication.

Minimum System Storage and Device Requirements

The minimum MFT system requires 34K.

A computing system using MFT must have at least 128K bytes of main storage and the following devices (which are included in the MFT 34K nucleus):

- One selector channel
- Two direct access storage devices (except the 2302 Disk Storage Unit)
- One IBM 1052 Printer-Keyboard
- One IBM 3210 or 3215 Console Printer Keyboard
- One card reader or tape device
- One card punch or tape device
- One printer or tape device

The minimum MFT system must also include the resident queued sequential access method (QSAM) routines for use with system readers and system output writers. The 34K nucleus supports these QSAM routines. However, if QSAM is not included in the nucleus, the size of the reader or writer partition must be increased to accommodate these routines.

The 34K nucleus supports a maximum of 2 partitions. For each additional partition generated, the size of the nucleus must be increased by 1K bytes.

The following features are also supported in the 34K nucleus:

- One multiplexer channel
- A third direct access device
- Four additional tape devices
- Storage protection

For each additional channel, input/output device, or control program feature, the size of the nucleus must be increased by the amount of main storage required as listed in the **Storage Estimates** publication.

If the shared direct access device feature is selected, the system must also include one IBM 2314 Direct Access Storage Facility combined with a 2844 Auxiliary Storage Control unit, or one of the following units equipped with a two-channel switch:

- IBM 2305 Fixed Head Storage Unit
- IBM 2314 Direct Access Storage Facility
- IBM 2820 Control Unit (with IBM 2301 Drum Storage)
- IBM 2841 Storage Control Unit (with IBM 2311 Disk Storage Drive, 2303 Drum Storage, or 2321 Data Cell)
- IBM 3330 Disk Storage Drive

Common Considerations

Several considerations apply to all phases of the system; these must be considered regardless of the type of job that is being run. They include:

- Estimating storage requirements.
- Choosing SYSGEN macro instruction options.
- Using resident reenterable routines.
- Placing system libraries on direct access devices.
- Sharing direct access devices with other systems.
- Choosing the size of the scheduler for the system.
- Choosing the number and size of partitions.
- Specifying appropriate job classes.
- Assigning job names.

- Formatting problem program messages.
- Choosing input readers and output writers.
- Avoiding system interlocks.

Estimating Minimum Storage Requirements

MFT operates in a system with at least 128K bytes of main storage. The system nucleus requires approximately 34K bytes, exclusive of space for resident access methods (other than QSAM) or the resident BLDL table. (See "Main Storage Organization" in **Section I: The MFT Control Program** for the components of the nucleus). If these features are selected, the corresponding main storage requirements should be added to the size of the nucleus. For complete information on storage requirements, see the **Storage Estimates** publication.

Calculating System Configurations

The configurations possible at a given installation may be roughly calculated as follows:

1. From the total main storage capacity, subtract the size of the system area.

2. Subtract the size of the scheduler-size problem program partition which is required for the system.

3. If resident writers are used, subtract 10K plus the input and output buffer sizes for each writer specified.

4. If a system input reader(s) is to be resident also, subtract the size of the reader for each one specified (30K or 44K plus the buffer sizes, depending on the scheduler chosen).

Remaining main storage may be apportioned through any combination of the following:

1. Inclusion of other optional features of the Operating System.

2. Increasing the size of the problem program partitions already defined.

3. Increasing the number of problem program partitions.

4. Increasing the number of system output writer partitions.

5. Increasing the number of system input reader partitions.

Note: If the system includes the storage protection feature, all storage assignment increases must be made in increments of 2K bytes.

Single Console vs. Multiple Consoles

Through multiple console support (MCS), an installation may use one primary (or master) console and multiple secondary consoles where each console is dedicated to one or more system functions (for example, tape library, disk library, or teleprocessing control). MCS services all consoles concurrently, creating an environment for operator/system interaction that gives each console the appearance of being the only console on the operating system. Each console operator receives only those messages from the system that are related to the commands that he enters and to his assigned functions.

MCS provides the mechanism to:

- Route messages to selected functional areas

- Allow a user-written exit routine to modify the message's routing and descriptor codes prior to the issuance of the message

- Switch to an alternate console if a primary console should fail

- Allow automatic message deletion on devices such as display tubes (graphics)

- Support a hard copy log for the recording of routed messages, operator commands, and system responses

Selective message routing, provided under MCS, is the ability to route both problem program and system-initiated messages to functional areas and the SYSLOG device. Messages appear only on consoles that have been specifically designated to receive the messages. In this manner, a console whose function is to receive tape messages, for example, is prevented from receiving messages not pertinent to that function. Routing codes are defined in the **Supervisor Services and Macro Instructions** publication.

A system generation option is provided to permit insertion of a resident, user-written exit routine in the communications task. The exit routine receives control prior to routing any WTO and WTOR messages whose routing codes will be used by the operating system. The exit routine may examine but not modify the message text; however, the exit routine may modify the message's routing and descriptor codes. Messages will be sent only to those locations specified in the modified routing codes. A complete explanation of the exit routine is in Section IV.

MCS permits console switching, which can be initiated automatically, by operator command, or when the operator manually presses the interrupt key on the system control panel:

- Automatic console switching to an alternate console occurs when permanent hardware errors are detected by the operating system.

- Command-initiated console switching occurs when the system accepts a valid VARY operator command. Command-initiated console switching is used to restructure the system console configuration and the hard copy log. Console switching to an alternate console can be performed by placing the original console either online or offline.

- Manual switching is limited to the master console (primary console device) and is initiated by pressing the interrupt key on the system control panel. Manual switching to a new master console is used when the master console is inoperative and the hardware failure cannot be detected by the system.

Messages can be automatically deleted from the screen on the Model 85 Operator Console with CRT Display by means of the DOM macro instruction. When a system or problem program no longer requires that a message be displayed -- for example, if a WTO macro instruction was issued and the message is no longer needed -- a DOM macro instruction should be issued to delete the message from the screen.

MCS allows buffered or immediate hard copy. Thus, no information is lost when messages and operator commands are deleted from graphic displays. In addition, the hard copy device can be used as a collection point for all messages and commands. Routing codes and the time, if the timer option is present, are prefixed to all messages and commands that are sent to the hard copy log. The system log -- the only buffered hard copy device supported -- must be specified at system generation, and can be modified at system initialization or during system operation. If hard copy is desired on a console, the hard copy device can be specified at system generation, at system initialization, or during system operation. Although the system log is supported with or without the MCS option, the hard copy log is only supported with the MCS option.

Using Resident Reenterable Routines

The resident reenterable load module feature allows problem programs to share reenterable code. It provides improved performance by placing frequently used modules in the resident reenterable routine area. It also provides reduced main storage requirements by placing modules in the area that are common to concurrently running jobs.

The feature uses the RAM parameter to make possible the pre-loading of both access method modules from the SVC library and any user-written reenterable modules from the link library. The feature can be specified at system generation by specifying RESIDNT=(RENTCODE,ACSMETH) in the SUPRVSOR macro instruction. At system initialization, the user can either cancel the feature entirely or provide up to four lists of access method and other reenterable modules of his own choosing to be loaded into the resident reenterable routine area.

In an operating environment where any problem program issues an ATTACH, LINK, LOAD, or XCTL macro instruction to request use of a reenterable module that is not resident in its partition, the supervisor will search the resident reenterable routine area for the module. No additional copies need be brought into main storage. Therefore, frequently used reenterable load modules should be loaded into the area. Any user-written reenterable load module and the loader modules from the link library can be loaded into the area by including it in one of the lists of modules specified; type III and IV SVC routines, however, must be loaded only into the RSVC area.

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 all six libraries are on the same device, throughput is decreased because of excessive arm interference. To increase throughput, libraries should be balanced on devices; devices should be balanced on channels. The ideal condition would be to have each library on a different direct access device, and each device on a separate channel. In installations with smaller systems, it would be best to have SYSJOBQE and LINKLIB on the same direct access device on channel 1, and SVCLIB, PROCLIB, PARMLIB, and MACLIB on another device on channel 2.

Whenever more than one library is to be placed on a 2311, 2314, 2319, or 3330 disk storage device, arm movement can be substantially reduced by placing the volume table of contents (VTOC) approximately midway between the first and last cylinders being used. The libraries, starting with the most frequently referenced, can then be alternately placed on both sides of the VTOC with the least referenced libraries furthest from the VTOC.

Blocking the Procedure Library

Blocking the procedure library conserves input/output storage space. The procedure library may be blocked during system generation or afterwards by using utilities.

Blocking the procedure library during system generation is done by pre-allocating the procedure library with RECFM=FB and BLKSIZE=(a multiple of 80). During Stage II of system generation, the iebcopy utility program blocks the procedure library on the new system. Blocking the procedure library may require that the reader partition size be increased (see

"Reader Partition Size Requirement"). If possible, the reader partition size should be considered prior to system generation -- that is, prior to the PARTITNS macro instruction -- to prevent redefining partitions at IPL just for procedure library blocking.

A preliminary study used a typical procedure library containing 54 procedures to determine the most efficient blocking factor to conserve input/output storage space. Blocking factors from 1 to 40 (BLKSIZE=80 to 3200) were used. It was found that blocking factors in the range 8 to 12 were most efficient. For example, on a 2311, an unblocked procedure library required 30 tracks but a blocked procedure library (with blocking factors in the range 8 to 12) required only 21 tracks, a reduction of 30% in I/O storage space.

Determining the Size of SYS1.SYSJOBQE

The size of the job queue data set (SYS1.SYSJOBQE) for a particular installation depends on several factors:

- Maximum number of jobs that will be queued at any time on the input and output queues.
- Average size of the jobs.
- Number of records in a logical track.
- Number of 176-byte records on a physical track.

For the formula to estimate the size of SYS1.SYSJOBQE, see the **Storage Estimates** publication.

Sharing Direct Access Storage Devices (DASD) With Other Systems

If the shared DASD feature is selected at system generation, considerations should be given to volume assignment and classification (read only or read/write) of common data sets. Volume handling and device reservation procedures must be carefully defined, because the shared DASD feature cannot prevent or resolve interlocks between systems. Detailed considerations for using the shared DASD option are given in Section IV.

Choosing the Size of the Scheduler

MFT provides two basic scheduler packages, 30K and 44K. (Detailed considerations of actual scheduler size are given in the **Storage Estimates** publication.) The choice of scheduler depends upon several factors: desired throughput, desired partition configuration, and main storage size. Usually, the primary factor is the amount of main storage available. In a 128K system, using the 30K scheduler allows specification of a greater number of partitions because 64K (with the 34K nucleus) would still be available for other problem program partitions, writer partitions, and/or reader partitions.

With the 44K scheduler in a 128K system, the user has 50K to establish either another problem program large parition, or several small problem program and/or writer partitions. The 44K scheduler increases throughput, but leaves less main storage for other partitions. If jobs are of short duration, it may be advisable to use the 44K scheduler, because scheduling activity will be high. However, if jobs are relatively long in execution time, the scheduler will not be needed as often; therefore, the 30K scheduler could be utilized.

The actual scheduler size needed to initiate a job may be specified during system generation (by use of the MINPART parameter) or during initialization (by use of the MIN parameter). This value may be equal to or greater than the scheduler design level selected.

Note: The size of the scheduler chosen at system generation determines which reader will be used by the system when START commands are entered for a reader (see "System Input Reader Partitions" in Section I).

Choosing Number and Size of Partitions

The number of partitions needed at an installation depends primarily on the number of different job **categories** (i.e., batch, graphics, telecommunications, and CPO) expected to run concurrently. At least one partition must be specified for each category. The number of partitions for each category, based on the number of jobs expected to be run in each, should then be established. In practice, the maximum number of partitions for which there is available main storage should be established. If fewer partitions are needed during operation, the number of partitions can be reduced by the operator either at system initialization or during operation. If necessary, partitions may be reestablished up to the limit specified at system generation.

Note: If the maximum number of partitions is established at system generation, the size of the system queue area (SQA) at system generation also should reflect this maximum number. Then, if the number is reduced at system initialization, the SQA can also be reduced, by replying to message IEA101A "SPECIFY SYSTEM PARAMETERS".

Within the limits of the system, the maximum number of partitions that can be specified depends on the size of the selected scheduler in relation to the amount of main storage available. At least one partition must be large enough to accomodate the selected scheduler. If a job is known to exceed the size of its intended partition, an adjacent partition can be eliminated and its storage reassigned to the other partition. Reassignment of contiguous partitions can be accomplished without interfering with unaffected partitions.

Choosing Appropriate Job Classes

A system installation can be set up for maximum efficiency and throughput by properly using the partition job class concept. Particularly, the processing characteristics of jobs likely to execute concurrently must be examined. Failure to consider job mix can lead to degrading system performance. Multiprogrammed jobs can, under certain circumstances, run slower than they would if processed sequentially. Because the previous version of MFT could not recognize processing characteristics of a particular group of jobs, it was necessary for job streams to be balanced so that concurrently operating jobs were complementary rather than conflicting. However, now the required balance can be achieved with proper use of the CLASS parameter by:

- Establishing the job characteristics to be controlled, based on the typical processing workload

- Establishing a suitable partition structure compatible with the job characteristics to be monitored

- Establishing the convention that jobs having certain characteristics are to be directed, through the CLASS parameter, to the appropriate partitions

Typical job characteristics are:

- High compute, low input/output time
- Balanced compute and input/output time
- Low compute time, high input/output time

- Use of specific types of input/output equipment, such as 2250 terminals, magnetic tape only, or telecommunications terminals
- Large main storage requirements
- Small main storage requirements
- Setup or non-setup jobs
- Use of preallocated data sets
- Time-slicing considerations

With this type of categorization, job mix can be balanced for improved throughput. For example, one partition can be established for high-input/output jobs and another for high compute-time jobs. Process-limited jobs (such as compilers) can then be assigned to the high compute-time partition, and jobs with high input/output requirements (such as sort programs, and reading and writing of data sets) to the input/output partition. Normal job scheduling should then produce a satisfactory job mix. Because jobs are queued by the CLASS parameter, and because each partition is scheduled for its next job immediately after the preceding one is complete, the system as a whole tends to execute complementary jobs concurrently.

The CLASS parameter may also be used to direct jobs that are to be time-sliced to partitions that were defined as time-slicing partitions. However, when using the time-slicing feature, caution should be taken when assigning a job class to a particular job. If a job is to be time-sliced, it must be assigned a job class that will be serviced by a time-slicing partition. Likewise, if the job is not to be time-sliced, it should not be assigned a job class that a time-slicing partition has been assigned to service.

A partition should be established to service each job class specified on a JOB card. If a job is assigned to an unserviced job class, it remains on the input queue for that class indefinitely, or until the operator discovers (by use of the DISPLAY N command) that the job has not been executed.

Default Job Class

If **no** CLASS parameter is specified on the JOB card, the system assigns job class A to the job. Therefore, a small partition should not be assigned to service job class A unless all jobs run at the installation will fit into that small partition. It is advisable to make job class A either a secondary or tertiary job class in one or more partitions, to ensure that any jobs that are assigned the default job class will be executed.

The default job class is given to a job only when **no** CLASS parameter is specified, not when an incorrect job class is given. For example, if P2 is specified as job classes M and L (Figure 10), P3 as C, J, and A, and P4 as N, C, and D, the following JOB card illustrates an invalid job class specification:

```
//MFT  JOB ,'MYJOB',MSGLEVEL=0,CLASS=G
```

Because job class G is an invalid job class for this particular configuration, the job will not be assigned job class A. It is placed on the CLASS=G queue, and is never initiated. It remains there indefinitely until the operator discovers that the job has not been executed. Therefore, extreme caution should be used when choosing a job class for the job to ensure that a partition has been specified for that job class. To prevent delays in processing jobs with "invalid" job class designators, the operator should enter DISPLAY N periodically to obtain a listing of the jobs on the hold and input work queues.

Priority Scheduling Within Job Classes

Jobs **within** job classes can be initiated according to a priority specified in the PRTY parameter on the JOB card. For example, several jobs may be designated job class B. Within this group of jobs, some are to be initiated before others. Therefore, higher priorities can be assigned to these jobs with the PRTY parameter. This affects only the way the job is initiated, **not** dispatched. If no PRTY parameter is specified, jobs are assigned the default priority established in the reader procedure and are initiated FIFO for each job class. Therefore, each group of jobs for a particular job class should be analyzed to determine if some are to be initiated before others, and to assign these preferred jobs higher priorities.

Assigning Job Names

Any valid job name is acceptable to the MFT system. Therefore it is possible that jobs with identical job names could be in the system at the same time. These duplicate-name jobs could be on the same input queues, different input queues, or executing in different partitions. The existence of these duplicate-name jobs could cause confusion when using the DISPLAY, CANCEL, HOLD, RELEASE, and RESET commands. For example, if a CANCEL command is entered for a job in the hold queue or an input queue, the system will cancel the first job encountered if duplicate job names are in the queue.

To prevent this confusion, a procedure should be established which will ensure that all jobs have unique names. This could be done, for example, by varying a portion of the jobname, i.e., JOBPAY1, JOBPAY2, etc., to reflect sequence of input. Other methods of unique identification for jobs could be derived from application, programmer's name, time-of-day, date, or any combination of these which would satisfy the needs of an installation.

In addition, job names of P0, P1, ... P51 should not be assigned because these are the partition identifiers. For example, if a job is assigned a name of P4, and a CANCEL command is entered for this job, both the job named P4 and the job that is running in Partition 4 will be canceled.

Formatting Problem Program Messages

In MFT it is necessary to relate WTO and WTOR messages to the problem program which issued them. In order to do this, a partition identifier is added to each message issued by a problem program partition. The maximum length of WTO messages is 122 characters. The maximum length of WTOR messages is 119 characters.

Example: WTO messages appear in the following form:

PHASE A ENTERED Pn

Similarly, WTOR messages in MFT include the partition identifier as follows:

id REPLY 8 CHARACTER NAME Pn

Choosing System Input Readers

The size of the system input reader is determined by the scheduler design level selected at system generation (30K or 44K); i.e., the scheduler design level determines which reader is used by the system. (See "Choosing the Size of the Scheduler" in this section.) Additional reader considerations include whether to use blocked input, whether to block the procedure library, which cataloged reader procedure to use, whether to use resident or transient readers, and whether to use more than one reader.

Blocking Input

Blocking the input stream records reduces the time required to access and/or to process them. Also, if the records are blocked, auxiliary storage space is conserved in that more records can occupy the same amount of space than if the records were not blocked. There are two types of blocking that should be considered relative to the input reader.

- Input to the reader from the job stream.
- Output from the reader (i.e., input to the processing program.)

If a resident reader is used, input to the reader from the job stream can be blocked; with a transient reader it **cannot** be blocked. If the input is to be blocked, the number of buffers and their sizes may be specified on the IEFRDER statement in a reader cataloged procedure. Alternatively, the buffer sizes may be overridden by specifying the new buffer sizes and number in the START command for the reader. If blocked input is used, the size of the reader partition should be increased accordingly.

The output from the reader is the input stream data that is transferred from the input stream to a direct access device for subsequent retrieval by the processing program; e.g. all data records between a DD * statement and a /* statement. This data can be blocked by both the resident reader and the transient reader. The block size is indicated in the IEFDATA statement in the cataloged procedure and may be overridden by specifying on the DD statement for the job the new buffer size and length for the input stream data. With this facility, blocking characteristics would be on a DD statement basis instead of a reader basis.

Choosing the Reader Procedure

There are three IBM-supplied cataloged reader procedures that can be specified in a START command: RDR, RDR400, and RDR3200. The significant difference in these readers is the block size (BLKSIZE), buffer length (BUFL), and the number of buffers (BUFNO) on the IEFDATA statement. (All three readers specify unblocked input to the reader.) The IEFDATA statement is used to indicate to the reader whether the input stream data is to be blocked or unblocked. Reader procedure RDR specifies unblocked records; procedure RDR400 specifies blocked records with a block size of 400; procedure RDR3200 specifies blocked records with a block size of 3200.

The choice of appropriate IBM-supplied reader procedure to use at an installation depends on whether the input stream is to be blocked or unblocked. If a START command specifying either procedure RDR400 or RDR3200 is entered, the IBM-supplied or user-written programs being read must expect blocked input or must override the blocking factor on DD * (or DD DATA) statements for these programs. If one of the IBM-supplied reader procedures is not used, a user-written writer procedure may be used. (See Section IV).

To efficiently use main storage and to increase the efficiency of accessing or processing the input stream, it is best to use a reader procedure that blocks the input stream data. If the job stream is intermixed with IBM-supplied or user-written programs that cannot accomodate blocked input stream data, the DD statements for these particular jobs can specify unblocked data records. For a complete description of how to override the block sizes in the cataloged procedure, see the **Job Control Language Reference** publication.

Resident reader efficiency will be improved if the job stream is blocked, and if a reader procedure is used that blocks the input stream data. Also, multiple buffers will improve the efficiency, especially in the case of card input.

Note: If a transient reader is used, only **one** input buffer should be specified.

Resident Reader or Transient Reader

In addition to blocking input and choosing the reader procedure, the decision to use a resident or transient reader depends upon the amount of main storage available, and the quantity of work to be read from any one input device. A resident reader improves performance, but reduces the amount of main storage available for general use. Whether the advantages of a resident reader compensate for its overhead depends on the size of the system and the type of job most frequently run. The following general considerations apply:

1. The system input reader can be resident in any MFT system (128K or larger), and would probably be resident in any system larger than 128K.
2. In any system of sufficient size, the reader should be resident if a high-intensity job stream is typical. A high-intensity job stream is one which has a number of relatively-short jobs in terms of CPU time required; input-stream processing time is an appreciable percentage of overall job time.

With a 128K system, it may not be feasible to use a resident reader because of the small amount of main storage available. Therefore, a transient reader should be considered. A transient reader makes more main storage available for general use, at the cost of reduced overall performance. A transient reader should be considered also if the user has input streams on more than one device, and no device has a majority of the work.

Note: A combination of resident and transient readers may be chosen, but only three readers may operate in the system at one time.

Single Reader or Multiple Readers

In determining whether to have more than one reader, the size of the machine and the number of problem program partitions necessary for the installation should be considered. If the user has 256K bytes or more of main storage, it might be advisable to specify more than one resident reader. For example, he could specify one reader for cards, one reader for magnetic tape, and a third for disk.

The primary consideration is to analyze the jobs and to determine which input device will have the majority of the input in terms of CPU time. If this device is the card reader, then one reader partition should probably be specified to read the input stream from the card reader continuously. If, on the other hand, there is a long input stream on magnetic tape and/or direct access storage, a reader partition(s) should be specified for these devices, and a transient reader for the card reader. If there is only one long input stream, it would be advisable to specify one reader partition for that particular device, and start a transient reader for the other devices.

Choosing System Output Writers

If possible, records to be written by a system output writer should be blocked. This improves throughput, because less input/output time is required, and disk arm interference is reduced. However, additional main storage must be provided within the problem program partition, where the records are initially blocked, and within the system output writer partition, to which the logical records are read. (The additional space required, in each case, is equal to the logical record length times the blocking factor plus the input buffer space.)

Because there is the choice of specifying partitions as resident writers or having the writers operate in problem program partitions (see "System Output Writers" in Section I), there are several factors that must be considered when deciding which type of writer to use.

Resident or Non-Resident Writers

With a resident writer, machine size is not as large a factor as with a resident reader because the minimum system writer is only 10K plus the input and output buffer space, and can operate in a small partition. The size difference between the writers varies only with the amount of buffer space specified for the output data sets (i.e., blocked versus unblocked records).

If a resident writer is not used and, instead, a writer is started in a small problem program partition, throughput may be decreased because the system does not start the writer until it terminates the problem program in the small partition. Thus, the writer must wait until a large partition is free to terminate the small partition and to start the writer. The advantage of a resident writer is that it need not wait until a problem program partition is free. Once the START command is entered, the writer is scheduled by the first scheduler-size partition that is free to perform scheduling services. It continues to write until a STOP command is entered. The writer can then be restarted when it is necessary.

A non-resident writer could also be started in a scheduler-size partition, thus avoiding the wait time required for a large partition to schedule the writer into a small partition. However, this could also waste main storage, because the standard output writer is much smaller (10K) than the required size of a large partition (that is, the size of the scheduler). In this case, the relative importance of time versus main storage available must be considered, and then the decision should be made whether to start a non-resident writer in a small or large partition.

If the system does not contain a resident writer, a writer must always be started in either a small or large problem program partition often enough to prevent the output work queues from being filled. If the output queues become filled, the output data not placed on the queues is lost. Frequent use of DISPLAY Q command will allow the operator to anticipate this condition. (See the **Storage Estimates** publication for the formula to determine the size of the output queue.)

A system-assigned non-resident writer may also be started in a problem program partition; i.e., START WTR.S,00E. However, the system-assigned writer does not operate in the same way as a system-assigned reader. The system-assigned writer does not leave the partition as the reader does, because the only way to stop a writer in a problem program partition is to enter a STOP command. Therefore, when using the system-assigned writer, the operator should ensure that the writer is not remaining idle, thus occupying the partition unnecessarily.

Choosing Direct System Output Writers

Since the user has 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 partition. This can be an advantage to the user that has a 128K system. The extra storage required by a resident system output writer could be used to add a small problem program partition or to include a larger scheduler in the system.

- Each direct system output writer can process only one output class per partition and needs one I/O device assigned to it. In a system with several active partitions, there will probably not be enough I/O devices to run direct system output writers in all partitions 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 queue, and jobs failed by the reader/interpreter. 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 in the active partitions. This would allow the system output writers to clear the output queue, without stopping work in the problem program partitions.

Use of Multiple Writers

The use of multiple output writers has several advantages. In general, a unique output writer can be used for each requirement in the system. For example, the following output classes might be assigned:

- An output class for all system messages
- An output class for all high-priority printed output, or for printed output requiring special forms
- An output class for all punched output
- An output class for all output to magnetic tape

By specifying the appropriate output class in the DD statement, the programmer selects the particular device on which his output is to be recorded. Because writers can share output classes, a writer can have a primary and a secondary function. For example, if output class B is assigned to a high-priority printer, and output class C to a "background" printer, the high-priority printer processes only high-priority output (SYSOUT=B).

If no high-priority data is waiting on the output work queue, the output writer performs its secondary function by taking a job from the SYSOUT=C queue. The advantage in this use of multiple writers is not only that it makes writers available for certain types of unique work, but that it also permits them to perform other work when circumstances permit.

Note:Problem programs that are device-dependent with respect to data sets written by a system output writer may have to be modified to run under MFT. If the data control block (DCB) for the system output device was coded for other than a direct access device, it may not be large enough for conversion to a direct access device DCB, as required by MFT.

Avoiding System Interlocks

A problem can occur when a task controls a resource, but is waiting for another resource which is under control of a second task. If the second task is waiting, and needs the resource now under control of the first task, a system interlock condition will occur. The first task cannot give up its resource until the second task relinquishes the resource it controls, and vice versa. The two tasks are therefore in a deadlock; processing cannot continue in either partition.

Two ways to avoid this problem are:

1. Request all resources initially; do not begin an irreversible course of action until all required resources have been obtained.

2. If holding a formerly obtained resource which may prevent acquisition of another resource, release the resource **before** requesting the other resource(s). If the former resource is still required, request it together with the other resource(s). (See the **Supervisor Services and Macro Instructions** publication for further information on system interlock.)

Data Set Integrity

An interlock may also occur during job initiation, if a job requests one or more data sets which are reserved for use by another job which is currently executing in the system. In order to prevent this interlock, the operator is notified of the condition through a series of console messages. The operator must then make a decision, based on his knowledge of the jobs in the system, the system configuration, and the data sets in use. He may wait for the requested data set(s) to become available, or he may cancel the job being initiated. For a complete description of the messages and replies involved with data set integrity, see the **Operator's Reference** publication.

System Generation Macro Instructions

Three system generation (SYSGEN) macro instructions are required to specify the characteristics of an MFT system: CTRLPROG is used to specify control program options; SCHEDULR is used to specify job scheduler options; PARTITNS is used to specify number, class, size, and time-slicing attributes of the partitions. The parameters required for each macro instruction are given below. For a complete description of the parameters, see the **System Generation** publication.

CTRLPROG Macro Instruction

The parameters required for generation of an an MFT system are:

TYPE
to specify the MFT control program.

MAXIO
to specify the maximum number of input/output operations that can be simultaneously processed.

The following parameters may also be specified for an MFT system:

FETCH
to specify the type of program fetch to be used.

HIARCHY
to specify the inclusion of main storage hierarchy support.

OVERLAY
to specify overlay supervisor options.

SYSQUE
to specify the size of the system queue area.

TMSLICE
to specify the time-slicing feature.

Example: The following example illustrates the use of the CTRLPROG macro instruction to specify an MFT control program. The maximum number of input/output operations that can be simultaneously processed is 20. The advanced overlay supervisor and PCI fetch are included. The system queue area is assigned a size of 4000 bytes.

```
CRTLPROG  TYPE=MFT,MAXIO=20,FETCH=PCI,OVERLAY=ADVANCED,SYSQUE=4000
```

SCHEDULR Macro Instruction

The SCHEDULR parameters required for generation of an MFT system are:

TYPE
to specify the MFT control program.

CONSOLE
to specify the primary console device.

The following options may also be included in an MFT system:

ACCTRTN
to specify a user-supplied accounting routine or SMF.

ALTCONS
to specify an alternate console for the primary console device, or the alternate console for the master console if MCS is being used.

CONOPTS
to specify the multiple console support (MCS) option and, if MCS is being used, whether a user exit is desired.

DESIGN
to specify the 30K or 44K job scheduler.

HARDCPY
to specify that a hard copy log is desired to record operator messages and commands (for MCS only).

JOBQFMT
to specify the format of the system job queue.

JOBQLMT
to specify the number of 176-byte records to be reserved for initiators.

JOBQRES
to specify the system residence device.

JOBQTMT
to specify the number of 176-byte records to be reserved for termination of jobs.

MINPART
to specify the number of 1K (1024-byte) blocks of main storage required to initiate a job.

OLDWTOR
to specify the routing codes to be assigned to any WTO or WTOR message without routing and descriptor codes (for MCS only).

OPTIONS
to specify system log and job scheduler options.

PROCRES
to specify the procedure library device.

REPLY
to specify the number of reply queue elements for WTOR routines.

ROUTCDE
to specify the routing codes to be assigned to the master console (for MCS only).

STARTI
to specify an automatic START INIT.ALL command after IPL.

STARTR
to specify an automatic START reader command after IPL.

STARTW
to specify an automatic START writer command after IPL.

WTLBFRS
to specify the number of WTL buffers.

WTLCLSS
to specify the output class of the system log data sets.

WTOBFRS
to specify the number of WTO buffers.

Example: The following example illustrates the use of the SCHEDULR macro instruction. The size of the job scheduler is 44K bytes. The address of the primary console device is 009. The address of an alternate console is 01F. The START commands for readers and writers are to be executed after the system is loaded into main storage. The devices to be started are located at 00C and 00E respectively. The START command for initiators is also to be executed automatically after IPL. An accounting routine has been supplied. The job queue and procedure library reside on the device located at 190. There are 10 data records per logical track on the job queue. The default value is accepted for all parameters not specified.

```
SCHEDULR  TYPE=MFT,DESIGN=44K,STARTR=A-00C, STARTW=A-00E,
    ACCTRTN=SUPPLIED,STARTI=AUTO,CONSOLE=009,
    ALTCONS=01F,JOBQRES=190,PROCRES=190,JOBQFMT=10
```

Note: If the multiple console support (MCS) option is selected when coding the SCHEDULR macro instruction, a SECONSLE macro instruction must be included for each console except the master console; that is, for the alternate to the master console and for each additional secondary console. For a complete description of the SECONSLE parameters, see the **System Generation** publication.

PARTITNS Macro Instruction

The PARTITNS macro instruction is unique to MFT and must be used. It establishes the maximum number of partitions for an installation. The maximum number of partitions that an installation **intends** to use should be established at system generation, because this number can be reduced during system initialization or during operation. Unnecessary system generations may be avoided because the number of partitions in use may **never** exceed the number established at system generation. Therefore, if the maximum number of partitions that might be used at some later date is generated, the system does not have to be regenerated to increase the number of partitions in the system. The attributes of each partition are established as follows:

Pn(C-class,S-nnK)
where the characters P, C-, S-, and K must appear as shown.

"n"
specifies the partition number, 0-51.

"class"
specifies the partition's function (either one to three job classes A-O for a problem program partition, or R or W for resident readers and resident writers, respectively).

"nn"
specifies the amount of main storage to be allocated to the partition. One problem program partition should be the size of the selected scheduler. (If there is not a resident reader in the system, one problem program partition **must** be the size of the scheduler.) Resident readers must be at least 30K or 44K, depending upon the scheduler design level selected. Resident writer partitions must be at least 10K.

Example: This example illustrates the use of the PARTITNS macro instruction to generate a five-partitionsystem. Partition 1 is to be a 34816-byte problem program partition. Jobs that specify CLASS=K or CLASS=E on their job cards are to be scheduled into Partition 1. Partition 0 is to be a 32768-byte resident reader partition (P0 need not be specified before P1). Partition 2 is to be a 12288-byte resident writer partition. Partition 3 is to be a 51200-byte problem program partition to service job classes J and A. Partition 4 is a resident writer partition of 14336 bytes.

```
PARTITNS  P1(C-KE,S-34K),P0(C-R,S-32K),P2(C-W,S-12K),
    P3(C-JA,S-50K),P4(C-W,S-14K)
```

If hierarchy support is included in the system, the format of the macro instruction is changed to:

```
    H0-nnK
    Pn(C-class,  H1-nnK )
    H0-nnK,H1-nnK
```

H0-nnK and H1-nnK

specifies the size of the partitions to be established in each hierarchy. H0 and H1 are designations for hierarchy 0 (processor storage) and hierarchy 1 (IBM 2361 Core Storage) respectively. Problem program partitions can be specified in hierarchy 0, hierarchy 1, or in both hierarchies. Partitions with a job class of R (reader) or W (writer) can be defined to exist in either hierarchy but not both. (Note, however, that if a reader or writer is placed in hierarchy 1 and a Model 50 is being used, overrun will occur.)

Example: This example illustrates the use of the PARTITNS macro instruction to generate the same five-partition system shown above, but with main storage hierarchy support. Partitions 1 and 4 are generated in hierarchy 1 (IBM 2361 Core Storage).

```
PARTITNS  P1(C-KE,H1-34K),P0(C-R,H0-32K),P2(C-W,H0-12K),
    P3(C-JA,H0-50K),P4(C-W,H1-14K)
```

Note: If P1 was segmented into both hierarchies, the format would be:

```
PARTITNS  P1(C-KE,H0-10K,H1-24K), ...
```

Special Considerations

Every installation varies in the amount and type of work processed. Special consideration must be given by application programmers and planning personnel to accommodate the work load that they anticapate. Some special considerations accommodated by the MFT control program include:

- Batch Processing
- Telecommunications
- Graphics
- Concurrent Peripheral Operations (Spooling)

Batch Processing

If the installation's work is primarily batch jobs, several factors must be considered when the system is initialized, and when it is operating. First, the number, size, and job class(es) for each partition must be determined. Then the decision must be made as to which partitions, if any,should be specified as resident reader and writer partitions and which partition should contain direct system output writers. The proper output classes for the installation must also be determined.

Choosing Number and Size of Partitions

The number and size of partitions depends upon the size of the installation's system. Naturally, a system with 512K bytes or more main storage has great flexibility, yet even an MFT system that has 128K bytes has considerable flexibility. There are several possible configurations for a 128K system. The best configuration for a particular installation usually depends on the type of job most frequently run. There must be at least one scheduler-size (problem program or resident reader) partition. (See "Choosing the Size of the Scheduler" in this section.) Remaining main storage can be assigned to other problem programs, reader, and/or writer partitions. (See "Typical System Configurations" in this section for examples of batch processing system configurations.)

Small Partitions

Small partitions are well suited to batch processing. However, small partitions require a large partition for scheduling (see "Job Initiation and Termination" in Section I), so total throughput may be reduced.

With the configuration shown in Figure 21, P0 could be assigned a primary job class of K reserved by the installation for "critical" small jobs; i.e., jobs that require small amounts of main storage and that must have fast throughput. By also assigning job classes G and J to P0, the partition need not remain idle when there is no critical work. When a scheduler is in P2 (at initiation and termination of jobs), it performs any scheduling duties that have been requested by small partitions P0, P3, and P4.

Note: Any job assigned a specific class must be able to run in any partition assigned to service that class.

Choosing Reader Partitions

A reader partition would probably be resident in any system over 128K, but may also be resident in a 128K system. If a resident reader is chosen for a 128K system, the 30K reader should be used. This makes more main storage available for problem program and/or writer partitions. If a resident reader is to be used, the best flow of input to the system, and the greatest amount of processing time for problem programs is gained by placing the reader in a high-priority partition. If there are at least 256K bytes of main storage, the 30K reader would probably not be used, because the 44K reader operates more efficiently.

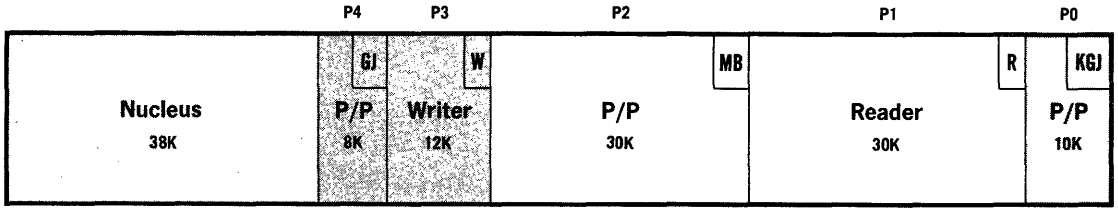


Figure 21. Sample 128K Small Partition Configuration

Assigning Job Classes to Jobs

Every job should be assigned a job class, using the CLASS parameter. For batch processing of input/output-limitedjobs, each job should be assigned a job class that corresponds to a high-priority partition. Process-limited jobs should be assigned to a lower-priority partition. In addition, jobs that can be executed without having any special input/output setups (i.e., "non-setup" jobs such as a FORTRAN compiler), or that have preallocated data sets, can be directed to a high-priority partition for fast throughput.

Assigning Partitions to Job Classes

After the job classes have been assigned to jobs, appropriate partitions must be assigned to service those jobs. If the partitions do not have the appropriate job classes specified, the job classes can be changed (see "Partition Redefinition" in Section I), or the CLASS parameter can be changed on the JOB card, before the job is entered in the input stream.

Choosing System Output Writer Partitions

If several partitions are specified as problem program partitions, it is advisable to have at least one resident writer in a higher-priority partition (Pl or P2). A resident writer will ensure that the output is continually being written. If another writer is needed, a non-resident writer can be started in a problem program partition, or a problem program partition can be redefined as a writer partition or a direct system output writer could be used.

Choosing Output Classes

At an installation it may be advisable to set up certain output classes for specific duties. For example, output class A could be for system messages, and class B for problem program output. Or, class A could be for system messages, class B for problem program output for the accounting department, class C for problem program output for the purchasing department, etc.

Note: System messages are assigned an output class through the MSGCLASS parameter on a JOB card. Problem program output is assigned an output class through the SYSOUT parameter on a DD card. (See the **Job Control Language Reference** publication.)

Another approach would be one in which output class A represents printer system message output, class B represents punched card output, class C represents magnetic tape output, and class D represents printer problem program output. Up to 36 output classes may be specified.

When using special forms on the printer, the operator should ensure that system messages are not written on the special forms. This possibility can be eliminated by establishing a different output class for output requiring the special forms.

Note: An identification problem may arise if system messages are assigned an output class different from problem program output. Therefore, it may be helpful for the programmer to print, as the first line of output, his name and department, if he chooses to use different classes for message and problem program output. This would also alleviate some operator problems. (See "Operating Considerations" in this section.)

When the system log option is present, system log data sets must be assigned an output class. The assignment can be made at system generation by using the WTLCLSS operand of the SCHEDULR macro instruction. Two log data sets are provided for recording the data sent to the log. To avoid the situation where the second data set becomes full before the first data set can be written, both the size of the data sets and the output writer class must be considered at system generation. To be sure that a full log data set is processed in a reasonable period of time, a unique output message class should be assigned for the log data sets and a writer should be assigned multiple output classes with the log class having the highest priority.

Telecommunications

MFT enables telecommunications jobs to be run concurrently with other types of jobs such as batch, graphics, and CPO. Several MFT considerations are of interest to the telecommunications user.

These considerations include the number of telecommunications partitions required, their placement in the system, their size, and their job class(es).

Choosing Number and Size of Partitions

Telecommunications jobs are considered unending in that they are scheduled only once, and are terminated only when a CANCEL command is entered, i.e., for partition redefinition. (See "Partition Redefinition" in Section I.) There must be at least one partition for each telecommunications job being run. The size of the partition depends upon the size of the telecommunications control program used by the installation.

To avoid delays in servicing lines, a telecommunications job should have unrestricted access to the resources of the central processor. For this reason, it is best to run telecommunications jobs in high-priority partitions. Because the telecommunications job is not alone in the system, its activities should cause minimum interference with jobs in other partitions, and it should not be susceptible to interference from these other jobs.

Small Partitions

If the telecommunications control program needs a small amount of main storage (i.e., less than the size of the chosen scheduler), use of a small partition would be efficient since it will have to be scheduled only once.

Choosing Reader Partitions

If the user is running primarily telecommunications jobs, it would not be necessary to specify a partition as a resident reader. Initially a transient reader could be brought into a batch problem program partition to read in the telecommunications jobs. (See Figure 22.) After the telecommunications partitions are activated, and a transient reader has read a job for P4, the reader will give control to the initiator to schedule a job into P4. (See "Input Readers" in Section I.)

Assigning Job Classes to Jobs

Each telecommunications job should have a unique job class assigned to it. The message control partition (P0) should have a different job class from the message processing partition. Caution must be taken to avoid assigning job classes to problem programs that correspond to the job class(es) of the telecommunications partitions. In Figure 22, a job is assigned a CLASS parameter of F if it is a telecommunications message control job. However, if the CLASS parameter for a message control job is K, the job is placed on input queue K, but never initiated.

Assigning Partitions to Job Classes

Each telecommunications partition should also have a unique job class so that the appropriate jobs may be directed to that partition. In Figure 22, P0 is assigned job class F, Pl is assigned E, and P2, D. If the job classes are to be changed, a CANCEL command must first be entered to terminate the unending job, and then the system may be redefined. (See "Partition Redefinition" in Section I.) Likewise, if the partition is not assigned to the telecommunications job class, the telecommunications job may never be initiated.

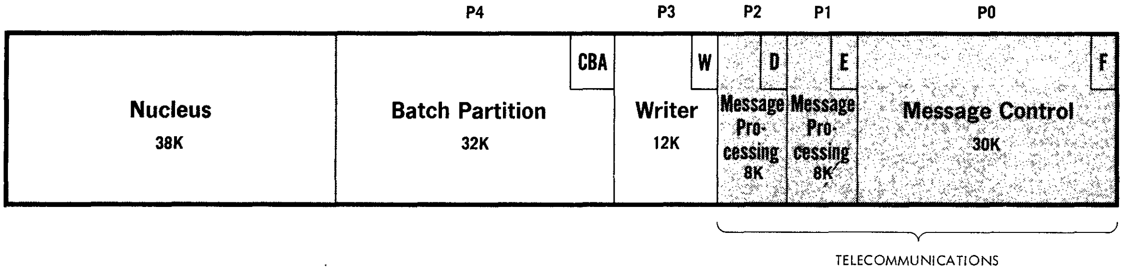


Figure 22. Sample 128K Telecommunications-Oriented Configuration

Choosing Writer Partitions

Telecommunications jobs will have no real need for resident writers. However, when batch jobs are run concurrently with telecommunications, a resident writer or a direct system output writer can be assigned to service the batch partition, as in Figure 22. Direct system output writers should not be started in a telecommunications partition.

Graphics

Graphics jobs in an MFT environment are subject to several general considerations. A graphics job associated with an unbuffered IBM 2250 Display Unit may operate with reduced performance if high telecommunications activity interferes with its access to the central processor for regenerating the display. In this case the relative importance of the graphics and telecommunications jobs must be determined, and the decision made as to which to run in the higher-priority partition. Additional considerations for MFT include assigning job classes to jobs, choosing the partition to service graphics jobs, using the time-slicing option, and assigning partitions to job classes.

Choosing Number and Size of Partitions

There must be at least one partition for each graphics job being run. The partition size depends upon the size of the graphics job. Generally, graphics jobs should be run in a high-priority partition to cause minimum interference with other jobs. Figure 23 shows a graphics-oriented system configuration. Graphics jobs are executed in P0. P3 could be used for large compilers, with P1 used as a resident writer to service P2 and P3. If telecommunications and graphics are being run in the same system, the best performance would be gained by placing the telecommunications job in a high-priority problem program partition, and the graphics job in a relatively high-priority partition also.

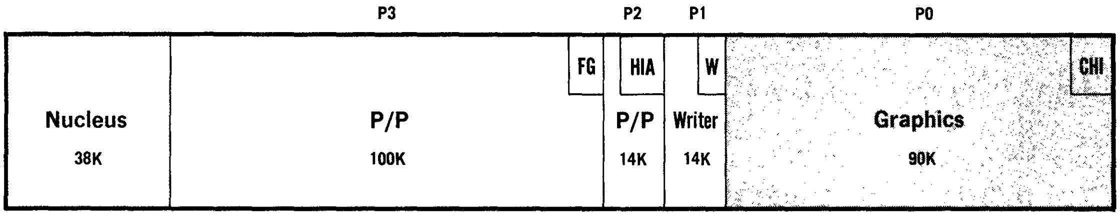


Figure 23. Sample 256K Graphics-Oriented System Configuration

Choosing Reader Partitions

In an installation running primarily graphics jobs, a resident reader should not be necessary. The system could utilize a transient reader. After the graphics job has been scheduled, the partition containing the transient reader could process batch jobs.

Assigning Job Classes to Jobs

Graphics jobs should also have a unique job class assigned to them, to ensure that they are executed in the selected partition. In Figure 23, jobs are assigned a CLASS parameter of C if they are graphics jobs.

Assigning Partitions to Job Classes

A graphics partition should be assigned a unique job class that corresponds to the job classes assigned to the graphics jobs. This ensures that jobs will be enqueued on the proper input queue, and executed in the appropriate partition. The partition could also be assigned secondary and tertiary job classes to reduce idle time. In Figure 23, P0 is assigned a primary job class of C for graphics jobs, and secondary and tertiary job classes of H and I. If the partition's job class is to be changed, and a graphics job is being run, a CANCEL command must be issued for the graphics job, and then the partition may be redefined.

Using the Time-Slicing Feature

The time-slicing feature of assigning uniform intervals of CPU time to a group of consecutive partitions is provided at system generation. (The number of time-slicing partitions and the time interval for each task are specified in the TMSLICE parameter of the CTRLPROG macro instruction. See the **System Generation** publication.) The ability to get uniform response time is useful in a graphics environment, particularly for concurrent applications involving graphics terminals. To minimize contention for the CPU with other jobs, it is best to establish the higher-priority partitions as time-slicing partitions.

Spooling (Concurrent Peripheral Operation)

Spooling (Concurrent Peripheral Operation) is the capability of performing utility functions such as card-to-tape, tape-to-print, or tape-to-punch while other jobs in the system continue processing. Execution of spooling jobs in MFT involves the same general considerations for assigning job classes to jobs and partitions as for telecommunications and graphics jobs. Spooling jobs should be assigned a class that corresponds to that of the spooling partitions. Spooling jobs can be placed anywhere in the system. Figure 24 illustrates a system configuration containing one spooling partition, one telecommunications partition, one resident reader, two resident writers, and three batch partitions. The spooling partition (P2) is assigned job class D, and no other partition is assigned this class.

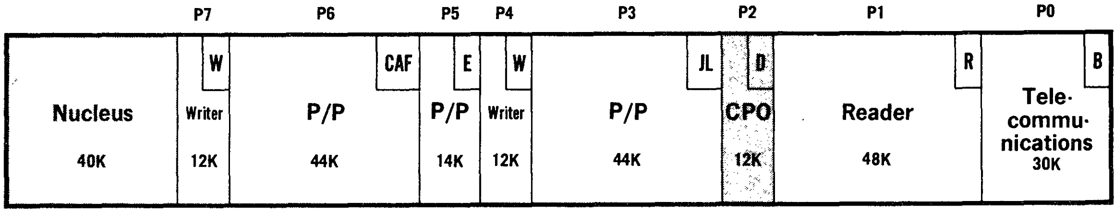


Figure 24. Sample 256K Telecommunications, Spooling, and Batch Processing Configuration

Typical System Configurations

This topic describes partition configurations for systems with 128K, 256K, and 512K bytes of main storage. These configurations are based on the considerations presented in the preceding four topics. Working configurations will depend on the individual requirements of each installation.

Systems With 128K Main Storage

A 128K system can support a variety of configurations. The configuration best for a particular installation usually depends upon the type of job most frequently run. Two examples follow.

Long-Duration Jobs

For applications where the typical job is of relatively long duration, a transient reader may be best. For such an application, where input stream processing requires a small percentage of total operating time and output requires two printers, or a printer and a punch most of the time, the configuration shown in Figure 25A might be used. This configuration has:

- Two problem program partitions, each handling batched jobs.
- Two system output writers, servicing the batch partitions.
- A transient system input reader.

This configuration permits all of the advantages of MFT to be realized, including independent partition scheduling and concurrent operation of problem programs and system output writers.

It would be advantageous to assign a single job class to each of the two problem program partitions, with one partition to be used for small batch jobs and the other partition for larger jobs. Or, jobs could be directed to one partition or the other on the basis of characteristics other than size (such as input/output device requirements).

High-Intensity Job Stream

A second configuration, shown in Figure 25B, has:

- One resident system input reader partition.
- Two system output writer partitions, servicing the problem program partition.
- One problem program partition, occupying all storage remaining available after reader and writer assignment.

This configuration should provide extremely efficient processing of a high-intensity job stream, where the typical workload involves a relatively large number of short-duration jobs. With the reader resident, input stream processing and job execution is concurrent with output writer processing.

Because a high-intensity job stream usually requires more processing time than input/output time, the characteristics of the problem programs and the reader are highly complementary, with a corresponding improvement in throughput.

Systems with 256K Main Storage

A 256K system makes possible a wider variety of configurations. Depending on the requirements of the installation, the most likely configurations will include two large (80K to 90K) batch partitions, or three to four medium-size (44K or greater) batch partitions. In either case, several system output writers could be provided to support the batch partitions. Figure 26A illustrates the first case with a configuration of two large batch partitions, a resident reader, and two resident writers. Figure 26B illustrates the second case: three medium-size batch partitions, a resident reader, and three resident writers.

	P3	P2	P1	P0
Nucleus 36K	P/P 30K	Writer 12K	P/P 38K	Writer 12K

A. Two Problem Program Partitions - Two Resident Writers

	P3	P2	P1	P0
Nucleus 36K	Writer 12K	P/P 36K	Writer 12K	Reader 32K

B. One Problem Program Partition - One Resident Reader - Two Resident Writers

Figure 25. Sample 128K Batch System Configurations

	P4	P3	P2	P1	P0
Nucleus 38K	Writer 12K	P/P 82K	Writer 12K	P/P 80K	Reader 32K

A. One Resident Reader - Two Resident Writers - Two Problem Program Partitions

	P6	P5	P4	P3	P2	P1	P0
Nucleus 40K	P/P 44K	Writer 12K	P/P 46K	Writer 12K	Writer 12K	P/P 44K	Reader 46K

B. One Resident Reader - Three Resident Writers - Three Problem Program Partitions

Figure 26. Sample 256K Batch System Configurations

Systems With 512K Main Storage

The choice of configurations available with 512K bytes of main storage is so great that no "typical" system can be defined. Such a system can support three or four 80K to 90K partitions, and whatever combination of supporting output writers and input readers is needed. Figure 27 shows a possible 512K configuration: four large (80K and 90K) batch partitions, two resident readers, two resident writers, and one small batch partition.

System With IBM 2361 Core Storage

The choice of configurations available with IBM 2361 Core Storage units frequently depend upon the size of the system's processor storage. With 128K bytes of processor storage, the partition segments in processor storage will necessarily be fewer and/or smaller than the partition segments would be in systems with 256K or 512K.

The type of processing to be done and typical job characteristics are as important to the choice of configurations as is obtaining a good partition balance. Throughput can be increased if the partition structure permits problem programs to take advantage of the IBM 2361 Core Storage environment in the following ways:

- Larger tables, work areas, and data areas can be used which should yield more direct, more efficient, and less complex programs.

- Larger program segments can be resident in processor storage, reducing or eliminating loading and linking to nonresident segments. In the problem program, the use of larger data areas can reduce the number of input/output transfers required.

- Placing part of a program in IBM 2361 Core Storage frees a corresponding amount of processor storage, thus permitting more jobs to be executed concurrently in processor storage.

Programs assembled and linkage edited to include hierarchies will execute in systems generated with or without a hierarchy structure. Conversely, programs assembled and linkage edited without hierarchies will execute in systems generated with a hierarchy structure.

Note: If the system configuration includes IBM 2361 Core Storage, caution should be taken when planning the sizes of the partitions to be included in processor storage. If the partition is generated in two segments, the segment in processor storage will determine whether the partition is treated as a small partition or as a large partition. For example, if Partition 4 is generated with a size of 24K in processor storage, and a size of 400K in IBM 2361 Core Storage, it would be treated as a small partition, thus requiring a large partition for scheduling services. If a partition is generated in only one hierarchy, it will follow the regular small/large partition guidelines.

Operating Considerations

The operator of an MFT system must be aware of several considerations related primarily to program execution, partition definition, output class reassignment restarting the system, and operator commands. If the system has the shared DASD option, the operator must also consider shared volume handling. These considerations are explained in the following paragraphs. For a complete discussion of operator requirements unique to MFT, see the **Operator's Reference** publication.

	P8	P7	P6	P5	P4	P3	P2	P1	P0
Nucleus	P/P	P/P	P/P	Reader	P/P	Writers		P/P	Reader
50K	90K	80K	8K	46K	88K	12K	12K	80K	46K

Figure 27. Sample 512K Batch Processing System Configuration

Program Execution

Because 15 problem programs can be executed concurrently, the system places additional responsibility on the MFT operator. At times he may become busy replying to system messages and problem program messages, placing special forms in the printer, etc. Therefore, whenever possible, he should perform preparatory work, such as obtaining and/or mounting required volumes, ahead of the required time. When responding to problem program messages, the operator should respond to the highest priority task first; i.e., the message from the partition with the lowest number. The operator must also remember that problem program and system messages may be intermingled on the console device.

In addition, because jobs may not be completed in the same order as they were entered in the system, the operator must ensure that the correct output is returned to each programmer.

The operator may also be required to start system input readers and output writers at certain times during operation. He may be given a specific time each day, or may have to use his judgment based on work load for the system.

Partition Definition

Even though the installation may not intend to use the maximum number of partitions at all times, the system must be regenerated if the number of partitions originally specified is increased. Therefore, the maximum number of partitions expected to be used should be specified at system generation. Partitions can then be redefined to decrease the number in use.

Caution must be observed when redefining partitions. Before redefining partitions, the operator should check the job class(es) of all pending jobs and ensure that the prospective partition definitions have job classes corresponding to the jobs that will be executed. This includes knowing the job classes of jobs which have already been placed on the input or hold queues, but have not been executed. (This can be accomplished by issuing the DISPLAY Q command.) If possible, he should also check pending jobs for their size requirement (by checking the job class versus the size of the partition assigned to service that job class) and compare this with the size of the job partitions. If they are originally assigned a CLASS parameter that corresponds only to a large partition, they should be reassigned to a large partition.

If the time-slicing feature is included in the system, the operator should not specify the same job class for both a time-sliced partition and a partition that is not time-sliced. For example, do not specify a partition with job classes B, C, D in a time-sliced group, and a partition with job classes D,E,F outside the group. Doing so would allow a job with a CLASS parameter of D to be executed either inside or outside the time-sliced group regardless of the programmer's intentions for that job. Also, a partition in a time-sliced group should not be assigned to service jobs with job classes of A, because A is the default job class, and the same problem could arise. For a complete description of how to redefine partitions, see the **Operator's Reference** publication.

After all redefinitions have been completed, message IEF805I "DEFINITION COMPLETED" is issued. The operator must enter either a START INIT command for each of the partitions that have been redefined, or a START INIT. ALL command.

Changing Output Classes

The output classes with which a writer is associated can be changed at any time, through proper use of the MODIFY command, or a combination of STOP and START commands. A program with a special forms requirement can obtain exclusive use of a printer by informing the operator to enter a MODIFY command. A STOP command followed by a START command for the same writer, but specifying a unique output class could also be entered. The STOP command causes the writer to stop at the end of the job it is currently executing. The operator then inserts the required forms and issues the new START command. That command would limit use of the printer to the data set associated with the new output class until another STOP and START command sequence for the printer is issued. The MODIFY command can also be used to change the conditions under which the output writer pauses for servicing of its device.

For example, a writer in partition 2, originally established to service output classes A, B, and C, could be changed to service only data sets for output class D by issuing the command:

```
MODIFY WTR.P2,CLASS=D
```

Handling Shared Direct Access Volumes

If the shared DASD feature is selected at system generation, additional responsibilities are imposed on the operator. Volume mounting and dismounting instructions are normally issued by the operating system. In a shared DASD environment, volume handling must be initiated by the operator and must be conducted in parallel on both sharing systems. Thus thorough operator communication from system to system must be maintained. For more information on these and other detailed operating procedures in a shared DASD environment, see the **Operator's Reference** publication.

Restarting the System

To restart the system after it has been shut down, the same steps taken in initially starting the system are followed, except when the SET command is entered. Either the "F" suffix from the "Q=unitname" parameter is omitted, or the entire "Q=unitname" parameter is omitted.

The following command illustrates this procedure:

```
SET DATE=yy.ddd,CLOCK=hh.mm.ss
```

By omitting the "Q=unitname" parameter, job queue data set information is saved. When restarting the system to save the information, the operator must make certain that all auxiliary storage volumes which were in use remain available. This insures that the job queue data set, output data sets, and input data sets accurately reflect the conditions which existed when a restart became necessary.

Operator Commands

The SHIFT command, used with the first version of MFT, is not recognized. The following commands, and their respective abbreviations, may be used in an MFT system:

CANCEL	C	RELEASE	A
CONTROL	K	REPLY	R
DEFINE	N	RESET	E
DISPLAY	D	SET	T
DUMP			
HALT	Z	START	S
HOLD	H	STOP	P
LOG	L	SWAP	G
MODE		UNLOAD	U
MODIFY	F	VARY	V
MOUNT	M	WRITELOG	W

Note: The commands are subject to the following restrictions:

- The DUMP, DEFINE, HALT, MODE, and SWAP commands are not allowed in the input stream; they must be entered through a console.
- The DUMP and MODE commands cannot be abbreviated.
- The MODE command can be used only with System/360 Model 85 and System/370 Models 135, 145, 155, and 165.
- The CONTROL command can be used only with the Model 85 Operator Console with CRT Display.

Be sure to use the correct abbreviations for operator commands. For example, at system initialization, if you inadvertently key in S for SET, the system assumes you are giving a START command. It queues the command, and waits for a SET command. The formats, functions, and parameters of the commands can be found in the **Operator's Reference** publication.

Section IV: Modifying the System

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

- Reader/Interpreter and Output Writer Cataloged Procedures
- Resident Routines Option
- Job Queue Format
- Output Separation
- Writing System output Writer Routines
- Adding SVC Routines to the Control Program
- Message Routing Exit Routines
- Handling Accounting Routines
- The Must Complete Function

Reader/Interpreter and Output Writer Cataloged Procedures

IBM supplies cataloged procedures for reader/interpreters and for output writers. You can:

- Use the IBM-supplied procedures.
- Use the IBM-supplied procedures, and override given parameters.
- Write and use your own cataloged procedures.
- Write and use your own cataloged procedure, and override given parameters.

The START command starts a reader/interpreter or an output writer, and designates the cataloged procedure to be used. If you use the START command to start a problem problem, there will be no SMF recording, or Checkpoint/Restart done for that job. Users can override given parameters in the cataloged procedure by specifying the desired paramters in the START command. (The **Operator's Guide** publication contains a complete description START command.)

An installation's parameters may differ consistently from those in the IBM-supplied procedure. If so, users can write cataloged procedures, rather than respecifying the parameters in every START command. To write cataloged procedures:

- Write the procedure in the required format.
- Add the procedure to the procedure library.
- Specify the procedure name in the START command.

To test the procedure by reference in another job but before adding it to the procedure library, format it as an in-stream procedure. See the **Job Control Language Reference** publication for a description of in-stream procedures. (In-stream procedures can be used with any reader that uses the IEFIRC reader/interpreter program or the IEFVMA ASB reader program.)

If the parameter values in a cataloged reader or writer procedure change frequently, use symbolic parameters in place of ordinary parameters. Then assign values to the symbolic parameters in the START operator command. (For a description of symbolic parameters and their use, see Appendix A: Cataloged procedures, in the **Job Control Language Reference** publication (GC 28-6704. For a description of the START operator command, see the **Operator's Reference** publication (GC28-6691).) An illustration of the use of symbolic parameters is given in this section under "Example of the Use of Symbolic Parameters."

To obtain a SYSABEND dump when a reader or writer is abnormally terminated, a DD statement describing the data set to be used must be added to the corresponding procedure. The format of the DD statement is described in this chapter under the title "Optional SYSABEND Data Set".

Reader/Interpreter Procedures

A cataloged procedure for reader/interpreters requires four job control statements: an EXEC statement and three DD statements. The names and purposes of these statements are listed below:

- An EXEC statement with the step name IEFPROC specifies the reader/interpreter program.
- A DD statement named IEFRDER provides the reader/interpreter with a description of the input stream.
- A DD statement named IEFPDSI describes the procedure library.
- A DD statement named IEFDATA defines the spooling or CPP (concurrent peripheral processing) data set that is used for intermediate storage of input stream data.

The standard reader/interpreter procedure supplied by IBM is named RDR. It specifies a block size of 80 bytes for the spooling data set. The complete standard procedure is:

```
                             Procedure: RDR
//IEFPROC     EXEC PGM=IEFIRC,REGION=48K,                           X
//                 PARM='8010300500102490501OSYSDAbbbE00001A'
//IEFRDER     DD   UNIT=244,LABEL=(,NL),VOLUME=SER=SYSIN,            X
//                 DISP=OLD,
//                 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,(500,500),RLSE,CONTIG),                 X
//                 DCB=(BLKSIZE=80,LRECL80),BUFL80),                 X
//                 BUFNO=2,RECFM=F,DSORG=PS)
```

IBM supplies three other cataloged procedures for reader/interpreters. Two provide different block size specifications for the spooling data set, and one is used to process job control statements for jobs being restarted. A procedure named RDR400 provides a block size of 400 bytes for the spooling data set. The RDR400 procedure is:

```
                             Procedure: RDR400
//IEFPROC     EXEC PGM=IEFIRC,REGION=50K,                           X
//                 PARM='8010300500102490501OSYSDAbbbE00001A'
//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)
```

A procedure for reader/interpreters named RDR3200 provides a block size of 3200 bytes for the spooling data set. The RDR3200 procedure is:

```
                             Procedure: RDR3200
//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)
```

The discussion of the special procedure used during restart follows separately under "Procedures Used by Restart."

When creating a reader/interpreter procedure, comform to the procedure format and the statement requirements. Use the IBM-supplied procedures as examples. The statement requirements are explained individually in the following paragraphs.

The EXEC Statement

The EXEC statement specifies the reader/interpreter program. It also passes a set of parameters to the reader/interpreter program. The format for the EXEC statement is:

```
//IEFPROC      EXEC      PGM=IEFIRC,                                     X
//                       PARM='bppttooommmiiicccrlssssssssaaaaefh'
```

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

PGM=IEFIRC
specifies the reader/interpreter program. The name of the program must be IEFIRC, as shown.

The PARM Field in the EXEC Statement of the Reader/Interpreter

PARM='bppttooommmiiicccrlssssssssaaaaefh'
is a set of parameters for the reader/interpreter program. This parameter field must consist of 35 characters, but the last seven have default values and need not be specified. Their meanings are explained in the following text.

b
character from 0 through 9 or A through F. Value not used by MFT but must be present.

pp
two numeric characters from 00 to 14. Indicates default priority for jobs from this input stream. Default priority is assigned to the job if not on JOB statement.

ttt
three numeric characters.

ooo
three numeric characters indicating the default for the primary number of tracks assigned for SYSOUT data sets. This primary allocation should be made sufficient for most 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. Value not used by MFT but must be present.

ccc

three numeric characters. Value not used by MFT but must be present.

r

numeric character from 0 to 3. Specifies the disposition of commands read from this input stream. Used by the reader/interpreter whether or not the command is authorized to be entered into the input stream (see the aaaa parameter). The reader/interpreter, if r is:

0 - passes the command to the Command Scheduling routine for execution.

1 - displays the command (via a WTO macro instruction). Passes it to the Command Scheduling routine for execution.

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 in the affirmative.

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

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

l

a numeric character 0 or 1 specifies the bypass label processing options. 0 signifies the BLP parameter in the label field of a DD statement to be ignored. Label parameter processed as NL. 1 signifies BLP not to be ignored. Label parameter processed as it appears.

ssssssss

eight alphameric characters specifying default device for SYSOUT. Becomes UNIT subparameter in the DD statement that defines SYSOUT (if the UNIT field is omitted from the DD statement). Must be padded to the right with blanks to full eight characters.

Note: 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 preserve some devices for private volumes, you should define a UNIT group which is a subset of the available direct access deivces. You may specify the name SYSOUT as the default unit name for the system output data sets if it was specified at system generation time; when this default is used, a unit count of 1 is implied. UNITNAME SYSOUT is fully described in the **System Generation** publication.

aaaa

for systems with the multiple console support (MCS) option: four hexadecimal numbers from 0000 to E000 indicate which operator command groups are to be executed if read from this input stream. Four blanks default to X"E000". Systems without the multiple support option: Set to X"E000", permitting all commands except DEFINE and HALT to be entered into the input stream.

Figure 28 shows the operator commands that are affected by the aaaa parameter in an MCS environment. 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: Information commands (Group 0) are always valid when entered into the input stream.

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

Figure 28. Operator Command Groups

- 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.

Bit settings for the aaaa parameter are:

Bytes	**Bits**	**Bit Settings**	**Meaning**
0 (aa)	0	1	Group 1 commands executed
	1	1	Group 2 commands 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"

ef

MSGLEVEL default value in absence of a value in the JOB statement. Job control statements and allocation/termination messages are recorded in the system output data set according to this value.

e

Kinds of job control statements recorded.

0 - JOB statements 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 ABEND condition. (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) in absence of a value in 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

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

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

This statement must be named 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 read. Can be any device supported by the queued sequential access method (QSAM). Device specified by its address, type, or group.

LABEL=(,type)
describes the data set label (needed only for **tape** data sets). If omitted, default to a standard label.

Note: Label types AL and AUL (American National Standard label types) should not be used.

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

Note: The volume serial numbers should not identify a volume that contains a data set written in ASCII.

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. If the procedure is going to be used for transient readers, the input must be unblocked 80-byte records. Other subparameter fields may be specified as needed; otherwise, the QSAM defualt attributes are assigned, as follows:

BUFNO - two buffers. (If the procedure is to used for transient readers, BUFNO=1 must be specified.)

RECFM - U-format, with no control characters.

TRTCH - odd parity, no data conversion, and no translation.

DEN - lowest density.

DSNAME=name,DISP=disposition
specifies the name and disposition of the input stream data set to be read. Should be used only with direct access input stream.

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

Note: OPTCD = Q should not be coded.

DD Statement for the Procedure Library

The reader/interpreter procedure 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 statement must be named 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, follow the IEFPDSI DD statement with other unnamed DD statements 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

The reader/interpreter procedure must include a DD statement that defines the spooling or CPP (concurrent peripherial processing) 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 spooling 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)
```

This statement must be named 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 in accordance with a reader/interpreter algorithm that is based on priorities and optimum access. If you want all the input stream data sets written on the same device, use the VOLUME parameter 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 RLSE subparameter releases all unused space to the system when the data set is closed. The CONTIG subparameter ensures that space is allocated in contiguous tracks or cylinders.

Note: The first space allocation made by the system will be for the reader/interpreter program itself, which does not need or use the space.

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 spooling 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 spooling 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.

Reader/Interpreter Procedure Used by Restart

The procedure, named IEFREINT, used to process job control statements for a job being restarted, is a skeleton of the normal reader/interpreter procedures. Its main function are to define the restart reader/interpreter program, named IEFVRRC, and to make the procedure library accessible to that program. The procedure is:

```
                 Procedure:     IEFREINT
//IEFPROC     EXEC     PGM=IEFVRRC,   RESTART READER PROGRAM          X
//                                    RESTART READER REGION           X
//                     PARM=RESTART
//IEFRDER     DD       DUMMY
//IEFPDSI     DD       DSNAME=SYS1.PROCLIB,DISP=OLD  PROCEDURE LIBRARY
//IEFDATA     DD       DUMMY
```

When creating your own restart reader/interpreter procedure, you must conform to the procedure format and the statement requirements. Use the IBM-supplied procedures as examples. The statement requirements are explained individually in the following paragraphs.

The EXEC Statement

The EXEC statement specifies the reader/interpreter program. It also passes a parameter to the reader/interpreter program. The format for the EXEC statement is:

```
//IEFPROC      EXEC       PGM=IEFVIIC,PARM=RESTART
```

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

PGM=IEFVRRC
specifies the reader/interpreter program. The name of the program must be IEFVRRC, as shown.

PARM=RESTART
must be coded as shown.

DD Statement for the Input Stream

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

```
//IEFRDER      DD    DUMMY
```

This statement must be named IEFRDER, as shown. The parameter requirements are as follows:

DUMMY
Must be coded as shown. System input is taken from the SYS1.JOBQE data set which is open already.

DD Statement for the Procedure Library

Your procedure for the restart reader/interpreter 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=OLD
```

This statement must be named 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=OLD
specifies that the procedure library is an existing data set.

DD Statement for the Spooling Data Set

The procedure for the restart reader/interpreter must include a DD statement that defines the CPP (concurrent peripheral processing) data set. Since the data is already in the checkpoint data set, DUMMY serves as operand. The format for this statement is:

```
//IEFDATA      DD    DUMMY
```

This statement must be named IEFDATA, as shown. The parameter requirement is as follows:

DUMMY
must be coded as shown.

Output Writer Procedures

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 names IEFRDER defines the output data set.

System Output Writer

The standard output writer procedure supplied by IBM is named WTR. The standard procedure is:

```
                                 Procedure:      WTR
//IEFPROC      EXEC      PGM=IEFSD080,                                 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 an output writer procedure, 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.

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,                                  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.

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 (printer) or C (punch). 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. Specify the type of output the writer can process; also establish the priority of the output classes, with the highest priority on the left. Included class name parameters in the START command override this entire set of class names in the cataloged procedure.

seprname

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

DD Statement for the OUTPUT Data Set

The 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)
//                 UCS=(code[,FOLD][,VERIFY]),                         X
                                [ ALIGN  ]
//                 FCB=(image-id[,VERIFY])                             X
```

This statement must be named 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, 3400-2, 3400-3, 3400-4, 2520, 2540, or 3211.

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). Not required for printer or card punch devices.

DSNAME=anyname

specifies a name for the output data set (tape only), so it can be referred to by subsequent job steps. 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. BLKSIZE and LRECL subparameter fields must be specified in all cases. 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 zero and one to be ignored when comparing characters between the UCS buffer and the print line buffer. This option allows lowercase character codes to be printed in uppercase by an uppercase chain/train. VERIFY causes the UCS image specfied to be output for the printer. The UCS parameter is optional and is valid only when the output device is a 3211 printer or a 1403 printer.

FCB=(image-id[,ALIGN][,VERIFY]

causes a 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. ALIGN and VERIFY each allow the operator to align forms. VERIFY also causes the FCB image to be output for the printer. The FCB parameter is optional and is valid only when the output device is a 3211 printer.

By using a certain kind of procedure, it is possible to reduce the amount of CPU time needed by the 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 3 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 other than a 3211 printer, or punch.

It should be noted that if a command chaining procedure is used to a punch, there is no automatic punch recovery even though there are more than 3 buffers.

Note: When using the UCS parameter with the START WTR command for a 1403 printer with the universal character set feature, it lasts until the completion of the writer. The UCS parameter cannot be overridden for a specific data set when using the (asynchronous) Sysout Writer.

Direct Sysout Writer -- The Synchronous System Output Writer Job

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 final output data set.

The procedure supplied by IBM is named DSO and is described in the following. To create this procedure, follow its format.

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

The EXEC Statement

The EXEC statement specifies the direct SYSOUT writer. It is also used to give the writer program necessary operating information.

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

IEFPROC
Name of the EXEC statement. Required as shown.

IEFDSO
Name of the writer program.

PARM=
Information for the IEFDSO program.

c
P for printer or C for card punch. Describes the final hard-copy medium. Must be given.

x
The SYSOUT class to be processed. Overriden to value in START command. If not stated here, must be given in the START command.

,jjjjjjj
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 started here.

If none are named here, then the job classes will be those assigned to the partition for which this writer is started.

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 message 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 explicity 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.

Optional SYSABEND Data Set

If the user desires an ABEND dump in the event that the reader, writer or initiator task is abnormally terminated, a //SYSABEND DD statement may be included in the respective procedures. It must be of the following form:

```
//SYSABEND DD SYSOUT=x
```

This statement defines the system output class for printed output if the task whose procedure contains the //SYSABEND DD statement is abnormally terminated. The "x" must be the alphabetic or numeric character that represents an output class for printed output. Any printed output class can be specified.

In addition to the SYSOUT parameter, the user may include a UNIT parameter to specify the intermediate direct access device and/or a SPACE parameter to specify the amount of intermediate direct access space required for the dump data set before it is printed. The default device type provided if the UNIT parameter is SYSDA; and the default allocation provided if the SPACE parameter is omitted is 5 tracks primary and 1 track secondary. (The default space allocation is only intended for a partial dump.)

Cataloging the Procedure

The IEBUPDTE utility program adds reader or direct sysout writer procedures to the cataloged procedure library (SYS1.PROCLIB). Use of this program is fully explained in the **Utilities** publication.

The following example shows the control statements needed to add a reader/interpreter procedure and in output writer procedure to the procedure library. For this example, the reader/interpreter procedure is named RDPROC4, and the output writer procedure is named WTPROC2.

The EXEC statement in this example specifies the IEBUPDTE program. The PARM=NEW parameter indicates that all input to the utility program is contained in the data set defined by the SYSIN statement.

The ADD control statement furnishes the name of the member to be added to the procedure library. The three numbers following the member name indicate:

- The level of modification (00 indicates first run).
- The source of the modification (0 indicates user-supplied).
- The printed output desired (ALL indicates print entire updated member and control statements).

The NUMBER statement specifies the sequence numbers for records within the new member. With this statement, the number 00000010 is assigned to the first record of the new procedure, and subsequent records are incremented by 00000010.

```
//NEWPROCS     JOB        09#770,D.P.BROWN
//             EXEC       PGM=IEBUPDTE,PARM=NEW
//SYSPRINT     DD         SYSOUT=A
//SYSUT2       DD         DSNAME=SYS1.PROCLIB,DISP=OLD
//SYSIN        DD         DATA
./             ADD        RDPROC4,LEVEL=00,SOURCE=0,LIST=ALL
./             NUMBER     NEW1=10,INCR=10
```

```
//IEFPROC        EXEC        PGM=IEFIRC                                         X
//                           PARM='80101001501024905O1SYSDA
//IEFRDER        DD          UNIT=2400-2,LABEL=( ,NL),                          X
//                           VOLUME=SER=SYSIN,                                  X
//                           DCB=( BLKSIZE=80,LRECL=80,BUFL=80,                 X
//                           BUFNO=1,RECFM=F,TRTCH=C,DEN=0 )
//IEFPDSI        DD          DSNAME=SYS1.PROCLIB,DISP=SHR
//IEFDATA        DD          UNIT=2311,                                         X
//                           SPACE=( 80,( 500,500 ),RLSE,CONTIG ),              X
//                           VOLUME=SER=222222,DISP=OLD,                        X
//                           DCB=( BLKSIZE=80,LRECL=80,BUFL=80,                 X
//                           BUFNO=2,RECFM=F )
./               ADD         WTPROC2,LEVEL=00,SOURCE=0,LIST=ALL
./               NUMBER      NEW=10,INCR=10
//IEFPROC        EXEC        PGM=IEFSD080                                       X
//                           PARM='PAC'
//IEFRDER        DD          UNIT=2400-2,LABEL=( ,NL ),VOLUME=( , , ,40 ),      X
//                           DSNAME=SYSOUT,DISP=( NEW,KEEP ),                   X
//                           DCB=( BLKSIZE=133,LRECL=133,RECFM=F,               X
//                           TRTCH=C )
/*
```

Example of the Use of Symbolic Parameters in Cataloged Reader and Writer Procedures.

Symbolic parameters in a cataloged procedure started via the START operator command may be assigned values in the START command that starts the procedure. This assigns a value to any parameter in the EXEC or in any DD statement when the procedure starts

A cataloged procedure that uses symbolic parameters may also have a PROC statement that shows the default values for the symbolic parameters. Keywords that may be used in a JOB, EXEC, or DD statement cannot be used as symbolic parameters. (For example, you cannot say that DISP is equal to & REGION.) However, subparameter keywords of the DD statement can be used as symbolic parameters. (For example, you may code BUFNO= & BUFNO.)

The following example shows a reader/interpreter procedure that contains symbolic parameters.

```
//RDPR5          PROC        REG=48,STIME=030,MCS=E000,MSGL=01,
//                           PDSI='SYS1.PROCLIB',BLK=80,BUFNO=2
//IEFPROC        EXEC        PGM=IEFIRC
//                           PARM='801&STIME.05001024905010SYSDAbbb&MCS&MSGL'
//IEFRDER        DD          UNIT=2400,LABEL=( ,NL ),VOLUME=SER=SYSIN,
//                           DCB=( BLKSIZE=80,LRECL=80,BUFL=80,
//                           BUFNO=1,RECFM=F )
//IEFPDSI        DD          DSNAME=&PDSI,DISP=SHR
//IEFDATA        DD          UNIT=SYSDA,
//                           SPACE=( 80,( 500,500 ),RLSE,CONTIG ),
//                           DCB=( BLKSIZE=&BLK,LRECL=80,BUFL=&BLK,
//                           BUFNO=&BUFNO,RECFM=F,DSORG=PS )
```

The PROC Statement

In the preceding illustration the PROC statement assigns default values to the symbolic parameters & REG, & STIME, & MCS, & MSGL, & PDSI, & BLK, & BUFNO.

The START Command

These same symbolic parameters are assigned values with the following START command:

```
START RDPR5,REG=50,STIME=035,MCS=E000,MSGL=11,PDSI='SYS1.USER'
                                              ,BLK=400,BUFNO=1
```

SYSIN and SYSOUT Data Blocking

Performance advantages can be gained by blocking SYSIN and SYSOUT data. Blocking reduces interference on the devices containing the intermediate data and improves direct access space use. The IBM-supplied reader procedures provides three levels of SYSIN blocking; review the blocking provided by the cataloged procedures of the various processors. Figure 29 shows the data blocking that is accepted by processors operating under MFT.

Blocking is obtained by including in the appropriate DD statement DCB information in the general form

DCB=(RECFM=x,LRECL=x,BLKSIZE=x)

The various programmer's guides should be consulted to determine options that need not be specified in individual cases. LRECL must be specified for the PL/I and FORTRAM H SYSLIN DD cards when these files are blocked. Assembler F and FORTRAN G and H are effectively unlimited. Sort is limited by assembled-in values. The utilities and RPG are limited by assembled-invalues of LRECL but may have a blocking factor other than 1. SYSIN and SYSOUT for the FORTRAN E compiler cannot be blocked through the system input reader and output writer, although the SYSOUT DD cards must include DCB=BLKSIZE=121.

When instituting data blocking, consider the following variables:

SIZE option

REGION values

MINPART value

Default REGION value provided by the reader procedure

The FORTRAN H SIZE parameter is independent of blocking and buffering considerations, although the REGION value must be 8K larger than the SIZE value.

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

Figure 29. Data Blocking Accepted by Processors Under MFT

Notes to Figure 29:

For compile-load-go cases, only the compile step must include complete SYSIN (SYSGO) DCB specifications.

F=Fixed, FA= Fixed, USASI control characters, FB=Fixed blocked, FBA=Fixed blocked, USASI control characters, FBM=Fixed blocked, machine control characters, VBA=Variable blocked, USASI control characters, FT=Full track, U=Undefined.

Partition sizes must be adequate to accommodate the specified blocking. The user should consult the individual programmer guides.

Blocking the Procedure Library

Blocking the procedure library may, in some cases, improve the use of direct access space and improve performance. It may be blocked at system generation or subsequently by using the operating system utilities. Block size must be a multiple of 80. Increased buffer size necessary for a blocked procedure library must be provided in the region parameter of the reader procedures for MFT. The partition size must be increased by the block size rounded to the next higher multiple of 2K.

In cases where the partition size has been increased for blocked SYSIN/SYSOUT in excess of that actuallly required (due to rounding) and the excess is greater than the block size for the procedure library, a further increase in partition size may not be necessary for processing blocked records from the procedure library.

The following example shows the control statements needed to block the procedure library using the IEBCOPY and IEHPROGM utility programs. Step C1 of job BLOCK copies the procedure library and blocks it to 400. It deletes the old copy and catalogs the new copy under the name of LIBCOPY. Step R1 renames the procedure library to SYS1. PROCLIB and catalogs it under that name.

```
//BLOCK        JOB      ACCT,D82,MSGLEVEL=1
//C1           EXEC     PGM=IEBCOPY
//SYSUT1       DD       DSNAME=SYS1.PROCLIB,UNIT2311,DISP=( OLD,DELETE
                        ,KEEP )
//SYSUT2       DD       DSNAME=LIBCOPY,UNIT=2311,VOLUME=SER=111111,      X
//                      DISP=( NEW,CATLG,DELETE ),DCB=( RECFM=FB,LRECL=80,X
//                      BLKSIZE=400 ),SPACE=( TRK,( 50,1,10 ) )
//SYSPRINT     DD       SYSOUT=A
//SYSIN        DD       DUMMY
/*
//R1           EXEC     PGM=IEHPROGM
//DD1          DD       UNIT=2311,VOLUME=SER=111111,DISP =OLD
//SYSPRINT     DD       SYSOUT=A
//SYSIN        DD       *
  RENAME DSNAME=LIBCOPY,VOL=2311=111111,NEWNAME=SYS1.PROCLIB
  CATLG DSNAME=SYS1.PROCLIB,VOL=2311=111111
/*
```

Resident Routines Options

The resident routines options are the BLDL feature, the resident reenterable modules feature, and the RSVC and RERP features. These features permit preloading into main storage routines (or at least their addresses) that otherwise would be repeatedly loaded each time the routines are requested. The Link list feature, also described in this chapter, permits references to the Link library to be extended to other data sets.

Nucleus Resident Library Routines

The BLDL, reenterable modules, RSVC and RERP options, allow placing in the nucleus area of main storage (make resident):

1. All, or a selection of, Link or SVC library directory entries.
2. A selected group of access method routines.
3. A selected group of type 3 and 4 SVC routines.
4. A selected group of error recovery procedures.
5. User-written reenterable routines from the Link library, the OS Loader, reenterable GSP routines, and the PL/1 shared library load module.

Placement occurs during the inital program load (IPL) process. The main storage area that resident routines occupy becomes part of the "fixed storage" area of the system. In effect, the nucleus is expanded.

These options are included in the system when it is generated. The **System Generation** publication describes the procedure. **The resident SVC routine option requires that the Transient SVC Table option also be included in the system.** To exercise control over the other options at IPL time, specify the operator communication facility for these options when the system is generated.

Specify the Link library (SYS1.LINKLIB) and SVC library (SYS1.SVCLIB) routines and directory entries, the access method routines, the type 3 and 4 SVC routines, and the error recovery procedures to be made resident through lists of linkage library, access method, SVC routine, and the error recovery procedures load module names placed in the parameter library (SYS1.PARMLIB).

A standard list and alternative lists of load module names may exist for the options. The standard list (so called because its member name in the parameter library is predefined) is automatically referred to during the IPL process when the operator communication facility **is not included** in the system with the options. When the operator communication facility is **included**, the operator must designate which list is to be used. IBM provides suggested standard lists for the resident access method modules and resident SVC routine options. These lists are in the starter system parameter library. Specify operator communication at system generation if you intend to use both SVC and Link library BLDL lists.

The operator communication facility provides full control over all the options at IPL time, i.e., selection of alternative or standard lists, and suppression of the options until the next IPL, or the options are in effect at every IPL, using the standard lists. The operator communication facility is required for the resident Link library modules option of MFT. Unless the operator refers to load list (or lists) for this option in his RAM= reply, none of the modules named on a load list is made resident.

This chapter discusses the function of each option, the creation of the parameter library lists, and, lists the content of the resident access method modules and resident type 3 and 4

SVC routines standard lists. The **Messages and Codes** publication describes the message (message number IEA101A) and replies associated with the options.

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 of those 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 40 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 system nuclues. 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. The 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 the 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
```

Resident Reenterable Modules Option

The resident reenterable modules options make it possible to pre-load reenterable access method module and user-written reenterable Link library modules, the OS Loader, reenterable GSP routines, and the PL/1 shared library load module. These two otherwise inpendent options -- the resident access method modules option and the resident Link library modules option -- use similarly named load lists (IEAIGG..) and share an operator reply (RAM=) at IPL time to refer to their separate lists.

The resident access methods modules option uses a list or list (named IEAIGG..) to name the modules to be preloaded and the RESIDNT=ACSMETH entry (in the system generation statements) to cause use of the pre-loaded modules when a DCB is being completed. The system comes with a standard list (IEAIGG00) which is used, unless it was changed or ask for the use of another list in the operator reply.

The resident Link library modules option uses a list or lists (also named IEAIGG.., but ending in a pair of characters other than the ones used to name the resident access methods option lists) to name the modules to be pre-loaded. The RESIDNT=RENTCODE entry (in the system generation statements) causes the pre-loaded modules to be used when a LINK, ATTACH, or XCTL macro instruction refers to the name of a resident module. Because there is no standard list, no modules are loaded unless you provide such a list.

To use both access method modules and Link library modules options, the system generation statement entry is: RESIDNT=ACSMETH,RENTCODE and the operator reply is RAM=kk,ll,mm,nn. Each pair of letters is a pair of numbers (like 01) that identify a list of either access method or user-written Link library modules and the OS Loader.

The Resident Access Method Modules Option

This option places access method load modules in the system nucleus and creates a resident list of these modules. 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 for 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 next section discusses some considerations pertaining to the use of the access method option.

ABEND and ABDUMP Requirements

The control program uses three BSAM modules (IGG019BA, IGG019BB, and the device-dependent EOB module) to provide main storage dumps during abnormal termination processing. These modules should be made resident using the RAM option. This avoids bypassing the dump facility when there is insufficient space available within the partition of the failing task into which these modules must be loaded.

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.

Users can 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 or the 2305 I/O devices in your system, add the following modules to the standard RAM list (IEAIGG00):

IGG019C4, IGG019FP, and IGG019EK

IGG019C0 must also be resident and is on the standard RAM list.

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

With MFT, the system management facilities (SMF) option requires these additional modules to be resident:

IGG019BA, IGG019BB

If the SMF recording is on tape, IGG019CC must be resident; if the SMF data set is on a direct access device, IGG019CD must be resident.

With MFT, the use of the log facility requires IGG019BA, IGG019BB, and IGG019CD to be resident. If the system log data sets are on a device with rotational position sensing (RPS), you must also make IGG019CJ and IGG019FN 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 alphameric 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. If RAM is not specified, they are loaded into the RAM area (even if the operator cancels use of the RAM option).

Resident Link Library Modules Option

This option permits preloading user-written reenterable Link library modules, the IBM-supplied OS/360 Loader, reenterable GSP routines, and the PL/1 shared library load module. The use of a LINK, ATTACH, LOAD, or XCTL macro instruction causes the contents supervision routines to find out whether the module is main storage resident already. If it is, the module already resident is used for that partition in which the macro instruction was used. If it is not, the module is loaded from the Link library into the requesting partition.

IBM-supplied modules, except those of the OS/360 Loader, GSP routines, and the PL/1 shared library load module cannot be used with this option. Any user-written routine that is reenterable may be used, for example, a user-written reader routine that is reenterable.

Adding the Resident Link Library Option

To add the option:

- Code RESIDNT=RENTCODE in your system generation statements to have the contents supervision routines find out whether the load module is already resident in main storage.
- Code OPTION=COMM in your system generation statements to allow the operator to have the modules preloaded at IPL time.
- Add to the Parameter library, a list or lists of names of reenterable modules to be preloaded. Each module name must be followed by its alias names (separated by commas).
- Have the operator specify the list or lists in his RAM= reply at IPL time.

Code RESIDNT=RENTCODE and OPTION=COMM to include certain IBM supplied coding in your system.

Name the list of reenterable Link library modules IEAIGGxx. The final two characters (xx) of the name may be any EBCDIC characters but should be different from any pair used to name a list of modules for the resident access method modules option. (The lists for the latter option are also named IEAIGG..). Add the list, or lists, to the Parameter library as described later in this section.

The modules are finally and actually preloaded if the operator includes the last two characters of the list name in his answer RAM= at IPL time. (For example, the name of the list of names of operator's reply must be of the form: RAM=..,..,aa,..) Since there is no standard list for this option, no modules are loaded unless you have constructed a list of names, added it to the Parameter library, and the operator refers to it (as described) in his RAM= response.

The Resident SVC Routines Option

This option places any of the type 3 and 4 SVC routine load modules in main storage. 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 had no assigned graphic in EBCDIC. This is b (the blank character). In BCD, 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.

Creating Parameter Library Lists

Use the IEBUPDTE utility program to construct the required lists of load module names in the parameter library. Standard member names for these lists are:

IEABLD00	for the BLDL table option
IEAIGG00	for the access method option
IEARSC00	for the SVC routine option
IEAIGE00	for the error recovery procedure option
LNKLST00	for the link library list option

These are the member names that the nucleus initialization program recognizes at IPL time in the absence of any other specification, i.e., when the operator communication facility is not incorporated.

Note: 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 nucleus.

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 identification. 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:

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

Subsequent records do not contain the library name.

SYS1.LINKLIB indicates that linkage library load module names follow.

SYS1.SVCLIB indicates that SVC library module names follow.

You may construct alternative lists for all but the LNKLST00 option and place them in the parameter library. Member names for these alternative lists are of the form:

IEABLDxx	for the BLDL option
IEAIGGxx	for the resident access method option
IEARSVxx	for the resident SVC routine option
IEAIGExx	for the resident error recovery procedure option
LNKLST00	for the link library list option

where xx can be any two alphameric characters.

Use of the alternative lists is indicated by the operator at IPL time and requires that the communication facility be present. When the communication facility is present, the operator must indicate that the standard list is to be used; that alternative lists are to be used; or that, for this IPL, the option(s) will not be used. In the latter case, no resident BLDL table, access method routines, SVC routines or error recovery procedures are placed in the nucleus.

Example

The following coding illustrates the format and content of a BLDL option list that might be used to support the resident BLDL table option. The operator, at IPL time, would have to indicate the member name, IEABLDAE to the system. The load module names listed are from the Assembler (E), Linkage Editor, and scheduler components of the operating system. Note that the module names are listed in ascending collating sequence as required for the resident BLDL option. Resident access method or SVC modules should be listed in order of anticipated frequency of use.

```
//BLDLIST EXEC PGM=IEBUPDTE,APRM=NEW
//SYSPRINT DD SYSOUT=A
//SYSUT2 DD DSNAME=SYS1.PARMLIB,DISP=OLD
//SYSIN DD *
./  ADD  NAME=IEABLDAE,LIST=ALL
./  NUMBER NEW1=01,INCR=02
SYS1.LINK GO,IEEGSTO,IEEGK1GM,IEEICIPE,IEEIC2NQ,IEEIC3JF,              X
              IEEQOT00,IEFINTQS,IEFK1|IEFSD008,IEFW21SD,IEFXA,         X
              IETASM,IETDI,IETE1,IETE2,IETE2A,IETE3,IETE3A,IETE4M,     X
              IETE4P,IETE4S,IETE5,IETE5A,IETE5E,IETE5P,IETINP,IETMAC,  X
              IETPP,IETRTA,IETRTB,IET07,IET071,IET08,IET09,IET09I,     X
              IET10,IET10B,IET21A,IET21B,IET21C,IET21D,IEWL,IEWSZOVR
./  ENDUP
/*
```

Note: During IPL the operator reply "L" may be used in conjunction with a list specification and causes the content of the list to be printed. Use this feature initially (especially with extensive lists) to easily identify format errors, e.g., a 9 character name, or incorrect name specifications.

Example of the ERP Option List

The following coding illustrates the format and content of an ERP option list that may be used to support the resident ERP option. The operator, at IPL time, would have to indicate the member name, IEAIGE01, to the system. The load module names listed are the optical reader ERPs, write-to-operator, statistics update, I/O purge, OBR and SDR/CCR modules. Note that the standard resident ERP list (IEAIGE00) contains no error routine member names and that IEAIGE01 is an alternate list.

```
//ERPLIST EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD SYSOUT=A
//SYSUT2 DD DSNAME=SYS1.PARMLIB,DISP=OLD
//SYSIN DD *
./       ADD    NAME=IEAIGE01,LIST=AL
./       NUMBER NEW1=01,INCR=02
SYS1.SVCLIB   IGE0011B,IGE0011C,IGE0011D,                              X
              IGE0025C,IGE0125C,IGE0225C,                              X
              IGE0025D,IGE0025E,IGE0025F,                              X
              IGE0125F,IGE0525F
./  ENDUP
/*
```

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. In MFT, the operator will receive a message if there is insufficient space for a reader/interpeter. He may wait for space or cancel the reader. **An initiator, however, must have sufficient queue records available to complete the initiation and running of a job or the job is canceled.** 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.
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 unpredictable, 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) x 2 + 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 exit, 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 spacc, 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 thsm 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 messages -- 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 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'soutput.

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 in the beginning of 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 size requirement of the output writer. If the job falls into a job class 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, 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 the 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 you may 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 MFT, the routine is linked (via the LINK macro instruction) when a data set requiring the routine is to be processed. The standard linkage conventions for linking 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 30 describes the information in this parameter list.

	Output Device	Indicator.
Byte 0	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	
Byte 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 30. 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 30, 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 **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 topic provides a general description of the procedures followed by the standard writer. (See Figure 31.) When writing a writer routine, delete, modify, or add items to some of these procedures, depending on the characteristics 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 C.

   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 agree with the length of the output buffer, the standard writer makes the proper adjustment.

3. The third aspect to consider is an 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.

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.

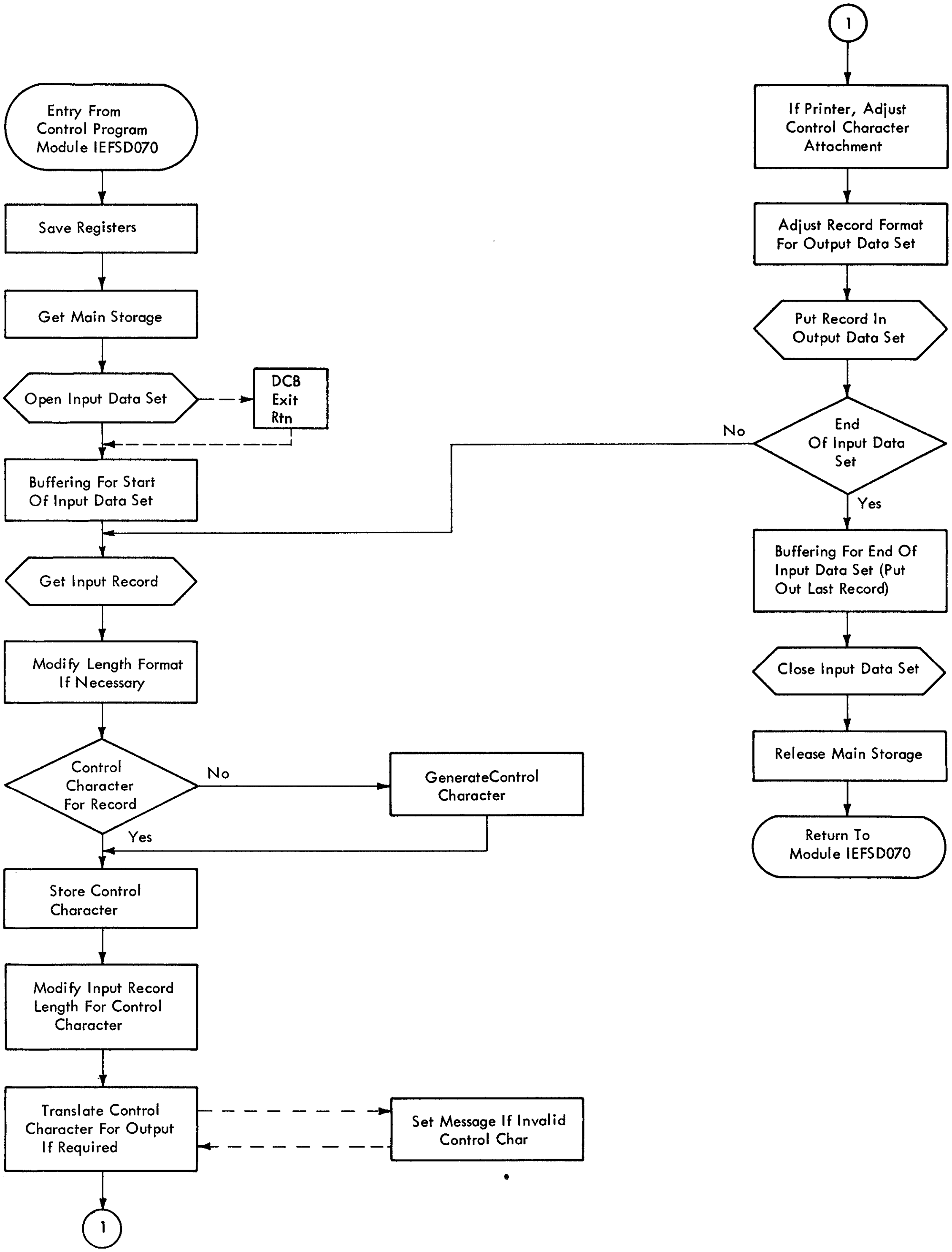

Entry From Control Program Module IEFSD070
Save Registers
Get Main Storage
Open Input Data Set
DCB Exit Rtn
Buffering For Start Of Input Data Set
Get Input Record
Modify Length Format If Necessary
Control Character For Record
No
Yes
GenerateControl Character
Store Control Character
Modify Input Record Length For Control Character
Translate Control Character For Output If Required
Set Message If Invalid Control Char
1
1
If Printer, Adjust Control Character Attachment
Adjust Record Format For Output Data Set
Put Record In Output Data Set
End Of Input Data Set
No
Yes
Buffering For End Of Input Data Set (Put Out Last Record)
Close Input Data Set
Release Main Storage
Return To Module IEFSD070

Figure 31. General Logic of Standard Output Writer

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 32. Details about many of the conventions are in the reference notes that follow the figure. The notes are referred to by the numbers in the last column of the figure. 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 32. 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	Following 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 (SVCB), 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.

The 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 routines 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 have been described.

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 MFT 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 a 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 33. Details about many of the conventions are in the reference notes that follow that figure. The notes are referred to by the numbers in the last column of the figure.

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 33. Programming Conventions for WTO/WTOR Routing

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

Reference Code	Reference Notes

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 (including 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 necessary 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 progarm. 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 routines must be serially reusable.

CSECT Name and Entry Point: The control section containing the entry point of the accounting routine, and the entry point, must be named IEFACTRT.

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.

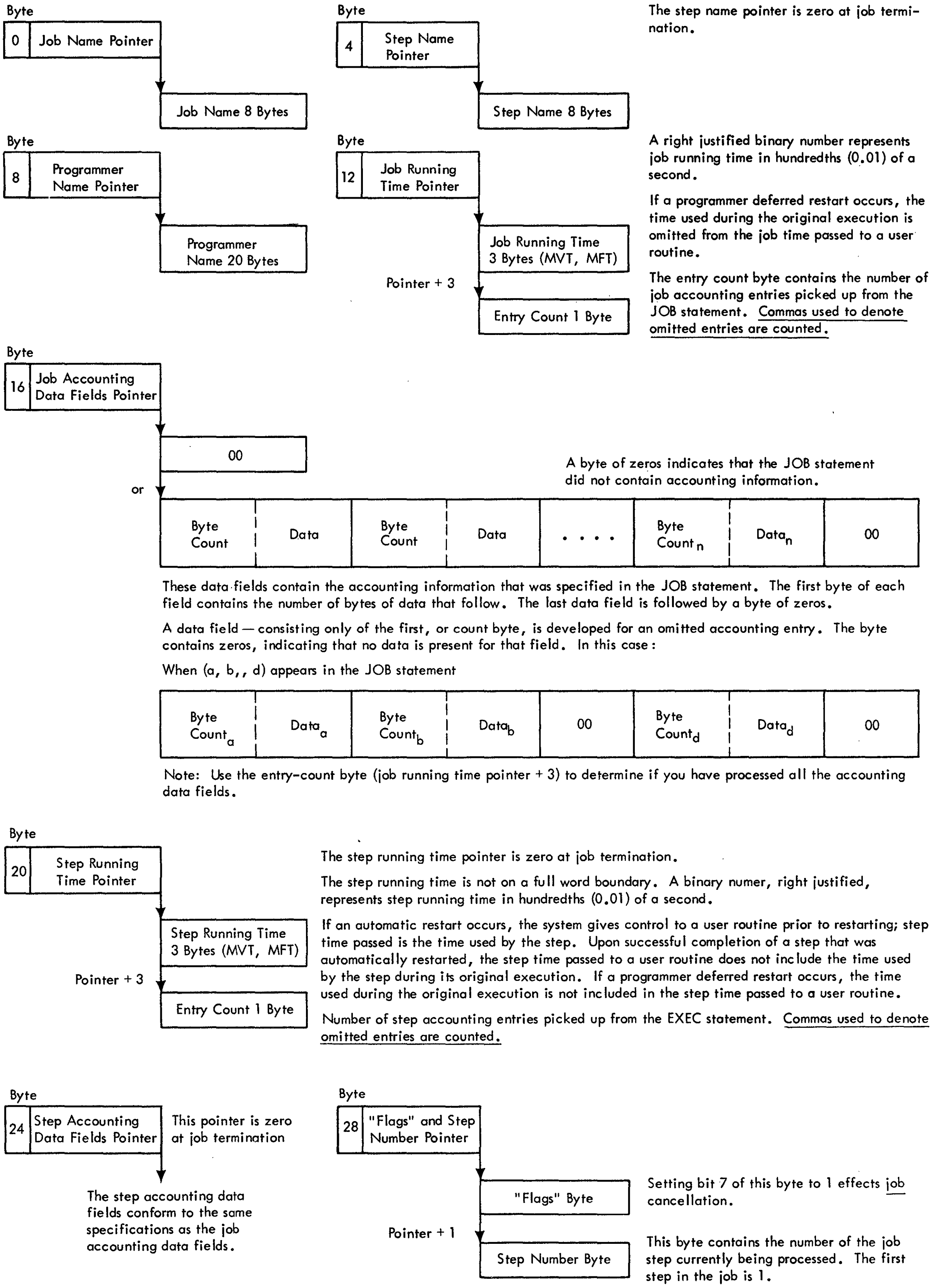
Register 0 = 8: Step Initiation

= 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.


Byte
0 Job Name Pointer
Job Name 8 Bytes
Byte
4 Step Name Pointer
Step Name 8 Bytes
Byte
8 Programmer Name Pointer
Programmer Name 20 Bytes
Byte
12 Job Running Time Pointer
Job Running Time 3 Bytes (MVT, MFT)
Pointer + 3
Entry Count 1 Byte
Byte
16 Job Accounting Data Fields Pointer
00
or
Byte
20 Step Running Time Pointer
Step Running Time 3 Bytes (MVT, MFT)
Pointer + 3
Entry Count 1 Byte
Byte
24 Step Accounting Data Fields Pointer
Byte
28 "Flags" and Step Number Pointer
"Flags" Byte
Pointer + 1
Step Number Byte

The step name pointer is zero at job termination.

A right justified binary number represents job running time in hundredths (0.01) of a second.

If a programmer deferred restart occurs, the time used during the original execution is omitted from the job time passed to a user routine.

The entry count byte contains the number of job accounting entries picked up from the JOB statement. Commas used to denote omitted entries are counted.

A byte of zeros indicates that the JOB statement did not contain accounting information.

Byte Count	Data	Byte Count	Data	. . . .	Byte Count$_n$	Data$_n$	00

These data fields contain the accounting information that was specified in the JOB statement. The first byte of each field contains the number of bytes of data that follow. The last data field is followed by a byte of zeros.

A data field — consisting only of the first, or count byte, is developed for an omitted accounting entry. The byte contains zeros, indicating that no data is present for that field. In this case:

When (a, b,, d) appears in the JOB statement

Byte Count$_a$	Data$_a$	Byte Count$_b$	Data$_b$	00	Byte Count$_d$	Data$_d$	00

Note: Use the entry-count byte (job running time pointer + 3) to determine if you have processed all the accounting data fields.

The step running time pointer is zero at job termination.

The step running time is not on a full word boundary. A binary numer, right justified, represents step running time in hundredths (0.01) of a second.

If an automatic restart occurs, the system gives control to a user routine prior to restarting; step time passed is the time used by the step. Upon successful completion of a step that was automatically restarted, the step time passed to a user routine does not include the time used by the step during its original execution. If a programmer deferred restart occurs, the time used during the original execution is not included in the step time passed to a user routine.

Number of step accounting entries picked up from the EXEC statement. Commas used to denote omitted entries are counted.

This pointer is zero at job termination

The step accounting data fields conform to the same specifications as the job accounting data fields.

Setting bit 7 of this byte to 1 effects job cancellation.

This byte contains the number of the job step currently being processed. The first step in the job is 1.

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 34. 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, place the routine in the SYS1.CI505 data set, prior to the start of the system generation, using the linkage editor. 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 apporpriate 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 porgram, 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 MFT schedulers (30K, 44K) follows:

MFT Configurations

30K Scheduler:

- Load module IEFSD520 -- step initiation
- Load module IEFSD515 -- step/job termination
- Load module IEFZA -- step initiation

44K Scheduler:

- Load module IEFW21SD -- step initiation
- Load module IEFSD510 -- step/job termination

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     EXEC PGM=IEWL, (parameters)
//SYSPRINT     DD   SYSOUT=A
//SYSUT1       DD   UNIT=SYSDA,SPACE=(parameters)
//SYSLMOD      DD   DSNAME=SYS1.LINKLIB,DISP=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 (WTOR) macro instruction. Write-to-programmer (WTO with routing code 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 36 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.

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.

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 H'____'-- 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 condidtion 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.

The Must Complete Function

System routines (routines operating under a storage protection 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 partition, only the requesting task is allowed to execute. All other tasks in the partition, 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 compplete 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.

```
Volume Serial       , ▲ , ▲ ,  Device Type    ,▲  ▲ Optional
Number 6 Bytes        │   │    4 Bytes         │  │ Information
                      │   │                    │  └────Blank--1 Byte
                      │   │                    └─Mount Priority--1 Byte
                      │   └─────Allocation Characteristic--1 Byte
                      └────Mount Characteristic--1 Byte
```

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

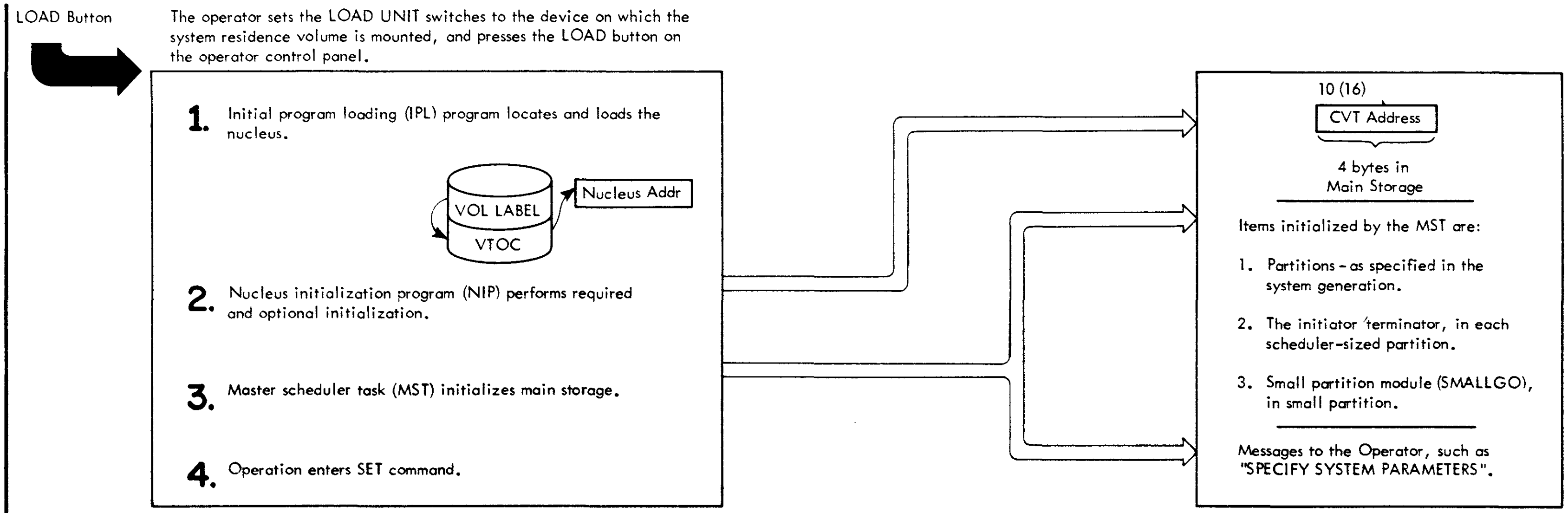
The IEBUPDTE utility program places the list (under the member name PRESRES) in the system parameter library, SYS1.PARMLIB. This utility is also used to maintain the list.

Section V: Logic Summary

This section contains two parts. Part I describes the overall logic of the MFT control program with diagrams. This part gives special emphasis to the master scheduler, communications task, readers, writers, and the supervisor. Part II describes the initialization of the control program, showing how main storage is segmented. Planning personnel and others who need to know the logic and learn the internal processing of the control program can look at this section.

Part I: Theory of Operation

Figures 35 through 39 describe the overall processing flow through each job cycle. These figures describe the processing performed by various components of the control program as it loads the nucleus, reads control statements, initiates the job step, causes processing to begin or end in other partitions, and terminates the job step.



The operator sets the LOAD UNIT switches to the device on which the system residence volume is mounted, and presses the LOAD button on the operator control panel.

1. Initial program loading (IPL) program locates and loads the nucleus.

2. Nucleus initialization program (NIP) performs required and optional initialization.

3. Master scheduler task (MST) initializes main storage.

4. Operation enters SET command.

Items initialized by the MST are:

1. Partitions - as specified in the system generation.
2. The initiator/terminator, in each scheduler-sized partition.
3. Small partition module (SMALLGO), in small partition.

Messages to the Operator, such as "SPECIFY SYSTEM PARAMETERS".

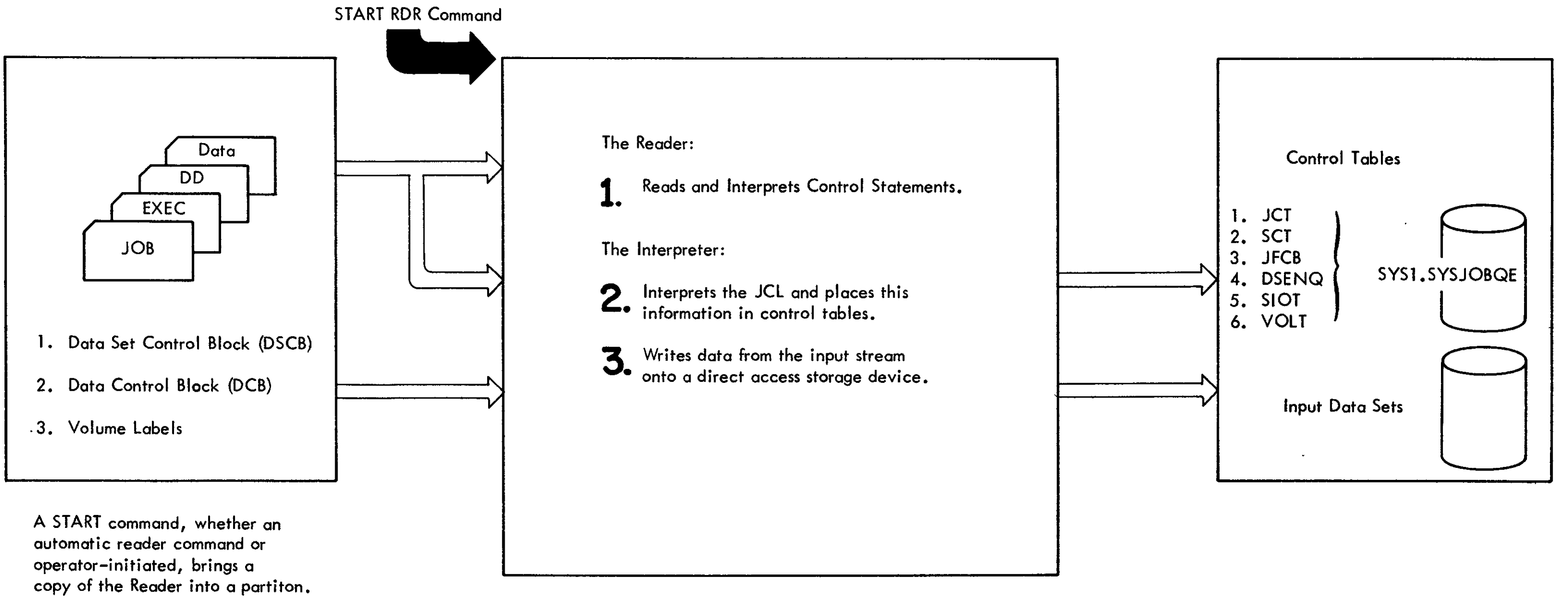
1. An IPL record is read and given control. This record causes the second IPL record to be read, which in turn, enables the rest of the IPL program to be read into main storage. The IPL program searches the volume label of the system residence volume to locate the volume table of contents (VTOC). The VTOC is then searched for the address of the nuclues data set (SYS1.NUCLEUS).

2. The nucleus is brought into the system area, and NIP is brought into the dynamic area. NIP receives control from the IPL program. It performs both required and optional initialization for control program operation, including initializing the communication vector table (CVT) and general system initialization, such as determining user options and opening system data sets. After completing its processing, NIP passes control to the master scheduler task (MST), which initializes main storage.

3. The MST gives control to the communications task, which communicates with the operator to request partition changes. The MST then initializes partitions by placing a copy of the initiator/terminator into each scheduler-sized partition; a copy of the small partition (SMALLGO) is placed in each small partition.

4. When the required SET command is entered, the communications task calls the master scheduler command scheduling routine to have the command executed. An automatic START reader command or a subsequent operator entered START reader command causes a copy of the reader/interpreter (reader) to be brought into its appropriate partition. If a START writer command is entered, a copy of a writer is also brought into the specified partition(s).

Figure 35. System Initialization


START RDR Command
Data
DD
EXEC
JOB
1. Data Set Control Block (DSCB)
2. Data Control Block (DCB)
3. Volume Labels
The Reader:
1. Reads and Interprets Control Statements.
The Interpreter:
2. Interprets the JCL and places this information in control tables.
3. Writes data from the input stream onto a direct access storage device.
Control Tables
1. JCT
2. SCT
3. JFCB
4. DSENQ
5. SIOT
6. VOLT
SYS1.SYSJOBQE
Input Data Sets

A START command, whether an automatic reader command or operator-initiated, brings a copy of the Reader into a partiton.

1. The reader gets control and reads control statements and data from the input job stream. It passes the control statements and data to the interpreter.

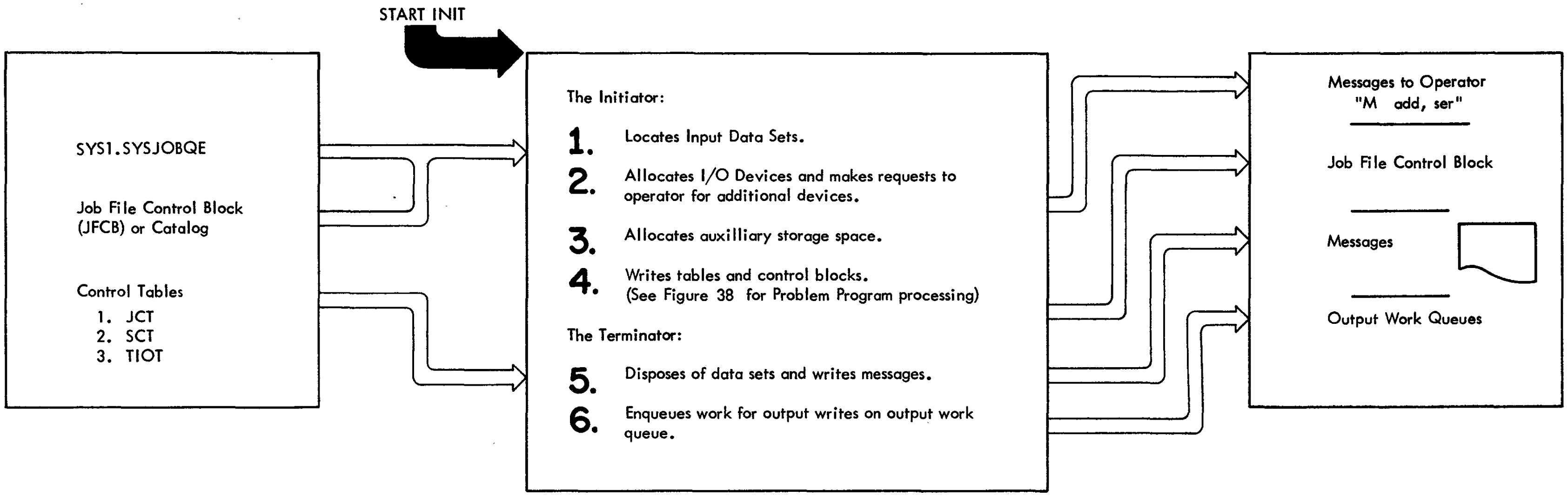
2. The interpreter analyzes the JOB, EXEC, and DD statements and places this information in the following tables:

- A job control table (JCT) for each job being read.
- A step control table (SCT) for each step being read. This is chained to the JCT.
- A job file control block (JFCB) and step I/O table (SIOT) for each DD statement. These are chained to the SCT.
- A data set enqueue table (DSENQ) is made from every JFCB used by the job.
- A volume table (VOLT) for each volume serial number used by the job.

3. The interpreter then places these updated control blocks into the SYS1.SYSJOBQE corresponding to the CLASS and PRTY parameters on the JOB statement.

Data sets in the input stream are written onto a direct access storage device for later use by the problem program. The address of the data is placed in the JFCB of the SYSIN DD statement.

Figure 36. Reader/Interpreter


START INIT
SYS1.SYSJOBQE
Job File Control Block (JFCB) or Catalog
Control Tables
1. JCT
2. SCT
3. TIOT
The Initiator:
1. Locates Input Data Sets.
2. Allocates I/O Devices and makes requests to operator for additional devices.
3. Allocates auxilliary storage space.
4. Writes tables and control blocks. (See Figure 38 for Problem Program processing)
The Terminator:
5. Disposes of data sets and writes messages.
6. Enqueues work for output writes on output work queue.
Messages to Operator "M add, ser"
Job File Control Block
Messages
Output Work Queues

1. The allocation routine, running as a subroutine of the initiator, determines the volume containing a given input data set (SYSIN) by examining the JFCB, or by searching the catalog. A catalog management routine is used to perform this search.

2. A job step cannot be initiated unless there are enough I/O devices to fill its needs. Allocation determines whether the required I/O devices are available, and makes specific assignments. If necessary, messages are issued to the operator to request the mounting of volumes.

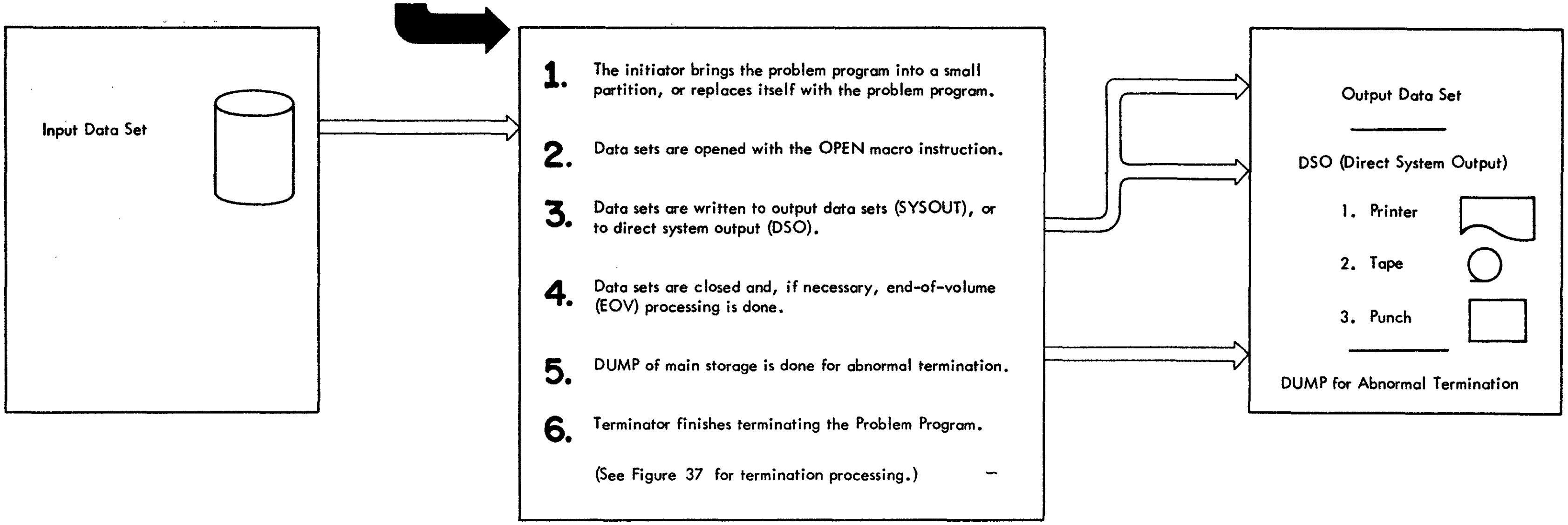
3. Working under the control of the allocation routine, direct access device space management (DADSM) routines acquire direct access volume space needed for output data sets, new data sets, and work data sets for jobs not using direct system output (DSO).

4. The JFCBs are updated by the open routines with information concerning the data sets to be used by step execution, and are written back into SYS1.SYSJOBQE. This information is used later when a data set is closed, when EOV conditions occur, and when the job step is terminated.

5. The terminator releases the I/O devices, and disposes of data sets used by the job step. The tables prepared during initiation (JCT, SCT, TIOT, etc.) contain the information used to release the I/O devices. These tables include information such as disposition of data sets. It then executes an installation accounting routines if one is provided.

6. At termination of a job not using direct system output processing, an entry is made on the user-specified output work queue; later the problem program output data can be written from a system direct access storage device to a user-specified by a system output writer. The initiator then initiates the next job step.

Figure 37. Initiator/Terminator



1. The initiator brings the problem program into a small partition, or replaces itself with the problem program.

2. Data sets are opened with the OPEN macro instruction.

3. Data sets are written to output data sets (SYSOUT), or to direct system output (DSO).

4. Data sets are closed and, if necessary, end-of-volume (EOV) processing is done.

5. DUMP of main storage is done for abnormal termination.

6. Terminator finishes terminating the Problem Program.

(See Figure 37 for termination processing.)

1. The initiator causes itself to be replaced by the problem program it is initiating (if the problem program is a large partition), or initiates the job in a small partition. The problem program can be an IBM-supplied processor, such as COBOL, linkage editor, or a user-written program. The problem program uses control program services for operations such as loading other programs and performing I/O operations.

2. The OPEN macro instruction gives control to the OPEN/CLOSE/EOV routines. These routines open the data set's DCB and bring the access method routines into the problem program partition. The problem program now processes until it terminates normally or abnormally, though it may not retain exclusive control of the CPU. Control always is received by the highest priority task ready to execute.

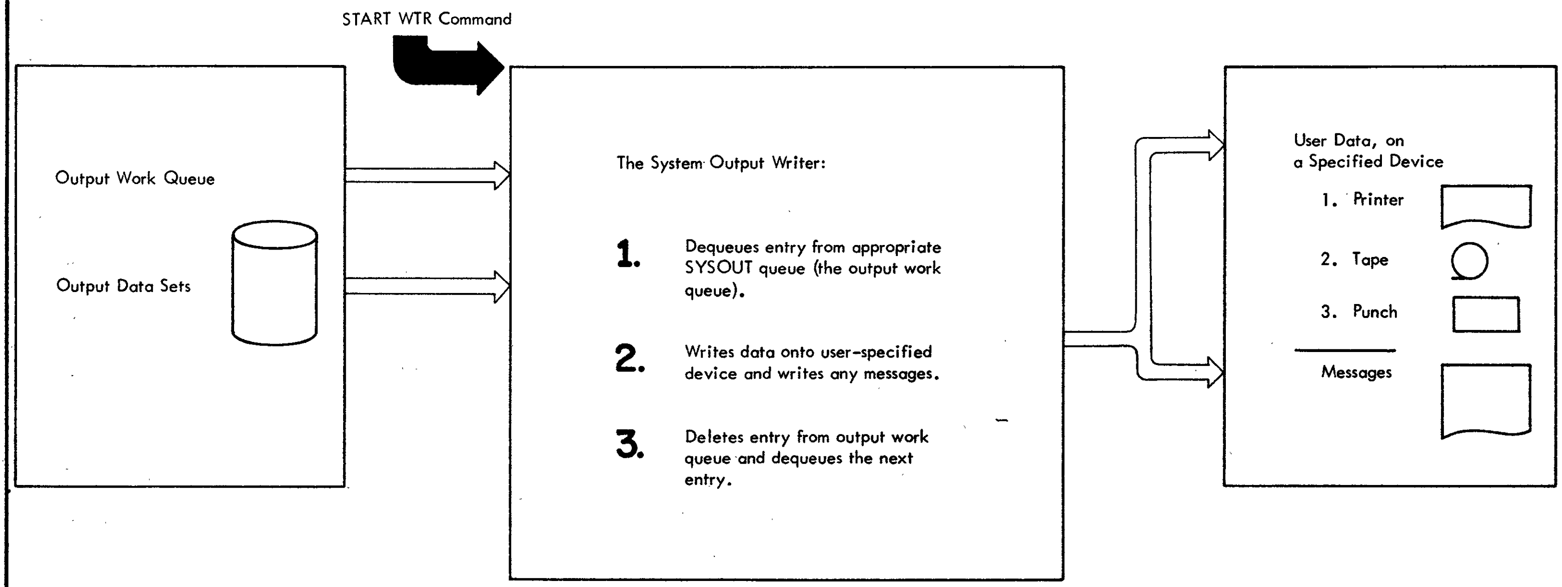
3. Output is written to the output data set (SYSOUT) or to the direct system output (DSO) data sets.

4. When the problem program terminates, the supervisor receives control. The supervisor uses the OPEN/CLOSE/EOV routines to close any open data control blocks.

5. Under abnormal termination conditions, the supervisor may also provide special termination procedures, such as a storage dump if a SYSABEND DD statement is in the input stream.

6. The supervisor then passes control to the terminator, which is either brought into the partition in which termination is to occur, or is brought into a large partition to terminate a small partition.

Figure 38. Processing a Problem Program


START WTR Command
Output Work Queue
Output Data Sets
The System Output Writer:
1. Dequeues entry from appropriate SYSOUT queue (the output work queue).
2. Writes data onto user-specified device and writes any messages.
3. Deletes entry from output work queue and dequeues the next entry.
User Data, on a Specified Device
1. Printer
2. Tape
3. Punch
Messages

1. An output writer operates concurrently with readers, problem programs, and other writers. When the START command is issued for a writer, the writer dequeues the first entry in the specified output (SYSOUT) queue.

2. If no requests have been enqueued in that output queue from the problem programs, the writer is placed in a wait condition until a job is terminated that has system messages or output data sets. After the entry is dequeued from the output queue, the writer transmits the data sets to the specified card punch, magnetic tape unit, or printer.

3. When the last record has been processed, the writer deletes the queue entry before dequeuing the next entry.

Figure 39. System Output Writer

Part II: Initialization of the Operating System

When the system is loaded, routines perform required and optional initialization of functions needed for control program operation. When the Nucleus Initialization Program (NIP) has defined the fixed area, it then assigns the rest of main storage to the master scheduler task to be prepared as the dynamic area for control program operation.

Main Storage Preparation

When NIP completes its functions it constructs a request block (RB) and an XCTL macro instruction (specifying master scheduler initialization routine IEFSD569) at the low address of the temporary master scheduler area. NIP places the address of this RB in master scheduler task TCB field TCBRBP. (The original contents of TCBRBP are saved and passed to IEFSD569 in a parameter list along with the original master scheduler task boundary box contents.) NIP sets master scheduler task TCB field TCBFLGS to make the master scheduler task dispatchable, and then branches to the dispatcher.

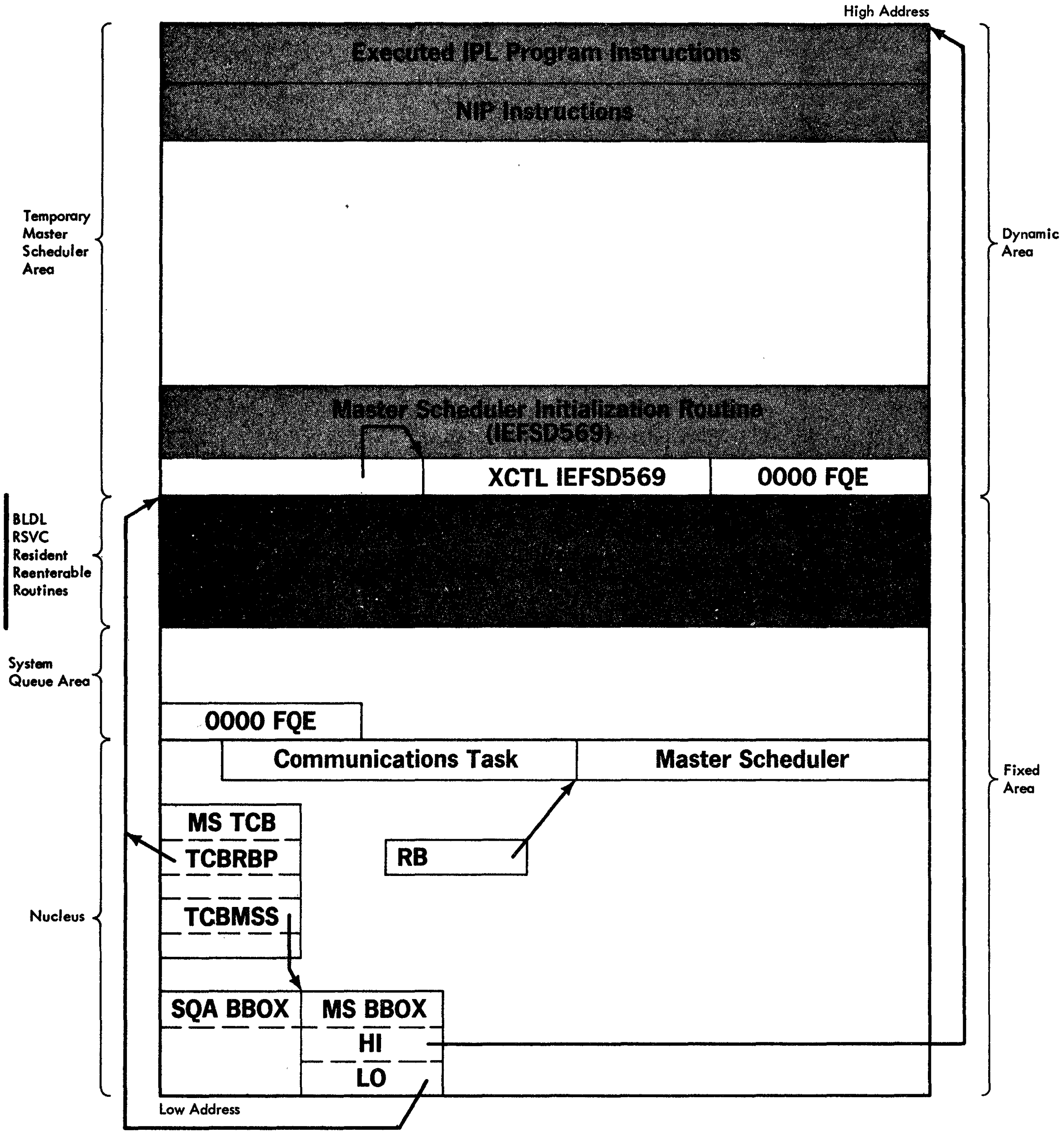
The dispatcher gives control to the master scheduler task causing execution of the XCTL instruction which NIP placed in the temporary master scheduler area. The master scheduler initialization routine is brought into the temporary master scheduler area and begins executing. Figure 40, excluding the medium shaded area, illustrates main storage at completion of NIP before branching to the dispatcher. Figure 40, excluding the light shaded area, illustrates main storage when the master scheduler initialization routine receives control from the dispatcher.

Figure 41 illustrates main storage (four partition example) at completion of master scheduler initialization. When the initialization routine completes processing, it branches to the dispatcher.

Initializing the Partitions

During master scheduler initialization the operator must accept automatic START commands or enter START commands manually. When a START command is processed, the partition number specified in the command is determined, and a command scheduling control block (CSCB) is built. The CSCB is used for communication between the command scheduling routines (SVC 34) and the command execution routines. The address of the CSCB is placed in the partition information block (PIB) of the specified partition, and the partition is posted. The PIB for each partition contains information used by command processing and scheduler routines.

After the initialization routine completes processing, the dispatcher gives control to the master scheduler router routine. When this routine completes processing, it returns to the dispatcher which begins searching the TCB queue. The highest priority task posted through START command processing receives control. The XCTL macro instruction addressed by the partition's RB is executed and the Job Select module (IEFSD510) or Small Partition module (IEFSD599) is brought into the partition. When an interruption occurs and the partition can no longer retain control, the dispatcher gives control to the next posted partition. This process continues, enabling all posted partitions to receive control and to execute the XCTL instruction placed in them by the initialization routine.



Legend:

▨ Contents of the Dynamic Area During IPL and NIP.

▨ Contents of the Dynamic Area After The Master Scheduler Task Receives Control on Completion of NIP.

■ Optional Features

Figure 40. Main Storage During Execution of NIP

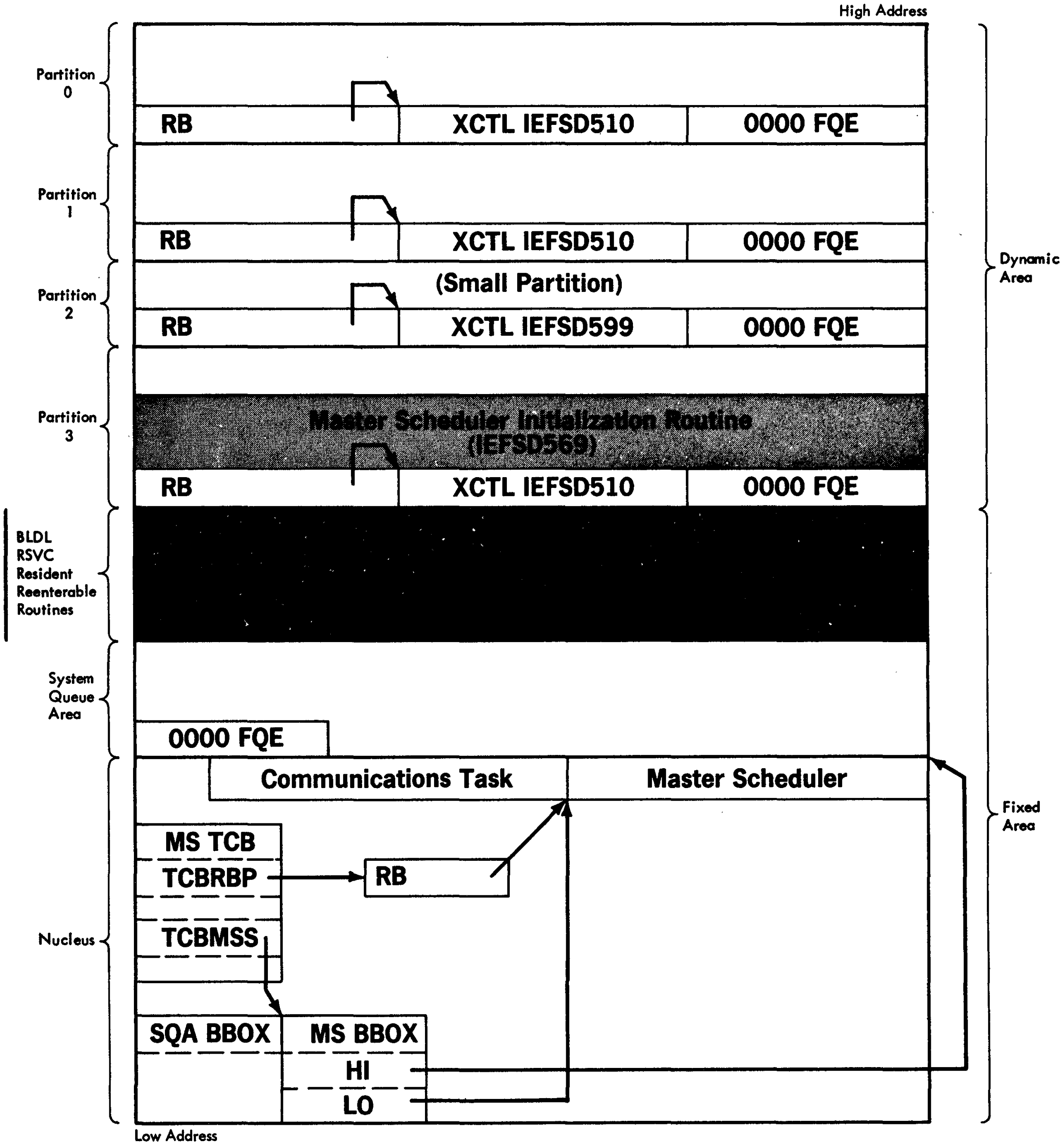


Figure 41. Main Storage at Termination of Master Scheduler Initialization

Appendix A: 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 utility program, described in the **Utilities** publication.

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 MFT.

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 **MFT 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, and 75 (and with MVT, for certain other models, not supported by MFT). 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 **MFT Supervisor PLM.**

Machine-Check Handler

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 Models 65 and 85, and for System/370 Models 135, 145, 155, and 165.

MCH for Model 65

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.

MCH is described in more detail in the **Machine-Check Handler for System/360 Model 65 PLM.**

MCH for Models 85, 135, 145, 155, and 165

MCH programs for Models 85, 135, 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 Models 135 and 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 Models 135 and 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 APR and DDR are optional. CCH is required or optional, depending on the CPU model.

When a system does not include CCH, channel checks are handled by the sysytem'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 135, 145 and 155. CCH is required for System/360 Model 85 and for System/370 Models 145, 155, and 165; it is optional for System/360 Models 65 and 75. (With MVT, CCH is required or optional for certain other models, which are not supported by

MFT.) 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, Models 135 and 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.

For APR to be effective, however, 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 for MFT. The VARY PATH function of APR is standard for MFT.

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 intermittant 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.

DDR is optional for all systems supported by MFT. 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 the supervisor FINCH routine or the resident DASD error recovery procedure. For other devices (tape and direct access), DDR receives control from the outboard recorder (OBR) or statistical data recorder (SDR) 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.**

Appendix B: System Macro Instructions

This chapter contains the description and formats of macro instructions that allow you either to modify control blocks or to obtain information from control blocks and system tables.

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 }  ,{  }  [,PARAM=list address]
                                     {CT}
                  [       {YES}]  [,PURGE= {QUIESCE}]  [,ASYNCH= {NO }]
                  [,XCTL= {   }]  [        {HALT   }]  [         {YES}]
                  [       {NO }]  [        {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 exit 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 been 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.
12	No work area was obtained.
16	No I/O processing was requested.

- 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 SVC 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 }
```

Ordinary ATTACH macro instruction parameters. See the description in the **Supervisor Services and Macro Instruction** publication (GC28-6646).

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 - This option is not available in MFT. The save area will be provided if SVAREA=NO is specified.

,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.

Inter-Partition POST -- Post a Nonresident Routine

The inter-partition POST macro instruction is primarily for MVT systems with TSO. (In those systems, the macro provides a way for programs to issue POST requests for TSO tasks that are currently swapped out of main storage.) However, you may use the list and execute forms of the macro in MFT systems also, in addition to the standard form of the regular POST macro. The MFT systems simply ignore the TJID and the TCB operands that are outlined below.

The **Supervisor Services and Macro Instructions** manual outlines the use of the standard form of the regular POST macro instruction, and tells generally how to use the list and execute forms of macro instructions. An outline for the use of the list and execute forms of the inter-partition POST macro appears below.

List Form of the Inter-Partition POST Macro Instruction

The list form of the inter-partition POST macro instruction constructs a parameter list that can be passed to the control program.

```
Name     Operation Operand

[symbol] POST      [ecb address] [,TJID=address]
                   ,TCB=0
                   ,TCB=address     ,MF=L
```

ecb address

The address of the ECB representing the event to be posted.

MF=L

Indicates the list form of this macro instruction.

Note: This list form is valid only for inter-partition POST requests; also, you must use a-type address constants for all address operands that you specify.

Execute Form of the Inter-Partition POST Macro Instruction

The execute form of the inter-partition POST macro instruction allows the user to issue inter-partition POST macro instructions for a nonresident control program routine by using a parameter list. The list form of the macro constructs the parameter list.

```
Name     Operation Operand

[symbol] POST      ,TJID=address
                   ,TJID=address,TCB=0
                   ,TJID=address,TCB=address
                   ,MF=(E,control program list address)
                   ,MF=(E,(1))
```

MF=(E,control program list address)

Indicates the execute form of the macro instruction, and specifies the address of the remote parameter list specified for the control program by the list form of the inter-partition POST macro instruction. You may load this address into any of the general registers, 2-12.

MF=(E,(1))
Indicates the execute form of the macro instruction, and specifies that register 1 must contain the address of the parameter list.

Note: The execute form of the inter-partition macro instruction is valid only for inter-partition POST requests.

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
specifies 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 C: 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 Table 41. 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.

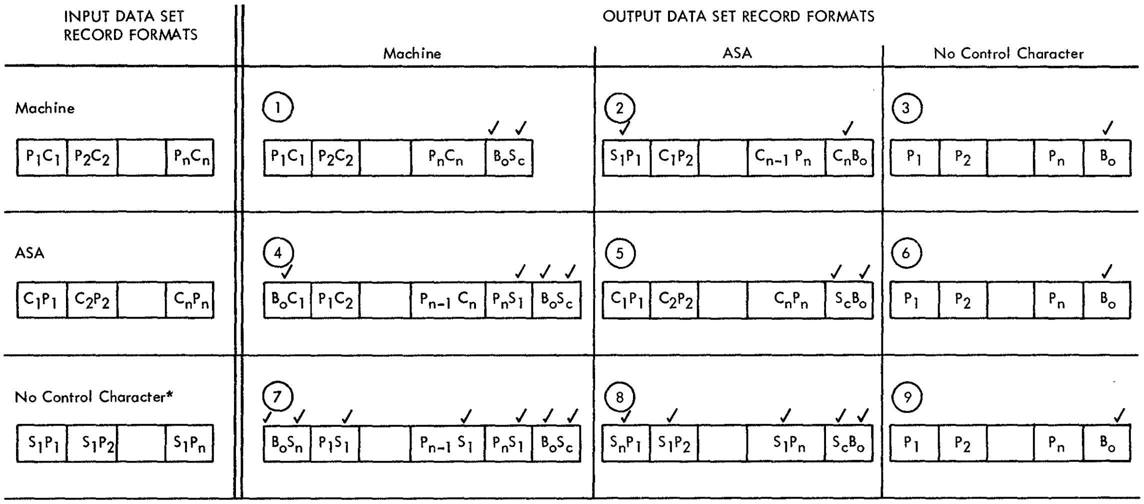
	Machine Control Characters			
	3525/			USASI
Stacker Unit	2540 Punch	2520 Punch	1442 Punch	Control Characters
1. P1	00000001	00000001	10000001	11100101 (V)
2. P1 Column Binary	00100001	00100001	10100001	
3. P2	01000001	01000001	11000001	11100110 (W)
4. P2 Column Binary	01100001	01100001	11100001	
5. RP3	10000001			
6. RP3 Column Binary	10100001			

Figure 42. 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 43. 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 43, 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.



= 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 43. Symbolic Representation of Record Formats

In cases 2 and 4 of Figure 43, 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.

The standard writer translation of printer-type control characters is given in Figure 44. In this table, 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 table.

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 44. 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 D: RESERVE Macro Instruction Used with the 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┘
                                      ┌     ┌TEST┐┐
         [rname length],SYSTEMS)      │,RET=┤USE ├│,UCB=pointer address
                                      └     └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.

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. The job or jobs will be canceled. 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, therfore 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 whcih 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 convert addresses into 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 prime 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
MCVERNAM MVC   RNAME+1(0),0(6)
ADDRTIOT DC    F'0'
SAVE     DS    18F
QNAME    DS    2F
RNAME    DS    CL256
RNLEN    DC    F'0'
         END
```

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.

Where more than one page reference is given, the first page number indicates the major reference.

GC27-6939-10

System/360 OS MFT Guide (S360-36) Printed in U.S.A. GC27-6939-10

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]

IBM Technical Newsletter

This Newsletter No.	GN28-2546
Date	April 16, 1973
Base Publication No.	GC27-6939-10
File No.	S360-36 (OS Release 21.7)
Previous Newsletters	None

IBM System/360 Operating System: MFT Guide

This Technical Newsletter, a part of release 21.7 of IBM System/360 Operating System, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and/or removed are:

Cover, 2
9-12, 12.1
21, 22
51, 52
57, 58
99, 100
111, 112
117, 118
123, 124
157, 158
187, 188, 188.1
193, 194, 194.1
207, 208

A change to the text or to an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments

The following topics are new or include changes for OS release 21.7:

Main Storage Organization Change
Job Step Timing
EXEC Statement
SYSIN and SYSOUT Data Blocking
The Resident Access Method Modules Option
STAE - Specify Task Asynchronous Exit
Inter-Partition POST - Post a Nonresident Routine

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications Development, Department D58, Building 706-2,
PO Box 390, Poughkeepsie, New York 12602

Printed in U.S.A.

READER'S COMMENT FORM

IBM System/360 Operating System:
MFT Guide

Order No. GC27-6939-10

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.

GC27-6939-10

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.

Cut Along Line

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]