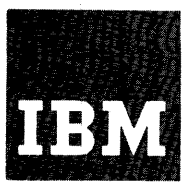
File Number S360(Mod.20)-36
Form C24-9006-4

DPS



Systems Reference Library

IBM System/360 Model 20 Disk Programming System Control and Service Programs

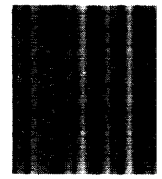
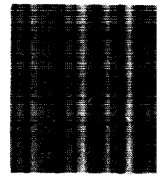
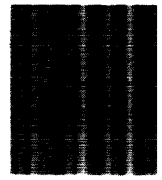
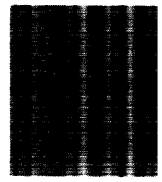
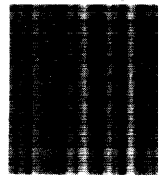
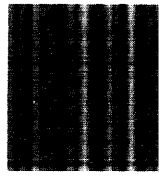
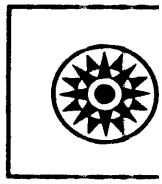
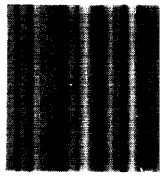
This publication describes the purpose and functions of the Control and Service programs of the IBM System/360 Model 20 Disk Programming System (DPS).

The publication is designed to help you use the DPS Control and Service programs to build, run, and maintain a disk-resident system.

The disk-resident system controls the compilation, assembly, and execution of problem programs. It offers you many advantages: a core-image library to store your programs, batch job processing for continuous operation, inquiry for access to specific information while another program is running, multiphase loading for particularly long programs, and symbolic addressing of I/O devices for greater flexibility.

To benefit from this publication, you should have a basic knowledge of the IBM System/360 Model 20, and be familiar with either the Assembler language or RPG.

You can find the titles and abstracts of related publications in IBM System/360 Model 20, Bibliography, Form A26-3565.

Fifth Edition (March, 1969)

This is a major revision of, and obsoletes C24-9006-3 and Technical Newsletters N33-9045 and N33-9049 and N33-9042.

Most of the text has been rewritten and reorganized to make the publication easier to understand. These improvements are not marked.

The technical changes incorporated in the publication relate to the delivery of IBM System/360 Model 20, Submodel 5, and to the addition of the COPSYS and BACKUP programs to the DPS. These technical changes and additions are marked in the following way: Changes to the text, and small changes to illustrations, are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption; added pages are flagged by the symbol • to the left of the page number.

This edition applies to the following components of IBM System/360 Model 20 Disk Programming System and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters.

	Version	Modification
Backup and Restore Program	1	0
Copy System Disk Program	1	0
Core-Image Maintenance Program	4	0
Core-Image Service Program	3	0
Directory Service Program	3	0
Disk-Resident Control Programs	3	0
Library Allocation Organization Program	3	0
Linkage Editor Program	3	0
Load System Disk Program	3	0
Macro Library Service Program	1	0
Macro Maintenance Program	3	0
Physical and Logical Unit Tables Service Program	3	0

Changes are continually made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 Model 20 SRL Newsletter, Form N20-0361, for the editions that are applicable and current.

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

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

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Laboratories, Programming Publications, 703 Boeblingen/Germany, P.O. Box 210.

Contents

This publication is designed to help you use the Control and Service programs of the IBM System/360 Model 20 Disk Programming System (DPS). The publication is divided into four main sections:

Introduction
DPS Control Programs
Libraries
DPS Service Programs
IBM Distribution Package.

The glossary defines the terms and abbreviations used in this publication.

The appendixes contain:

- Information on the System/360 magnetic tape and disk labeling conventions.
- An alphabetic list of the Model 20 DPS program and phase names.
- A diagram indicating the use of the Disk Programming System.

The purpose of the DPS Control and Service programs is to help you operate a Model 20 configuration equipped with disk drives and an optional selection of other input/output devices including magnetic tape units and the printer-keyboard.

The DPS Control and Service programs enable you to build and maintain a disk-resident control system which supervises:

(1) the translation and execution of problem programs written in the DPS Assembler or the RPG language, or PL/I,

(2) the execution of IBM-supplied Model 20 Disk Sort/Merge and Utility programs, and

(3) the execution of IBM-supplied Model 20 DPS Tape Sort/Merge and Utility programs.

Your disk-resident system will make data processing easier. It enables you to load and retrieve programs rapidly, reduces card handling to a minimum, and provides program and data storage space adequate to your needs.

Prerequisites

You should be familiar with the characteristics of the Model 20 as described in the publication

IBM System/360 Model 20
Functional Characteristics
Form A26-5847

and learn about the use of disk storage equipment from the publication

IBM System/360
Component Descriptions
Form A26-5988.

You will find a detailed description of the procedure for generating a Monitor in the publication

IBM System/360 Model 20
Disk Programming System
System Generation and Maintenance
Form C33-6006.

If you have an IBM Binary Synchronous Communications Adapter (BSCA), refer to the publications:

General Information
Binary Synchronous Communications
Form A27-3004

IBM System/360 Model 20
Input/Output Control System for the Binary Synchronous Communications Adapter
Form C33-4001.

Depending on the language you wish to use, you should be thoroughly familiar with the appropriate Systems Reference Library (SRL) publications listed below.

IBM System/360 Model 20
Disk and Tape Programming Systems
Report Program Generator
Form C24-9001

IBM System/360 Model 20
Disk and Tape Programming Systems
Assembler Language
Form C24-9002

IBM System/360 Model 20
Disk Programming System
Input/Output Control System
Form C24-9007

IBM System/360 Model 20
Disk Programming System
PL/I
Form C33-6007

If you intend to use the Model 20 Disk Sort/Merge or Utility programs or the Model 20 DPS Tape Sort/Merge or Utility programs

Dotted line flags planning information

supplied by IBM, you should be familiar with the pertinent SRL publications.

Titles and abstracts of other related publications are listed in the SRL publication IBM System/360 Model 20, Bibliography, Form A26-3565.

Terms and Abbreviations

The Glossary contains a summary of the terms and abbreviations used in this publication and offers a brief definition of each entry.

Principle Features and Advantages

The following description briefly details the prime features and advantages of the Model 20 disk-resident system.

PROGRAM LIBRARY FACILITY

The disk-resident system contains a core-image library. This library can be used to store all programs that you have written to meet your specific requirements and that IBM has written to perform generally useful operations. Each of the programs stored in the library can be loaded into main storage and executed at any time. You can add new programs to the library at any time, and you can delete programs you no longer require.

MACRO LIBRARY FACILITY

In addition to the core-image library, your disk-resident system may also contain a macro library. This library is used to store macro definitions that you have written or that IBM has supplied. The macro definitions can only be used within programs written in the DPS Assembler language. They are called from the macro library through macro instructions such as GET, PUT, or FETCH. A maintenance program is available from IBM that enables you to update the macro library by adding or deleting macro definitions according to your needs.

BATCH JOB PROCESSING

The disk-resident system makes it possible to execute a number of problem programs consecutively in one system run. The control programs provide for automatic job-to-job transition. The operator's activities are reduced to (1) determining the sequence of programs to be executed, (2) supplying the system with certain information by means of control statements, and (3) handling card, disk or magnetic tape devices.

MULTIPHASE LOADING

Because of this feature of the disk-resident system, your program written in the Assembler language may consist of a number of independent or inter-dependent phases to be loaded and executed consecutively and selectively. The loading of one such phase is initiated by the preceding phase.

SYMBOLIC ADDRESSING OF INPUT/OUTPUT DEVICES

The DPS programs use symbolic addresses to refer to input/output (I/O) devices. These symbolic I/O device addresses consist of standardized names, some of which can be placed in the problem program to refer to disk or magnetic tape units. The assignment of physical device addresses to the symbolic addresses takes place at execution time. It is achieved by means of ASSGN control statements and the logical and physical unit tables of the Monitor program.

Machine Requirements

MINIMUM SYSTEM CONFIGURATION

Submodel 2

- An IBM 2020 Central Processing Unit, Model BC2 (12,288 bytes of main storage);
- an IBM 2311 Disk Storage Drive, Model 11 or 12;
- one of the following card reading devices:

  IBM 2501 Card Reader, Model A1 or A2,
  IBM 2520 Card Read-Punch, Model A1,
  IBM 2560 Multi-Function Card Machine (MFCM), Model A1;
- one of the following printers:

  IBM 1403 Printer, Model N1, 2, or 7,
  IBM 2203 Printer, Model A1.

Submodel 4

- An IBM 2020 Central Processing Unit, Model BC4 (12,288 bytes of main storage);
- an IBM 2311 Disk Storage Drive, Model 12;
- an IBM 2560 MFCM, Model A2;
- an IBM 2203 Printer, Model A2.

Submodel 5

- An IBM 2020 Central Processing Unit, Model BC5 (12,288 bytes of main storage);
- an IBM 2311 Disk Storage Drive, Model 11 or 12;
- an IBM 2560 MFCM, Model A1;
- an IBM 2203 Printer, Model A1.

MAXIMUM SYSTEM CONFIGURATION

Submodel 2

- An IBM 2020 Central Processing Unit, Model D2 (16,384 bytes of main storage); with or without IBM Binary Synchronous Communications Adapter, Feature No. 2074;
- two IBM 2311 Disk Storage Drives, Model 11 or 12 (both must be the same model);
- an IBM 2415 Magnetic Tape Unit, Model 1 through 6;
- an IBM 2501 Card Reader, Model A1 or A2;
- an IBM 1442 Card Punch, Model 5;
- one of the following card units:

  IBM 2520 Card Read-Punch, Model A1,
  IBM 2520 Card Punch, Model A2 or A3,
  IBM 2560 MFCM, Model A1;

- one of the following printers:

  IBM 1403 Printer, Model N1, 2, or 7,
  IBM 2203 Printer, Model A1;

- an IBM 2152 Printer-Keyboard;
- one of the following magnetic character readers:

  IBM 1419 Magnetic Character Reader, Model 1 or 3,
  IBM 1259 Magnetic Character Reader, Model 1, 31, or 32;

Submodel 4

- An IBM 2020 Central Processing Unit, Model D4 (16,384 bytes of main storage); with or without IBM Binary Synchronous Communications Adapter, Feature No. 2074;
- two IBM 2311 Disk Storage Drives, Model 12;
- an IBM 2560 MFCM, Model A2;
- an IBM 2203 Printer, Model A2;
- an IBM 2152 Printer-Keyboard;

Submodel 5

- An IBM 2020 Central Processing Unit, Model E5 (32,768 bytes of main storage); with or without IBM Binary Synchronous Communications Adapter, Feature No. 2074;
- four IBM 2311 Disk Storage Drives, Model 11 or 12;
- an IBM 2415 Magnetic Tape Unit, Model 1 through 6;
- an IBM 2501 Card Reader, Model A1 or A2;
- an IBM 1442 Card Punch, Model 5;
- one of the following card units:

  IBM 2520 Card Read-Punch, Model A1,
  IBM 2520 Card Punch, Model A2 or A3,
  IBM 2560 MFCM, Model A1;

- one of the following printers:

  IBM 1403 Printer, Model N1, 2, or 7,
  IBM 2203 Printer, Model A1;

- an IBM 2152 Printer-Keyboard;
- one of the following magnetic character readers:

  IBM 1419 Magnetic Character Reader, Model 1 or 3,
  IBM 1259 Magnetic Character Reader, Model 1, 31, or 32;

Summary of System Programs

The primary element of the Model 20 DPS is the disk-resident system built to meet your particular requirements. The disk-resident system contains the disk-resident Monitor program and a core-image library. It may also contain a macro library.

The core-image library contains the disk-resident Job Control program and may contain any of the following IBM-supplied programs:

- service programs,
- language translator programs,
- utility programs, and
- other DPS programs.

You may also place any of your own programs in the library of the disk-resident system.

A disk-resident system can be used to control the execution of programs contained in its own core-image library. Before a disk-resident system can be used, the disk-resident Monitor program must be loaded into main storage. This is achieved by means of the Initial Program Loader.

Figure 1 shows the IBM-supplied programs which make up the Model 20 Disk Programming System. The Model 20 DPS program names are listed in detail in Appendix I.

CONTROL PROGRAMS

The term "control programs" refers to the IBM-supplied Initial Program Loader, the Monitor program, and the Job Control program, each of which can be used in a card and a disk version. The functions of each version of the three control programs are essentially the same. The disk-resident control programs supervise the compilation (or assembly) and execution of all programs running under the control of the system.

The following is a brief discussion of each type of control program.

Disk Initial Program Loader (IPL)

The function of the disk Initial Program Loader is to initiate the system operation. It consists of two parts; the first part, which comprises three punched cards, causes the second part to be read into main storage from the system disk pack.

The IPL loads the Monitor into main storage from the system disk pack. The Monitor remains in main storage throughout the system run, and calls the Job Control program into storage whenever a program reaches the end of the job. Hence, the Initial Program Loader need not be used again during the system run. The Initial Program Loader is discussed in detail in the section DPS Control Programs.

Disk Monitor Program

Throughout a system run, the Monitor resides in main storage to provide information and perform operations required by all jobs. The Monitor consists of the following parts, depending on the desired features you specify at Monitor generation time:

1. Communication Region -- This area contains information required by all programs. It includes fields for recording the date, the name of the current job, and the current state of program switches.

2. Logical Unit Table -- An area for recording the current I/O device assignments.

3. Physical Unit Table -- An area containing the physical disk and magnetic tape device addresses.

4. Physical Disk I/O Routines -- A set of routines that perform disk I/O operations for the Monitor and problem programs, including the disk and magnetic tape XIO routine.

5. Fetch Routine -- A routine that loads problem programs from the core-image library into main storage so that they can be executed.

6. Monitor I/O Area -- An area that is used by Monitor routines for reading from disk.

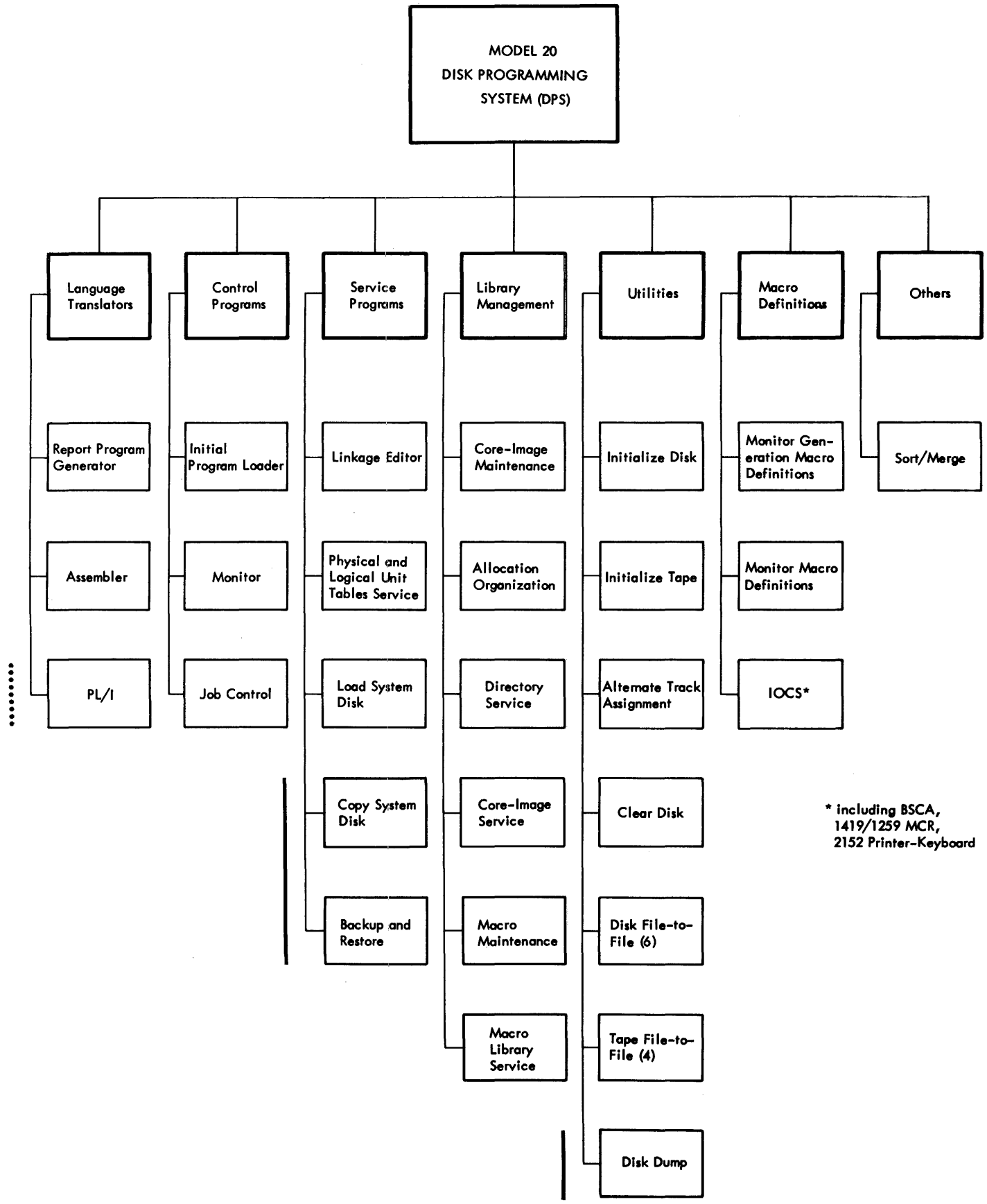
7. Tape Error Recovery Routine -- A set of routines to control the execution of error-recovery procedures when magnetic tape I/O errors occur.

8. Tape Error Statistics Routine -- A routine that analyzes the interrupts and magnetic tape I/O errors occurring during the execution of a program. This routine provides a useful service aid.

These routines and areas correspond to the standard version of the Monitor. You can tailor a Monitor with fewer or more routines according to your needs. The generation of a Monitor is described in the section Generative Monitor Concept.

The storage requirements of the Monitor vary with the features it supports. The minimum version of the Monitor occupies about 3300 bytes of main storage, the maximum version about 4270 bytes. For details refer to the SRL publication IBM System/360 Model 20, Disk Programming System, Performance Estimates, Form C33-6003.

The Monitor is described in detail in the section DPS Control Programs.



● Figure 1. Summary of IBM-Supplied Model 20 DPS Programs

Dotted line flags planning information

Lisk Job Control Program

The Job Control program is executed between jobs to prepare the system for the next job to be executed. It is first called by the Monitor program at IPL time, and after that it is called each time a program reaches the end of the job. The job control statements you insert supply the system with the information required to direct the operations of the Job Control program.

The Job Control program changes the temporary I/O device assignments and certain areas of the Monitor communication region in main storage. It also stores disk and tape label information for label checking. The Job Control program is discussed in detail in the section DPS Control Programs.

Card-Resident Control Programs

The control programs are also used in a card version, which is referred to as a card-resident system.

An installation that operates primarily under the control of the disk-resident system can use the card-resident system:

- to test problem programs that are available in absolute card format before they are placed into the core-image library,
- to execute problem programs that are available in absolute card format.

SERVICE PROGRAMS

The term "service programs" refers to the Library Management programs, the Physical and Logical Unit Tables Service program, the Linkage Editor program, the Load System Disk program, the Copy System Disk program, and the Backup and Restore program.

Library Management Programs

The Library Management programs are a group of programs that maintain the disk-resident libraries.

The programs enable the libraries to be modified by:

- adding or deleting information,
- redefining the limits of the libraries,
- printing the contents of the library directories, and
- printing or punching the contents of the libraries.

A program is placed temporarily in the core-image library during load-and-go, compile-and-execute, or assemble-and-execute operations. If the programs are run frequently, the Core-Image Maintenance program (CMAINT) can be used to make them a permanent part of the core-image library. Otherwise, the area in the library where they are placed is overlaid by the next program that is temporarily or permanently included in the core-image library.

The Library Management programs are discussed in detail in the section DPS Service Programs.

Three libraries are used with the disk-resident control system. They are:

1. A required core-image library that contains the Job Control program, and may also contain the Library Management programs, the Physical and Logical Unit Tables Service program, the Linkage Editor, the Assembler program, RPG, PL/I, the Sort/Merge programs, the Utility programs, and problem programs in core-image format, i.e., in immediately executable form. A program is stored in the core-image library as one or more program phases.
2. An optional macro library that contains Logical IOCS, Monitor macro definitions, Monitor generation macro definitions, and user-written macro definitions.
3. An optional relocatable area that is used to temporarily hold the output of one assembly or one RPG or PL/I compilation that is to be edited and executed immediately.

The libraries are discussed in detail in the section Libraries and Relocatable Area.

Physical and Logical Unit Tables Service Program (PSERV)

The main function of the PSERV program is to list or change the permanent device assignments in the Monitor on the system disk pack. Its other functions are to alter the configuration byte in the Monitor communication region and to display information about the Monitor type and features that have been generated. The PSERV program is discussed in detail in the section DPS Service Programs.

Linkage Editor Program (LNKEDT)

Programs that are output from the Assembler may be processed by the Linkage Editor program before they are placed in the core-image library. The Linkage Editor program combines separately assembled programs or

Dotted line flags planning information

phases into an integral program that can be executed under control of the Monitor program.

The Linkage Editor program is discussed in detail in the section DPS Service Programs.

Load System Disk Program (LDSYS)

The Load System Disk program loads the disk-resident system onto the disk pack destined for system residence. It can be executed under control of the disk-resident or card-resident control system.

The minimum disk-resident system you build must include the disk-resident part of IPL, the Monitor, and the core-image library containing the Job Control program. At your discretion, the system may also include the Assembler program, the Report Program Generator, PL/I, the Linkage Editor, and other DPS programs. You may also allocate disk areas for a macro directory, a macro library, and a relocatable area.

The Load System Disk program is discussed in detail in the section DPS Service Programs.

Copy System Disk Program (COPSYS)

The Copy System Disk program writes the contents of a system disk pack onto another disk pack.

The Copy System Disk program is described in detail in the section DPS Service Programs.

Backup and Restore Program (BACKUP)

The Backup and Restore program copies the contents of a disk pack onto magnetic tape for backup and writes the contents back onto disk. Optionally, a card file can be included on the backup tape and restored to its original medium.

The Backup and Restore program is discussed in detail in the section DPS Service Programs.

LANGUAGE TRANSLATORS

The three DPS language translator programs are the Report Program Generator (RPG), the Assembler/IOCS program, and PL/I.

The DPS Report Program Generator

The DPS Report Program Generator compiles source programs written in the RPG language into object programs. The object program can be obtained either in punched cards and executed in a subsequent run, or it can be stored permanently in the core-image library through a CMAINT run and called into main storage repeatedly for execution. The object program can also be executed immediately after compilation (compile-and-execute), but the CMAINT program is also required on the system disk pack because the object program must be stored temporarily in the core-image library. In addition, as for all RPG runs, a relocatable area must have been specified.

Note that RPG programs cannot be linked or relocated. Therefore, the Linkage Editor program is not used with RPG programs.

DPS Assembler/IOCS Program

The DPS Assembler program translates source programs written in the DPS Assembler language into object programs. The object program can be obtained either in punched cards and executed in a subsequent run, or it can be stored permanently in the core-image library through a CMAINT run and called into main storage repeatedly for execution. The object program can also be executed immediately after assembly (assemble-and-execute), but the CMAINT program is also required on the system disk pack because the object program must be stored temporarily in the core-image library. In addition, a relocatable area must have been specified.

Any macro definition to be used, IBM-supplied or user written, must be incorporated in the macro library of your disk-resident system. The macro definitions are called from the macro library by means of the macro instructions you issue in your problem program. The macro instructions of the DPS IOCS and the BSCA IOCS and the IOCS for the 1419/1259 MCRs are described in the pertinent SRL publications.

DPS PL/I Compiler

The PL/I compiler translates source programs written in PL/I into machine language and performs the necessary link-editing to transform compiled object modules into an executable object program.

By submitting certain program control statements to the compiler, the user can either

(1) compile
or (2) compile and link-edit
or (3) compile, link-edit, and execute
or (4) link-edit and execute
or (5) just link-edit

a source program in one job.

OTHER IBM-SUPPLIED PROGRAMS

The remaining Model 20 DPS programs supplied by IBM are described in the paragraphs that follow.

DPS Disk Sort/Merge Program and DPS Tape Sort/Merge Program

The DPS Disk Sort/Merge program and the DPS Tape Sort/Merge program allow you to:

(1) Sort a single data file, which contains data in an unknown or unordered sequence, and arrange the logical records according to the sequence you specify.

(2) Merge two to four previously sequenced input files into a single ordered file.

(3) Sequence-check a single file while copying it. You can also reblock the records if you wish.

The DPS Disk Sort/Merge program and the DPS Tape Sort/Merge program can be executed under control of the card-resident system if they are contained in punched cards. Otherwise, they must be contained in the core-image library of the disk-resident system.

For further details on these two programs, refer to the SRL publications:

IBM System/360 Model 20
Disk Programming System
Disk Sort/Merge Program
Form C26-3806, and

IBM System/360 Model 20
Disk and Tape Programming Systems
Tape Sort/Merge Program
Form C26-3804.

DPS Disk Utility Programs and DPS Tape Utility Programs

IBM supplies fifteen Utility programs for Model 20 systems with disk storage equipment. Ten of the Utility programs transfer data from one storage medium to another, for instance from disk to magnetic tape, or from card to disk. The remaining programs are used for preparing and maintaining the format of the disk pack or magnetic tape reel, for example, to initialize the tape reel, or to allocate alternate tracks.

The DPS Disk Utility programs and the DPS Tape Utility programs can be executed under control of the card-resident system if they are contained in punched cards. Otherwise, they must be contained in the core-image library of your disk-resident system and executed under control of the disk Monitor program.

For further details on the Utility programs refer to the SRL publications:

IBM System/360 Model 20
Disk Programming System
Disk Utility Programs
Form C26-3810, and

IBM System/360 Model 20
Disk and Tape Programming Systems
Tape Utility Programs
Form C26-3808.

Organization of the System Disk Pack

The system disk pack contains your disk-resident system. The disk-resident system consists of the IPL part 2, the Monitor program, and the core-image library.

Figure 2 illustrates the organization of the system disk pack and indicates the extents of the IPL part 2, the Monitor, and the core-image library.

The standard IBM volume label, which identifies the volume, and the label information area (LIA) for Job Control, which contains the disk label information for I/O files, reside on track 1 (or tracks 1 and 2) of cylinder 0. The Volume Table of Contents (VTOC) contains a disk file label that indicates the extent of the disk area occupied by the system. The VTOC resides on tracks 2-9 (or tracks 3-9) of cylinder 0, if you make no other assignment. For precise details of the format of disk labels, refer to Appendixes A and C.

The Monitor is stored in a fixed location (cylinder 4, tracks 0-1). The libraries, however, are variable in extent. The core-image directory will always begin at cylinder 4, track 4, and the libraries and other directories will follow consecutively. The core-image and macro libraries and directories are discussed in the section Libraries and Relocatable Area.

The library work area is used by the Core-Image Library Maintenance program to update the Monitor or IPL, and to store tape label information in assemble-and-execute runs.

Alternate tracks (cylinders 1-3) are available for use when a track on the system disk pack becomes defective.

The system directory contains information on the extent and location of the disk areas allocated to the core-image directory, the core-image library, the macro directory, the macro library, and the relocatable area. Each of the five entries in the directory takes up 14 bytes. The format of a directory entry is:

Byte	Contents
0-1	Entry identifiers and indicators.
2-5	Disk address of the beginning of the area
6-9	Disk address of the last sector allocated to the area.
10-13	Disk address of the last currently occupied sector of the area.

The disk addresses are in the form chhr, where c represents the cylinder number, hh the track number, and r the record number.

The total area occupied by the system directory is 70 bytes.

Contents	Begin Address			End Address			Sector Total
	Cylinder	Track	Sector	Cylinder	Track	Sector	
Disk IPL	000	0	0	000	0	9	10
Volume Label	000	0	0	000	1	0	1
LIA (Standard)	000	1	1	000	1	9	9
VTOC (Standard)	000	2	0	000	9	9	80
Alternate Track Area	001	0	0	003	9	9	300
System Directory	004	0	0	004	0	0	1
Monitor	004	0	1	004	1	9	19
Library Work Area	004	2	0	004	3	9	20
Core-Image Directory*	004	4	0				
Core-Image Library*							
Macro Directory*	*Specified according to your actual requirements. These areas immediately follow the library work area and are adjacent to one another. The area occupied by a directory must not be greater than 100 sectors.						
Macro Library*							
Relocatable Area*							

Figure 2. Organization of the System Disk Pack

DPS Control Programs

The DPS Control programs are the Initial Program Loader, the Monitor program, and the Job Control program. Each of these three programs can be used in a card and a disk-resident version. The functions of the two versions of each control program are essentially the same. Their differences are described in the appropriate sections below.

The Initial Program Loader loads the Monitor program into main storage and transfers control to it, causing it to load the Job Control program. After execution of this program, control is returned to the Monitor program. The Job Control program is used (1) to assign physical input/output addresses to the symbolic addresses used in the programs, (2) to specify other environmental data (for example, the date or the storage capacity), and (3) to place the name of the next program to be executed into the communication region of the Monitor.

The card-resident control programs are used to control the execution of object programs that are contained in punched cards.

The disk-resident control programs are required for the execution of the programs supplied by IBM -- such as the DPS Report Program Generator -- and for the execution of any other programs contained in the core-image library of your disk-resident system.

The subsequent text describes the control programs and their functions.

Monitor Program

The Monitor program is the main control program. Its principal functions are (1) to load the program phases to be executed into main storage, (2) to allow interprogram communication, (3) to store physical I/O addresses, (4) to process requests for magnetic tape, disk and printer-keyboard I/O operations, (5) to control the initialization of inquiry programs, and (6) to schedule interrupts.

You can generate a Monitor that is tailored to your requirements. This section explains the generative concept. It also describes several areas and routines that are part of the Monitor.

GENERATIVE MONITOR CONCEPT

The generative Monitor concept permits you to tailor a Monitor to the actual tasks it must perform in executing programs, and to adapt the Monitor to the installed system.

Not all available I/O devices (magnetic tape units, disk units, etc.,) may be attached to your system. Through Monitor generation, you can generate the routines that support your I/O devices, and omit those routines from the Monitor that support I/O devices your system does not have.

The Monitor is generated by means of Monitor generation macro definitions in an assembly run. The special features the Monitor is to have are submitted to the Assembler by means of macro instructions, operands, and detail specifications.

The generated Monitor is classified into four types:

1. Card-resident
   The generated Monitor is always punched into cards. In this form, it executes non-relocatable object modules contained in cards.

2. Disk-resident
   The generated Monitor is written onto the system disk pack and loaded into main storage by the IPL program. It executes object modules contained in the system libraries.

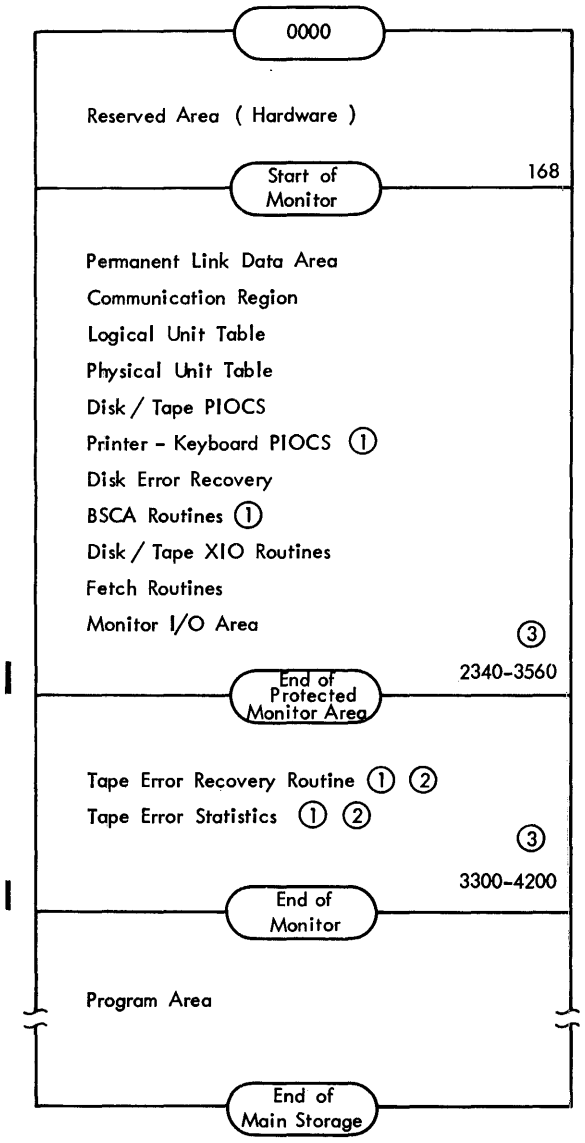
3. Disk-resident with transient routines
   This Monitor is used in the same way as the disk-resident Monitor. Routines that are only temporarily needed reside on the system disk pack (for example, the Fetch routine). These transient routines are executed in the transient area of the Monitor, an area two sectors in length. This technique reduces the storage requirements of the Monitor to a minimum.

4. Disk-resident with inquiry support
   Basically, this Monitor is like the Monitor with transient routines, but it also supports inquiry. The major part of the routines used for inquiry support are transient and reside on the system disk pack.

The usage of Monitor generation macro instructions, the Monitor generation procedure, and the insertion of the Monitor in the system disk pack are described in the publication IBM System/360 Model 20, Disk Programming System, System Generation and Maintenance, Form C33-6006.

Figures 3 and 4 show the storage maps of the DPS Monitor for a card-resident and a disk-resident system. The Monitor occupies the lower area of main storage. The communication region, the logical unit table, the physical unit table, and the physical disk I/O routines reside permanently in main storage.

The program area follows the Monitor area in main storage. This area is used for program compilation, assembly, and execution. Tape label information is stored at the end of main storage.

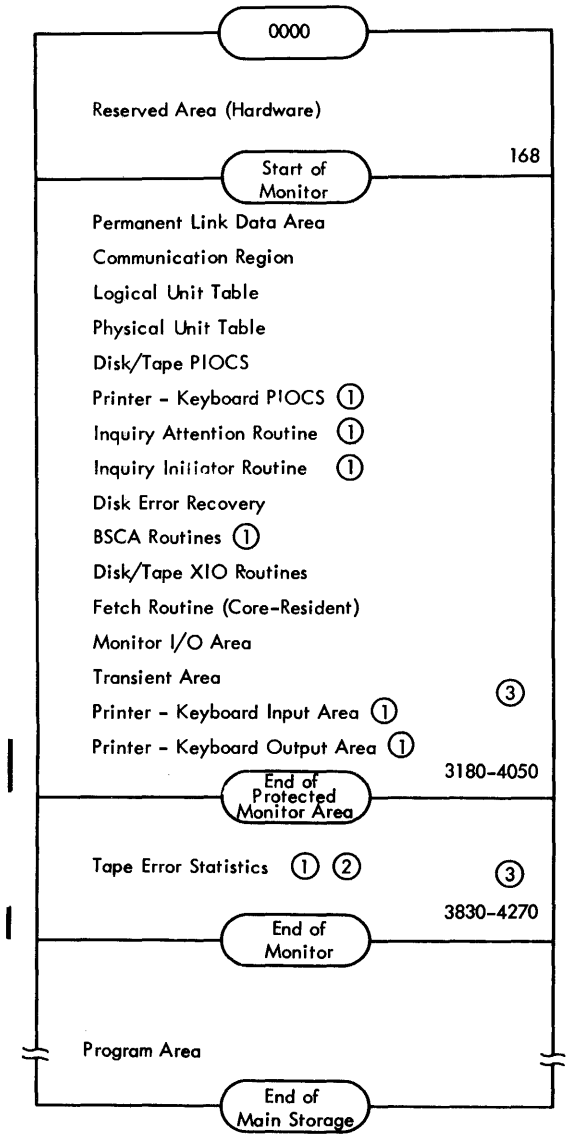


(1) Only included if specified during Monitor generation

(2) May be overlaid by user programs if tape I/O or tape error statistics are not required

(3) End address depends on the operands specified at Monitor generation time

● Figure 3. Storage Map Showing Areas Allocated to the Card-Resident or Disk-Resident Monitor with no Transient Routines

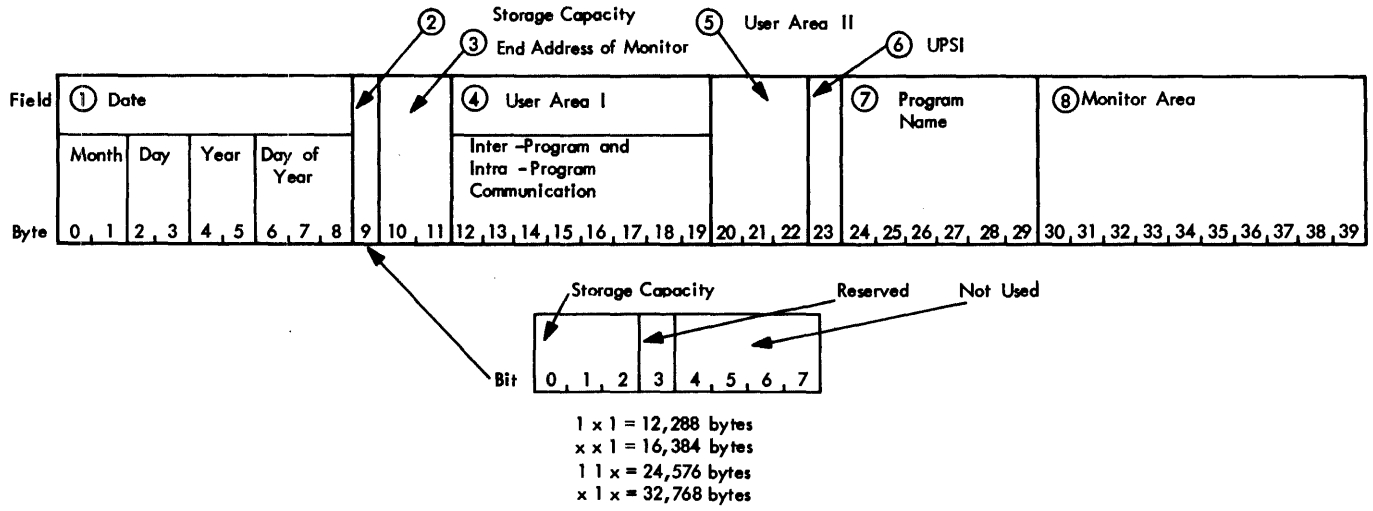


(1) Only included if specified during Monitor generation

(2) May be overlaid by user programs if tape error statistics are not required by the job

(3) End address depends on the operands specified at Monitor generation time

● Figure 4. Storage Map Showing Areas Allocated to the Disk-Resident Monitor with Transient Routines and with Inquiry Support



● Figure 5. Monitor Communication Region

You can specify the following features at Monitor generation time:

- number of additional logical unit blocks to assign the symbolic device addresses SYS000 to SYS019
- I/O support for up to four disk drives
- I/O support for up to six magnetic tape drives
- tape error statistics
- I/O support for printer-keyboard
- inquiry facilities
- BSCA support
- I/O request queuing
- Read/compute write/compute overlap.

COMMUNICATION REGION

The communication region is an area of 40 bytes within the Monitor area. You can address the communication region in problem programs written in the Assembler language by means of certain macro instructions described in the publication IBM System/360 Model 20, Disk Programming System, Input/Output Control System, Form C24-9007.

You can use the communication region to insert certain information into problem programs, or exchange information between various problem programs.

Figure 5 shows the layout of the communication region. The circled numbers identify its various fields.

The fields in the communication region are filled in by the Job Control program from information you supply by means of control statements. The following list describes the contents of each field. The item numbers refer to the areas circled in the figure.

Field	Byte(s)	Contents
(1) Date	0-8	The Job Control program sets this field from information you supply in the DATE control statement. It records the month, day, year, and day of year in zoned decimal format. August 14, 1969, for example, is recorded as 081469227. The field is in printable format. You can access the field, or a portion of the field, by means of Assembler move instructions and use it to date reports. The last five bytes of the field are used internally by the label processing routines to check the expiration dates of files.
(2) Storage capacity	9	This byte records the storage capacity of your IBM 2020 CPU. It is set to the specified value during Monitor generation time or when you submit a CONFG control statement in a PSERV run. The first three bits (0, 1, 2) contain a code indicating the storage capacity:

1x1 = 12,288 bytes
xx1 = 16,384 bytes
11x = 24,576 bytes
x1x = 32,768 bytes

Bit 3 is reserved, bits 4 through 7 are not used.

(3) End Address of Monitor — 10-11 — This field gives the end address of the storage area used for the Monitor. The address is supplied automatically.

(4) User Area I — 12-19 — This area can be used for inter-program and intra-program communication. It is not reset by the Job Control program between jobs. This area cannot be used during the execution of PL/I object programs.

(5) User Area II — 20-22 — This area can be used for intra-program communication. It is reset to binary zeros at the end of a job by the Job Control program.

(6) UPSI — 23 — This field contains user program switch indicators. Each time the Job Control program is called, it initializes the byte to binary zero. From the information you supply in the UPSI control statements, it then sets the specified bits (switches) to 1. Each bit in the field can be set and tested in a problem program.

(7) Program Name — 24-29 — This field contains the name of the program phase to be executed. It is set by the Job Control program from information you supply in the JOB control statement, or by the FETCH and EOJ macro instructions.

(8) Monitor Area — 30-39 — This field is used only by the Monitor.

LOGICAL AND PHYSICAL UNIT TABLES

The logical unit table is a component of the Monitor and resides in main storage throughout the system run. It relates symbolic addresses used in problem programs to physical addresses (for example, it relates SYSIPT to the physical address of the card reading device in your configuration).

The DPS Control and Service programs and the other IBM-supplied programs use symbolic addresses to refer to input/output devices. Some of these symbolic I/O addresses can be employed in problem programs.

The logical unit table consists of up to 26 entries, one associated with each device name, each entry occupying an area of 2 bytes. These entries are called logical unit blocks (LUBs). Each LUB is reserved for information about one specific symbolic device address. Thus, a particular LUB, although its contents may change, always refers to the same symbolic device address (SYSRDR or SYSIPT, and so on).

When a disk or magnetic tape drive is assigned to a symbolic device address (by means of an ASSGN control statement), the Job Control program records a pointer to the physical unit block (PUB) containing the physical address of the device to be assigned. When a card I/O device or a printer is assigned, actual information about the device is placed in the appropriate LUB. Whenever a statement containing a symbolic device address is encountered during the execution of a job, the appropriate LUB is consulted to determine the associated physical address. The I/O device defined in this manner then executes the specified input/output operation under control of the Monitor program.

Figure 6 shows the sequence of the LUBs in the logical unit table.

Device assignments are either permanent or temporary. Permanent device assignments are written onto the system disk pack. They are loaded into main storage as part of the Monitor. Temporary device assignments are made at Job Control time. They change the assignments currently in main storage, but do not alter the permanent device assignments on the system disk pack.

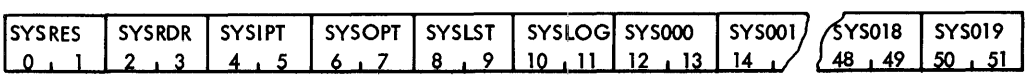


Figure 6. Sequence of LUBs in the Logical Unit Table

Dotted line flags planning information

You can make permanent device assignments at Monitor generation time. By means of the ASSGN macro instructions you issue, the assignments for all physical devices (disk drives, magnetic tape drives, printer, etc.,) in the configuration can be inserted into the respective LUBs and PUBs. Permanent device assignments for a disk-resident system may also be made by running a LDSYS or PSERV program.

The permanent device assignments of the Monitor resident in main storage during a system run remain in effect for all jobs until the Job Control program encounters an ASSGN control statement indicating a temporary device assignment. The temporary device assignment then remains in effect until it is changed by another ASSGN control statement. The permanent device assignments (on the system disk pack) are not affected by ASSGN control statements submitted to the Job Control program. Such ASSGN statements affect only the Monitor resident in main storage. The section Job Control Program describes the format of the ASSGN control statement.

The following table lists the symbolic addresses you must use to refer to I/O devices:

Symbolic Address	Refers to
SYSRES*	Disk-resident system: disk drive on which system disk pack is mounted; card-resident system: disk drive on which Job Control writes label information
SYSRDR*	Card reading device for control statements
SYSIPT	Card reading device, disk or tape drive used as input device for programs
SYSOPT	Card punching device, disk or tape drive used as output device for programs
SYSLST	Printer that lists the output of programs
SYSLOG	Printer that logs control statements
SYS000 . . SYS019	Disk and tape drives used as input and output for all programs; card I/O devices for Sort/Merge program

*SYSRES and SYSRDR must be assigned at IPL time. In a disk-resident system, SYSRDR may be reassigned during any following Job Control run.

PHYSICAL IOCS FOR THE PRINTER-KEYBOARD

The physical IOCS for the printer-keyboard controls all printer-keyboard input and output operations.

These physical IOCS routines interpret programmed commands to the printer-keyboard and direct the transfer of information between the printer-keyboard and main storage. They also test for errors in the transmission of data, and control the execution of error recovery procedures.

INQUIRY ROUTINES

The Inquiry routines are included in the Monitor if you specify TYPE=INQRY at Monitor generation time.

To initiate an inquiry, the operator presses the Request key on the printer-keyboard. If the execution of the mainline program can be suspended for the execution of an inquiry program, a Read instruction is initiated on the printer-keyboard, whereupon the operator types in the name of the inquiry program. A second Read instruction is initiated to read data that might be required by the inquiry program. The mainline program is rolled out of main storage onto the system disk pack. Then the inquiry program is called into main storage for processing. After execution is completed, the mainline program is loaded back into main storage and resumes execution. Each of the routines involved in processing inquiry programs is described in detail in the paragraphs that follow.

INQUIRY ATTENTION ROUTINE

This routine is called after the operator presses the Request key on the printer-keyboard. It first checks the status of the inquiry control bits in the communication region of the Monitor to test whether the current job can be interrupted by an inquiry program. If it can, the routine prints the message ENTER PROGNAME on the 2152, provided you specified INQMSG=YES at Monitor generation time.

If the current job cannot be interrupted, the routine prints the message NOINQ and returns control to the mainline program.

If the mainline program requires data transmission by means of BSCA, the Inquiry Attention routine also tests whether BSCA transmission is in progress. If so, the routine informs the remote terminal that transmission is to be discontinued.

INQUIRY INITIATOR ROUTINE

If a printer-keyboard input area is specified, control is transferred to this routine when the operator presses the End of Transmission (EOT) key on the printer-keyboard for the second time after having initiated an inquiry. If no printer-keyboard input area is specified, control is transferred to this routine after the operator presses the EOT key once.

This routine saves all data significant for the current status of the mainline program and loads the Rollout routine from the core-image library into the transient area of the Monitor.

PHYSICAL DISK AND TAPE I/O ROUTINES

The physical disk and tape I/O routines perform input/output operations.

They are used by the DPS IOCS, the Report Program Generator, PL/I, the disk utility programs, and other IBM-supplied programs.

The functions of these routines are (1) to handle the initiation of disk and magnetic tape I/O operations, (2) to analyze interrupts and errors, and (3) to control the execution of error recovery procedures in case of I/O errors.

The Tape Error Recovery and the Tape Error Statistics routines are included in the Monitor if you specify the pertinent operands at Monitor generation time. You can omit them if you do not need tape I/O support.

In a card-resident and a disk-resident Monitor, these routines are generated behind the Monitor I/O area. In a Monitor with transient routines, they reside on the system disk pack and are executed in the transient area of the Monitor.

The count fields for tape error statistics are located in the last positions of the Monitor.

If they are generated as an integral part of the Monitor, the Tape Error Statistics routines may be overlaid by every program phase if magnetic tape I/O is not required for a specific job. They are always restored at end-of-job time by the Job Closing routines.

JOB CLOSING ROUTINES

The functions of the Job Closing routines are:

- printing tape error statistics
- printing BSCA communications error statistics
- restoring the Tape Error Recovery and Tape Error Statistics routines
- calling the Job Control program which prepares for the next job by analyzing the control statements that follow.

These routines are executed and restored at the end of a job before the transition to the next job.

By specifying certain operands when you generate the Monitor, you determine which of these routines will be generated.

Tape error statistics are printed only if you include the control statements OPTN TES and LOG in the job control statements preceding the job.

BSCA communications error statistics are printed only if a LOG control statement was submitted in a preceding Job Control run and is still effective, that is, not negated through a NOLOG statement submitted in a subsequent Job Control run.

FETCH ROUTINE

The Fetch routine loads program phases into main storage so that they can be executed.

- In a card-resident Monitor, the Fetch routine loads only phases contained in punched cards.
- In a disk-resident Monitor, the Fetch routine loads only phases that are written on the system disk pack.
- In a Monitor with transient routines, the Fetch routine consists of a core-resident part and a transient part that resides in the core-image library. The core-resident part loads only the transient routines, for example the transient Fetch routine. The transient part loads any other program phases that reside in the core-image library.
- In the disk-resident system, all programs must be placed in the core-image library by the CMAINT program, either permanently or temporarily, before they can be loaded by the Fetch routine. Any phases that require relocation or linking must be processed by the Linkage Editor program before they are placed in the library. The Fetch routine scans the core-image directory and compares the phase names in the entries of this directory against the phase name sup-

plied in the JOB control statement. It locates the program phase in the core-image library and determines its loading address from information supplied in the core-image directory. Then it loads the appropriate phase into main storage and initiates execution.

Loading Conventions

You must observe the following rules when loading problem programs from the core-image library or from the device assigned to SYSIPT (in cards or card-image format)

1. You will encounter a variety of situations:

   - If the phase load address is within the Tape Error Statistics routine:

     This routine is switched off by the Fetch routine and no magnetic tape error analyzation and printout is performed.

   - If the phase load address is within the Tape Error Recovery routine (applies only to card-resident Monitors and disk-resident Monitors with no transient routines):

     This routine is switched off by the Fetch routine and no magnetic tape I/O requests can be performed during this phase.

   - If the phase load address is lower than the end address of the Monitor I/O Area (and lower than the transient area and printer-keyboard input area, depending on the type of Monitor generated):

     A halt is displayed to prevent an overlay of the protected Monitor area. The job must be cancelled. A Linkage Editor run or a reassembly or recompilation is required.

2. The general registers 8, 14, and 15 are used in the loading process. For this reason, any values in these registers that must be saved have to be moved to an intermediate storage area, from where they can be reloaded into the appropriate registers after completion of the loading process.

3. If you use the DPS IOCS to handle the card I/O requirements in a problem program, a WAITC macro instruction must precede each FETCH macro instruction in the program.

4. The Fetch routine of the card-resident Monitor can only load object programs that are contained in punched cards and require neither relocating nor linking. If relocation or linking is required, this must be done by means of the Linkage Editor program before loading.

   The input to the Fetch routine can be Assembler output, RPG output, PL/I output, or Linkage Editor output. However, only the card types listed below are recognized by the Fetch routine (all other card types are ignored).

   The TXT cards contain the instructions and constants to be loaded. Each TXT card also contains the address of the first byte of the storage area into which the text is to be loaded.

   The REP cards are used to substitute new text for portions of assembled text. Each REP card also contains the assembled address of the first byte to be replaced. No diagnostic is made on the format of the REP card.

   An XFR card is inserted at the end of each phase. It contains an instruction that causes a transfer to the entry point of the phase loaded. This makes it possible to execute the phase immediately after loading. If the XFR card does not contain a branch address, it is ignored.

   An END card appears at the end of each program. It causes a transfer to the entry point of the program. If the END card does not contain a branch address, it is ignored.

MONITOR I/O AREA

A Monitor I/O Area is generated for all card-resident and disk-resident Monitors. This area is 270 bytes in length; it is used by the Monitor routines.

TRANSIENT AREA

This area is generated for disk-resident Monitors with transient routines and Monitors with inquiry support. The transient area is used

- by Monitor routines as I/O area
- for executing transient routines
- for intercommunication between transient routines.

Dotted line flags planning information

The transient area is 580 bytes in length. A total of 540 bytes are used for executing transient routines; the remaining 40 bytes serve as intercommunication area for the transient routines. The first 270 bytes of the transient area are also used as Monitor I/O Area.

If inquiry facilities are supported, a printer-keyboard input area is generated behind the transient area.

TRANSIENT ROUTINES

Transient routines are only generated for Monitors with the transient feature. They are generally two sectors in length and reside in the core-image library.

The basic transient phases are

- transient Fetch routine
- Tape Error Recovery routine
- Rollout routine
- Rollin routine.

The Fetch routine fetches each transient phase from the core-image library when needed and loads it into the 580-byte transient area of the Monitor; each transient phase overwrites any previously loaded transient phase. The transient concept optimizes the efficient use of main storage allotted to the Monitor.

TRANSIENT TAPE ERROR RECOVERY

If the Monitor contains transient routines, the Fetch routine loads the first phase of the Tape Error Recovery routine into the transient area whenever an error occurs during magnetic tape I/O operations. If the error is a read error or an irrecoverable tape error, the Fetch routine loads the second or third phase of the Tape Error Recovery routine into the transient area for execution.

ROLLOUT ROUTINE

The Rollout routine is called by the Inquiry Initiator routine when an inquiry program is to be executed.

The Rollout routine writes the whole of main storage located behind the Monitor into the dummy phase of the core-image library reserved for this purpose at Monitor generation time. Then the Rollout routine fetches the inquiry program whose name the operator entered on the printer-keyboard.

If you submitted a CONFG control statement in a previous job to temporarily change the storage capacity specification in main storage, the amount of main storage rolled out into the dummy phase depends on the storage capacity specification in main storage and on the system disk pack.

If the storage specification in main storage is lower than the storage specification on the system disk pack, the Rollout routine automatically rolls the lower storage area into the dummy phase. If the storage specification in main storage is higher than the storage specification on the system disk pack, a halt occurs, and the operator can either discontinue or continue the inquiry run. If he continues it, the Rollout routine rolls the storage area specified on the system disk pack into the dummy phase. The remaining part of the storage area is not rolled out, and is therefore not protected during the inquiry run.

The Rollout routine is always transient and resides in the core-image library.

ROLLIN ROUTINE

After the inquiry program has been executed, the Fetch routine reads the Rollin routine into the transient area of the Monitor. The Rollin routine restores the status of the mainline program that prevailed before the mainline program was interrupted by the inquiry program, and the mainline program resumes execution. The Rollin routine is always contained in the core-image library as a transient phase.

Job Control Program

This section describes the functions of the Job Control program and shows you how to code the control statements required to perform these functions.

For a disk-resident system, the Job Control program is contained in the core-image library of the system disk pack.

The disk-resident Job Control program is executed between programs to prepare the system for the next program run. It obtains information about the next job from the job control statements you supply. It processes these control statements, records this information in the communication region of the Monitor, processes tape and disk file labels, and returns control to the Monitor so that the prepared job can be executed.

The Job Control program is called into storage initially by the Monitor program.

Subsequently, each time an EOJ macro instruction is encountered, indicating the end of a job, the Job Control program is called to prepare the system for the next job to be executed.

For a card-resident system, the Job Control program is generated together with the card-resident IPL and card-resident Monitor program.

The card-resident Job Control program deck must precede every problem program deck. Control statements may be placed between the Job Control deck and the problem program deck if the I/O device assignments for the job are to be changed. The Fetch routine of the Monitor loads the Job Control program into main storage. After the Job Control program has processed the job control statements, it returns control to the Monitor. The storage area in which the Job Control program was stored can be overlaid by the problem program.

CONTROL STATEMENT CONVENTIONS

Control statements that are common to all programs are called job control statements. They are read by the Job Control program before each job is executed. Their meaning and formats are described in the section Format of Job Control Statements.

Some control statements are applicable only to certain system programs. They instruct the program about the functions it is to perform. Since they are read by the program itself, they are called program control statements. Each program control statement is described together with the program to which it applies in the section DPS Service Programs.

The following conventions serve as a guideline to help you code the control statements described in this publication:

1. Upper-case words (example: ASSGN) must appear in the control statement exactly as shown in the format.

2. Lower-case words symbolize additional information you must supply. For example, if the word "program" appears in the format, you must supply the name of the program to be processed. The text below the format explains what kind of information the lower-case words represent.

3. Brackets [ ] indicate optional information.

4. Three dots indicate that information may be repeated.

The general format of all control statements described in this publication is:

Name	Operation	Operands
//	operation	[operand] [,operand] ...

//
Identifies the statement as a control statement. The slashes must appear in the first two columns of the control statement and must be followed by at least one blank column.

operation
Indicates the function of the control statement. For example, the word ASSGN indicates that the control statement specifies an I/O device assignment. The operation field can be up to five characters long and must be followed by at least one blank.

operands
Supply additional information about the control statement. For example, the operands of an ASSGN control statement specify the symbolic device address and the characteristics of the actual device. The operand field may be blank or may contain one or more operands, separated by commas, with no intervening blanks. A blank to indicate the end of the field must follow the last operand in the field. The field must not extend beyond column 71 of the punched card.

Programmer's Comments

To help you document your program, you may insert comments to the right of the control statements described in this publication. In doing so, follow these rules:

1. If the control statement has one or more operands, leave at least one blank column between the last operand and your comment. Example:

   ```
   // FILES SYS008,2 SKIP 2 TAPEMARKS
   ```

2. If an operand is not allowed, insert at least one blank column between the operation code and your comment. Example:

   ```
   // PAUSE PUT CARDS IN HOPPER
   ```

3. If you wish to omit an optional operand, you must insert a comma in place of the operand and then leave at least one blank column between the comma and your comment. Example:

   ```
   // DELET, DELETE ALL PERMANENT LABELS
   ```

Programmer's comments have no effect on the program. They are printed together with the control statements on the device assigned to SYSLOG. The comments must not extend beyond column 71 of the statement.

ORDER OF INPUT

The Job Control program precedes every job except the execution of an inquiry program.

It prepares the system for the execution of the following job by reading and processing the set of job control statements that you supply. These job control statements are always read on the device whose symbolic address is SYSRDR.

Figure 7 is a summary of these job control statements and their functions. A detailed description of each statement follows in the section Format of Job Control Statements.

Normally the first statement of a set of control statements for a particular job is a JOB statement. Only PAUSE, LOG, and NOLOG statements may precede a JOB statement. The last statement of a set of job control statements must be an EXEC statement.

Except where noted, the remaining job control statements of a set may be arranged in any order between the JOB and EXEC statements.

Operation Specification	Function
ASSGN	changes I/O device assignments in logical unit table
CONFG	places storage capacity specification into communication region
DATE	places date into communication region
DELET	causes permanent labels to be deleted from the label information area
DLAB	supplies disk label information for individual file
DSPLY	causes listing of all permanent labels contained in the label information area

Figure 7. Job Control Statements and Their Functions, Part 1 of 2

Operation Specification	Function
EXEC	indicates end of control statements and returns control to Monitor
FILES	positions magnetic tape reel by skipping specified number of tapemarks or by rewinding it
JOB	specifies name of program
LOG	causes listing of control statements on SYSLOG; allows tape error statistics, communications error statistics, and permanent labels to be listed on SYSLOG if listing has been requested
NOLOG	causes listing of control statements to be discontinued; prevents tape error statistics, communications error statistics, and permanent labels from being listed even if listings are requested
OPTN	indicates that the printout of tape error statistics is required by the job; indicates that the execution of a job is not to be interrupted by an inquiry program
PAUSE	causes immediate halt
TPLAB	supplies tape label information for individual file
UPSI	changes setting of UPSI byte in communication region
VOL	specifies name of file to be processed; specifies the symbolic address of the drive on which this file is mounted
XTENT	defines the extents used by a file on a disk pack; specifies the symbolic address of the drive on which this pack is mounted

Figure 7. Job Control Statements and Their Functions, Part 2 of 2

Figure 8 shows an example of how job control statements are inserted into the input deck.

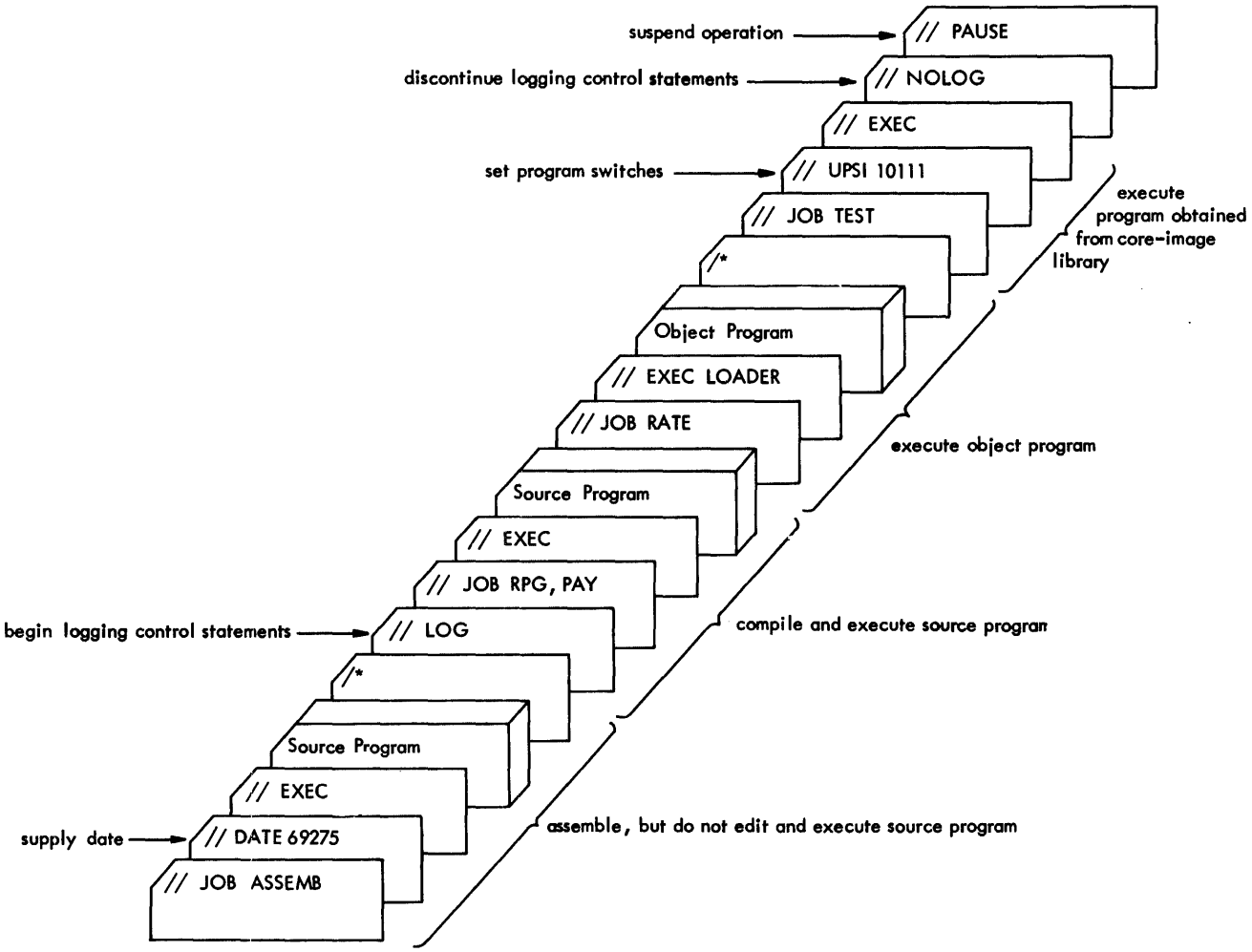


Figure 8. Example of Using Job Control Statements

FORMAT OF JOB CONTROL STATEMENTS

This section describes the format of each job control statement and explains its function.

JOB Control Statement

The JOB control statement indicates to the Job Control program that a set of control statements follows. The JOB control statement specifies the name of the program to be executed; the program name is placed into the communication region (bytes 24 through 29). The format of the JOB control statement indicates whether the program is to be executed, or assembled, or compiled. The JOB control statement has one of the following formats:

Name	Operation	Operands
//	JOB	program
//	JOB	ASSEMB,program
//	JOB	RPG,program
//	JOB	PL1,program

program

The name of the program to be executed. The name is not restricted in length. However, if the length exceeds six characters, only the leftmost six characters are recognized.

If the name appears by itself, it is the name of either a phase in the core-image library or an object program stored on cards or magnetic tape.

Dotted line flags planning information

If the name is preceded by the word ASSEMB, the program is a source program to be first assembled, then executed, provided no linking or relocation is necessary. If the name is preceded by the word RPG, the program is a source program to be first compiled, then executed. If the name is preceded by the word PL1, the program is a source program to be first compiled and link-edited, and then executed.

If the program is to be only assembled or compiled and not executed at once, the word ASSEMB or RPG appears alone as the program name. If the program is to be only compiled and/or link-edited, but not executed, the word PL1 appears alone as the program name.

If you wish to assemble or compile a source program and then execute it immediately (// JOB ASSEMB,program, // JOB RPG, program, or // JOB PL1, program) you must fulfill two requirements. First, a relocatable area must be on your system disk pack. The relocatable area is required to temporarily store the output from the Assembler, RPG, and PL/I programs. Second, the CMAINT program must be permanently stored in the core-image library. This program is used internally to store the Assembler, RPG, or PL/I output in the core-image library as a temporary entry. The object program is then loaded from the core-image library and executed automatically.

UPSI Control Statement

The UPSI (user program switch indicator) control statement is used to set the program switches in the communication region (byte 23). Each time the Job Control program is called it initializes the byte to binary zero. From the information you supply in the UPSI control statement, it then sets the specified bits (switches) to 1.

You can include one or several UPSI control statements at Job Control time. The program that follows must be one that is to be executed immediately, not just assembled or compiled. During the execution of the program, you can test the UPSI byte by using the COMRG or MVCOM Monitor macro instructions, and make further processing dependent on this test. For example, you can include in the problem program a routine that causes two output listings to be printed if the UPSI byte is set to 1. Whenever you run this problem program, you can insert the control statement // UPSI 1 into the deck of job control statements if you want two printouts, or omit the control statement if you want only one.

The format of the UPSI control statement is:

Name	Operation	Operand
//	UPSI	xxxxxxxx

xxxxxxxx
One to eight characters except blank or comma;

1 = corresponding bit set in UPSI byte; not 1 = corresponding bit remains unchanged.

You don't have to specify trailing not-1 bits.
Example: 1xx11xxx = 1xx11

DATE Control Statement

The DATE control statement is required for the first job that follows IPL. This control statement must contain the last two digits of the current year and the day of the year. The Job Control program records the month, the day, the year, and the day of the year in the communication region. This information is used in the jobs that follow to check the expiration dates of disk and tape labels.

RPG automatically dates reports from the date supplied in the DATE control statement. In programs written in the Assembler language, you can access the date by means of Monitor macro instructions and Assembler move instructions and use it for dating output listings and reports.

The format of the DATE control statement is:

Name	Operation	Operand
//	DATE	yyddd

yy
last two digits of the current year

ddd
the day of the year. Obtain this by adding the day of the month to the following constants:

Jan = 0	Jul = 181
Feb = 31	Aug = 212
Mar = 59	Sep = 243
Apr = 90	Oct = 273
May = 120	Nov = 304
Jun = 151	Dec = 334

Add one extra day after Feb 28 for Leap Year.

CONFG Control Statement

When you generate the Monitor, you also specify the storage capacity of your system. This specification is permanently stored in byte 9 of the communication region, and loaded into main storage as part of the Monitor. If you wish to change the storage specification temporarily from job to job (not on the system disk pack), you can insert a CONFG control statement at Job Control time with the format:

Name	Operation	Operand
//	CONFG	xx

xx
bytes of main storage
12 = 12,288
16 = 16,384
24 = 24,576
32 = 32,768

Since tape label information is stored in upper main storage, the CONFG control statement must precede all VOL and TPLAB statements.

OPTN Control Statement

The OPTN control statement has the format:

Name	Operation	Operands
//	OPTN	NOINQ,TES

NOINQ
No program may be executed as an inquiry program until the next Job Control run is completed.

TES
Indicates that tape error statistics are to be kept during the job that follows and printed before the next Job Control run.

You may specify one operand or two operands in any order separated by a comma.

We recommend that you take advantage of the Tape Error Statistics routine because it provides you with a powerful service aid. If your job does not require magnetic tapes, you can overlay the routine with the problem program.

As the sample printout shown in Figure 9 indicates, the addition of tape error statistics provides you with the means of determining the total activity on each magnetic tape unit and the number of errors encountered.

You may use the TES option in:

- all IBM-supplied system programs that support magnetic tape I/O
- problem programs written in RPG
- problem programs written in Assembler/IOCS.

LOG Control Statement

You must insert a LOG control statement with the format:

Name	Operation	Operand
//	LOG	

into the input deck if you want a listing of

- all control statements starting with the LOG statement itself
- tape error statistics

Tape Error Statistics

Tape Unit	Total No. of Requests	Read Error	Irrecoverable Read Error	Noise Record	Write Error	Erase Gaps
80	320	0	0	0	0	0
81	100	12	3	6	0	0
82	752	0	0	0	0	0
83	110	0	0	0	1	1

Units 84 and 85 were not used, and therefore no tape error statistics were printed for these devices.

Figure 9. Sample Tape Error Statistics Printout

- BSCA communications error statistics
- permanent labels.

You may place the LOG control statement either before the JOB statement or anywhere between the JOB statement and the EXEC statement.

The LOG control statement is effective only if a printer is assigned to SYSLOG.

The Job Control program continues printing the listings for all jobs until it encounters a NOLOG control statement.

NOLOG Control Statement

When the Job Control program encounters a NOLOG control statement, it stops the listing of subsequent control statements, tape error statistics, BSCA communications error statistics, and permanent labels for all jobs. The format of the NOLOG control statement is:

Name	Operation	Operand
//	NOLOG	

It may be placed either before the JOB statement or anywhere between the JOB statement and the EXEC statement.

The Job Control program resumes printing the listings only if it encounters another LOG control statement.

PAUSE Control Statement

The PAUSE control statement immediately interrupts processing. It only suspends operation and does not affect the contents of main storage. To resume operation, merely press the Start key on the CPU console.

The PAUSE control statement has the format:

Name	Operation	Operand
//	PAUSE	

The Job Control program executes the PAUSE statement as soon as it encounters it. You may place PAUSE anywhere in the set of job control statements, except within the control statement groups VOL, DLAB, XTENT and VOL, TPLAB.

EXEC Control Statement

The EXEC control statement indicates to the Job Control program that the reading of a set of control statements has been completed and that control is to be returned to the Fetch routine in the Monitor program. The EXEC control statement has the following formats:

Name	Operation	Operand
//	EXEC	
//	EXEC	R
//	EXEC	LOADER
//	EXEC	LOADER,R

No operand
: Disk-resident system: The program has been assembled or compiled and stored in the core-image library. Now it is read from the core-image library into main storage and executed. Card-resident system: The object program (in cards) is to be read on the card reading device assigned to SYSRDR. This is the only format you may use in a card-resident system.

R
: Used only for CMAINT and LNKEDT runs. The input is read from the relocatable area.

LOADER
: The execute-loader function is to be used. The problem program is read from the card reading device or magnetic tape drive assigned to SYSIPT. The CMAINT program, which must be on the system disk pack in this case, is internally used to store the problem program in the core-image library as a temporary entry. The program is then automatically loaded from the core-image library and executed.

LOADER,R
: The execute-loader function is to be used. The problem program is read from the relocatable area and executed. Since the problem program is temporarily stored in the core-image library before it is loaded into main storage, the CMAINT program must be contained in the core-image library.

The table in Figure 10 shows you the relationship between the JOB and EXEC control statements for a variety of jobs. The comments in the right column indicate what I/O device assignments are required to do each job.

1. // JOB ASSEMB // EXEC	Input from SYSIPT = Card or Tape Output onto SYSOPT = Card or Tape, and RL
2. // JOB LNKEDT // EXEC [R]	Input from SYSIPT = Card or Tape. If R is specified, input from RL. Output onto SYSOPT = Card or Tape, and RL
3. // JOB CMAINT // EXEC [R]	Input from SYSIPT = Card, Tape, or Disk. If R is specified, input from RL.
4. // JOB program // EXEC	Disk-resident system: Load program from core-image library and execute. Card-resident system: Load program from SYSRDR and execute.
5. // JOB program // EXEC LOADER[,R]	Read program from SYSIPT, temporarily include in core-image library, and execute. Input in card image from SYSIPT = Card or Tape. If R is specified, input from RL.
6. // JOB ASSEMB,program // EXEC	Assemble, temporarily include in core-image library, execute program. Input same as 1. Output from Assembler in RL.
7. // JOB RPG // EXEC	Input from SYSIPT = Card Output onto SYSOPT = Card and RL
8. // JOB RPG,program // EXEC	Compile, temporarily include in core-image library, execute program. Input same as 7. Output from RPG into RL and on SYSOPT.
9. // JOB PL1 // EXEC	Compile and/or link-edit source program, but do not execute. SYSIPT = Card or Tape. SYSOPT = Card or Tape
10. // JOB PL1,program // EXEC	Compile, link-edit and execute source program. SYSIPT = Card or Tape. Output on SYSOPT = Card or Tape and RL.

RL = Relocatable Area
Tape = Magnetic Tape

Figure 10. Relationship Between the JOB and EXEC Control Statements

As mentioned under points 5, 6, and 8 of Figure 10, phases may also be included temporarily in the core-image library. That means they are deleted automatically when the CMAINT program is used again. These temporary phases can only be executed in the job that immediately follows.

When the procedure listed in point 5, 6, or 8 is used, the Job Control program sets special switch indicators. From this information, the CMAINT program sets a temporary job switch. This switch is always tested by the Fetch routine. If it is on, only temporary phases can be loaded. Therefore, mixed fetching of permanent and temporary phases is never possible. The advantage of temporary phases is that they are deleted from the core-image library after execution, leaving more room in the library for permanent phases that are executed frequently.

I/O DEVICE ASSIGNMENTS

When you generate a Monitor, you also generate up to 26 permanent device assignments.

Devices are assigned by means of the LUB table and the PUB table in the Monitor. The LUB table contains the symbolic device addresses of all I/O devices and a pointer to their physical device addresses.

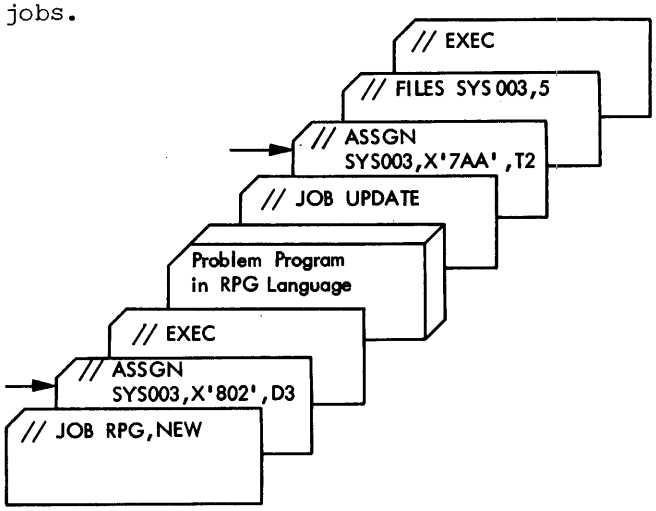
If a LUB refers to a disk or tape drive, the pointer points to the address of the PUB that contains the physical device address. If a LUB refers to any other device, the pointer points to the physical device address contained in the LUB itself.

Before each program is executed, the ASSGN control statements submitted for each job cause the Job Control program to insert into the respective LUB the pointer that

Dotted line flags planning information

points to the physical device address submitted in the ASSGN control statement. This freedom gives you great flexibility in switching from one physical device to another, without altering the logic of the program.

Figure 11 shows an example of how I/O devices were assigned for two specific jobs.



●Figure 11. Example of I/O Device Assignment

ASSGN Control Statement

When you generate a Monitor, you also generate permanent device assignments by means of ASSGN macro instructions. These permanent device assignments are written onto the system disk pack and loaded into main storage together with the Monitor at IPL time.

Permanent device assignments loaded into main storage are in effect for all jobs, unless you alter them by submitting ASSGN control statements to the Job Control program. These new device assignments then remain in effect for all subsequent jobs unless you change them again by submitting other ASSGN control statements. If the operator reloads the Monitor into main storage through an IPL run, the permanent device assignments written on the system disk pack are reloaded into main storage and take effect, cancelling any assignments that may have been made by ASSGN control statements.

The temporary device assignments alter only the device assignments in main storage, not the permanent device assignments on the system disk pack. If you want to alter the permanent device assignments, you either have to run the PSERV program or generate a new Monitor.

The formats of the ASSGN control statement used to alter device assignments temporarily are:

Name	Operation	Operands
//	ASSGN	SYSxxx,X'cuu',dd[,X'ss']
//	ASSGN	SYSxxx,UA

SYSxxx	symbolic device address:
SYSRES	system disk pack containing disk-resident system (assigned only at IPL time)
SYSRDR	card reading device for reading job control statements
SYSIPT	input device
SYSOPT	output device
SYSLST	printer for output listings
SYSLOG	printer for logging statements and statistics
SYS000 : SYS019	other I/O devices

X'cuu'	physical device address:
c	attachment point:
1	2501 Card Reader
2	2520 or 2560
3	1442 Card Punch
4	1403 or 2203 Printer
7	2415 Magnetic Tape Unit
8	2311 Disk Storage Drive
uu	unit:
01	disk
02	disk
03	disk
04	disk
08 . . FD	magnetic tape
00	all other devices

dd	device type:
D3	2311 Model 11
D4	2311 Model 12
L1	1403 Printer
L3	2203 Printer
P2	1442 Card Punch
P3	2520 Card Punch
R4	2501 Card Reader
R5	2520 Card Read Punch
R6	2560 MFCM Primary Feed
R7	2560 MFCM Secondary Feed
T1	2415 7-track magnetic tape
T2	2415 9-track magnetic tape

X'ss' specification for 7-track tape:				
Code	BPI	Parity	Translate	Convert
X'10'	200	odd	off	on
X'20'	200	even	off	off
X'28'	200	even	on	off
X'30'	200	odd	off	off
X'38'	200	odd	on	off
X'50'	556	odd	off	on
X'60'	556	even	off	off
X'68'	556	even	on	off
X'70'	556	odd	off	off
X'78'	556	odd	on	off
X'90'	800	odd	off	on
X'A0'	800	even	off	off
X'A8'	800	even	on	off
X'B0'	800	odd	off	off
X'B8'	800	odd	on	off

X'ss' specification for 9-track tape:	
C0	1600 BPI
C8	800 BPI

UA	unassigned
The pointer to the corresponding PUB is changed to a dummy entry.	

Examples: The following examples illustrate how to use the ASSGN control statement.

1. // ASSGN SYS001,X'708',T2,X'C0'

   This control statement assigns SYS001 to a 2415 9-track magnetic tape drive whose unit number is 08, specification 1600 bytes per inch.

2. // ASSGN SYS002,X'802',D3

   This control statement assigns SYS002 to a 2311 disk drive, Model 11, whose unit number is 02. We assumed that this Monitor supports at least two disk drives. Otherwise the assignment would have to read X'801'.

3 // ASSGN SYS003,UA

   This control statement releases SYS003 from a device assignment.

FILES Control Statement

The FILES control statement rewinds a reel of magnetic tape or positions it on the tape drive. For example, a tape can be positioned at the beginning of the first file or the fourth file on the reel by using this control statement. Since the FILES control statement is executed as soon as it is encountered, the magnetic tape drive to which the FILES statement refers must be correctly assigned. The formats of a FILES control statement are:

Name	Operation	Operand
//	FILES	SYSxxx,nnn
//	FILES	SYSxxx,REW

SYSxxx
   Symbolic device address of the tape drive on which the reel of tape is mounted. For the correct symbolic device address, see the ASSGN control statement.

nnn
   A number from 1-999 representing the number of tapemarks to be skipped, counting from the position of the tape when the statement is read. (In unlabeled files, a tapemark follows each file. In labeled files, a tapemark also follows the label).

REW
   Causes the specified tape to be rewound.

LABEL INFORMATION PROCESSING

Whenever you use disk or labeled tape files in a program, you must include information about the labels of these files in the job control statements that precede the problem program. The Job Control program processes this information and supplies it to the label processing routines of system programs.

For each disk file, you must supply at least three control statements: VOL, DLAB, and XTENT. For each indexed sequential

file, however, you must supply at least two XTENT statements, one for the prime data area and one for the cylinder index.

For each labeled tape file, you must supply two control statements: VOL and TPLAB.

The order in which these control statements must be inserted is mandatory:

1. VOL
2. DLAB or TPLAB
3. XTENT for every disk area, in the order in which these areas are to be used.

The following sections describe how to code each of these control statements.

VOL Control Statement

You must insert a VOL control statement for every labeled tape and disk file used in the program. The VOL control statement indicates the symbolic address of the device on which the volume (disk pack or tape reel) is mounted, and identifies the file by name.

The format of the VOL control statement is:

Name	Operation	Operand
//	VOL	SYSxxx,filname

SYSxxx
Symbolic address of the device on which the volume is mounted. For the correct symbolic device address, see the ASSGN control statement.

filname
Name of the file to be processed. May be 1-7 characters long. The file name is the same as the file name on an RPG File Description Specifications form or in the DTF file definition statement of the IOCS.

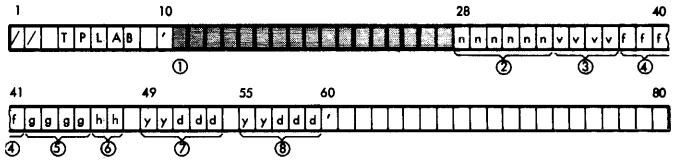
If SYSxxx refers to a magnetic tape drive, a TPLAB control statement is expected as the next control statement. If it refers to a disk drive, a DLAB statement is expected to follow.

TPLAB Control Statement

The TPLAB control statement contains information about the tape file specified in the VOL control statement. This information is the same as the information contained in fields 3-10 of the standard IBM tape file label fully described in Appendix D.

The format of the TPLAB control statement is shown below. Note that the TPLAB statement, like all other job control statements, is variable in format, not fixed. This means that you may insert more than one blank between the name field and the operation field, and between the operation field and the operand field. But to help you count the columns, only one blank has been used to separate the fields.

The format of the TPLAB control statement is:



① Col.10-27. Insert an apostrophe followed by unique identification for file. Seventeen characters long including blanks.

② Col.28-33. Six-byte numeric code identical to number written on exterior of tape reel.

③ Col.34-37. Volume sequence number. 0001 for first volume of this file, 0002 for second volume, etc.

④ Col.38-41. File sequence number. 0001 for first file of multi-file set, 0002 for second file, etc.

⑤ Col.42-45. Code starting with 0001 representing which edition of the file this is.

⑥ Col.46-47. Code starting with 01 representing which version of the file this is. Followed by one blank.

⑦ Col.49-53. Creation date of file (for format see DATE control statement). Followed by one blank.

⑧ Col.55-60. Expiration date of file (for format see DATE control statement). Followed by apostrophe.

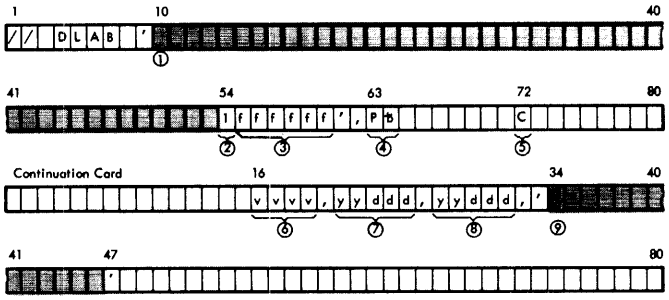
DLAB Control Statement

The DLAB control statement contains information about the disk file specified in the VOL control statement.

This information is similar to the information contained in fields 1-8 of the standard IBM disk file label fully described in Appendix E.

The format of the DLAB control statement is shown below. Note that the DLAB statement, like all other job control statements, is variable in format, not fixed. This means that you may insert more than one blank between the name field and the operation field, and between the operation field and the operand field. But to help you count the columns, only one blank has been used to separate the fields.

The format of the DLAB control statement is:



First Card:

① Col.9-53. Insert an apostrophe followed by unique identification for file. Forty-four characters long including blanks.

② Col.54. Insert decimal 1.

③ Col.55-62. Six-byte file serial number. Followed by apostrophe, followed by comma.

④ Col.63-64. Insert the code P followed by a blank if this label is to be permanent. Leave both columns blank if this label is to be temporary.

⑤ Col.72. You must always insert a continuation punch in this column.

Continuation Card:

⑥ Col.16-19. Must always begin in column 16. Four-byte volume sequence number followed by a comma.

⑦ Col.71-25. Creation date of file (for format see DATE control statement). Followed by a comma.

⑧ Col.27-31. Expiration date of file (for format see DATE control statement).

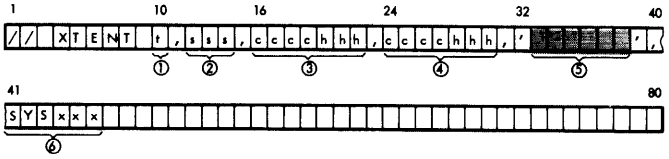
⑨ Optional code indicating which programming system is used. Code up to 13 characters long. If you use this field, you must insert a comma in col.32 and enclose the code in apostrophes.

XTENT Control Statement

In addition to a VOL control statement and a DLAB control statement, you must insert at least one XTENT control statement for sequential or direct access files and at least two XTENT control statements for an indexed sequential file (one for the prime data area, one for the cylinder index).

The format of the XTENT control statement is shown below. Note that the XTENT statement, like all other job control statements, is variable in format, not fixed. This means that you may insert more than one blank between the name field and the operation field, and between the operation field and the operand field. But to help you count the columns, only one blank has been used to separate the fields.

The format of the XTENT control statement is:



① Col.10. Insert code for type of extent followed by a comma:
Code 1 = data extent
Code 2 = independent overflow extent
Code 4 = cylinder-index extent

② Col.12-14. Extent sequence number 000-255 followed by a comma.

③ Col.16-22. Begin address of extent. Followed by a comma.

④ Col.24-30. End address of extent. Followed by a comma and an apostrophe.

⑤ Col.33-38. Volume serial number followed by an apostrophe and a comma.

⑥ Col.41-46. Symbolic address of drive on which disk pack containing this extent is mounted (see ASSGN control statement).

Here are some additional guidelines to help you code the XTENT control statement correctly.

For every direct-access file and for every sequential disk file, you must insert at least one XTENT statement with the code 1 in column 10.

For every indexed sequential file you must insert at least two XTENT statements. The first statement is for the prime data

area. It must contain the code 1 in column 10. The second statement is for the cylinder index. It must contain the code 4 in column 10.

If you specify an overflow extent (type 2) you must also specify a cylinder-index extent (type 4). In this case the data extents (type 1) must be on cylinder boundaries, that is, the begin address must be cccc000 and the end address must be cccc009.

The volume serial number of the first XTENT statement must be identical to the file serial number of the DLAB statement (col. 55-62).

If several extents refer to the same file, they must be specified in ascending order. Example:

```
XTENT 1,001,0011003,0011006,'ssssss',SYS002
XTENT 1,003,0201000,0202001,'ssssss',SYS005
```

One disk volume can contain many files, each with its own XTENT statement. The volume serial number ('ssssss') must be the same for all these files. The symbolic device addresses (SYSxxx) may be different for these files, but the physical device assignments (X'cuu') in the ASSGN statements must be the same. The correct assignments in our example would be:

```
ASSGN SYS002,X'801',D3
ASSGN SYS005,X'801',D3
```

When you create a disk file and specify its extents, don't attempt to overwrite the volume label, the label information area (LIA), the Volume Table of Contents (VTOC), or the alternate track area.

The volume label described in Appendix C always begins on cylinder 0, track 0 of the disk pack. The alternate track area always begins on cylinder 0, track 1 and extends over three tracks. The extents occupied by the LIA and the VTOC depend on how you initialized your disk pack.

If you attempt to overwrite these areas, Job Control will print a halt and issue a warning message.

Figure 12 shows you several examples of how to code VOL, TPLAB, DLAB, and XTENT control statements. If you study these examples, what we discussed in this section will become clearer.

PERMANENT AND TEMPORARY DISK LABEL INFORMATION

For every disk file to be opened during a job, the pertinent label information must be available in the label information area of the system disk pack so that the label-processing routines can check or create the file labels in the job that follows.

The disk label information you supply by means of job control statements may be either temporary or permanent.

Permanent label information is located at the beginning of the label information area. It remains in that area until it is deleted by the DELET control statement described later. Temporary label information immediately follows permanent label information in the label information area. It is cleared in each successive Job Control run.

Permanent label information as well as temporary label information may be added to the label information area in any Job Control run.

To identify permanent label information, the DLAB control statement must contain the code P% in the two-character field following the file serial number. If both permanent and temporary label information control statements are specified in the same Job Control run, the control statements providing permanent label information must precede those providing temporary label information. The sequence is checked and a halt occurs when a sequence error is detected, causing the Job Control run to be discontinued.

The file names of different label information blocks in the label information area must never be identical. The Job Control program checks the file names of any label information to be added against the file names of all permanent label information blocks already located in the label information area. If the file name of the label information to be added differs from the file name of all existing permanent label information blocks, the new label information is added adjacent to the last label information block. If the new label information has the same file name as a permanent label block, a halt occurs. The operator then has the option of ignoring the new label information or else overwriting the old permanent label block, but only if the new label information is to be stored permanently. If the new label information is to be stored temporarily, the Job Control run is discontinued.

We recommend that you use permanent label information (1) in cases where a job that processes the same disk file is executed frequently, and (2) for inquiry programs.

```
|1        1         2         3         4         5         6         7         8|
|         0         0         0         0         0         0         0         0|
|// JOB   FIRST                                                                   |
|// ASSGN SYS001,X'708',T2                                                        |
|// FILES SYS001,1                                                                |
|// VOL   SYS001, KEY                                                             |
|// TPLAB 'KEY TAPE FILE    00038400010001000102 65085 66085'                     |
|// EXEC                                                                          |
|                                                                                 |
|// JOB   SECOND                                                                  |
|// ASSGN SYS002,X'801',D3                                                        |
|// ASSGN SYS005,X'801',D3                                                        |
|// VOL   SYS002,INPUT        MULTI-VOL, SINGLE DRIVE                             |
|// DLAB  'INPUT DATA SET                              1SSSSSS',P       C         |
|               0001,65032,66033,'0000000000000'                                  |
|// XTENT 1,001,0011003,0011006,'SSSSSS',SYS002                                   |
|// XTENT 1,003,0201000,0202009,'SSSSSS',SYS005                                   |
|// XTENT 1,008,0050008,0060006,'TTTTTT',SYS002                                   |
|// EXEC                                                                          |
|                                                                                 |
|// JOB   THIRD                                                                   |
|// ASSGN SYS019,X'802',D3                                                        |
|// VOL   SYS002,OUTPUT       MULTI-VOL, TWO DRIVES                               |
|// DLAB  'OUTPUT DATA SET                              1SSSSSS',        C         |
|               0001,65032,66033,'0000000000000'                                  |
|// XTENT 1,004,0023001,0023001,'SSSSSS',SYS005                                   |
|// XTENT 1,099,0050008,0060006,'A12BCZ',SYS019                                   |
|// EXEC                                                                          |
```

●Figure 12. Examples of VOL, TPLAB, DLAB, and XTENT Control Statements

DELET Control Statement

You have the option of deleting permanent labels from the label information area by means of the job control statement:

Name	Operation	Operands
//	DELET	[filename1,filename2,...]

If the operand field is blank, all permanent labels will be deleted from the label information area.

If the operand field contains a file name, the permanent label information for that file will be deleted.

The DELET control statement enables you to make room for new labels if the label information area is full, or if the file to which the label refers has expired.

You may insert the DELET control statement anywhere between the JOB and EXEC statements, but (1) not after the statement group VOL, DLAB, XTENT, and (2) not between VOL and TPLAB.

DSPLY Control Statement

You have the option of displaying all permanent labels by using the contrcl statement:

Name	Operation	Operand
//	DSPLY	

You may insert the DSPLY control statement anywhere between the JOB and EXEC control statements, but (1) not between VOL, DLAB, and XTENT, and (2) not between VOL and TPLAB.

The DSPLY statement is executed as soon as it is encountered. The information displayed consists of the file name, the symbolic device address, the expiration date, the extent type, and the extents of the pertinent file.

The DSPLY statement enables you to determine which label information is permanent and to check the expiration date of your files.

Disk Initial Program Loader (IPL)

At the beginning of a system run, the disk Initial Program Loader (IPL) loads the Monitor from the system disk pack into main storage. The Monitor then remains in main storage throughout the system run in order to control processing.

IPL is an IBM-supplied program in two parts. Part 1 consists of three punched cards, Part 2 resides on the system disk pack on cylinder 0, track 0.

The card deck required to initiate program loading is shown in Figure 13.

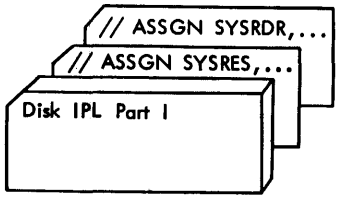


Figure 13. Card Deck Required to Initiate Program Loading

The first three cards are Part 1 of IPL. Then follow the ASSGN control statements for SYSRES and SYSRDR.

You load the first card by setting the address switches on the CPU console to an even address lower than X'1000' and pressing the Load key. The cards that follow are automatically loaded to a fixed location in main storage.

After IPL Part 1 is loaded, it loads IPL Part 2 from the system disk pack into main storage. IPL Part 2 then loads the Monitor into lower main storage.

The IPL then fills the logical unit blocks associated with SYSRES and SYSRDR and transfers control to the Fetch routine of the Monitor, which calls the Job Control program to begin processing the job control statements for the first job to be executed. After this, the Monitor remains in main storage throughout the current system run. The Job Control program is called automatically after each job.

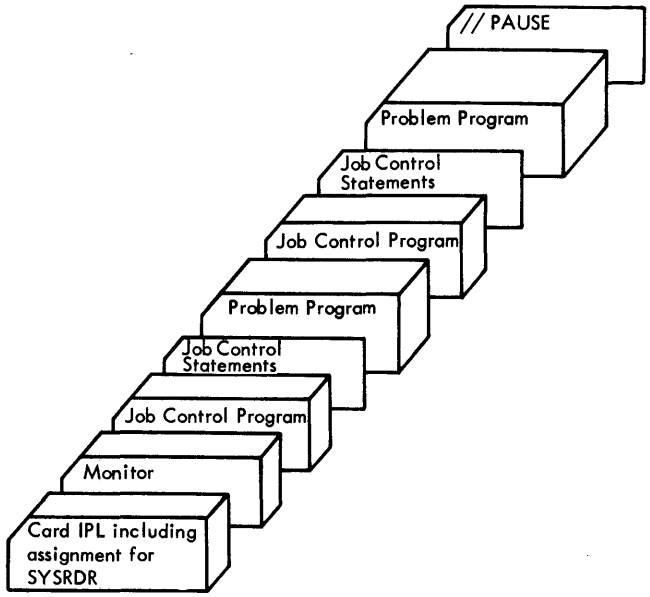
If a system run is discontinued (not temporarily suspended by a PAUSE control statement, or by one job being discontinued), the Initial Program Loader must be used to re-initiate system operation, that is, to initiate the subsequent system run.

Card Initial Program Loader (IPL)

The IPL program for the card-resident system is generated together with the card-resident Job Control program and the card-resident Monitor program.

The card-resident IPL program loads the card-resident Monitor into main storage.

Figure 14 shows you how to arrange the input deck to initiate program loading. Place the generated IPL deck on the card reading device to be assigned to SYSRDR. If you wish to change the generated assignments included in the IPL deck, replace the ASSGN cards for SYSRES and SYSRDR. If you do not wish to process disk file labels, you must unassign SYSRES (// ASSGN SYSRES,UA).



●Figure 14. Typical Input Deck for Card-Resident System

Libraries and Relocatable Area

As an introduction to the DPS service programs described in this publication, this section discusses the core-image library, the macro library, and the relocatable area, all three of which are allocated, managed, and maintained by the DPS service programs.

Core-Image Library

This section explains the purpose, organization, and use of the core-image library.

PURPOSE OF THE CORE-IMAGE LIBRARY

All executable programs are permanently stored in the core-image library. These programs are in core-image format; that is, they can be loaded directly into main storage by the Fetch routine and executed without preliminary cross-referencing and relocation.

You can store in the core-image library all IBM-supplied programs that you need for the operation of your system. You can also store problem programs there that you run frequently. Once the problem program is stored, each phase can be executed either by itself or in combination with other phases.

A problem program goes through the following states before it is placed into the core-image library.

1. A problem program is written as a set of source statements in RPG or Assembler language or PL/I. To make it easier to write and test large programs, a program in Assembler language can be written in parts, each part consisting of one or more complete control sections. Such parts are called source modules. An Assembler source program may consist of one or more source modules. An RPG source program can be only one source module.

2. The Assembler translates the source modules into a set of object statements (machine-language statements). The set of object statements, which again is made up of one or more complete control sections, is called an object module. An object program can consist of one or more object modules. An RPG compilation produces one object module with all linkages resolved.

3. Then the Linkage Editor processes the object module, performing cross-referencing and relocation where necessary. The output of the Linkage Editor is called an executable program and consists of one or more program phases.

4. Programs which are to be disk resident are then placed in the core-image library by means of the Core-Image Maintenance program.

The core-image library is also used internally by system programs to temporarily store an object module before it is executed. This special application is described in more detail in the section Use of the Core-Image Library.

ORGANIZATION OF THE CORE-IMAGE LIBRARY

The core-image library is stored in the library section of the system disk pack. Since the program phases in the library are variable in length and random in order, a core-image directory is used to record the location of each program phase in the core-image library. The phase-name entries in the core-image directory are arranged according to the collating sequence. A Library Management program (CMAINT) updates the directory whenever a phase is added to or deleted from the core-image library.

There is one 30-byte entry in the core-image directory for each program phase in the core-image library. Each entry contains the following information:

1. Identifier for permanent or temporary phase.

2. Name of the phase.

3. Starting disk address of the phase.

4. Starting main storage address for loading the phase.

5. Control transfer address.

6. Number of sectors completely occupied by the phase. (Each sector consists of 270 bytes, and there are 10 sectors to a track).

7. Number of bytes occupied by the phase in the last sector.

8. Version identification of the phase.

Dotted line flags planning information

The Fetch routine uses the core-image directory to determine where a phase to be loaded is located, how much storage it will occupy, and where it is to be placed in main storage.

USE OF THE CORE-IMAGE LIBRARY

The JOB and EXEC control statements and the FETCH macro instruction are used to initiate the loading of program phases from the core-image library into main storage for execution. The JOB and EXEC control statements are explained in the section Job Control Program; the Fetch routine called by the FETCH macro instruction is explained in the section Monitor Program.

The diagram in Appendix J shows the various methods of using the JOB and EXEC statements in relation to the DPS.

If a source module is to be both assembled and executed, the following control statements are required.

```
// JOB        ASSEMB,program
// EXEC
```

This module is then temporarily included in the core-image library and can only be executed immediately.

If a source module is to be both compiled and executed, the following control statements are required.

```
// JOB        RPG,program
// EXEC
```

or

```
// JOB        PL1,program
// EXEC
```

This module is then temporarily included in the core-image library and can only be executed immediately.

If an object module stored on a medium other than the system disk pack is to be executed, the following control statements are required.

```
// JOB        program
// EXEC       LOADER
```

This module is then temporarily included in the core-image library and can only be executed immediately.

If a phase is a permanent item of the core-image library and is a complete program or the first phase of a multiphase program, the following control statements are required.

```
// JOB        phasnam
// EXEC
```

If a phase is a permanent item of the core-image library and is the second or a subsequent phase of a multiphase program, the following macro instruction must be used in the text of the preceding phase.

```
FETCH    phasnam
```

Within a job, either permanent or temporary phases can be loaded from the core-image library, not both. Mixed fetching of permanent and temporary phases is not possible.

Each phase can consist of up to nine subphases. Each subphase can be called into main storage only by means of the FETCH macro instruction without an operand. This loads the next consecutive subphase.

Phases of programs compiled by the PL/I compiler -- called segments -- are loaded by issuing the PL/I statement CALL OVERLAY.

Macro Library

This section explains the purpose, organization, and use of the macro library.

PURPOSE AND USE OF THE MACRO LIBRARY

A macro definition is a sequence of Assembler and macro-language statements that is given an identifier and placed into the macro library. A macro instruction in an Assembler-language program causes the generation of a sequence of Assembler-language statements depending on parameters which are specified as operands in the macro instruction.

The macro library may contain IBM-supplied macro definitions that refer to the Monitor and/or the logical IOCS, and user-written macro definitions.

Macro definitions must be written in the macro language, which is an extension of the Assembler language. The macro language is described in the publication IBM System/360 Model 20, Disk and Tape Programming Systems, Assembler Language, Form C24-9002.

ORGANIZATION OF THE MACRO LIBRARY

The macro library is stored in the library section of the system disk pack. A macro directory is used to record the location of macro definitions in the macro library. The macro-name entries in the macro direc-

tory are arranged according to the collating sequence.

The macro directory consists of 15-byte entries, one entry for each macro definition in the library. The entry contains the following information:

1. Macro name.
2. Starting disk address of the macro definition.
3. Number of sectors occupied by the macro definition. (Each sector consists of 270 bytes, and there are 10 sectors to a track.)
4. Number of bytes used in the last sector.
5. Version and modification level of the macro definition.

The Assembler program uses the macro directory to retrieve the macro definition from the library when the program that contains the macro instruction is assembled.

Relocatable Area

The relocatable area is located on the system disk pack. You can allocate its size through the Library Allocation Organization program (AORGZ).

The relocatable area is used as work area by the RPG, the Linkage Editor program, and by PL/I.

When you assemble, compile, or link-edit a problem program, the object program is stored in the relocatable area. This enables you to perform additional job steps, such as cataloging the object program in the core-image library, with a minimum of card handling.

You also have the option of using the relocatable area as input area for the Linkage Editor and Core-Image Maintenance programs. These options are described in the sections that deal with these programs.

The diagram in Appendix J shows several ways in which the relocatable area may be used.

This section describes the functions of the DPS service programs, and shows you what job and program control statements to use in order to perform a certain task.

In the first part, the service programs that deal with the libraries and directories are grouped together. These are: the Directory Service program (DSERV), the Core-Image Service program (CSERV), the Macro Service program (MSERV), the Core-Image Maintenance program (CMAINT), the Macro Library Maintenance program (MMAINT), and the Library Allocation Organization program (AORGZ).

In the second part, individual system programs that will help you operate the disk-resident system are described one by one. These are: the Physical and Logical Unit Tables Service program (PSERV), the Load System Disk program (LDSYS), the Backup and Restore program (BACKUP), the Linkage Editor program (LNKEDT), and the Copy System Disk program (COPSYS).

The boxes drawn with heavy black lines contain the job control and program control statements you need to run the program in question. All statements that require further explanation are described in detail below these boxes.

Directory Service Program (DSERV)

The Directory Service program (DSERV) lists the contents of the system directory, the core-image directory, and the macro directory on the printer. Each directory entry is listed on a separate line.

JOB CONTROL STATEMENTS

The job control statements required for a DSERV run are:

```
// LOG            optional
// JOB            required
// DATE           1st job after IPL only
// ASSGN SYSLST   required
// ASSGN SYSLOG   optional
// EXEC           required
```

JOB Control Statement

The JOB control statement used to request a directory service function has the following format:

Name	Operation	Operand
//	JOB	DSERV

DSERV
Indicates that one or more directories are to be listed.

ASSGN Control Statements

You must assign SYSLST to the printer so that the directories can be listed. We recommend that you also assign SYSLOG to the printer. If these assignments are already effective, you need not submit them again.

The remaining job control statements are not described here. You can find their formats in the section Job Control Program.

PROGRAM CONTROL STATEMENTS

The program control statements required for a DSERV run are:

```
// DSPLY          required
// END            required
```

DSPLY Control Statement

The DSPLY control statement specifies which directories are to be listed. One, two, or all three of the directories can be specified. The DSPLY control statement has one of the following formats:

Name	Operation	Operands
//	DSPLY	ALL
//	DSPLY	SD [,CD] [,MD]

ALL
Indicates that all three directories are to be listed.

SD
Indicates that the system directory is to be listed.

CD
Indicates that the core-image directory is to be listed.

MD

Indicates that the macro directory is to be listed.

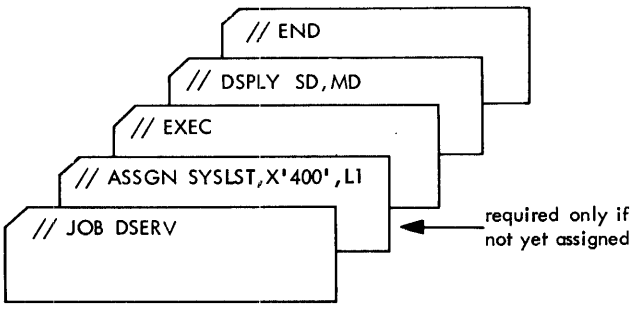
When two or three directories are specified on one DSPLY control statement, the order in which they are specified is not significant. You may also use separate control statements.

END Control Statement

The format of the END control statement is

Name	Operation	Operand
//	END	

Figure 15 is an example of using job and program control statements to list the system directory and the macro directory.



●Figure 15. Example of Using Control Statements to Request Directory Service Functions

Core-Image Service Program (CSERV)

The Core-Image Service program (CSERV) lists the contents of the core-image library on the printer, or writes or punches the contents on the device assigned to SYSOPT.

When the contents are listed, the printer lists the entries in the core-image directory, and then lists each phase in the core-image library together with its subphases in hexadecimal notation.

When the contents are punched or written, the output consists of a card deck or a card-image file on magnetic tape or disk, depending on your specifications.

You can use the output of a CSERV run as input to the CMAINT and LDSYS programs.

JOB CONTROL STATEMENTS

The job control statements required for a CSERV run are:

```
// LOG            optional
// JOB            required
// DATE           1st job after IPL only
// ASSGN SYSOPT   depends on job
// ASSGN SYSLST   depends on job
// ASSGN SYSLOG   optional
// VOL
// DLAB           depends on job
// XTENT
// OPTN TES       recommended for tapes
// EXEC           required
```

JOB Control Statement

The JOB control statements used to request a core-image service function has the following format:

Name	Operation	Operand
//	JOB	CSERV

CSERV

Indicates that one or more phases, with their subphases, or the entire contents of the core-image library are to be listed, written, or punched.

ASSGN Control Statements

To punch or write the output, assign SYSOPT to a card punching device, a magnetic tape drive, or a disk drive.

To obtain a listing of the output, assign SYSLST to the printer.

We recommend that you also assign SYSLOG to the printer.

If these assignments are still effective, you need not submit them again.

VOL, DLAB, XTENT Control Statements

You must supply disk label information if SYSOPT refers to a disk drive. The file name you insert must be SERVO.

The remaining job control statements are not described here. You can find their formats in the section Job Control Program.

PROGRAM CONTROL STATEMENTS

The program control statements required for a CSERV run are:

```
// DSPLY  }
// PUNCH  }      depends on job
// IPL    }
// MONTR  }
// END           required
```

DSPLY Control Statement

The DSPLY control statement specifies which phase is to be listed. The DSPLY control statement has one of the following formats:

Name	Operation	Operands
//	DSPLY	ALL
//	DSPLY	phase
//	DSPLY	phase.ALL
//	DSPLY	

ALL
Indicates that all phases are to be listed.

phase
Name of the phase to be listed together with its subphases. The name may be up to six characters long. The first character must be alphabetic, the others may be alphabetic or numeric.

phase.ALL
Indicates that all phases whose names begin with the characters preceding .ALL are to be listed. The name may be up to six characters long.

No Operand
Indicates that another control statement follows immediately. This control statement must have one of the following formats:

Name	Operation	Operand
//	IPL	
//	MONTR	

IPL
Indicates that the disk-resident IPL program is to be listed.

MONTR
Indicates that the disk-resident Monitor program is to be listed together with its Job Closing routines (SYSEND) and transient phases ($$$...).

PUNCH Control Statement

The PUNCH control statement specifies which phase is to be punched into cards or written onto disk or magnetic tape. The PUNCH statement has one of the following formats:

Name	Operation	Operands
//	PUNCH	ALL
//	PUNCH	phase
//	PUNCH	phase.ALL
//	PUNCH	

ALL
Indicates that all phases are to be punched or written.

phase
Name of the phase to be punched or written together with its subphases. The name may be up to six characters long. The first character must be alphabetic, the others may be alphabetic or numeric.

phase.ALL
Indicates that all phases whose names begin with the characters preceding .ALL are to be punched or written. The name may be up to six characters long.

No Operand
Indicates that another control statement follows immediately. This control statement must have one of the following formats:

Name	Operation	Operand
//	IPL	
//	MONTR	

IPL
Indicates that the disk-resident IPL program is to be punched or written.

MONTR
Indicates that the disk-resident Monitor program is to be punched or written together with its Job Closing routines (SYSEND) and transient phases ($$$...).

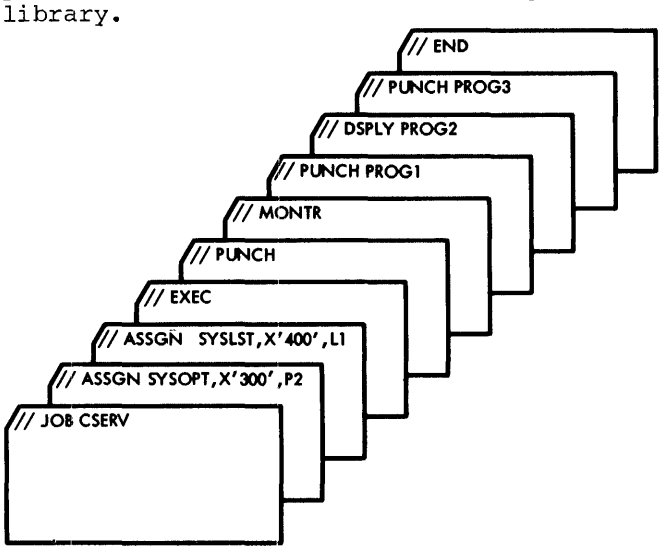
PUNCH statements for the IPL and the Monitor program must precede all other PUNCH statements. If both IPL and MONTR statements are to be used, IPL must be specified before MONTR.

END Control Statement

The format of the END control statement is:

Name	Operation	Operand
//	END	

Figure 16 is an example of using job and program control statements to list and punch the contents of the core-image library.



●Figure 16. Example of Using Control Statements to Request Core-Image Service Functions

OUTPUT ON SYSOPT

The output of the CSERV program on SYSOPT contains the following cards or card-image records:

IPL program
- // IPL identification
- TXT cards
- END card

Monitor
- // MONTR identification/features
- TXT cards
- END card
- all transient and Job Closing phases belonging to Monitor

Other phases
- // CATAL
- PHASE card
- TXT cards
- END card
- TXT cards for all subphases
- END card

The formats of these cards are described in the section Linkage Editor Program.

The CSERV program writes or punches an EOF record at the end of the output file.

Output	EOF Record
card deck	/* and // END
magnetic tape	// END and tapemark
disk	/*

Macro Service Program (MSERV)

The Macro Service program (MSERV) lists the contents of the macro library on the printer, or writes or punches the contents on SYSOPT.

When the contents are listed, the printer lists the entries in the macro directory, and then lists each macro definition either in internal format or in source format, whichever you specify.

When the contents are written or punched, the output consists of a card deck or a card-image file on magnetic tape or disk, depending on your specifications.

You can use the output of an MSERV run as input to the MMAINT program.

JOB CONTROL STATEMENTS

The job control statements required for an MSERV run are:

// LOG	optional
// JOB	required
// DATE	1st job after IPL only
// ASSGN SYSOPT	depends on job
// ASSGN SYSLST	depends on job
// ASSGN SYSLOG	optional
// ASSGN SYS000	depends on job
// VOL	
// DLAB	
// XTENT	
// OPTN TES	recommended for tapes
// EXEC	required

JOB Control Statement

The JOB control statement used to request a macro service function is:

Name	Operation	Operand
//	JOB	MSERV

MSERV
Indicates that one or more macro definitions or the entire contents of the macro library are to be listed, written, or punched.

ASSGN Control Statements

To punch or write the output, assign SYSOPT to a card punching device, a magnetic tape drive, or a disk drive.

If the output is to be in source format, assign SYS000 to a disk drive.

To obtain a listing of the output, assign SYSLST to the printer. We recommend that you also assign SYSLOG to the printer.

If these assignments are still effective, you need not submit them again.

VOL, DLAB, XTENT Control Statements

You must supply disk label information if SYSOPT refers to a disk drive. The file name you insert must be SERVO.

If the output is to be in source format, you must assign a work area with the file name WORKM to SYS000, and supply the corresponding disk label information.

The remaining job control statements are not described here. You can find their correct format in the section Job Control Program.

PROGRAM CONTROL STATEMENTS

The program control statements required for an MSERV run are:

```
// DSPLY   )
// PUNCH   )
// INTERN  )      depends on job
// SOURCE  )
// END            required
```

DSPLY Control Statement

The DSPLY control statement specifies which macro definition is to be listed. The DSPLY control statement has one of the following formats:

Name	Operation	Operand
//	DSPLY	ALL
//	DSPLY	macro
//	DSPLY	macro.ALL

ALL
Indicates that all macro definitions are to be listed.

macro
Name of the macro definition to be listed. The name may be up to five characters long. The first character must be alphabetic, the others may be alphabetic or numeric.

macro.ALL
Indicates that all macro definitions whose names begin with the characters preceding .ALL are to be listed. The name may be up to five characters long.

PUNCH Control Statement

The PUNCH control statement specifies which macro definition is to be punched into cards or written onto disk or magnetic tape. The PUNCH statement has one of the following formats:

Name	Operation	Operands
//	PUNCH	ALL
//	PUNCH	macro
//	PUNCH	macro.ALL

ALL
Indicates that all macro definitions are to be punched or written.

macro
Name of the macro definition to be punched or written. The name may be up to five characters long. The first character must be alphabetic, the others may be alphabetic or numeric.

macro.ALL
Indicates that all macro definitions whose names begin with the characters preceding .ALL are to be punched or written. The name may be up to five characters long.

INTERN Control Statement

The INTERN control statement specifies that the macro definitions named in the following control statements up to the next SOURCE control statement are to be listed, written, or punched in internal format. By internal format we mean the format in which the macro definition is stored in the macro library.

The INTERN control statement has the format:

Name	Operation	Operand
//	INTERN	

If you want to obtain all macro definitions in internal format in an MSERV job, you don't have to insert the INTERN control statement. The internal format will be automatically assumed even if you leave the control statement out.

However, if you want one macro definition in source format and the following macro definitions in internal format, you must insert the INTERN control statement immediately before the DSPLY or PUNCH control statement that refers to the second macro definition. Example:

```
// SOURCE
// DSPLY MACR1
// INTERN
// PUNCH MACR2
// DSPLY MACR3
```

When you wish to punch macro definitions for backup purposes, we recommend that you specify the internal format. Fewer cards will be punched, and both the current MSERV job and the MMAINT job you may wish to run later will be much faster.

SOURCE Control Statement

The SOURCE control statement specifies that the macro definitions named in the following control statements up to the next INTERN control statement are to be listed, written, or punched in source format. By source format we mean the macro language in which the macro definition was written.

The format of the SOURCE control statement is:

Name	Operation	Operand
//	SOURCE	

If you want to obtain a macro definition in source format, insert the SOURCE control statement immediately before the first DSPLY or PUNCH control statement that refers to this macro definition.

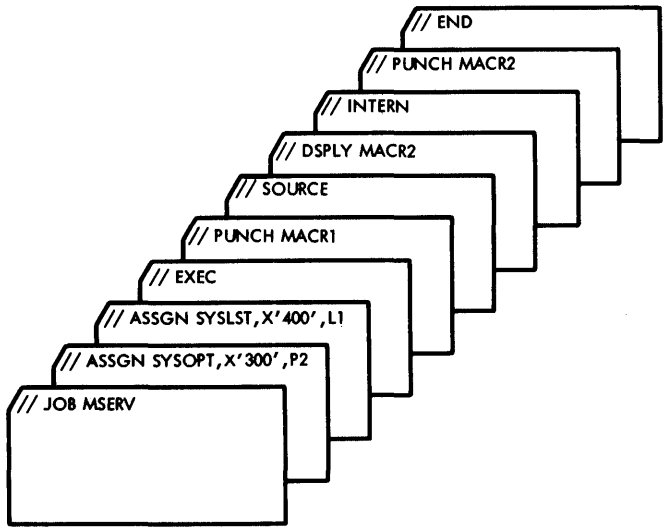
We recommend that you specify the source format if you wish to use the output listing or output card deck to modify the macro definition.

END Control Statement

The format of the END control statement is:

Name	Operation	Operand
//	END	

Figure 17 is an example of using job and program control statements to list and punch the contents of the macro library.



●Figure 17. Example of Using Control Statements to Request Macro Service Functions

OUTPUT ON SYSOPT

The output of the MSERV program contains the following cards or card-image records for each macro definition:

in internal format
- // INCLD
- MACRO card
- IMT cards
- MND card

For the format of these cards, see the section Macro Maintenance Program.

In source format
- // CATAL
- MACRO header statement
- prototype statement
- model statements
- conditional assembly instructions
- MEND trailer statement

For the format of these cards, see the SRL publication IBM System/360 Model 20, Disk and Tape Programming Systems, Assembler Language, Form C24-9002.

The MSERV program writes or punches an EOF record at the end of the output file.

Output	EOF Record
card deck	// END
magnetic tape	tapemark
disk	/*

Core-Image Maintenance Program (CMAINT)

The Core-Image Maintenance program (CMAINT) maintains the core-image library. It adds any number of phases to, and deletes any number of phases from, the core-image library in one job.

JOB CONTROL STATEMENTS

The job control statements required for a CMAINT run are:

// LOG	optional
// JOB	required
// DATE	1st job after IPL only
// ASSGN SYSIPT	depends on job
// ASSGN SYSLST	depends on job
// ASSGN SYSLOG	optional

// VOL
// DLAB
// XTENT
	depends on job

// OPTN TES	recommended for tapes
// FILES	may be required
// EXEC	required

JOB Control Statement

The format of the JOB control statement is:

Name	Operation	Operand
//	JOB	CMAINT

CMAINT
: Identifies the operation as a maintenance operation on the core-image library.

ASSGN Control Statements

If the input to the CMAINT program is not in the relocatable area, you must assign SYSIPT to an input device so that the phase to be added can be read. This input device can be a card reading device, a magnetic tape drive, or a disk drive.

If the input to the CMAINT program is the output of an Assembler run, an RPG compilation, or a Linkage Editor run, this input can be read directly from the relocatable area on the system disk pack. In this case, SYSIPT need not be assigned.

The assignments of SYSLST and SYSLOG are optional, but we recommend that you use them.

If these assignments are already effective, you need not submit them again.

VOL, DLAB, XTENT Control Statements

You must supply disk label information if SYSIPT refers to a disk drive. The file name you use must be SYSIF.

FILES Control Statement

If the input text is stored on magnetic tape, you must position the tape to the first record of the phase to be added, or to the beginning of the volume or header label that precedes the phase. Use the FILES control statement for this purpose if necessary.

EXEC Control Statement

The EXEC control statement required for the CMAINT program has one of the two following formats:

Name	Operation	Operands
//	EXEC	
//	EXEC	R

No operand
: The input is read from the device assigned to SYSIPT.

R
: The input is an object program read from the relocatable area.

The remaining job control statements are not described here. You can find their correct formats in the section Job Control Program.

PROGRAM CONTROL STATEMENTS

The program control statements required for a CMAINT run are:

// DELET
// CATAL
// IPL
// MONTR
	depends on job

// END	required

DELET Control Statement

The DELET control statement is used to specify which phase is to be deleted from the core-image library. The DELET control statement has one of the following formats:

Name	Operation	Operand
//	DELET	phase
//	DELET	phase.ALL

phase
The name of the phase that is to be deleted from the core-image library.

phase.ALL
Indicates that all phases whose names begin with the characters preceding .ALL are to be deleted from the core-image library. The name may contain from one to six characters.

Phases whose names begin with SYS or $$$ must not be deleted.

CATAL Control Statement

The CATAL control statement is used to add a phase to the core-image library. The text of the new phase is read from the system input device (SYSIPT) to which a card reading device, a magnetic tape drive, or a disk drive may be assigned. The CATAL control statement has one of the following formats:

Name	Operation	Operand
//	CATAL	phase
//	CATAL	

phase
The name of a particular phase to be read from the system input device (SYSIPT). The name may contain up to six characters. If SYSIPT and SYSRDR are assigned to different devices this form of the CATAL statement causes the CMAINT program to scan the input medium for a PHASE statement with a matching name. The phase named is then added to the core-image library. If SYSIPT and SYSRDR are assigned to the same card reading device the operand of the CATAL statement is ignored.

If more than one statement of this format is used, the statements must be in the same order as the corresponding phases on the input medium.

No Operand
Indicates that the text of the next sequential phase is to be read from the system input device (SYSIPT).

When phases are to be added to the core-image library, a PHASE card is required in addition to the TXT cards produced by the Assembler or RPG programs. The PHASE card provides information for the CMAINT program. The format of the PHASE card is described in the section Linkage Editor Program. If SYSIPT and SYSRDR are not assigned to the same card reading device, the input on SYSIPT must be terminated by the /* card.

IPL Control Statement

The IPL control statement is used to replace the Initial Program Loader on the system disk pack. The IPL control statement indicates that the card deck containing the new IPL program is to be read on the device assigned to SYSIPT. The IPL is loaded into its fixed location on the system pack (cylinder 0, track 0). The IPL control statement has the following format:

Name	Operation	Operand
//	IPL	

If both the IPL and the Monitor are to be replaced, the IPL statement must precede the MONTR statement and every additional CATAL statement.

MONTR Control Statement

The MONTR control statement is used to replace the Monitor program on the system disk pack. The MONTR control statement indicates that the card deck containing the new Monitor program and its transient and Job Closing routines, if any, is to be read on the device assigned to SYSIPT. The Monitor is loaded into its fixed location on the system disk pack (beginning at sector 1, track 0 of cylinder 4), and the corresponding transient and Job Closing routines, if any, are included in the core-image library. The MONTR control statement has the following format:

Name	Operation	Operand
//	MONTR	

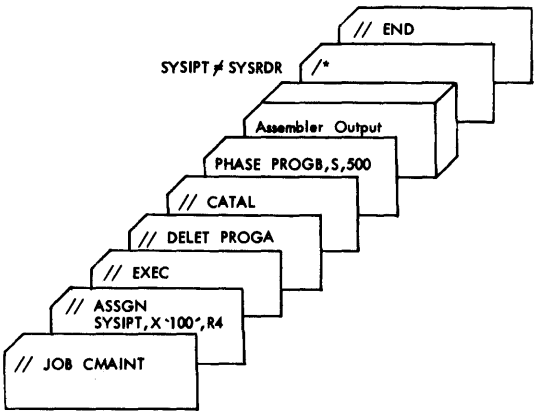
The MONTR statement must precede every additional CATAL statement.

END Control Statement

The format of the END control statement is:

Name	Operation	Operand
//	END	

Figure 18 shows an example of using job and program control statements in a maintenance operation on the core-image library.



●Figure 18. Example of Using Control Statements to Request a Core-Image Library Maintenance Function

Rules for Subphases

Normally, a phase that is to be placed into the core-image library must consist of one or more complete control sections. The end of a phase is indicated to the CMAINT program by means of the PHASE statement that precedes the subsequent phase, or by means of an END control statement, or by means of an EOF record (/* or tapemark).

Normally, only one END or XFR statement appears within one phase. However, you may insert up to nine XFR or END cards into the TXT cards of one phase. This allows you to build additional subphases within one control section. The following guidelines will help you in building subphases:

1. Literals must be defined within the phase or subphase in which they are required.

2. The load point for each subphase is derived from the load address of the first TXT statement within each subphase.

3. The entire phase (including all subphases) is placed in the core-image library under the phase name in the associated PHASE statement.

4. Thus, all subphases are listed under the name of the phase to which they belong. However, the last byte of the subphase name stored in the core-image directory contains one of the digits 1 to 9 to identify each subphase.

5. All subphases of a phase can be fetched in sequential order by means of FETCH macro instructions without an operand.

6. A phase that contains one or more subphases is added to or deleted from the core-image library by means of only one CATAL or DELET control statement.

The formats of the XFR, TXT and END cards are described in the section Linkage Editor Program.

Macro Maintenance Program (MMAINT)

The Macro Library Maintenance program (MMAINT) maintains the macro library. It adds any number of macro definitions to, and deletes any number of macro definitions from, the macro library in one job.

JOB CONTROL STATEMENTS

The job control statements required for an MMAINT run are:

```
// LOG              optional
// JOB              required
// DATE             1st job after IPL only
// ASSGN SYSIPT     depends on job
// ASSGN SYSLST     optional
// ASSGN SYSLOG     optional
// VOL    }
// DLAB   }         depends on job
// XTENT  }
// OPTN TES         recommended for tapes
// FILES            may be required
// EXEC             required
```

JOB Control Statement

The format of the JOB control statement is:

Name	Operation	Operand
//	JOB	MMAINT

MMAINT
Identifies the operation as a maintenance operation on the macro library.

ASSGN Control Statements

If you wish to add or replace a macro definition, you must assign SYSIPT to a card reading device, a magnetic tape drive, or a disk drive so that the input text can be read.

If you only wish to delete macro definitions in a job, you must unassign SYSIPT (// ASSGN SYSIPT,UA).

The assignments of SYSLST and SYSLOG are optional, but we recommend that you use them.

If these assignments are already effective, you need not submit them again.

VOL, DLAB, XTENT Control Statements

You must supply disk label information if SYSIPT refers to a disk drive. The file name you insert must be SYSIF.

FILES Control Statement

If SYSIPT refers to a magnetic tape drive, you must position the tape to the first record of the macro definition to be added. Use the FILES control statement for this purpose if necessary.

The remaining job control statements are not described here. You can find their correct formats in the section Job Control Program.

PROGRAM CONTROL STATEMENTS

The program control statements required for an MMAINT run are:

// DELET
// CATAL
// INCLD
	depends on job

// END	required

DELET Control Statement

To delete macro definitions from the macro library, use the following control statement:

Name	Operation	Operand
//	DELET	macro
//	DELET	ALL

macro
The name of the macro definition that is to be deleted from the macro library. The name may be up to five characters long.

ALL
Indicates that all macro definitions are to be deleted from the macro library.

CATAL Control Statement

If the macro definition you wish to insert is in source format, that is, written in the macro language, use the control statement:

Name	Operation	Operand
//	CATAL	

The text of the macro definition you wish to add as a new entry, or wish to insert in the place of an old entry, is read from the device assigned to SYSIPT and cataloged in the macro library. The macro definition statements are listed on the printer assigned to SYSLST.

A typical macro definition in source format consists of a header statement, a prototype statement, model statements, conditional assembly instructions, and a trailer statement. Details of the DPS Assembler macro language are contained in the SRL publication IBM System/360 Model 20, Disk and Tape Programming Systems, Assembler Language, Form C24-9002.

INCLD Control Statement

If the macro definition you wish to insert is in internal format, that is, the format in which it is written into the macro library, use the control statement:

Name	Operation	Operand
//	INCLD	

The text of the macro definition you wish to add as a new entry, or wish to insert in the place of an old entry, is read from the device assigned to SYSIPT and cataloged in the macro library.

A typical macro definition in internal format consists of a header statement, IMT cards, and an MND card. You can also change the text of a macro definition by inserting MOD cards into the input stream. The purpose and format of the IMT, MND, and MOD cards are described in the following section.

END Control Statement

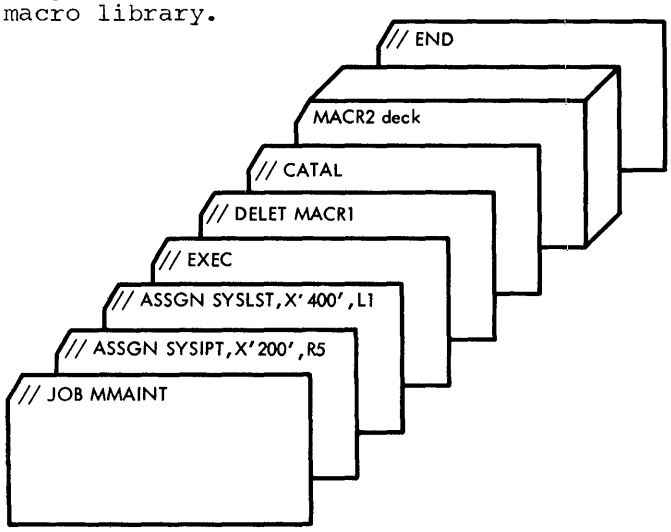
The format of the END control statement is:

Name	Operation	Operand
//	END	

If you wish to delete old macro definitions and add new macro definitions in the same job, delete the old macro definitions first. This is advisable because the macro library is automatically compressed before the first addition is made.

If you wish to replace old macro definitions with new macro definitions, delete the old macro definitions first. No error will occur if you don't do it this way. However, compression takes place more than once, and this has a negative effect on performance.

Figure 19 is an example of using job and program control statements to maintain the macro library.



●Figure 19. Example of Using Control Statements to Request a Macro Library Maintenance Function

Input Card Formats

If the input to the MMAINT program is in cards, you have the option of changing the input text by means of MOD cards. This can only be done if the output of the MSERV run was punched into cards.

If the output was written onto disk or magnetic tape, the output records have the same format as the output cards, but they cannot be modified by means of MOD cards.

The following sections describe the IMT and MND cards generated by the MSERV program and used as input to the MMAINT program, and explain how you can modify the input text by means of MOD cards.

IMT Card

IMT cards contain macro definitions in internal format. The cards also specify the location of the text within the macro definition.

MOD Card

You have the option of punching MOD cards and inserting them into the macro definition deck to replace text contained in an IMT card. Each MOD card contains the location of the first byte to be replaced. It may contain from 2 to 22 bytes of text. The text is substituted, byte for byte, for the original text, beginning at the specified location. Both the address and the new text must be written in hexadecimal notation. Remember that MOD cards must never precede the IMT cards they are supposed to modify.

MND Card

The MND card indicates the end of the macro definition.

Figures 20, 21, and 22 show the format of IMT, MOD, and MND cards, respectively.

Columns	Contents
1	12-3-9 punch
2-4	IMT
5	blank
6-8	sequential number of first byte of text in this card
9-10	blank
11-12	number of bytes of text in this card
13-16	blank
17-72	up to 56 bytes of macro definition text in internal format
73-80	optional identification (MSERV inserts the first and the last two characters of the macro name and a card serial number)

●Figure 20. Format of the IMT Card

Columns	Contents
1	12-3-9 punch
2-4	MOD
5-6	blank
7-12	sequential number of the first byte to be replaced (in hexadecimal notation with leading zeros, right-justified)
13-16	blank
17-70	from one to eleven 4-digit fields (in hexadecimal notation), separated by commas, each field replacing one half-word of text. A blank column indicates the end of information in this card.
71-72	blank
73-80	optional identification (If an identification is punched in these columns, the entries in columns 73-76 must not differ from the entries in the corresponding IMT card)

Figure 21. Format of the MOD Card

Columns	Contents
1	12-3-9 punch
2-4	MND (indicating end of macro definition)
5-72	blank
73-80	optional identification (MSERV inserts the first and the last two characters of the macro name and a card serial number)

●Figure 22. Format of the MND Card

Library Allocation Organization Program (AORGZ)

The Library Allocation Organization program (AORGZ) redefines the limits of the core-image library and directory, the macro library and directory, and the relocatable area on the system disk pack.

By means of the AORGZ program, these areas can be allocated to or eliminated from the system disk pack, or their size can be increased or decreased.

JOB CONTROL STATEMENTS

The job control statements required to request an allocation function are:

```
// LOG            optional
// JOB            required
// DATE           1st job after IPL only
// ASSGN SYSLST   optional
// ASSGN SYSLOG   optional
// EXEC           required
```

JOB Control Statement

The JOB control statement used to request a reallocation operation has the following format:

Name	Operation	Operand
//	JOB	AORGZ

AORGZ
Identifies the operation as reallocation.

The remaining job control statements are not described here. You can find their correct formats in the section Job Control Program.

PROGRAM CONTROL STATEMENTS

The program control statements required for an AORGZ run are:

```
// LIMIT     required
// END       required
```

LIMIT Control Statement

The LIMIT control statement specifies which libraries and directories are to be reallocated. Any number of areas can be reallocated within a single job. The format of the LIMIT control statement is:

Name	Operation	Operands
//	LIMIT	xx,nnn [,xx,nnn...]

xx
Code indicating which of the libraries or library directories are to be reallocated.

Code	Area
CD	Core-Image Directory
CL	Core-Image Library
MD	Macro Directory
ML	Macro Library
RL	Relocatable Area

nnn
Code in decimal notation indicating how many tracks are to be allocated. Leading zeros are not used.

It doesn't make any difference in what order you specify the areas to be allocated. But each area must be followed by a comma and the number of tracks it is to occupy. If you want to reallocate three areas, the operand will be: xx,nnn,xx,nnn,xx,nnn.

If you specify a 0 after the area code xx, the area will be eliminated from the system disk pack. If you want to leave an area unchanged, don't specify it in the control statement.

The total number of tracks you allocate must not exceed the number of tracks on the disk pack (1986 tracks for Model 11, 986 tracks for Model 12).

If the total number of tracks you allocate extends into a data file that has not expired, a halt occurs. If the data file has expired, it is overwritten.

The macro directory and the core-image directory must not occupy more than ten tracks apiece.

If you eliminate the macro library from the system disk pack, eliminate the macro directory as well (ML,0,MD,0).

Never eliminate the core-image library and the core-image directory.

END Control Statement

The format of the END control statement is:

Name	Operation	Operand
//	END	

Figure 23 is an example of using job and program control statements to reallocate several areas on the system disk pack.

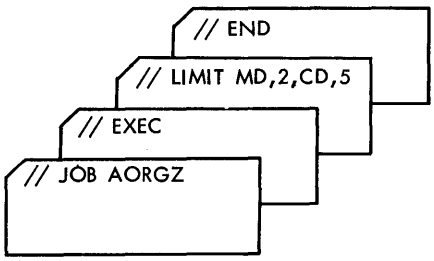


Figure 23. Example of Using Control Statements to Request Library Allocation Functions

Physical and Logical Unit Tables Service Program (PSERV)

This section explains the functions of the Physical and Logical Unit Tables Service program (PSERV) and describes the control statements required to request these functions. The PSERV program is used to list or change the permanent device assignments of the Monitor resident on the system disk pack. It can also be used to change the configuration byte in the communication region of the Monitor.

JOB CONTROL STATEMENTS

The job control statements required for a PSERV run are:

```
// LOG             optional
// JOB             required
// DATE            1st job after IPL only
// ASSGN SYSLST    depends on job
// ASSGN SYSLOG    optional
// EXEC            required
```

JOB Control Statement

The format of the JOB control statement is:

Name	Operation	Operand
//	JOB	PSERV

PSERV
Identifies the operation as a PSERV function.

ASSGN Control Statements Preceding EXEC

If you want a listing of information from the PSERV run, you must assign SYSLST to the printer. Remember that this ASSGN control statement must precede the EXEC

control statement. The assignment of SYSLOG is optional. If these assignments are already effective, you need not submit them again.

EXEC Control Statement

The control statements that follow the EXEC statement indicate to the PSERV program which functions it is to perform.

The remaining job control statements are not described here. You can find their correct formats in the section Job Control Program.

PROGRAM CONTROL STATEMENTS

The program control statements required for a PSERV run are:

```
// DSPLY  }
// ASSGN  }  depends on job
// CONFG  }
// END       required
```

DSPLY Control Statement

The DSPLY control statement instructs the PSERV program to list all entries of the logical unit table, all entries of the physical unit table, and information about the type of Monitor resident on the system disk pack. This operation requires a printer, which must be assigned to SYSLST.

The format of the DSPLY control statement is:

Name	Operation	Operand
//	DSPLY	

The logical unit table is listed first. The entries are listed in the same format as the corresponding ASSGN statements. The listing includes the device types that the entries refer to. This printout is followed by a listing of all device addresses contained in the physical unit table, and finally by information about the type of Monitor resident on the system disk pack.

ASSGN Control Statements Following EXEC

You can insert ASSGN control statements after the EXEC statement to change the permanent device assignments in the Monitor resident on the system disk pack. The physical and logical unit tables of the Monitor in main storage are not affected by the resulting assign operations. Changes of the permanent assignments will affect processing operations only after an IPL procedure has transferred the modified Monitor from disk to main storage.

The format of the ASSGN control statement is described in the section Job Control Program. All ASSGN control statements accepted by the Job Control program are also accepted by the PSERV program.

The ASSGN control statements may be submitted to the PSERV program in any order. ASSGN control statements are required only for those entries in the logical unit table whose standard assignments are to be changed. However, if the standard assignment or the unit number of one magnetic tape drive is to be changed, all magnetic tape drives must be reassigned by changing the corresponding entries.

CONFG Control Statement

You can use the CONFG control statement to alter the storage-capacity byte in the communication region of the Monitor.

However, if the Monitor currently on the system disk pack supports inquiry facilities, its storage-capacity information must not be altered by a PSERV run, because the dummy phase used to save the contents of main storage while the inquiry program is executed was generated to accommodate the original storage capacity specified at Monitor generation time.

The format of the CONFG control statement is described in the section Job Control Program.

END Control Statement

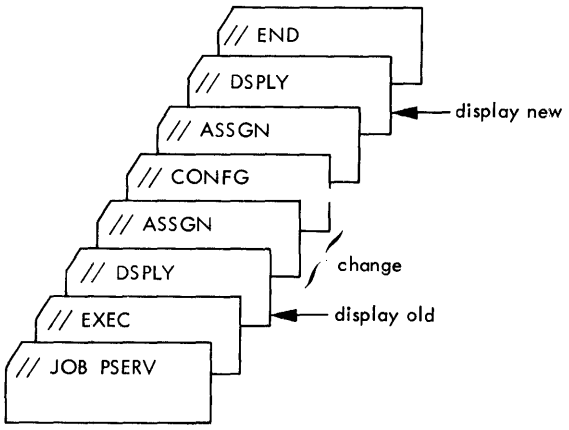
The format of the END control statement is:

Name	Operation	Operand
//	END	

You may insert all three control statements -- DSPLY, ASSGN, and CONFG -- after the EXEC control statement. The order in which you arrange them, however, must be meaningful.

The first step is to display the current permanent assignments, the second is to alter them, and the third step is to display the modified assignments.

Figure 24 shows an example of using job and program control statements to request a PSERV function.



●Figure 24. Example of Using Control Statements to Request a PSERV Function

Linkage Editor Program (LNKEDT)

The Linkage Editor program is used to link separately assembled programs or phases into an integral program that can be executed under control of the Monitor program. The Linkage Editor program can also be used to relocate assembled programs.

The Linkage Editor program is not required to edit

- RPG output because programs written in the RPG language cannot be relocated or linked
- PL/I output because the PL/I compiler itself edits the source program
- a single-phase program assembled in one Assembler run whose starting address is higher than the end address of the Monitor.

The following sections discuss in detail what the functions of the Linkage Editor program are, what control statements you must supply to request these functions, what input statements you must supply, what output you will obtain, and how you can use the Linkage Editor program effectively.

FUNCTIONS OF THE LINKAGE EDITOR PROGRAM

The Linkage Editor program performs the following functions:

1. It links separately assembled program phases into one executable object program that can be placed into the core-image library. (The output may also be executed under control of the card-resident Monitor program.)
2. It relocates assembled programs, if required. By means of the PHASE statement you supply, you can load the program into any location of main storage. The Linkage Editor program will cross-reference all symbols defined as EXTRN and ENTRY statements in the various control sections.
3. It checks the input cards for certain types of errors and determines
   - any unresolved External Reference (ER) items,
   - the number of unresolved ER and RLD items,
   - the number of REP and TXT cards that are outside the bounds of a phase.

It also prints a linkage table, provided that you assigned SYSLST to a printer.

JOB CONTROL STATEMENTS

The job control statements you must supply to request a Linkage Editor function are:

// LOG	recommended
// JOB	required
// DATE	1st job after IPL only
// ASSGN SYSIPT	depends on job
// ASSGN SYSOPT	depends on job
// ASSGN SYSLST	recommended
// ASSGN SYSLOG	recommended
// OPTN TES	recommended for tapes
// FILES	may be required
// EXEC	required

JOB Control Statement

The format of the JOB control statement is:

Name	Operation	Operand
//	JOB	LNKEDT

LNKEDT
Identifies the operation as linking or relocating separately assembled programs.

ASSGN Control Statements

Input to the Linkage Editor program may be from the relocatable area, from punched cards, or from magnetic tape. If you do not want input from the relocatable area, you must assign SYSIPT to a card reading device or to a magnetic tape drive.

The output from the Linkage Editor run is always written into the relocatable area. If you also want the output to be punched into cards or written onto magnetic tape, you must assign SYSOPT to a card punching device or a magnetic tape drive. If SYSOPT is not assigned, the output will only be written into the relocatable area.

The assignments of SYSLST and SYSLOG are optional, but we recommend that you make them.

If these assignments are still effective, you need not submit them again.

FILES Control Statement

If you furnish the input on a magnetic tape drive, you must position the tape to the first data record. Use the FILES control statement for this purpose if necessary.

EXEC Control Statement

The EXEC control statement has one of two formats, depending on how you furnish the input to the Linkage Editor run:

Name	Operation	Operand
//	EXEC	
//	EXEC	R

No operand
Indicates that the input is to be read from the device assigned to SYSIPT.

R
Indicates that the input is to be read from the relocatable area.

The remaining job control statements are not described here. You can find their correct formats in the section Job Control Program.

INPUT TO THE LINKAGE EDITOR PROGRAM

The input to the Linkage Editor program (that is, the output of one or more assembly runs) can be in the form of punched cards, or in card-image format on magnetic tape, or in card-image format in the relocatable area.

If the input is in cards, you have the option of replacing portions of assembled text with new text by means of REP cards.

In the following sections, we assume that the input to the Linkage Editor program is in cards, in order to better describe the input and how you can modify portions of assembled text.

This section describes the input cards used for a Linkage Editor run. Figure 25 is a list of these cards. The first four cards shown in this figure -- PHASE, ACTION, REP, and ENTRY -- are the ones you must furnish. The remaining cards are produced by the Assembler in the assembly run.

Card	Function
Furnished by User	
PHASE	indicates the beginning of a phase, specifies name and load address of the phase
ACTION	indicates whether XFR cards are to be duplicated
REP	substitutes new text for portions of assembled text
ENTRY	indicates end-of-input to the Linkage Editor program; may be used to specify transfer point that replaces the address in the (output) END statement of the first phase
Produced by Assembler	
ESD	contains external symbol dictionary
TXT	contains program instructions and constants
RLD	contains relocation dictionary
END	indicates the end of a program and may specify a program entry address
XFR	specifies program entry address

Figure 25. Summary of Input Cards to the Linkage Editor Program

If the source deck to be assembled contains a PHASE or ACTION card, you can instruct the Assembler to reproduce these cards in the object deck by submitting a REPRO statement. If you submit an AOPTN statement to the Assembler, the Assembler will generate an ENTRY card without an operand in the object deck.

The following sections describe each of the input cards listed in Figure 25.

PHASE Card. Any program that is either to be processed by the Linkage Editor or to be

included in the core-image library of the disk-resident system must be preceded by a card that contains a PHASE statement. If a program consists of more than one phase, each of the program phases must be headed by a PHASE card.

The PHASE card must precede the phase, and not appear somewhere within it. This would cause an undetectable error and result in incorrect program loading.

The PHASE card furnishes the name and the load address of a phase. The load address of a phase can be relative (1) to the last location of main storage, or (2) to the end address of the Monitor, or (3) to a previously defined symbol. It can also be an absolute address. The format of the PHASE card is:

Name	Operation	Operands
	PHASE	name,i,ddddd [,address,vvmm]

PHASE
Leave at least one column blank before and after PHASE

name
The name of the phase. This name may consist of 1 to 6 characters. It must be different for each phase if more than one phase is to be link-edited. We urge you not to choose a name whose first three characters are identical with the first three characters of the phase names of IBM-supplied programs (see Appendix I). Should the remaining characters be identical as well, the system program is in jeopardy. The phase name must not include embedded blanks or special characters. The first character of the name must be alphabetic, the remainder may be alphabetic or numeric (the symbols #, @, and $ are considered alphabetic characters).

i
The indicator for the load address. This can be one of the letters A, C, L, or S.

- A -- indicates that the phase is to be loaded at the absolute address specified in the next operand (ddddd).
- C -- indicates that the address is relative to the last location of storage.
- L -- indicates that the address is a name defined in a previous phase.
- S -- indicates that the address is relative to the end address of the Monitor.

ddddd
The displacement. This operand permits you to modify the load address you specified in the preceding operand (i). You can specify a positive displacement if the load address is A, L, or S. (If the load address is A, the displacement is the absolute address.) You may specify a negative displacement if the load address is C or L.

Positive and negative displacements can be expressed in decimal or hexadecimal notation:

Positive
nnnnn (up to five digits)
X'nnnn' (four digits)

Negative
-nnnnn (up to five digits)
-X'nnnn' (four digits)

address
Symbolic address used only if the load address is L. The symbolic address must have been defined in a previous phase either in an ENTRY, CSECT, or START Assembler statement.

vvmm
Four numeric characters identifying the version and modification level of the phase. This operand is optional.

Separate the operands from each other by commas. Do not insert any blanks. If an operand is not required, insert a comma in its place (refer to the third example in Figure 26).

The Linkage Editor program derives the load address of a phase from the information contained in the PHASE statement.

Figure 26 shows five examples of PHASE statements. The circled numbers in the left-hand margin of this figure refer to the text below.

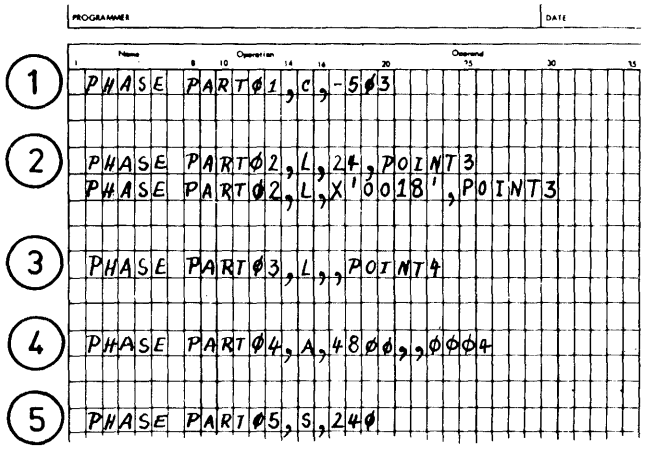


Figure 26. Examples of the PHASE Statement

1. This PHASE statement causes loading to begin 503 bytes below the last byte of storage.

2. Loading begins at the address POINT3+24. POINT3 is a name defined in a preceding control section. The displacement may also be specified in hexadecimal notation as shown in the second line of this example.

3. Loading of PART03 begins at the address of POINT4, which is a previously defined symbol.

4. Loading of PART04 begins at storage location 4800. The last operand is used as a version and modification level of the phase.

5. Loading of PART05 begins 240 bytes beyond the last byte occupied by the Monitor.

ACTION Card. This card enables you to create subphases. The ACTION card instructs the Linkage Editor program to duplicate into the output deck all XFR and END cards encountered in the input deck. The formats of the ACTION card are:

Name	Operation	Operand
	ACTION	DUP
	ACTION	NODUP

ACTION must be preceded and followed by at least one blank column. The card ACTION DUP instructs the Linkage Editor to duplicate this card and all XFR and END cards subsequently encountered, and to discontinue duplicating as soon as a card ACTION NODUP has been read and duplicated.

You can place ACTION cards anywhere in the input stream behind the first PHASE card. You will normally instruct the Assembler to reproduce the ACTION cards from the source deck by means of a REPRO statement, but you may supply an ACTION card yourself and insert it into the input stream.

If the ENTRY card for the Linkage Editor contains an operand, an exception is made to the above description. Refer to the section Output of the Linkage Editor Program.

REP Card. The REP card is used to substitute new text for portions of assembled text. Each REP card contains the assembled address of the first byte to be replaced. It may contain from 2 to 22 bytes of text. The text is used to replace the original text byte for byte, beginning at the specified address. Both the address and the new text must be stated in hexadecimal notation. The format of the REP card is shown in Figure 27.

Column	Contents
1	12-2-9 punch
2-4	REP
5-6	blank
7-12	assembled address of the first byte of text to be replaced (in hexadecimal notation with leading zeros; right-justified)
13-14	blank
15-16	ESID* number of the control section containing the text (in hexadecimal notation)
17-70	from 1 to 11 four-digit fields (in hexadecimal notation) separated by commas, each field replacing one half-word of text. A blank column indicates the end of information in this card
71-72	blank
73-80	program identification (optional)

*See the section ESID Numbers

Figure 27. Format of the REP Card

The REP card is accepted by all programs that read TXT cards (card-resident IPL, card-resident Monitor, LDSYS, CMAINT, LNKEDT). For all programs except the Linkage Editor program, columns 15-16 of the REP card may be left blank.

The REP card must be placed behind the text that is to be modified.

If the segment in which this text is contained consists of input for more than one phase, place the REP card within the bounds of the phase to which the text belongs. The Linkage Editor program will produce a new TXT card that has been correctly relocated. This card contains the information and data needed to modify the text.

Do not modify address constants by means of REP cards. RLD relocation applies only to TXT cards contained in the input stream.

A REP or TXT card must not cause information to be loaded into locations outside the limits of a phase or subphase. Howev-

er, the Linkage Editor will also process such cards.

ENTRY Card. The ENTRY card indicates the end of input to the Linkage Editor program. The ENTRY card has the following format:

Name	Operation	Operand
	ENTRY	[address]

ENTRY
Must be preceded and followed by at least one blank column.

address
The operand is a symbol defined as the operand of an ENTRY statement or the name of a CSECT or START Assembler statement in any phase of the input. The symbol to which the operand refers must be unique. use of the operand is optional. If the operand is used, the ENTRY card causes the name specified in it to replace the transfer point specified in the first phase processed.

ESD Card. ESD cards contain the information required to link one segment to other segments. It contains all symbols defined in one segment but referred to in another segment. It also contains all symbols referred to in one segment but defined in another segment.

The Linkage Editor program recognizes three types of ESD cards: SD, LD, and ER.

1. SD -- Section Definition.
   This type of ESD card is produced whenever a START or CSECT Assembler instruction is contained in the source program. The SD-type of ESD card is used to specify the name, the load address, the ESID number, and the length of the control section.

2. LD -- Label Definition.
   The LD-type of ESD card is produced whenever the source program contains an ENTRY Assembler instruction. The LD-type of ESD card defines a name that may be used in any other segment as an entry point, or as the name of a constant or storage area.

3. ER -- External Reference.
   This type of ESD card is produced whenever the source program contains an EXTRN Assembler instruction. The ER-type of ESD card specifies a name that is used in this segment to refer to a point in some other segment(s).

The format of the ESD card is shown in Figure 28.

Columns	Contents
1	12-2-9 punch
2-4	ESD
5-10	blank
11-12	number of bytes of information contained in this card
13-14	blank
15-16	ESID number of the first SD or ER in this card* (blank for LD)
17-24	name (defined in a START, ENTRY, EXTRN, or CSECT statement)
25	code to indicate SD, LD, or ER type SD -- 12-0-1-8-9 punch LD -- 12-1-9 punch ER -- 12-2-9 punch
26	12-0-1-8-9 punch
27-28	ER -- 12-0-1-8-9 punches SD,LD -- load address
29	blank
30	12-0-1-8-9 punch
31-32	SD -- length of control section LD -- ESID* number of the SD containing the name ER -- blank
33-72	blank
73-80	program identification (optional)

*See the section ESID Numbers

Figure 28. Format of the ESD Card

ESID Numbers. External Symbol Identification (ESID) numbers are pointers assigned by the Assembler. These pointers are used by the Linkage Editor program to correctly recompute the constants referred to in the RLD entries. ESID numbers are also used to identify the cards belonging to a particular control section within a segment. Assignment of ESID numbers in a segment begins with 01 and continues sequentially to the maximum of 31. The first ESID number assigned in a control section is a pointer to this control section and its internal constants. The remaining ESID numbers assigned within a control section pertain to the external symbols defined in this control section.

Columns	Contents
1	12-2-9 punch
2-4	TXT
5	blank
6	12-0-1-8-9 punch (binary zero)
7-8	load address of first byte of text in this card*
9-10	blank
11-12	number of bytes of text to be loaded*
13-14	blank
15-16	ESID number of the control section containing the text*
17-72	up to 56 bytes of text (instructions and/or constants) to be loaded*
73-80	program identification (optional)

*in EBCDIC card code

Figure 29. Format of the TXT Card

TXT Card. The TXT cards contain the problem program in machine language. A TXT card contains the load address of the instructions or constants in the card. It also contains a reference to the control section in which this information occurs, enabling the Linkage Editor program to derive the relocation factor involved. TXT cards can be modified by the Linkage Editor with the information contained in RLD cards. The format of the TXT card is shown in Figure 29.

RLD Card. The RLD cards identify portions of the text (address constants) that require modification owing to relocation. They furnish the Linkage Editor with information required to perform the relocation.

The format of the RLD card is shown in Figure 30.

END Card. The END card specifies the transfer address for program execution and indicates the end of a phase or subphase. To the Linkage Editor program, it indicates that the end of a segment has been reached.

Columns	Contents
1	12-2-9 punch
2-4	RLD
5-10	blank
11-12	number of bytes of information contained in variable field (multiple of 8) *
13-16	blank
17-72	Variable Field* This field contains from one to seven 8-column groups of information. Each group contains: 2 cols. -- ESID number assigned to the control section in which the contents of the constant occur. 2 cols. -- ESID number assigned to the control section in which this constant occurs. 1 col. -- 12-4-9 punch indicating that the address correction factor is to be added to the constant; or 12-6-9 punch indicating that the address correction factor is to be subtracted from the constant 1 col. -- 12-0-1-8-9 punch 2 cols. -- assembled address of load constant
73-80	program identification (optional)

*in EBCDIC card code

Figure 30. Format of the RLD Card

When processed by the Linkage Editor, the transfer address in an END or XFR card need not be a transfer point defined in the pertinent program. It may be defined by means of an EXTRN statement that refers to an entry point in another program. In this case, the transfer address cannot be modified by means of a displacement relative to a symbol.

When processed by another program, the transfer address must not be symbolic; it must be a fixed (assembled) storage location or blank.

If the transfer address is blank, the address in the PHASE card or the load address of the first TXT card of a sub-

phase is assumed to be the control transfer point.

The END card is produced by the Assembler when an END Assembler instruction is encountered. The format of the END card is shown in Figure 31.

Column	Contents
1	12-2-9 punch
2-4	END
5	blank
6-8	Control transfer address (assembled address of the operand in the Assembler END statement) or blank. If blank, col.17-24 may contain a symbolic transfer point.
9-14	blank
15-16	ESID number of the control section to which this END card refers.* Or blank if Assembler END statement has no operand.
17-24	symbolic transfer point if this was the operand of an EXTRN statement. Or blank.
25-72	blank
73-80	program identification (optional)

*in EBCDIC card code

Figure 31. Format of the END Card

The section Output of the Linkage Editor Program describes how the Linkage Editor handles the END and XFR cards.

XFR Card. The XFR card contains the transfer address for program execution and indicates the end of a phase or subphase. It has the same function as the END card except that it does not indicate the end of a segment to the Linkage Editor. (Any number of XFR cards may be contained in each segment of a program.) For further details, refer to the description of the END card.

The XFR card is produced by the Assembler when an XFR Assembler instruction is encountered in the source program. Figure 32 shows the format of the XFR card.

Columns	Contents
1	12-2-9 punch
2-4	XFR
5	blank
6-8	Control transfer address (assembled address of the operand in the Assembler XFR statement) or blank. If blank, col.17-24 may contain a symbolic transfer point.
9-14	blank
15-16	ESID number of the control section in which the transfer occurs.* Or blank if Assembler XFR statement has no operand.
17-24	symbolic transfer point if this was the operand of an EXTRN statement.* Or blank.
25-72	blank
73-80	program identification (optional)

*in EBCDIC card code

Figure 32. Format of the XFR Card

OUTPUT OF THE LINKAGE EDITOR PROGRAM

The output of the Linkage Editor program can be punched into cards or written onto magnetic tape in card-image format.

To make it easier to understand the contents of the output, we will assume in this section that the output is in punched cards.

Each output phase produced by the Linkage Editor program consists of the following in the order shown:

- one CATAL card,
- one PHASE card,
- one ESD card,
- a number of TXT cards,
- one END card,
- possibly a number of ACTION and XFR or END cards reproduced by means of ACTION statements,
- one /* card,
- one // END card.

The output is in the form of a single control section per phase, even if the input consisted of several control sections.

The Linkage Editor program generates a CATAL card for every PHASE card. (The

CATAL card is required if the output of the Linkage Editor program serves as input to the CMAINT program.) All CATAL cards and blank cards that were already in the input deck prior to the Linkage Editor run are ignored.

The Linkage Editor program produces one SD-type ESD card per phase immediately following the PHASE card.

The output TXT cards contain the information from the input TXT cards modified according to the input RLD cards for correct relocation and linking.

All ACTION cards are reproduced in the output deck.

The ACTION DUP card is effective for all following input cards as long as no ACTION NODUP card is encountered. When ACTION DUP is effective, all input XFR and END cards are reproduced in the output deck. The transfer addresses are modified for correct relocation or linking. When ACTION NODUP is effective, XFR and END cards are not reproduced in the output deck. As long as no ACTION card is encountered in the input stream, ACTION NODUP is assumed.

The Linkage Editor produces one END card at the end of each phase, that is, at the end of the last subphase. This END card determines the transfer point of the last (or only) subphase of a phase.

The transfer address inserted into this END card is determined in the following way:

1. If the phase contains an XFR or END card that (a) does not follow an ACTION DUP card (that is, ACTION NODUP is effective), and (b) specifies a definite address (address fields not blank), the transfer address from the first of these cards is taken.

2. If no such card as specified under point 1 exists, the load address of the phase is inserted into the END card.

If the last card of a phase is an XFR or END card and ACTION DUP is effective, this card will be reproduced into the output deck and the Linkage Editor will not generate an END card as described above.

If the ENTRY card contains an operand, this address replaces the transfer address of the END card at the end of the first phase, and ACTION DUP is ineffective for this END card.

The output deck does not contain an ENTRY card.

If SYSLST has been assigned, the Linkage Editor program prints a listing of information concerning the output of the Linkage Editor run.

USE OF THE LINKAGE EDITOR PROGRAM

The output of the Linkage Editor program is always written into the relocatable area. This object program can be loaded into main storage for execution by supplying the two job control statements // JOB program and // EXEC LOADER,R. It can also be cataloged in the core-image library by the CMAINT program using the control statement // EXEC R.

If you assigned SYSOPT to a card punching device, the output of the Linkage Editor run is also punched into cards. This object program can be executed under the control of the card-resident Monitor, or placed into the core-image library on the system disk pack.

If you assigned SYSOPT to a magnetic tape drive, the output of the Linkage Editor run is also written onto tape. The output tape contains an object program in card-image format. To execute this object program, it must first be placed into the core-image library by means of a CMAINT run, or loaded into main storage by means of the control statement // EXEC LOADER.

The Linkage Editor program can be executed only as a separate job. The following example illustrates the manner in which the Linkage Editor program is used for an assemble-and-execute run.

```
|Name|Operation|Operands                  |
|----|---------|--------------------------|
|//  |JOB      |ASSEMB                    |
|    |         |                          |
|//  |EXEC     |                          |
|    | |       |                          |
|    | |       |                          |
|    | V       |                          |
|//  |JOB      |LNKEDT                    |
|    |         |                          |
|//  |EXEC     |R                         |
|    | |       |                          |
|    | |       |                          |
|    | V       |                          |
|//  |JOB      |program                   |
|    |         |                          |
|//  |EXEC     |LOADER,R                  |
```

The output of the Assembler program is placed in the relocatable area. Thus, it is available as direct input for the Linkage Editor program. Since the output of the Linkage Editor program is also placed in the relocatable area, the problem program is temporarily included in the core-image library and can be executed without the need for card handling or tape positioning.

The program name in the last JOB statement is only checked for presence. It is not compared with the name in the PHASE statement, if any.

The problem program may be permanently included in the core-image library by means of the control statements shown in the following example.

```
|Name|Operation|Operands                  |
|----|---------|--------------------------|
|//  |JOB      |ASSEMB                    |
|    |         |                          |
|//  |EXEC     |                          |
|    | |       |                          |
|    | |       |                          |
|    | V       |                          |
|//  |JOB      |LNKEDT                    |
|    |         |                          |
|//  |EXEC     |R                         |
|    | |       |                          |
|    | |       |                          |
|    | V       |                          |
|//  |JOB      |CMAINT                    |
|    |         |                          |
|//  |EXEC     |R                         |
|    | |       |                          |
|    | |       |                          |
|    | V       |                          |
|//  |JOB      |program                   |
|    |         |                          |
|//  |EXEC     |                          |
```

The output of the Assembler program is placed in the relocatable area. Thus, it is available as direct input to the Linkage Editor program. Since the output of the Linkage Editor program is also placed in the relocatable area, it is available as input to the CMAINT program. The object program is now a permanent entry in the core-image library and can be executed whenever required with only a minimum of card handling.

EXAMPLES

This section contains three examples that illustrate the use of the Linkage Editor program and its functions. Each of the sample programs must be processed by the Linkage Editor program before it can be executed or placed into the core-image library.

Example 1

Figure 33 shows a typical example of two separately assembled programs that are assumed to be linked through ENTRY and EXTRN Assembler statements. Figure 33 also shows the associated output of the Linkage Editor program.

The object program (Linkage Editor output) has been relocated according to the specifications in the PHASE statement. Moreover, the object program has been assigned fixed storage locations beginning with 4700. Further details concerning this example are listed in the column Comments of Figure 33.

Example 2

Figure 34 illustrates a multiphase source program for single assembly, the output of the Assembler, and the output of the Linkage Editor program. The final object program has been relocated according to the specifications in the PHASE statement. Multiple control sections within an input phase have been combined into one control section per output phase.

Example 3

Figure 35 also illustrates a multiphase source program, but in this example the phases are assembled separately. Figure 33 shows the Assembler output and the output of the Linkage Editor program.

The final object program has been relocated. It can be executed under the control of the card-resident Monitor program. It can also be placed in the core-image library of the disk-resident system.

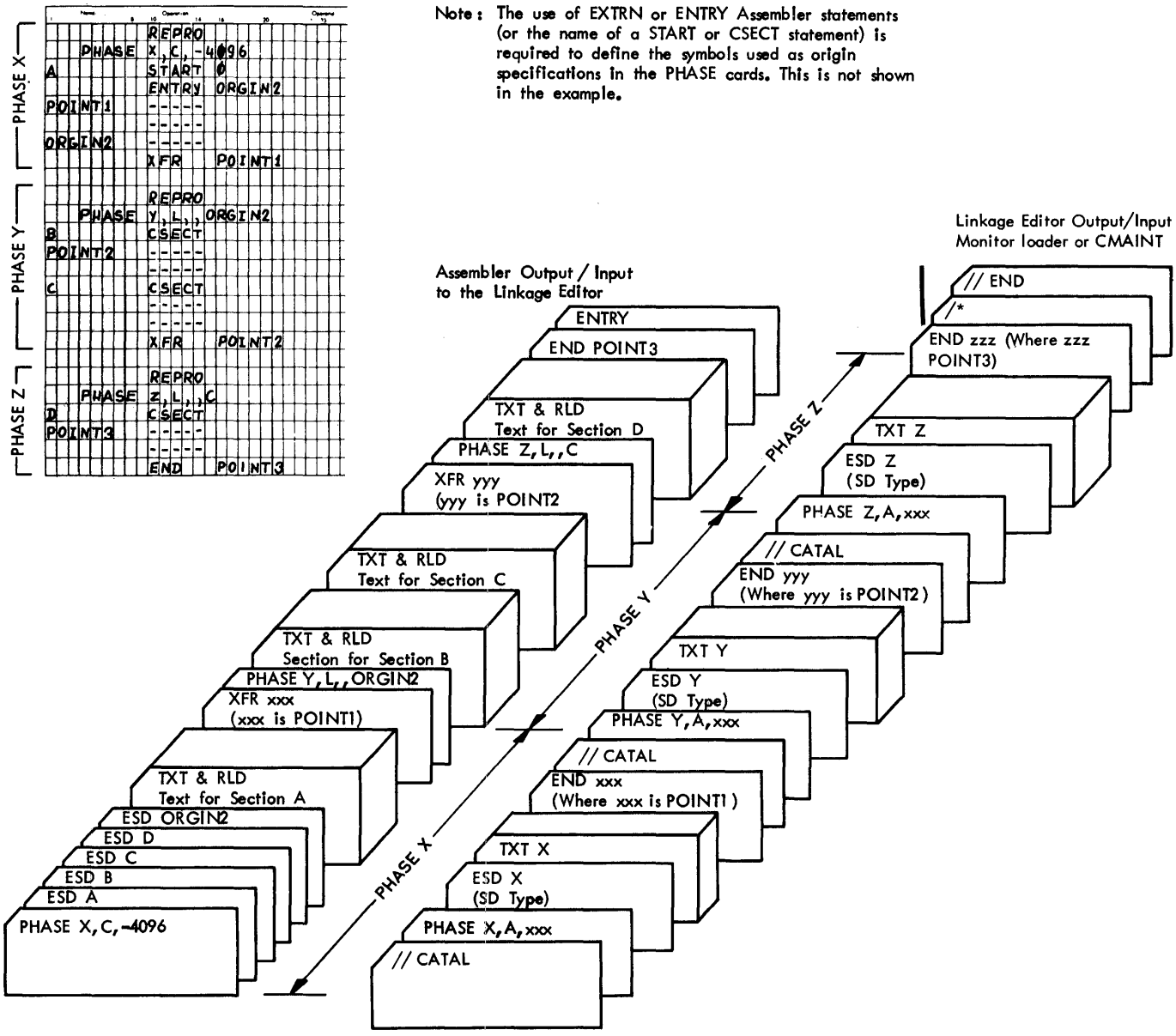
Source Program

```
LOCATN     OBJECT CODE    ADD1   ADD2   STMT       SOURCE STATEMENT

                                        0001              REPRO
                                                          PHASE   PROGA,A,4700
0000                                    0002   A          START   0
                                        0003              EXTRN   B
                                        0004              EXTRN   C
0000   0D80                             0005   BEG        BASR    8,0
0002                                    0006              USING   *,8

00CA   48A0   819A          019C       0010              LH      10,=Y(B)
00CE   48B0   819C          019E       0011              LH      11,=Y(C)
00D2   0D9B                            0012              BASR    9,11

019C   0000                            0016         # Y(B)
019E   0000                            0017         # Y(C)
0000                                    0018              END     BEG
```

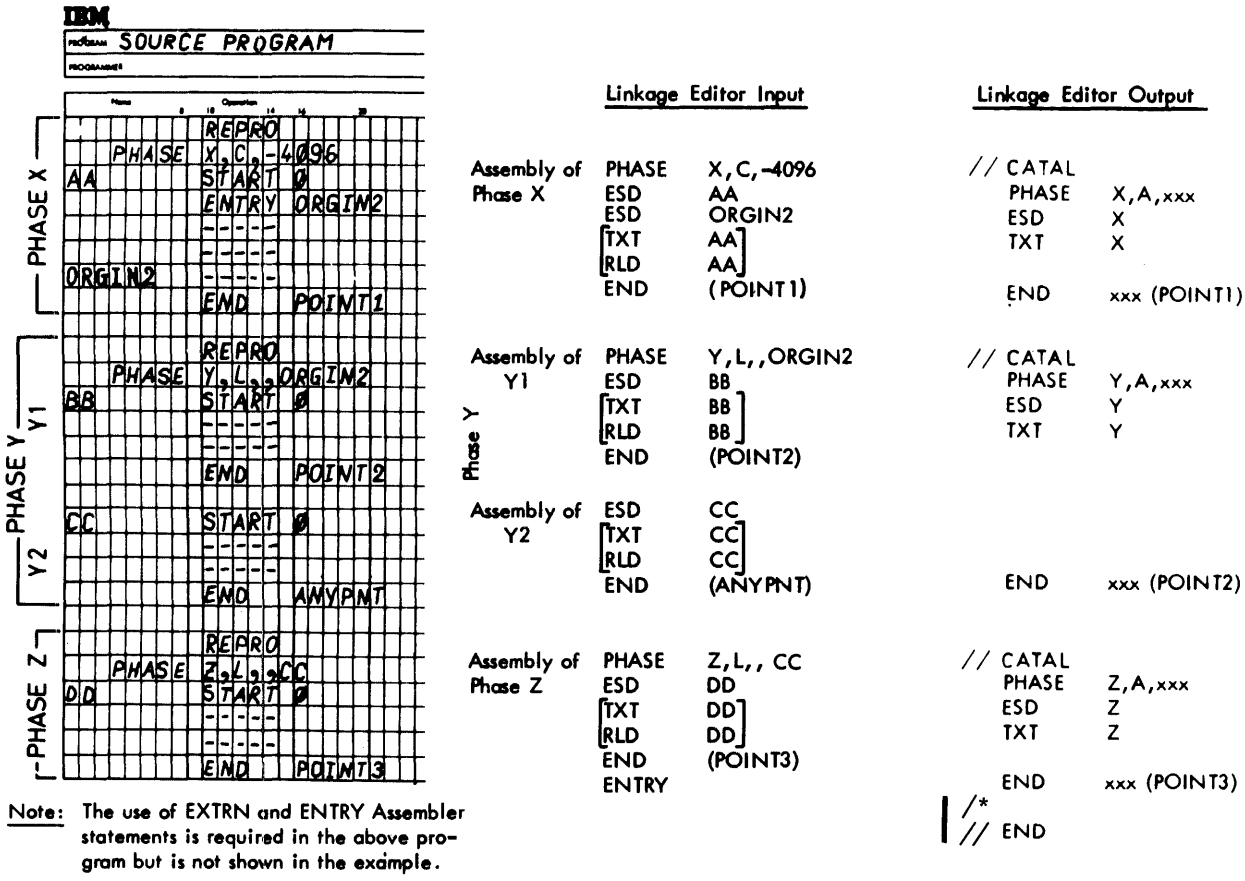
```
                                        0001              AOPTN   ENTRY
07D0                                    0002   B          START   2000
07E4                                    0003              ENTRY   C
07D0                                    0004              USING   *,11

07E4   4AE0   B400          0BD0       0008   C          AH      14,=H'5'

0BD0   0005                            0011         # H'5'
0BD2                                    0011   D          CSECT
0BD2   0DB0                            0012              BASR    11,0
0BD4                                    0013              USING   *,11

1084   07F9                            0017              BR      9
07D0                                    0018              END     B
```

Assembler Output/Input to the Linkage Editor	Output of the Linkage Editor	Comments
	// CATAL	Linkage Editor generates this card.
PHASE PROGA,A,4700	PHASE PROGA,A,X'125C'	Card is reproduced. Indicates the name and begin address of the phase.
ESD A(SD,ER)	ESD (SD)	SD's and ER's are relocated, ER's are combined with LD's of section B. Only one SD is produced for the whole phase.
TXT A	TXT A	TXT for section A.
RLD A		RLD's are relocated.
END A		
ESD B(SD,LD)		ESD information for sections B and D has been relocated and SD has been combined with ER of section A.
ESD D(SD)		
TXT B	TXT B	TXT for section B.
TXT D	TXT D	TXT for section D.
END B	END A	END B is rendered ineffective since an END card was previously encountered in this phase. END A remains effective since the ENTRY statement has no operand.
	/*	
ENTRY	// END	

●Figure 33. Using the Linkage Editor Program, Example 1



●Figure 34. Using the Linkage Editor Program, Example 2



Note: The use of EXTRN and ENTRY Assembler statements is required in the above program but is not shown in the example.

●Figure 35. Using the Linkage Editor Program, Example 3

Load System Disk Program (LDSYS)

This section describes the functions of the LDSYS program and the control statements you must supply to request these functions.

PURPOSE OF THE LOAD SYSTEM DISK PROGRAM

The LDSYS program enables you to build a disk-resident system that is especially tailored to your needs.

It loads the disk-resident portion of IPL and the Monitor into their fixed locations.

The LDSYS program furthermore enables you to build a core-image library that includes all IBM-supplied programs and executable problem programs that you need to operate your system effectively.

The LDSYS program can be executed under the control of both the disk-resident and the card-resident control systems.

In order to use the LDSYS program to create a disk-resident system, the input you furnish must be in punched cards or in card-image format on magnetic tape.

The minimum components your disk-resident system must include are:

- the disk-resident portion of IPL
- the Monitor
- the core-image library containing the Job Control program.

The disk-resident portion of IPL is a standard program supplied by IBM. You obtain this program in punched cards or in card-image format on magnetic tape by means of the CSERV program, and use this output as input to the LDSYS program.

The Monitor in the distribution package is the standard Monitor. If the features of the standard Monitor do not reflect your requirements, you can generate a Monitor tailored to your specific needs. The input you use for the LDSYS program can be (1) the standard Monitor obtained in punched cards or in card-image format on magnetic tape by means of the CSERV program, or (2) the output of a Monitor generation run.

The core-image library must contain the Job Control program. In addition, it may contain a selection of IBM-supplied programs and object programs that you will run frequently. You can obtain all IBM-supplied programs in punched cards or in card-image format on magnetic tape by means of the CSERV program. The problem programs you will run frequently must be in punched cards or in card-image format on magnetic tape and immediately executable (that is, already assembled or compiled) before you can use them as input to the LDSYS program.

You may also include the remaining components of the Disk Programming System in the system disk pack you create.

Before you can begin creating a disk-resident system, you must first initialize the disk pack as described in the SRL publication IBM System/360 Model 20, Disk Programming System, Disk Utility Programs, Form C26-3810.

If you wish to execute programs that are not already stored in your core-image library, you must include the CMAINT program on your disk pack. Then you can use the execute-loader function and read your program from SYSIPT (// EXEC LOADER) or from the relocatable area (// EXEC LOADER,R).

In addition, if you wish to assemble and execute or compile and execute source programs in one job, you must include the Assembler or RPG program and a relocatable area in your disk-resident system.

In order to store additional programs in the core-image library, you must also include the CMAINT program on the system disk pack.

JOB CONTROL STATEMENTS

The job control statements that are required for a LDSYS run are:

// LOG	optional
// JOB	required
// DATE	1st job after IPL only
// ASSGN SYSIPT	required
// ASSGN SYSOPT	required
// ASSGN SYSLOG	optional
// EXEC	required

JOB Control Statement

The format of the JOB control statement required to create a disk-resident system is:

Name	Operation	Operand
//	JOB	LDSYS

LDSYS
Indicates that a disk-resident system is to be built.

ASSGN Control Statements Preceding EXEC

Depending upon the medium on which input is stored, SYSIPT may be assigned either to a card reading device or to a magnetic tape drive.

SYSOPT must be assigned to the disk drive on which the disk-resident system is to be written.

We recommend that you assign SYSLOG to the printer so that all control statements, PHASE statements, and diagnostic messages can be printed.

If these assignments are still effective, you need not submit them again.

The remaining job control statements are not described here. You can find their correct formats in the section Job Control Program.

PROGRAM CONTROL STATEMENTS

The program control statements required for a LDSYS run are:

// LIMIT
// IPL
// MONTR
	required

// ASSGN	optional
// CONFG	optional
// END	required on SYSRDR if ≠ SYSIPT
// END	required on SYSIPT

LIMIT Control Statement

The LIMIT control statement specifies the areas of the system disk pack to be allocated to the directories, the libraries, and the relocatable area. The format of the LIMIT control statement is:

Name	Operation	Operands
//	LIMIT	xx,nnn[,xx,nnn...]

xx
Code indicating which of the libraries or library directories are to be allocated.

Code	Area
CD	Core-Image Directory
CL	Core-Image Library
MD	Macro Directory
ML	Macro Library
RL	Relocatable Area

nnn
Code in decimal notation indicating how many tracks are to be allocated. Leading zeros are not used.

It doesn't make any difference in what order you specify the areas to be allocated. But each area code xx must be followed by a comma and the number of tracks it is to occupy. If you want to allocate three areas, the operand will be:
xx,nnn,xx,nnn,xx,nnn.

If you specify a 0 after the area code xx, the area will be omitted from the system disk pack. A specification of zero has the same effect as omitting an area code entirely. The total number of tracks you allocate must not exceed the number of tracks on the disk pack (1986 tracks for Model 11, 986 tracks for Model 12).

The macro directory and the core-image directory must not occupy more than ten tracks a piece.

If you omit the macro library from the system disk pack, omit the macro directory as well (ML,0,MD,0 or omit both area codes).

Never omit the core-image library and the core-image directory.

IPL Control Statement

The IPL control statement instructs the LDSYS program that the disk-resident IPL deck follows and is to be loaded. The format of the IPL control statement is:

Name	Operation	Operand
//	IPL	

MONTR Control Statement

The MONTR control statement instructs the LDSYS program that the disk-resident Monitor program is to be loaded. The MONTR statement also informs the LDSYS program that ASSGN statements and possibly a CONFG control statement may follow. The format of the MONTR control statement is:

Name	Operation	Operand
//	MONTR	

ASSGN Control Statements Following EXEC

The ASSGN control statement is used to insert or change device assignments in the physical and logical unit tables of the Monitor of the disk-resident system to be created. The format of this control statement is described in the section Job Control Program.

CONFG Control Statement

The CONFG control statement is used to change the storage-capacity specification in the communication region of the Monitor of the disk-resident system to be created. It should not be used if the Monitor supports inquiry facilities. The format of this control statement is described in the section Job Control Program.

END Control Statements

Two END control statements are used in building the disk-resident system. One END control statement terminates the input from SYSIPT. The other END control statement terminates the input from SYSRDR. If SYSRDR refers to the same device as SYSIPT, the second END statement is not required.

The format of the END control statement is as follows:

Name	Operation	Operand
//	END	

SAMPLE LDSYS RUN

This example shows how a disk-resident system can be created by means of the LDSYS program operating under the control of the card-resident control system.

The following sequence of control statements and card decks are used to build the system. The comments below refer to Figure 36.

1. Card Initial Program Loader deck.
2. ASSGN control statements for SYSRES and SYSRDR.

3. Card-resident Monitor deck.

4. Card-resident Job Control deck.

5. JOB control statement for LDSYS program.

6. DATE control statement, to supply the date for the communication region.

7. ASSGN control statements affecting the card-resident Monitor in main storage, for example assigning SYSOPT to the disk unit that is to accommodate the disk-resident system to be created.

8. EXEC control statement. After this statement every interspersed blank card is ignored, therefore it is possible to partition the input card deck.

9. Load System Disk program deck.

10. LIMIT control statement for library and directory allocations.

11. IPL control statement to indicate the beginning of the disk-resident IPL.

12. Disk-resident IPL card deck.

13. MONTR control statement to indicate the beginning of the disk-resident Monitor.

Example A: Standard Monitor

14. Standard Monitor distributed by IBM.

15. Optional ASSGN control statements used to alter the permanent device assignments of the standard Monitor.

16. Optional CONFG control statement used to alter the standard storage specification in the communication region of the Monitor.

17. END control statement. Optional if SYSRDR=SYSIPT.

18. PHASE card for Job Closing routines.

19. Job Closing routines card deck.

20. PHASE card for Job Control program.

21. Job Control program deck.

22. Core-image library phases in card-image format beginning with a PHASE statement. A CATAL statement preceding the PHASE statement will be ignored.

23. END control statement signifying the end of phases for the core-image library. This card terminates the execution of the Load System Disk program.

Example B: Generated Monitor

14. Generated Monitor. Note that the first three cards in the output deck of a Monitor generation run are: // JOB CMAINT, // EXEC, and // MONTR. Since they apply only to a CMAINT run, they must be removed. The Monitor contains the permanent device assignments specified at generation time. It includes the Job Closing routines.

15. PHASE card for Job Control program.

16. Job Control program deck.

17. Core-image library phases in card-image format beginning with a PHASE statement. A CATAL statement preceding the PHASE statement will be ignored.

18. END control statement signifying the end of phases for the core-image library. This card terminates the execution of the Load System Disk program.

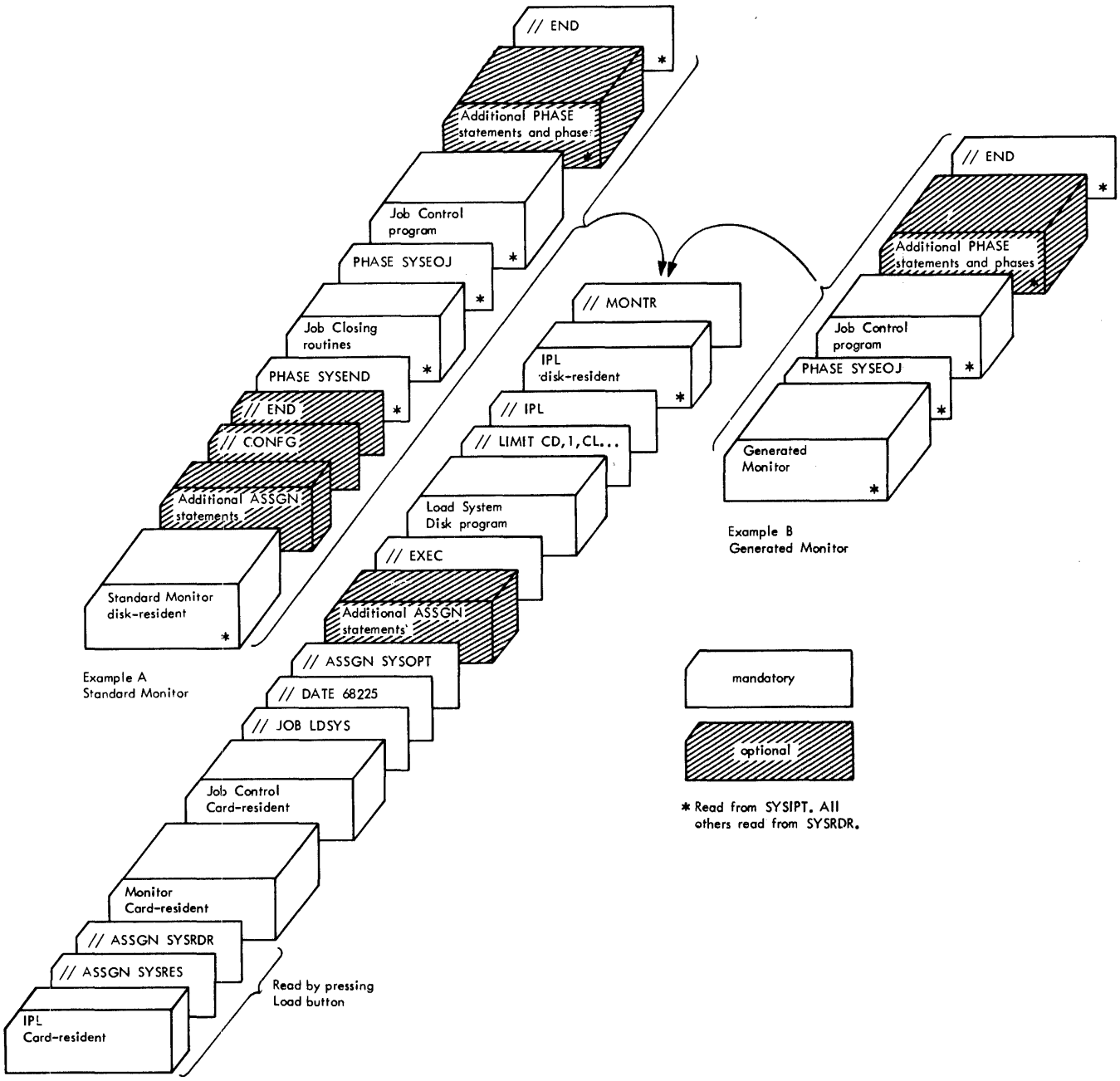


Figure 36. Sample LDSYS Run Under Control of the Card-Resident System

Copy System Disk Program (COPSYS)

The Copy System Disk program (COPSYS) is a service program that operates under the control of either the disk-resident or the card-resident Monitor.

The function of the COPSYS program is to transfer the system file from the system disk pack onto another disk pack for backup purposes.

The input to the COPSYS program is supplied by the system disk pack that is to be copied. The output of the program is written onto the disk pack assigned to SYSOPT. This disk pack must previously have been initialized by means of the INTDSK utility program so that it will contain a volume label and a VTOC.

JOB CONTROL STATEMENTS

The job control statements you must supply in order to request a COPSYS function are:

```
// LOG            optional
// JOB            required
// DATE           1st job after IPL only
// ASSGN SYSIPT   required
// ASSGN SYSOPT   required
// ASSGN SYSLOG   optional
// EXEC           required
```

LOG Control Statement

We recommend that you insert the LOG control statement so that all error messages and diagnostics can be printed on the device assigned to SYSLOG. In addition, a header statement and a concluding statement will then be printed on the printer by the COPSYS program to help you document your COPSYS run.

JOB Control Statement

The format of the JOB control statement required to request a COPSYS run is:

Name	Operation	Operand
//	JOB	COPSYS

COPSYS
Indicates that the system disk pack is to be copied.

ASSGN Control Statements

SYSIPT must be assigned to the disk drive on which the system disk pack to be copied is located.

SYSOPT must be assigned to the disk drive on which the system file is to be written.

We recommend that you also assign SYSLOG to the printer.

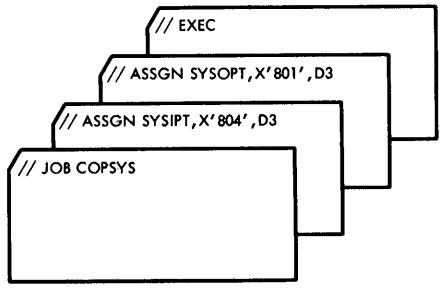
If these assignments are still effective, you need not submit them again.

The remaining job control statements are not described here. You can find their correct formats in the section Job Control Program.

PROGRAM CONTROL STATEMENTS

No program control statements are required for a COPSYS run.

Figure 37 shows a typical example of using job control statements to request a COPSYS function.



●Figure 37. Example of Using Control Statements to Request a COPSYS Function

Backup and Restore Program (BACKUP)

The Backup and Restore program enables you to create a backup tape from a disk file or, optionally, from a disk file and one or more card files, and to restore the backup information to its original media.

One application of the program is to make a backup tape of every disk file before you modify any data on that file. Then, if the modified disk file is accidentally destroyed, you can easily restore the backup file onto another disk pack.

The Backup and Restore program has four functions:

- create a backup tape
- initialize one or more disk packs
- restore one or more disk backup files
- punch or display one or more card backup files.

The first function, create a backup tape, runs under the control of the disk-resident system. It causes a tape-resident control system, the DPS Initialize Disk program, and the phases required for the remaining three functions to be written onto the backup tape in sequential order. It also punches a bootstrap card later used to initiate retrieval from the backup tape.

The last three functions then run under the control of the tape-resident control system on the backup tape. They are initiated by means of the bootstrap card punched when the backup tape was created.

Each function of the Backup and Restore program is described in detail in the following, together with the job control and program control statements you need to request each function.

CREATE A BACKUP TAPE

This function of the Backup and Restore program allows you to create a backup tape from a disk file and, optionally, from one or more card files. It also punches a bootstrap card that will initiate the retrieval of information from the backup tape when you require it.

Since a backup tape is created under the control of the disk-resident system, the Backup and Restore program must be contained in the core-image library of the system disk pack mounted on SYSRES. In addition, the DPS Initialize Disk utility program must also be contained in the core-image library, since this program is automatically written on every backup tape you create.

JOB CONTROL STATEMENTS

The job control statements required to create a backup tape are:

```
// LOG           recommended
// JOB           required
// DATE          1st job after IPL only
// ASSGN SYSIPT  required
// ASSGN SYSOPT  required
// ASSGN SYS000  depends on job
// ASSGN SYS001  depends on job
// UPSI 1        depends on job
// OPTN TES      recommended
// EXEC          required
```

Job Control Statement

The format of the JOB control statement required to create a backup tape is:

Name	Operation	Operand
//	JOB	BACKUP

BACKUP
Indicates that a backup tape is to be created.

ASSGN Control Statements

The use of ASSGN control statements depends on the job you wish to perform.

SYSIPT must be assigned to the disk drive on which the disk file to be copied is mounted. The assignments of SYSIPT and SYSRES may be identical.

SYSOPT must always be assigned to a magnetic tape drive. It is the symbolic device address of the drive on which the backup tape will be created.

SYS000 must be assigned to a card punching device if you want a bootstrap card to be punched. If you specify the option NOBOOT in the program control statements, you need not assign SYS000.

SYS001 must be assigned to a card reading device if you want to include one or more card files on the backup tape (OPTN CARDFILE included among the program control statements). Otherwise SYS001 need not be assigned. The assignment of SYS001 and SYSRDR may be identical.

The assignment of SYSLOG is optional. We recommend that you make this assignment so that error messages, tape error statistics, and the file identifications of backup files can be printed.

If any of these assignments are still effective, you need not submit them again.

UPSI Control Statement

An UPSI control statement with the format

Name	Operation	Operand
//	UPSI	1

is required if an uninitialized tape is at load point. This statement causes label checking to be skipped.

The remaining job control statements are not described here. You can find their correct formats in the section Job Control Program.

PROGRAM CONTROL STATEMENTS

With the exception of the END statement, all program control statements described here are optional. If you do not submit them, the program will copy the entire disk pack onto tape up to cylinder 102 or 202, and punch the bootstrap card later used to initiate retrieval.

By submitting program control statements, you can select the files you wish to write on the backup tape, and request or suppress certain options. These program control statements are:

```
// START          optional
// ENDTR          optional
// COPY           optional
// IDENT          optional
// OPTN           depends on job
// END            required
/& NAME           depends on job
/&                depends on job
```

Start Control Statement

By means of the START control statement, you indicate to the program the address of the first track of the disk pack to be written onto tape. The format of this control statement is:

Name	Operation	Operand
//	START	ccchh

ccc
Cylinder number in decimal notation

hh
Track number in decimal notation

The START control statement is optional. If you do not submit it, START 00000 is assumed.

The START control statement does not apply to the disk volume label and the VTOC, which are always written onto the backup tape.

ENDTR Control Statement

This control statement indicates to the program the address of the last track of the disk pack to be written onto tape. The format of the ENDTR control statement is:

Name	Operation	Operand
//	ENDTR	ccchh

ccc
Cylinder number in decimal notation

hh
Track number in decimal notation

The ENDTR control statement is optional. If you do not submit it, ENDTR 10209 or 20209 is assumed, depending on the model of the disk drive you assigned as input.

The ENDTR control statement does not apply to the disk volume label and the VTOC, which are always written onto the backup tape.

COPY Control Statement

The COPY control statement indicates which data files are to be written onto tape.

The formats of the COPY control statement are:

Name	Operation	Operand
//	COPY	ALL
//	COPY	UNEXP
//	COPY	EXP
//	COPY	NAME

ALL
All files completely within the limits defined in the START and ENDTR control statements are written onto tape

UNEXP
All unexpired files completely within the limits defined in the START and ENDTR control statements are written onto tape

EXP
All expired files completely within the limits defined in the START and ENDTR control statements are written onto tape

NAME
The disk files specified in the cards that immediately follow this statement are to be written onto tape.

The cards specifying the files have the format:

Col. 1-15 blank
Col. 16-59 file identification (44 characters as they appear in the DLAB statement)
Col. 60-80 comments or numbering

If you do not submit the COPY statement, COPY ALL is assumed.

IDENT Control Statement

By using the IDENT control statement, you have the option of writing an identifying code on the backup tape. The format of the IDENT control statement is:

Name	Operation	Operand
//	IDENT	code

The identifying code you insert into the operand field will be printed out on SYSLOG when the information on the backup tape is written back onto disk.

OPTN Control Statement

The format of the OPTN control statement is:

Name	Operation	Operands
//	OPTN	CARDFILE
//	OPTN	NOBOOT

CARDFILE
Use an OPTN statement with this operand if you want to write one or more card files on the backup tape. Remember that in this case SYS001 must refer to the card reading device on which the card files can be read.

NOBOOT
Use an OPTN statement with this operand if you do not want a bootstrap card punched. You may still have bootstrap cards available from previous runs.

END Control Statement

The END control statement is always required. Its format is:

Name	Operation	Operand
//	END	

/& NAME Control Statement

If you wish to include one or more card files on the backup tape (OPTN CARDFILE among the control statements), you must precede each card file on SYS001 with the control statement:

Name	Operation	Operand
/&	NAME	name

name
The name of the card file to be written onto tape behind the disk backup file. The name may be up to six characters long. The first character must be alphabetic, the others may be alphabetic or numeric.

The name of the card file is written onto the backup tape and later used to retrieve the file.

You can write several card files on the backup tape in one job. You must precede each card file with a /& NAME control statement, and the names must be in ascending order according to the collating sequence.

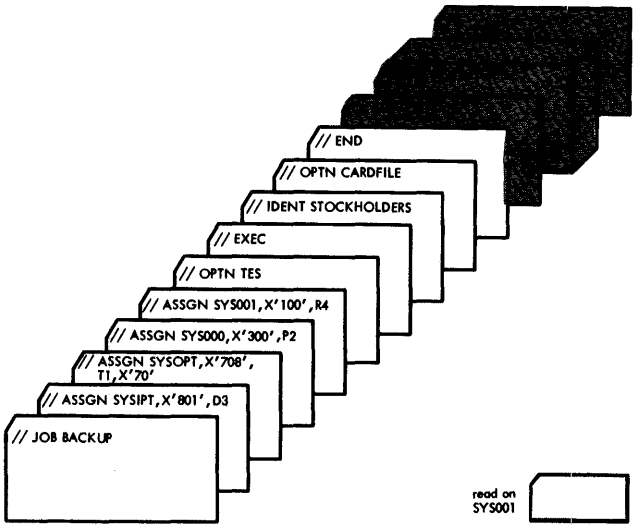
/& Control Statement

This control statement indicates to the program that no further card files are to be included on the backup tape. The format of the control statement is:

Name	Operation	Operand
/&		

This control statement must be inserted behind the last card file on SYS001.

Figure 38 is an example of using job control and program control statements to create a backup tape.



● Figure 38. Example of Using Control Statements to Create a Backup Tape

INITIALIZE ONE OR MORE DISK PACKS

The input for this function of the Backup and Restore program is the backup tape, on which the DPS Initialize Disk program was written at creation time.

This program runs under the control of the tape-resident control system included on the backup tape, therefore a system disk pack is not used. All you need is the backup tape and the bootstrap card. After you initiate the retrieval of information from the backup tape by means of the bootstrap card, you request the initialize function of the Backup and Restore program if your disk pack is not yet initialized.

By means of the initialize function you can initialize one disk pack or several disk packs in succession and write a volume serial number on each of them.

JOB CONTROL STATEMENTS

Use the following job control statements to initialize a disk pack using the Backup and Restore program:

// JOB	required
// DATE	required if the execution of the program immediately follows the loading of the bootstrap card
// ASSGN SYSRDR	depends on job
// ASSGN SYSOPT	required
// RPT	required for each additional disk pack you wish to initialize
// EXEC	required

Job Control Statement

To call the DPS Initialize Disk program from the backup tape into main storage, use the control statement

Name	Operation	Operand
//	JOB	INTDSK

INTDSK
Identifies the job as an initialize-disk run.

ASSGN Control Statements

To initialize a disk pack, you must assign SYSOPT to the disk drive on which the disk pack to be initialized is mounted.

If the card reading device in your configuration is not a 2501 Card Reader, and if initialization is the first job after loading the bootstrap card, you must assign SYSRDR to your card reading device.

RPT Control Statement

If you wish to initialize more than one disk pack in succession, use the control statement

Name	Operation	Operand
//	RPT	

For each additional disk pack you wish to initialize, you must repeat all the control statements required for JOB INTDSK and insert the RPT control statement before the EXEC statement.

The RPT control statement causes the backup tape to be rewound and spaced forward to the point where initialization of the additional disk pack can begin.

The remaining job control statements are not described here. You can find their

correct format in the section Job Control Program.

PROGRAM CONTROL STATEMENTS

The program control statements required to initialize the disk pack are described in the SRL publication IBM System/360 Model 20 Disk Programming System, Disk Utility Programs, Form C26-3810.

RESTORE ONE OR MORE DISK BACKUP FILES

The input for this function of the Backup and Restore program is the backup tape, on which the restore phases were written at creation time.

This program runs under the control of the tape-resident control system included on the backup tape, therefore a system disk pack is not used. If the first function you request is the restore function, you initiate the retrieval of information from the backup tape by means of the bootstrap card and submit job control and program control statements on the device assigned to SYSRDR.

By means of the restore function, you can restore the disk backup file onto one disk pack, or onto several initialized disk packs in succession.

JOB CONTROL STATEMENTS

Use the following job control statements to restore a disk backup file onto a disk pack:

// LOG	recommended
// JOB	required
// DATE	required if the execution of the program immediately follows the loading of the bootstrap card
// ASSGN SYSRDR	depends on job
// ASSGN SYSOPT	required
// RPT	required for each additional disk pack onto which you restore the backup file
// EXEC	required

LOG Control Statement

We strongly urge you to insert the LOG control statement.

Before the backup file is transferred to disk, the program compares the extents of the backup file with the extents of unexpired files already on the disk pack. If they overlap, the unexpired files will have to be deleted before the backup file can be transferred to disk. If you insert the LOG control statement, the program will print the file identification of such unexpired files so that you can decide whether to continue the job or not.

The program will also print information about the contents of the disk pack.

JOB Control Statement

The JOB control statement required to restore a disk backup file onto a disk pack is:

Name	Operation	Operand
//	JOB	RESTOR

RESTOR
Identifies the job as a restore operation.

ASSGN Control Statements

To restore a disk backup file onto disk, you must assign SYSOPT to the disk drive onto which the backup file is to be written.

Note that you need not submit an ASSGN control statement for SYSIPT. Instead, the operator will use the console switches to insert the physical device address of the backup tape drive.

If the card reading device in your configuration is not a 2501 Card Reader, and if restoring a disk backup file is the first job after loading the bootstrap card, you must assign SYSRDR to your card reading device.

RPT Control Statement

If you wish to restore the disk backup file onto more than one disk pack in succession, use the control statement

Name	Operation	Operand
//	RPT	

For each additional disk pack onto which you wish to restore the disk backup file, you must repeat all the control statements required for JOB RESTOR and insert the RPT control statement before the EXEC statement.

The RPT control statement causes the backup tape to be rewound and spaced forward to the point where the backup file can be restored to another disk pack.

PROGRAM CONTROL STATEMENTS

The program control statements used to restore a disk backup file onto a disk pack are:

// OPTN	optional
VOL1nnnnnn	optional
// END	required

OPTN Control Statement

By means of the OPTN control statement, you specify whether the information restored onto the output disk pack is to be verified, that is, compared with the contents of main storage.

The formats of the OPTN control statement are:

Name	Operation	Operands
//	OPTN	VERIFY=YES
//	OPTN	VERIFY=NO

The operands are self-explanatory. If you do not submit this control statement, VERIFY=YES is assumed.

VOL1 Control Statement

By means of the VOL1 control statement, you instruct the program to write the specified volume serial number onto the output disk pack. The format of this statement is:

Name	Operation	Operand
VOL1nnnnnn		name

nnnnnn
Columns 5-10. The volume serial number to be written onto the disk pack.

name
Columns 42-51. Owner's name code.

Use the VOL1 statement with discretion and with utmost care. Normally, the volume serial number of the output disk pack is compared with the volume serial number of the disk pack from which the backup tape was made. But when you use the VOL1 statement, this diagnostic is suppressed. Moreover, the volume serial number you specify in this statement is written on the output disk pack, and the file serial numbers of all files on this disk pack are changed accordingly. If you have multi-volume files and the disk pack assigned to SYSOPT is the first volume of such a file, all files on this disk pack are in jeopardy if the volume serial number (and hence the file serial number) is changed. So before you use this statement, make double sure that the volume serial number you specify is correct.

END Control Statement

The format of the END control statement is:

Name	Operation	Operand
//	END	

PUNCH OR DISPLAY ONE OR MORE CARD BACKUP FILES

The input for this function of the Backup and Restore program is the backup tape, on which the punch and display phases were written at creation time.

This program runs under the control of the tape-resident control system included on the backup tape, therefore a system disk pack is not used. If the first function you request is the punch or display function, you initiate the retrieval of information from the backup tape by means of the bootstrap card and request the punch and display function of the Backup and Restore program by submitting job control and program statements on the device assigned to SYSRDR.

The card files must be retrieved from the backup tape in the same order in which they were written onto the backup tape. If you request a retrieval sequence other than the collating sequence, a diagnostic halt will occur.

The punch and display function must be the last function you request when you use the backup tape as input, because when this function is completed, the backup tape is automatically rewound and unloaded, and no further functions can be requested.

JOB CONTROL STATEMENTS

Use the following job control statements to punch or display a card backup file:

// JOB	required
// DATE	required if the execution of the program immediately follows the loading of the bootstrap card
// ASSGN SYSOPT	depends on job
// ASSGN SYSLST	depends on job
// EXEC	required

JOB Control Statement

The JOB control statement required to print or punch a card backup file has the format:

Name	Operation	Operand
//	JOB	PUNCH

PUNCH
Indicates that a card backup file is to be punched or displayed.

ASSGN Control Statements

To punch a card backup file you must assign SYSOPT to a card punching device.

To display a card backup file, you must assign SYSLST to the printer.

Note that you need not submit as ASSGN control statement SYSIPT. The operator will use the console switches to insert the physical device address of the backup tape drive.

If the card reading device in your configuration is not a 2501 Card Reader, and if punching or displaying a card backup file is the first job after loading the bootstrap card, you must assign SYSRDR to your card reading device.

The remaining job control statements are not described here. You can find their correct formats in the section Job Control Program.

PROGRAM CONTROL STATEMENTS

The program control statements required to punch or display a card backup file included on the backup tape are:

// PUNCH
// DSPLY
// DSPCH
	depends on job

// END	required

PUNCH, DSPLY, and DSPCH Control Statements

The PUNCH control statement instructs the program to punch a card backup file.

The DSPLY control statement instructs the program to display a card backup file on the printer.

The DSPCH control statement instructs the program to display as well as to punch a card backup file. These operation are then performed simultaneously.

The formats of these control statements are:

Name	Operation	Operand
//	PUNCH	name
//	DSPLY	name
//	DSPCH	name

name
The name given to the card-image file when the backup tape was created.

You can insert these control statements in any number and any sequence, but the names specified in the operand fields must be in ascending order according to the collating sequence. You may, however, request a file more than once in succession.

END Control Statement

The format of the END control statement is:

Name	Operation	Operand
//	END	

The END control statement indicates that no further card backup files are requested, and causes the backup tape to be rewound and unloaded.

As we explained in the introduction to this chapter, the three functions

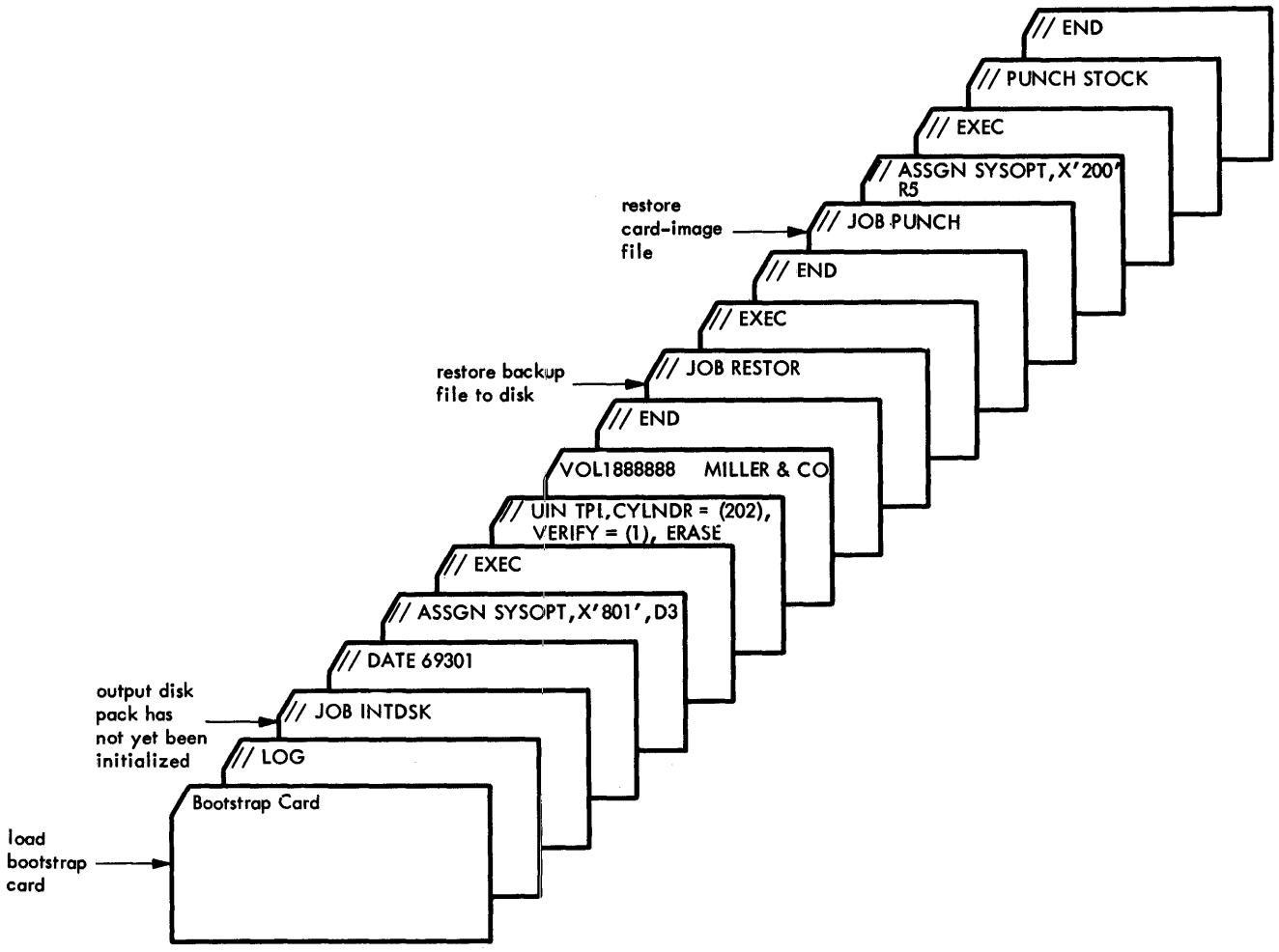
- initialize
- restore
- punch or display

are stored on the backup tape in sequential order. Once you load the bootstrap card that initiates the retrieval of information from the backup tape, you can request these functions by submitting the appropriate job control statements to SYSRDR. But remember that it is time-saving to request the func-

tions in the same order as they are stored on tape.

Always request JOB PUNCH last, because it causes the backup tape to be rewound and unloaded.

Figure 39 shows an example of job control and program control statements that request all three functions, using the backup tape as input.



● Figure 39. Example of Using Control Statements to Initialize a Disk Pack, Restore a Disk File to Disk, and Punch a Card File

IBM Distribution Package

IBM distributes the entire Model 20 Disk Programming System on a disk pack or magnetic tape supplied by the user. The distribution package includes a retrieval program to copy the disk-resident system and to punch the card-resident Initial Program Loader deck.

The distribution package on the disk pack or magnetic tape you supply includes:

- Disk IPL, Part 2.
- Disk-resident standard Monitor.
- Core-image library containing the IBM programs listed in Figure 40.
- Macro library containing Monitor generation macro definitions, IOCS macro definitions, and Monitor macro definitions.
- Sample programs in absolute card format.

Program	Name
Library Allocation Organization	AORGZ
Alternate-Track Assignment Utility	ATASGN
Backup and Restore	BACKUP
Card-to-Disk Utility	CARDSK
DPS Card-to-Tape Utility	CARTAP
Clear Disk Utility	CLRDSK
Copy System Disk	COPSYS
Core-Image Maintenance	CMAINT
Core-Image Service	CSERV
Disk Dump	DDUMP
Directory Service	DSERV
Disk-to-Card Utility	DSKCAR
Disk-to-Disk Utility	DSKDSK
Disk-to-Printer Utility	DSKPRT
Disk-to-Tape Utility	DSKTAP
DPS Initialize Tape Utility	INITTP
Initialize Disk Utility	INTDSK

●Figure 40. IBM Programs in the Core-Image Library of the Distribution Package, Part 1 of 2

Program	Name
Job Control	SYSEOJ
Job Closing Routines	SYSEND
Linkage Editor	LNKEDT
Load System Disk	LDSYS
Macro Maintenance	MMAINT
Macro Service	MSERV
PL/I Compiler	PL1
Physical and Logical Unit Tables Service	PSERV
Report Program Generator	RPG
Disk Sort/Merge	SORT
DPS Tape-to-Card Utility	TAPCAR
Tape-to-Disk Utility	TAPDSK
DPS Tape-to-Printer Utility	TAPPRT
DPS Tape Sort/Merge	TAPSRT
DPS Tape-to-Tape Utility	TAPTAP

Figure 40. IBM Programs in the Core-Image Library of the Distribution Package, Part 2 of 2

By means of the IBM programs contained in the core-image library, you can add your own programs and delete those components which are not required in your installation.

You can also build your own disk-resident system for special applications. For information on how to do this, refer to the section Load System Disk Program (LDSYS).

If you want to generate a Monitor tailored to your needs, refer to the SRL publication IBM System/360 Model 20, Disk Programming System, System Generation and Maintenance, Form C33-6006.

The contents of the distribution package can also be obtained in punched cards. Use the CSERV program to obtain IBM programs, use the MSERV program to obtain macro definitions, and use the Disk-to-Card Utility program to obtain the sample programs in punched cards.

Dotted line flags planning information

Because of its flexibility, the Disk Programming System can be adapted to the environment in which it is to operate.

Basically, the Disk Programming System can be either card-resident or disk-resident.

The disk-resident system, furthermore, can operate either as a fixed-job or as a variable-job system.

The following sections describe how each of these systems operate, and what components they contain.

Disk-Resident Control System

The disk-resident control system is used to translate problem programs written in Assembler language or RPG.

It consists of the standard programs and areas found on cylinders 0 and 4 of the system disk pack (see Figure 2) and the core-image library. The core-image library must contain the Job Control program. In addition, it must contain the programs you need to compile, assemble, or execute problem programs.

If you wish to compile-and-execute programs written in RPG, your disk-resident system must have the RPG program in the core-image library and a relocatable area somewhere on the disk pack.

If you wish to assemble-and-execute programs written in the Assembler/IOCS language, your disk-resident system must have the Assembler program in the core-image library, a relocatable area somewhere on the system disk pack, and a macro library for IBM-supplied macro definitions.

The core-image library may also contain a selection of IBM-supplied programs that will help you operate your disk-resident system effectively.

Card-Resident Control System

In a card-resident control system, all three control programs -- IPL, the Monitor, and Job Control -- are in punched cards and not on a system disk pack. The object programs (output of assembly runs or RPG compilations) are executed under the control of this card-resident system.

The card-resident system offers advantages especially if you have only one disk drive.

After the source programs are assembled or compiled under control of the disk-resident system, they can be executed without the system disk pack under control of the card-resident system. The disk drive is now available as I/O device for execution.

The card-resident system does not use any additional disk areas besides the two areas used for label checking: the label information area (cylinder 0, track 1 or tracks 1 and 2) and the area containing the Volume Table of Contents. (For more detailed information, refer to the subsequent section Single-Drive System Considerations and the section Disk Label Processing in Appendix A.)

Disk-Resident Fixed-Job System

A fixed-job system must include the standard programs and areas found on cylinders 0 and 4 of the system disk pack (see Figure 2), a core-image directory, and a core-image library containing the Job Control program and the problem programs to be run under control of the system.

The program to be executed is loaded from the system disk pack into main storage by means of the Fetch routine contained in the Monitor.

Disk-Resident Variable-Job System

A variable-job system must include the standard programs and areas found on cylinders 0 and 4 of the system disk pack (see Figure 2), a core-image directory, and a core-image library containing the Job Control program, the CMAINT program, and sufficient space to accommodate the largest problem program to be run under control of the system.

When a program is to be executed it is read into main storage from cards or tape, temporarily placed into the core-image library by the CMAINT program, and then loaded into main storage by the Fetch routine of the Monitor.

Estimated Disk-Storage Requirements

The following estimates are provided to assist you in planning your system. They refer to a minimum variable-job system.

No. of Tracks	Contents
1	IPL part 2
44	the standard programs and areas found on cylinders 0 and 4 of the system disk pack (see Figure 2)
1	Core-image directory
20	Core-image library

Each entry in the core-image directory is 30 bytes long, so that this track can accommodate entries for 90 program phases. Approximately 70 of these 90 phases may be user-program phases.

Program phases are written in the core-image library in fixed-length records of 270 bytes each. There are 10 records per track. A phase always begins with a new record, but need not begin with a new track. The above estimate (20 tracks) includes the requirements for the Job Control program and the CMAINT program, which together occupy approximately 10 tracks, leaving 10 tracks for the user's problem programs. These 10 tracks can, for example, hold approximately six 4K phases.

Single-Drive System Considerations

Model 20 systems with a single IBM 2311 Disk Storage Drive require at least a system disk pack to permit the assembly or compilation of programs. The user's object programs thus obtained can be executed under control of the card-resident system (no additional core-image library space on the disk packs) or by means of a disk-resident fixed-job or variable-job system.

Special consideration must be given to jobs that process files stored on more than one disk pack.

Single-Phase Programs. Single-phase programs processing files on a single drive that are stored on more than one disk pack can be executed under control of the card-resident system or a fixed-job or variable-job disk-resident system.

If the card-resident system is used, every disk pack used for the file must contain the label information required for label checking. This information must be written on each pack by means of separate Job Control runs under control of the card-resident system before the file can be processed.

The same applies if a fixed-job or variable-job disk-resident system is used. In addition, however, the first disk pack must contain the control programs and the problem program to be executed.

Multiphase Programs. Multiphase programs that process multi-volume disk files on a single drive should make use of the disk-resident fixed-job system. Here, too, the disk label information must be written on every pack by means of separate Job Control runs. In addition, the control programs and the program phases must be included in every disk pack for the file to permit phases to be retrieved selectively.

Alternate Track Area. An area of three cylinders on the disk pack in which tracks may be used as alternatives to defective tracks occurring elsewhere on the disk pack.

Assemble-and-Execute. An operation in which a program is first assembled and then executed immediately in the same job.

Backup and Restore Program (BACKUP). A DPS Service program. Enables you to create a backup tape from a disk file and one or more card files, and to restore each backup file to its original medium.

Binary Synchronous Communications Adapter (BSCA). A feature that may be built into the Central Processing Unit of a Submodel 2, 4, or 5. It permits the system to function on a switched or leased communications network as a processor terminal.

Card-Resident System. Consists of the card control programs (Initial Program Loader, Monitor, and Job Control). Used for the execution of object programs contained in punched cards.

Communication Region. An area of the Monitor. Contains date, storage-capacity specification, UPSI byte, user areas 1 and 2, and program-name area. Provides for inter-program and intra-program communication.

Communications Error Statistics (CES). A record of errors occurring in BSCA transmission. In generating a Monitor with BSCA support, the user automatically generates the routine that records and analyzes these errors.

Compile-and-Execute. An operation in which a program is first compiled and then executed immediately in one job.

Copy System Disk Program (COPSYS). A DPS Service program. Enables you to copy a system disk pack onto another disk pack.

Core-Image Directory. A table on the system disk pack containing the addresses and extents of the programs and/or program phases in the core-image library.

Core-Image Format. A format identical to that used in main storage. It facilitates rapid loading from the core-image library into main storage without intermediate processing.

Core-Image Library. A disk area containing the Job Control program, other IBM-supplied programs (except the Monitor and the IPL), and user's problem programs. Permits retrieval of programs and/or phases by the Monitor.

Core-Image Maintenance Program (CMAINT). A DPS Service program. Updates the core-image library and directory. Is used to add and/or delete phases.

Core-Image Service Program (CSERV). A DPS Service program. Permits the listing, writing, or punching of one or more entries of the core-image library.

Directory Service Program (DSERV). A DPS Service program. Causes printing of the system and/or core-image and/or macro directory.

Disk-Resident System. Contains the Monitor, the disk-resident portion of the IPL, and the Job Control program. May contain any IBM-supplied and/or user-written programs and/or macro definitions as well as a relocatable area.

DPS Control Programs. A collective term used to refer to the Initial Program Loader, the Monitor program, and the Job Control program.

DPS Service Programs. A collective term used to refer to the Library Management programs, the PSERV program, the Linkage Editor program, the AORGZ program, the Load System Disk program, the Copy System Disk program, and the Backup and Restore program.

External Symbol Identification (ESID). ESID numbers are Assembler-assigned pointers that are used by the Linkage Editor to correctly recompute the constants referred to in RLD entries.

Fixed-Job System. A fixed-job sytem must include the standard programs and areas found on cylinders 0 and 4 of the system disk pack, a core-image directory, and a core-image library containing the Job Control program and the user's problem programs. The programs to be executed are loaded into main storage from the core-image library.

Initial Program Loader (IPL). A DPS Control program. Available in a card and a

disk version. The card version is contained in punched cards, the disk version is partly contained in a deck of three punched cards, partly in an area at the beginning of the system disk pack. Loads Monitor into main storage. Is used to assign physical I/O device addresses to symbolic addresses SYSRES and SYSRDR. Required for the initialization of a card-resident or disk-resident system run.

Inquiry Programs. Inquiry programs are initiated by pressing the Request key on the printer-keyboard and typing in the name of the program. The current contents of main storage (excluding the Monitor) are rolled out on the system disk pack; then the inquiry program is loaded and processed; after execution is completed, the old status is restored and execution of the mainline program resumes. Inquiry programs can be executed only under control of a Monitor that supports inquiry facilities. The execution of inquiry programs is not preceded by a Job Control run.

Inter-Program Communication. The exchange of data between two or more programs. Facilitated by the communication region.

Intra-Program Communication. The exchange of data between two or more phases of a multi-phase program. Facilitated by the communication region.

Job Control Program. A DPS Control program. Resides in main storage between jobs and provides for automatic job-to-job transition. Performs I/O device assignment. Causes Monitor to load next program.

Library Allocation Organization Program. A DPS Service program. Used to redefine the limits of the core-image library and directory, the macro library and directory, and the relocatable area.

Label Information Area (LIA). An area on the system disk pack into which disk file label information, as contained in the VOL, DLAB, and XTENT statements, is placed by the Job Control program. This information is used by the label processing routines. Tape file label information is stored in the upper portion of main storage when magnetic tape I/O is required.

Library Management Programs. Collective term for six DPS Service programs: Core-Image Maintenance, Macro Maintenance, Core-Image Service, Macro Service, Directory Service, and Allocation Organization programs.

Library Work Area. An area on the system disk pack used by the Core-Image Maintenance program when updating the Monitor program or IPL, and for storing tape label information in assemble-and-execute and compile-and-execute runs.

Linkage Editor Program (LNKEDT). A DPS Service program. Relocates programs or phases and links separately assembled programs or phases.

Load System Disk Program (LDSYS). A DPS Service program. Creates a disk-resident system from card input. Is executed under control of the card-resident or the disk-resident system.

Logical Unit Block (LUB). An entry in the Logical Unit Table.

Logical Unit Table. A feature of the Monitor program. It has 26 logical unit blocks, each of which refers to one specific symbolic I/O address. These symbolic addresses are related to physical I/O device addresses by means of ASSGN control statements.

Macro Directory. A table on the system disk pack listing the macro names, begin addresses, and area sizes of the macro definitions contained in the macro library. Can be listed on a printer by means of the Directory Service program.

Macro Name. An entry in the macro directory that identifies the corresponding macro definition in the macro library. Serves as an operation code for the associated macro instruction.

Macro Library. A disk area containing the macro definitions required by the macro instructions issued in user-written programs.

Macro Maintenance Program (MMAINT). A DPS Service program. Updates the macro library and directory. Is used to add and/or delete macro definitions.

Macro Service Program (MSERV). A DPS Service program. Permits the listing, writing, or punching of one or more macro definitions from the macro library.

Monitor I/O Area. An area of main storage within the Monitor used as a buffer by the Fetch routine when loading problem programs.

Monitor Program. The main control program. Resident in core storage throughout a system run. IBM distribution package contains the standard Monitor and several Monitor macro definitions. Instead of employing the standard Monitor, the user can tailor a Monitor according to his system requirements by specifying certain macro instruc-

tions, and generate it by means of an assembly run.

Object Module. A set of statements produced as a result of the translation of the source statements of a complete control section.

Permanent Link Data Area. A part of the Monitor with a fixed location in main storage; used for inter- and intra-program communication by system programs.

Phase. The smallest addressable unit in the core-image library of a disk-resident system.

Physical and Logical Unit Tables Service Program (PSERV). A DPS Service program. This program is used to print and/or change the permanent device assignments, and/or to change the storage-capacity byte in the communication region of the Monitor stored on the system disk pack.

Physical Disk and Tape I/O Routines. A set of routines that is contained in the Monitor program and performs tape and disk I/O operations for the Monitor and problem programs.

Physical Unit Block (PUB). An entry in the Physical Unit Table.

Physical Unit Table. A feature of the Monitor program. It has up to ten physical unit blocks, each of which contains a physical device address. Pointers to these entries are inserted into the logical unit table by means of ASSGN control statements.

Relocatable Area. An area on the system disk pack to temporarily hold an object module, thus permitting the assembly or compilation and the execution of a program or program phase in one job.

Segment. A program or phase that has been separately assembled.

Subphase. A separately executable routine within a phase of a problem program. It may be overlaid after execution. The method of building a program from subphases is used when a large problem program is to be executed.

Symbolic Device Address. A symbol used in IBM-supplied and user-written programs to refer to an I/O device (e.g., SYSRES, SYSIPT, SYS005). This address is related to a physical device address by means of the logical unit table.

System Directory. A table on the system disk pack listing the addresses and sizes of the core-image library and directory, the macro library and directory and the relocatable area.

System Disk Pack. The disk pack on which the user's disk-resident system is stored.

Tape Error Recovery Routine (TER). A routine to control the execution of error recovery procedures in the case of magnetic tape I/O errors.

Tape Error Statistics Routine (TES). A routine to analyze the interrupts and magnetic tape I/O errors occurring during the execution of a program.

User Program Switch Indicators (UPSI). A field of one byte within the communication region of the Monitor program. Specified bits (switches) may be set by means of the UPSI control statement and tested in user's programs.

Variable-Job System. A variable-job system must include the standard programs and areas found on cylinders 0 and 4 of the system disk pack, a core-image directory and a core-image library containing the Job Control program, the CMAINT program, and sufficient space to accommodate the largest problem program to be run under control of the system. A program to be executed is read into main storage from punched cards or magnetic tape, temporarily placed into the core-image library by CMAINT, then loaded into main storage by the Fetch routine and executed.

Volume Label. The volume label identifies and protects the entire volume (disk pack or magnetic tape reel). It is fixed in length and format, and lies in a fixed location within the volume. The volume label contains the volume serial number, the address of the VTOC, the address of the last permanent label in the LIA, and an indicator that specifies whether the LIA occupies one or two tracks.

Volume Table of Contents (VTOC). A number of records on a disk pack, composed of disk file labels, specifying the extents of, and identifying all files on the pack.

VTOC File Label. The first label in the VTOC. It identifies the VTOC and specifies its limits.

Appendix A. Disk Labeling Conventions

The Model 20 Disk Programming System provides positive identification and protection of all disk files by recording labels on each disk pack. These labels ensure that the correct pack is used for input and that no current information is destroyed on output.

If the Model 20 Disk Programming System is used, standard disk labels are required on all disk packs.

The standard label set includes one IBM volume label for each pack and one or more file labels for each logical file on the pack.

Standard IBM Volume Label

The standard IBM volume label identifies and protects the entire volume (pack). It is always the first record on cylinder zero, track one. It is fixed in length and format.

The standard IBM volume label contains a volume serial number. This number is assigned to the disk pack when it is prepared for use in the system. The number is normally not changed.

The only fields in the standard volume label that are used by the Model 20 Disk Programming System are the volume serial number field and the field with the address of the area containing the file labels.

The standard IBM volume label for disk has the same length and format as that for tape (see Appendix C).

Creation of Volume Labels

The standard volume label is written by an IBM-supplied Utility program (Initialize Disk) at the time a disk pack is prepared for use. The Initialize Disk program is described in the SRL publication IBM System/360 Model 20, Disk Programming System, Disk Utility Programs, Form C26-3810.

Standard IBM Disk File Labels

A standard file label or a set of standard file labels (1) identifies a particular logical file, (2) gives its location(s) on the disk pack, and (3) contains information to prevent the premature destruction of the file.

The number and format of labels required for a file depend on the file organization and the number of separate areas of the pack (extents) used by the file.

Volume Table of Contents (VTOC)

All standard file labels are grouped together and stored in a specific area of the pack. Because each file label contains file limits, the group of labels on a pack is essentially a directory to all data records on the pack (or volume). Therefore, it is called the Volume Table of Contents (VTOC). The VTOC itself is a file of records (one or more standard label records per logical file in the volume) and is defined by a file label. The label of the VTOC is the first record in the VTOC. This label identifies the file as the VTOC and gives the file limits of the VTOC.

The location and length of the VTOC are determined by control statements submitted to the INTDSK Utility program when the disk pack was initialized.

The VTOC may be placed anywhere on the disk pack, with the following restrictions:

1. It cannot be located within the alternate track area.
2. If it is on the pack used for system residence, it must be outside the residence area.
3. It must occupy at least one full track.
4. It may be only ten tracks in length but is not restricted to cylinder boundaries.

The Initialize Disk program prepares the VTOC by (1) writing the volume label with a pointer to the VTOC, (2) writing the VTOC file label at the beginning of the VTOC area and (3) writing an EOF sector behind the VTOC file label.

Standard File Label Formats

All standard disk file labels used with the IBM 2311 are 135 bytes long. Normally, two file labels are written into one disk sector. However, a Format 1 label must always start at the beginning of a sector. This may cause some sectors to contain only one label.

More than one file label may be required to describe a file. If this is the case,

all file labels for this file form an integral area within the VTOC.

There are four different formats for standard disk file labels.

Format 1. This format is used for all logical files. It is always the first of a series of labels when a disk file requires more than one label.

The Format 1 label identifies the logical file (by a file identification assigned by the user and included in the label) and contains file and data-record specifications. It also provides the addresses of three separate disk areas (extents) for the file. If the file is contained in more than three separate areas on the pack, a Format 3 label must immediately follow the Format 1 or Format 2 label.

If a logical file is recorded on more than one disk pack, the Format 1 label must be the first label for the file in the VTOC of each pack.

The Format 1 label is described in Appendix E.

Format 2. This format is required for any file that is organized according to the Indexed Sequential File Management System. The remaining area contains specifications unique to this type of file organization.

If an indexed sequential file is recorded on two or more packs, the Format 2 label is used on the first pack only. It is not repeated on the second pack (as the Format 1 label is).

The Format 2 label is described in Appendix F.

Format 3. If a logical file uses more than three extents on any one pack, this format is used to specify the addresses of the additional extents. The Format 3 label is used only for extent information. As many as 12 additional extents can be specified in one label. If a file uses more than 12 additional extents on a pack, more than one Format 3 label is required for that pack.

The Format 3 label follows the Format 1 label for the logical file, or a preceding Format 3 label or a Format 2 label. Format 3 labels are written on the pack on which the extents they define are located.

The Format 3 label is described in Appendix G.

Format 4. This format is used to define the VTOC itself. The Format 4 label is always the first label in the VTOC. In addition to defining the VTOC, the label is used to specify the location and number of available tracks in the alternate track area.

The Format 4 label is described in Appendix H.

Disk Label Processing

All disk label processing is performed by the label processing routines. These routines use the information supplied in the control statements (VOL, DLAB, and XTENT) that was stored by the Job Control program in the label information area of the disk pack mounted on SYSRES. Therefore, the execution of all programs processing disk files must be preceded by a Job Control run. VOL and DLAB statements must be supplied for each logical file, and an XTENT statement must be supplied for each extent occupied by the file.

The label processing routines for sequential files process the labels of an input or output file one pack at a time. When the end of the last extent on a pack is reached and the file is not yet completed, the next pack for the file is automatically opened.

The label processing routines for direct-access or indexed sequential files require that all packs for a file be on line for initial opening.

The following cases require special consideration.

Inquiry Programs. Inquiry programs, which are initiated by pressing the Request key on the printer-keyboard, do not begin with a Job Control run. Therefore all disk label information required by the inquiry program must be provided by an earlier Job Control run. Normally, the label information for inquiry programs consists of permanent labels. Should the label information consist only of temporary labels, it must be provided in the last Job Control run since any intervening Job Control run would overwrite them (see Permanent and Temporary Disk Label Information).

Multi-Volume Files -- Two-Drive System. When processing multi-volume files in a system with two disk drives, the pack mounted on SYSRES must remain on line throughout processing, while the volumes containing the file (or the remainder of the file if the first portion is stored on the pack mounted on SYSRES) should be mounted successively on the other disk drive.

Multi-Volume Files -- Single-Drive System. The processing of multi-volume files on a

single-drive system requires additional preparation. Since the label information must be on line throughout processing, it must be written on each pack used for the file before processing can begin. This is done by means of separate card-resident Job Control runs for each pack, during which SYSRES must be assigned to the disk drive. Multi-volume files thus prepared can be processed under control of the card-resident system. If the disk-resident system is to be used, the file must either start on the system disk pack, or the problem program must begin with a programmed halt to permit the operator to remove the system disk pack and mount the first pack of the file.

Single-Volume Files -- Single-Drive System. Single-volume files can be processed in a system with only one disk drive without a previous separate Job Control run if either the card resident control system is used and the file is mounted on SYSRES, or the disk-resident control system is used and the file is stored on the system disk pack. If the disk resident control system is used and the file is stored on a pack other than the system disk pack, the label information must be written on the pack containing the file by means of a separate card-resident Job Control run, and the problem program must begin with a programmed halt to permit the operator to remove the system disk pack and mount the pack containing the file.

Once the label information has been written into the VTOC, it remains valid for an indefinite number of program executions.

Label processing consists of the checks described below.

Disk Input Files

- The volume serial number in the volume label is compared to the volume serial numbers in the XTENT statements.
- Fields 1-3 of the Format 1 label are compared to the corresponding fields in the DLAB statement. Fields 4-6 are then compared with their EBCDIC equivalents in the DLAB continuation statement.
- The extent definitions in the Format 1 and Format 3 labels are compared with the corresponding limit fields in the XTENT statements.
- In an inquiry program, the second half of the file type field generated by the Assembler in the DTF block is compared with Field 10 of the format-1 label to determine whether the file is protected.

Disk Output Files

- The volume serial number in the volume label is compared with the volume serial numbers in the XTENT statements.
- All extent definitions in the labels contained in the VTOC are checked to determine whether there is any overlap with the extents defined in the XTENT statements. If an overlap exists, the expiration date of the label concerned is checked against the date in the communication region. If the expiration date has passed, the VTOC is compressed, overwriting the entire set of labels for the file concerned. If the expiration date has not passed, a programmed halt occurs.
- In an inquiry program, the second half of the file type field generated by the Assembler in the DTF block is compared with Field 10 of the format-1 label to determine whether the file is protected.
- The labels of the output file are written in the VTOC behind the labels already present.

Appendix B. Tape Labeling Conventions

A tape file processed by IBM-supplied programs must conform to certain standards regarding labels and the placement of tape marks.

Tape files with standard labels, with non-standard labels, or without labels can be processed. If a reel of tape contains more than one file (multi-file reel), all labels for these files must be of the same type (standard, nonstandard, or none).

Standard IBM Tape Labels

Two basic label types are provided to identify and protect tape files: volume labels and file labels. Each of the standard volume and file labels is 80 characters long. A volume label identifies a reel of magnetic tape, which may contain one file, more than one file, or part of a file. Each tape file, in turn, is identified and protected by at least two file labels: a header file label and a trailer file label.

The standard set of tape labels consists of:

- one standard IBM volume label per reel
- up to seven additional volume labels per reel
- two standard IBM tape file labels (one header label and one trailer label) for each file on the reel
- up to seven additional header labels and up to seven additional trailer labels for each file on the reel
- up to eight user header labels and up to eight user trailer labels for each file on the reel.

If standard labels are specified for a file, a standard IBM volume label, a standard IBM header label, and a standard IBM trailer label must be provided. Additional volume labels as well as additional and user file labels are optional.

STANDARD IBM VOLUME LABEL

If standard labeling is used, the standard IBM volume label is always the first record on the reel. It is fixed in length and format. The label identifier (character positions 1-4) is VOL1.

The standard IBM volume label contains a volume serial number. This number is assigned to the reel when it is prepared for use. The number is never changed. It is repeated in the standard IBM file labels for all files on the reel.

The format of the standard IBM volume label is given in Appendix C.

Standard IBM volume labels are checked by IBM-supplied programs (e.g., IOCS, Utility programs, Sort/Merge).

ADDITIONAL VOLUME LABELS

The standard IBM volume label can be followed by up to seven additional volume labels. These labels are fixed in length (80 characters). The character positions 1-4 must contain one of the identifiers VOL2-VOL8, according to the relative position of the label within the group of volume labels. The remaining 76 character positions can contain whatever information the user requires.

Additional volume labels are bypassed by IBM-supplied programs.

CREATION OF VOLUME LABELS

All standard volume labels (the IBM label and any additional labels) are written by an IBM-supplied Utility program (Initialize Tape) when a reel is prepared for use. The Initialize Tape program is described in the SRL publication IBM System/360 Model 20, Disk and Tape Programming Systems, Tape Utility Programs, Form C26-3808.

STANDARD IBM TAPE FILE LABEL

If standard labels are specified for a file, the file must be preceded by a standard IBM header label and followed by a standard IBM trailer label. These labels are fixed in length and format.

The label identifier (character positions 1-4) is:

- HDR1 for a header label (preceding the data file),
- EOF1 for an end-of-file trailer label (following a data file),

- EOV1 for an end-of-volume trailer label (at the end of a tape reel to indicate that file is continued on another reel).

The format of the standard IBM tape file label is shown in Appendix D.

Standard IBM tape file labels are processed by IBM-supplied programs.

ADDITIONAL TAPE FILE LABELS

Each standard IBM tape file label may be followed by up to seven additional tape file labels. These labels are fixed in length (80 characters). The character positions 1-3 must contain a label identifier equal to that in the preceding IBM file label (HDR, EOF, or EOV). Character position 4 must contain a number from 2 to 8 to indicate the relative position of the label within the group of file labels. The remaining 76 character positions can contain whatever information you require.

Additional tape file labels are not read, processed or written by the IBM-supplied Model 20 programs. They are included only for reasons of compatibility with the programming systems for other System/360 models.

USER TAPE FILE LABELS

You may include 1 to 8 user header labels and 1 to 8 user trailer labels to further define your file. These labels have a fixed length of 80 characters. The character positions 1-4 must contain the label identifiers UHL1 to UHL8 for header labels, and UTL1 to UTL8 for trailer labels. The remaining 76 positions of each label may contain any information that you require.

User header and trailer labels are read and written, but not processed by the IBM-supplied Model 20 programs.

Tape Organization with Standard Tape Labels

Figure 41 illustrates the tape organization for files that use the standard label set. The sequence of items on the tape is:

1. Standard IBM volume label (required)
2. Additional volume labels (up to seven, optional)
3. Header label set:
   Standard IBM file header label (required)
   Additional file header labels (up to seven, optional)
   User header labels (up to eight, optional).
4. Tapemark between header label set and first data record.
5. Physical records of the file.
6. Tapemark between last data record and trailer label set.

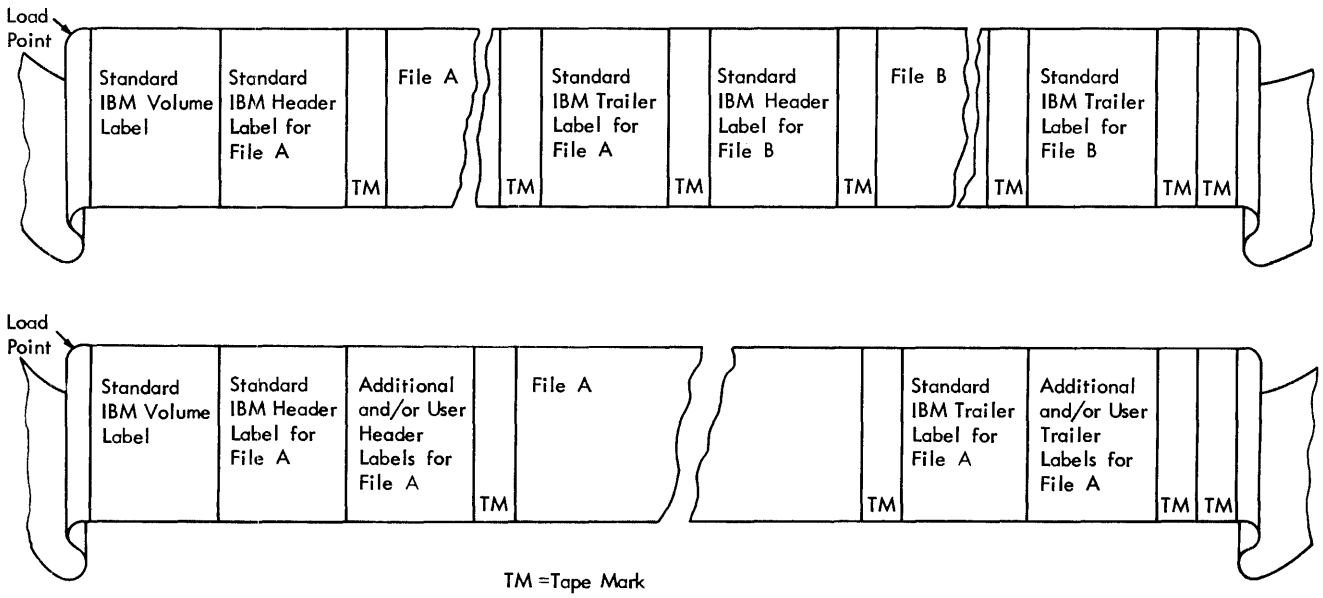


Figure 41. Tape Organization with Standard Labels

7. Trailer label set:
   Standard IBM file trailer label (required at end of file and end of volume)
   Additional file trailer labels (up to seven, optional)
   User trailer labels (up to eight, optional).

8. Tapemark after trailer label set.

9. If the file is on a multi-file reel but is not the last file on the reel (EOF label) the next standard IBM file header label is written in this position. If the file is on a single-file reel (EOF label) or is the last file on a multi-file reel, another tapemark is written in this position. If the file is a multi-reel file (EOV label) a tapemark is written in this position.

Standard IBM Tape Label Processing

Standard IBM Tape Label Processing is performed by the label-processing routines of the Model 20 Tape and Disk Programming Systems. These routines use the information supplied in the VOL and TPLAB statements that was stored by the Job Control program at the end of main storage. Only one VOL and one TPLAB statement need be supplied for each logical file, regardless of the number of reels on which the file is recorded.

While a program using the IOCS is being loaded, the tape label data is moved to the label-processing routines, making the upper area of main storage available to the user. (The upper area of main storage may be overwritten when an inquiry program is initiated, therefore tape labels cannot be processed by an inquiry program.)

Label processing routines normally consist of two main parts: one to read, check, and write header labels, the other to read, check, and write trailer labels.

The operations performed by the label-processing routines are described below.

Tape Input Files

- If the reel is positioned at the load point, the standard IBM volume label is read from the first or only reel used for the file, and the volume serial number in this label is compared to the file serial number in the TPLAB statement. If the numbers are not identical, the program halts and displays an error code. In case of a multi-reel file, the volume labels of all further reels used for the file are bypassed. If the reel is not positioned at the load point, the volume label is not checked.

- The standard IBM file header label is read from the first reel, and the contents of the TPLAB statement are compared to the corresponding fields in that label. If a multi-reel file is being processed, the standard IBM file header labels on the subsequent reels are also read and compared to the contents of the TPLAB statement after the preceding reel has been processed. The volume sequence number read from the TPLAB statement is increased by one for each additional reel.

- If a reel of tape contains more than one file (multi-file reel), the label processing routines use the file sequence number to position the file correctly. The file sequence numbers in the standard IBM file header labels are checked against the file sequence number in the TPLAB statement, and the corresponding files are bypassed until a match is found or the end of the tape is reached. If the tape is positioned beyond the desired file when the search is started, the program halts and displays an error code.

- If user header labels are specified, they are read into main storage and thus made available for processing by the user's label routines. To provide the necessary linkage, an exit address must be supplied by the user. User header labels are read one at a time, until all have been processed. If no exit address is specified the labels are bypassed.

- When a standard IBM file trailer label is read, the block count in this label is compared to a count accumulated by the IOCS during program execution.

- If user trailer labels are specified, they are treated in the same way as user header labels.

Note: If an input tape contains standard labels but the user does not want these labels to be checked, the entry FILABL=NSTD should be used in the corresponding file definition statement. This causes the standard label set to be bypassed by the label processing routines.

Tape Output Files

- If the reel is positioned at the load point, the standard IBM volume label is read from the first or only reel used for the file, and the volume serial

number in this label is compared to the file serial number in the TPLAB statement. If the numbers are not identical, the program halts and displays an error code. The volume labels of all further reels used for the file are bypassed. If the reel is not positioned at the load point, the volume label is not checked.

- If a standard IBM file header label is present on the tape onto which the output file is to be written, this label is read, and its expiration date is compared to the date in the communication region. If the expiration date has passed, the reel is backspaced to write the new standard IBM file header label. If the expiration date has not yet passed, the program halts and displays an error code. This check is performed for each reel of a multi-reel output file. If no file label is present (tapemark after volume label) the tape is considered expired.

- The new standard IBM file header label is written with the information supplied in the TPLAB statement. For a multi-reel file, the volume sequence number is increased by one for each successive reel. The creation date of the output file is taken from the TPLAB statement before the label is written. If the output file is to be written on a multi-file reel and is not the first file on the reel (i.e., if the output tape is not initially positioned at load point), the label processing routines of the IOCS will search for an IBM trailer label by reading backward from the point at which the tape is positioned. When the label is found, its file serial number and volume sequence number are compared to the corresponding information in the TPLAB statement. If the numbers are not equal, the program halts. If the numbers are equal, the file sequence number of the trailer label, increased by one, replaces the file sequence number read from the TPLAB statement. The IBM header label is then written immediately after the tapemark that follows the trailer label(s). Note that the label processing routines do not check whether the header labels destroy a file that started after the trailer label. The user must position the tape correctly before opening the output file.

- If user header labels are specified, the user's label routine is entered to furnish the labels as each file is opened. As many as eight user header labels can be written.

- If an end-of-reel condition is sensed before completion of the file, a standard EOV trailer label is written with the information supplied in the TPLAB statement and the block count accumulated during processing.

- When the end of file is reached, a standard EOF label is written with the same information as indicated for the EOV label above.

- If user trailer labels are specified, the user's label routine is entered each time an EOV or EOF trailer label has been written. As many as eight user trailer labels can be written.

Note: Standard labels on a 7-track tape are written in the same density as the data on that tape (all information on a tape reel must be written in the same density). The standard labels are written with even parity in the translation mode.

Nonstandard Tape Labels

Tape labels not conforming to the standard label specifications are considered nonstandard. Nonstandard tape labels must be followed by a tape mark. The Model 20 Tape and Disk Programming Systems do not read, check, or write nonstandard labels. Nonstandard tape labels are bypassed and processing begins at the first record following the tape mark.

Unlabeled Tape Files

Unlabeled tape files must conform to certain rules if they are to be processed by the Model 20 Tape and Disk Programming Systems. The first record may be a tapemark, the last record must be a tapemark. The end of a volume must be indicated by two tapemarks. All other records are data records.

If the first record of an unlabeled input tape file is not a tapemark, the record is assumed to be a data record.

When an unlabeled output tape file is specified, the label processing routines assume that the mounted output tape is unlabeled. No label checking is performed, and any labels present on the output tape are destroyed. A tapemark is written as the first record on the output file unless the entry TPMARK=NO is used in the appropriate file definition statement.

Appendix C. Standard IBM Volume Label, Tape or Disk

The volume label format for tape or disk is as follows:

Field	Bytes	Name and Length	Description
1	0-2	label identifier 3 bytes, EBCDIC	must contain VOL to indicate that the label is a volume label.
2	3	volume label number 1 byte, EBCDIC	indicates the relative position (in this case 1) of a volume label within a group of volume labels.
3	4-9	volume serial number 6 bytes, EBCDIC	a unique identification code which is assigned to a volume when it enters an installation. This code may also appear on the external surface of the volume for visual identification. It is normally a numeric field 000001 to 999999, however, any or all of the 6 bytes may be alphabetic.
4	10	volume security 1 byte, EBCDIC	not supported by the Model 20 Disk Programming System.
5	11-20	data file directory 10 bytes discontinuous binary	for disk only. The first 5 bytes contain the starting address (cchhr) of the VTOC. The last 5 bytes are blank. For tape reels, this field is not used and should be recorded as blanks.
6	21-30	reserved - 10 bytes	reserved for manufacturers.
7	31-40	reserved - 10 bytes	reserved for American Standards Association.
8	41-50	owner name and address code 10 bytes	indicates a specific customer, installation and/or system to which the volume belongs. This field may be a standardized code, name, address, etc.
9	51	volume protection - 1 byte	hexadecimal 0F prevents volume from being accessed by inquiry program
10	52-53	address of last permanent label - 2 bytes	disk address (hr) of last permanent label in label information area.
11	54-55	number of permanent labels - 2 bytes	total number of permanent labels in the label information area (hexadecimal notation).
12	56-58	reserved - 23 bytes	reserved for future use.
13	79	length indicator - 1 byte	indicates whether the label information area (LIA) is one or two tracks in length.

Note: All reserved fields should contain blanks to facilitate their use in the future. Any information appearing in these fields at the present time will be ignored by the Model 20 Disk Programming System.

Appendix D. Standard IBM Tape File Label

The standard IBM tape file label format and contents are as follows:

Field	Bytes	Name and Length	Description
1	0-2	label identifier 3 bytes, EBCDIC	identifies the type of label HDR = Header -- beginning of a data file EOF = End of File -- end of a data file EOV = End of Volume -- end of the physical reel
2	3	file label number 1 byte, EBCDIC	indicates the relative position (in this case 1) of a file label within a group of file labels.
3	4-20	file identifier 17 bytes, EBCDIC	uniquely identifies the entire file; may contain only printable characters.
4	21-26	file serial number 6 bytes, EBCDIC	uniquely identifies a file/volume relationship. This field is identical to the Volume Serial Number in the volume label of the first or only volume of a multi-volume file or a multi-file set. This field will normally be numeric (000001 to 999999) but may contain any six alphameric characters.
5	27-30	volume sequence number 4 bytes, EBCDIC	indicates the order of a volume in a given file or multi-file set. The first must be numbered 0001 and subsequent numbers must be in proper numeric sequence.
6	31-34	file sequence number 4 bytes, EBCDIC	assigns numeric sequence to a file within a multi-file set. The first must be numbered 0001.
7	35-38	generation number 4 bytes, EBCDIC	uniquely identifies the various editions of the file. May be from 0001 to 9999 in proper numeric sequence.
8	39-40	version number of generation 2 bytes, EBCDIC	indicates the version of a generation of a file.
9	41-46	creation date 6 bytes, EBCDIC	indicates the year and the day of the year that the file was created; Position / Code / Meaning 1 / blank / none 2-3 / 00-99 / year 4-6 / 001-366 / day of year (e.g., January 31, 1969 would be entered as 69031)
10	47-52	expiration date 6 bytes, EBCDIC	indicates the year and the day of the year when the file may become a scratch tape. The format of this field is identical to that of Field 9. On a multi-file reel, processed sequentially, all files are considered to expire on the same day.
11	53	file security 1 byte, EBCDIC	not supported by the Model 20 Disk Programming System.

12	54-59	block count 6 bytes, EBCDIC	indicates the number of data blocks written on the file from the last header label to the first trailer label exclusive of tape marks. Count does not include checkpoint records. This field is used in Trailer Labels.
13	60-72	system code 13 bytes	not supported by the Model 20 Disk Programming System.
14	73-79	reserved - 7 bytes	reserved for American Standards Association.

Appendix E. Standard Disk File Label, Format 1

The Format 1 disk file label is common to all data files on disk.

Field	Bytes	Name and Length	Description
1	0-43*	file identifier 44 bytes, EBCDIC	this field serves as identifier of the file. Each file must have a unique file identifier. Duplication of identification will cause retrieval errors. The Model 20 Disk Programming System compares the entire file identifier field against the identification given in the DLAB statement. The file identifier for the system disk pack (bytes 0-21 contain 'SYSTEM 360 MOD20 DPS and bytes 22-43 contain blanks) must never be used for other files.

The following fields (2-33) comprise the DATA portion of the file label:

Field	Bytes	Name and Length	Description
2	44*	format identifier 1 byte, EBCDIC	1 = Format 1
3	45-50	file serial number 6 bytes, EBCDIC	uniquely identifies a file/volume relationship. It is identical to the Volume Serial Number of the first or only volume of a file.
4	51-52	volume sequence number 2 bytes, binary	indicates the order of a volume relative to the first volume on which the data file resides.
5	53-55	creation date 3 bytes discontinuous binary	indicates the year and the day of the year the file was created. It is of the form ydd, where y signifies the year (0-99) and dd the day of the year (1-366).
6	56-58	expiration date 3 bytes discontinuous binary	indicates the year and the day of the year the file may be deleted. The form of this field is identical to that of Field 5.
7A	59	extent count 1 byte, binary	contains a count of the number of extents for this file on this volume.
7B	60	bytes used in last block of directory 1 byte, binary	not supported by the Model 20 Disk Programming System.
7C	61	reserved - 1 byte	reserved for future use.
8	62-74	system code 13 bytes	uniquely identifies the programming system. The character codes that can be used in this field are limited to 0-9, A-Z, or blanks. This field is optional for Model 20 DPS.
9	75-81	reserved - 7 bytes	this field is reserved for future use.
10	82-83	file type 2 bytes	the contents of this field uniquely identify the type of data file: Hex 4000 = Sequential organization Hex 2000 = Direct-access organization Hex 8000 = Indexed-sequential organization The second and fourth half-bytes of this field contain codes used for file protection.

11**	84	record format 1 byte	the contents of this field indicates the type of records contained in the file: Bit Position Contents Meaning 0 and 1 01 variable length records 10 fixed length records 11 undefined format 2 0 no track overflow 1 file is organized using track overflow (Operating System/360 only) 3 0 unblocked records 1 blocked records 4 0 no truncated records 1 truncated records in file 5 and 6 01 control character ASA code 10 control character machine code 00 control character not stated 7 0 records have no keys 1 records are written with keys.
12	85	option codes 1 byte	bits within this field are used to indicate various options used in building the file. Bit 0 If on, indicates data file was created using Write Validity Check. 1-7 unused.
13	86-87	block length 2 bytes, binary	indicates the block length for fixed-length records.
14	88-89	record length 2 bytes, binary	indicates the record length for fixed-length records.
15	90	key length 1 byte, binary	indicates the length of the key portion of the data records in the file.
16	91-92	key location 2 bytes, binary	indicates the high-order position of the key portion.
17	93	data set indicators 1 byte	not supported by the Model 20 Disk Programming System.
18	94-97	secondary allocation 4 bytes, binary	not supported by the Model 20 Disk Programming System.
19	98-102	last record pointer 5 bytes discontinuous binary	not supported by the Model 20 Disk Programming System.
20	103-104	reserved - 2 bytes	reserved for future use.
21	105	extent type indicator 1 byte	indicates the type of extent with which the following fields are associated: Hex Code 00 next three fields do not indicate any extent. 01 prime area (indexed sequential) or consecutive area, etc. (i.e., the extent containing the user's data records.) 02 overflow area of an indexed sequential file 04 cylinder index or master index area of an indexed sequential file

22	106	extent sequence number 1 byte, binary	indicates the extent sequence in a multi-extent file.
23	107-110	lower limit 4 bytes discontinuous binary	the cylinder and the track address specifying the starting point (lower limit) of this extent component. This field has the format cchh.
24	111-114	upper limit 4 bytes discontinuous binary	the cylinder and the track address specifying the ending point (upper limit) of this extent component. This field has the format cchh.
25-28	115-124	additional extent 10 bytes	these fields have the same format as the fields 21-24 above.
29-32	125-134	additional extent 10 bytes	these fields have the same format as fields 21-24 above.

The end of the active VTOC is indicated by a label that contains / and blank in the first three bytes and a binary zero in byte 44.

**These fields are not supported by the Model 20 Disk Programming System.

Appendix F. Standard Disk File Label, Format 2

The Format 2 disk file label is used only with indexed sequential data files. It is preceded either by a Format 1 label or by a Format 3 label.

Field	Bytes	Name and Length	Description
K1	0	key identification 1 byte	this byte contains the Hex Code 02 in order to avoid conflict with file name.
K2*	1-7	address of 2nd level master index 7 bytes discontinuous binary	this field contains the address of the first track of the second level of the master index, in the form mbbcchh.
K3*	8-12	last 2nd level master index entry address 5 bytes discontinuous binary	this field contains the address of the last index entry in the second level of the master index, in the form cchhr.
K4*	13-19	address of 3rd level master index 7 bytes discontinuous binary	this field contains the address of the first track of the third level of the master index, in the form mbbcchh.
K5*	20-24	last 3rd level master index entry address 5 bytes discontinuous binary	this field contains the address of the last entry in the third level of the master index, in the form cchhr.
K6	25-43	reserved - 19 bytes	reserved for future use.
D1	44	format identifier 1 byte, EBCDIC	2 = Format 2
D2	45	number of index levels 1 byte, binary	the contents of this field indicate how many levels of index are present with an indexed sequential file.
D3*	46	high level index development indicator 1 byte, binary	this field contains the number of tracks determining development of master index.
D4	47-49	first data record in cylinder 3 bytes discontinuous binary	this field contains the address of the first data record on each cylinder in the form hhr.
D5	50-51	last data track in cylinder 2 bytes, binary	this field contains the address of the last data track on each cylinder, in the form hh.
D6	52	number of tracks for cylinder overflow 1 byte binary	this field contains the number of tracks in cylinder overflow area.
D7*	53	highest "r" on high-level index track 1 byte, binary	this field contains the highest possible r on track containing high-level index entries.
D8	54	highest "r" on prime track 1 byte, binary	this field contains the highest possible r on prime data tracks for form F records.

D9	55	highest "r" on overflow track 1 byte, binary	this field contains the highest possible r on overflow data tracks for form F records.
D10*	56	"r" of last data record on shared track 1 byte, binary	this field contains the r of the last data record on a shared track.
D11	57-58	spare - 2 bytes	reserved for future use.
D12	59-60	tag deletion count 2 bytes, binary	this field contains the number of records that have been tagged for deletion.
D13*	61-63	non-first overflow reference count (RORG3) 3 bytes, packed decimal	this field contains a count of the number of random references to a non-first overflow record.
D14*	64-65	number of bytes for highest level index 2 bytes, binary	the contents of this field indicate how many bytes are needed to hold the highest-level index in main storage.
D15*	66	number of tracks for highest-level index 1 byte, packed decimal	this field contains a count of the number of tracks occupied by the highest-level index.
D16	67-70	prime record count 4 bytes, binary	this field contains a count of the number of records in the prime data area.
D17*	71	status indicator 1 byte	the eight bits of this byte are used for the following indications: bit — description 0 — last block full 1 — last track full 2-7 — must remain off
D18	72-78	address of cylinder index 7 bytes discontinuous binary	this field contains the address of the first track of the cylinder index, in the form mbbcchh.
D19*	79-85	address of lowest-level master index 7 bytes discontinuous binary	this field contains the address of the first track of the lowest-level index of the high level indexes, in the form mbbcchh.
D20*	86-92	address of highest-level index 7 bytes discontinuous binary	this field contains the address of the first track of the highest-level master index, in the form mbbcchh.
D21	93-100	last prime data record address 8 bytes discontinuous binary	this field contains the address of the last data record in the prime data area, in the form mbbcchhr.
D22	101-105	Last track index entry address 5 bytes discontinuous binary	this field contains the address of the last normal entry in the track index on the last cylinder in the form cchhr.
D23	106-110	last cylinder index entry address 5 bytes discontinuous binary	this field contains the address of the last index entry in the cylinder index in the form cchhr.

D24*	111-115	last master index entry address 5 bytes discontinuous binary	this field contains the address of the last index entry in the master index in the form cchhr.
D25	116-123	last independent overflow record address 8 bytes discontinuous binary	this field contains the address of the last record written in the current independent overflow area, in the form mbbcchhr.
D26*	124-125	bytes remaining on overflow track 2 bytes, binary	this field contains the number of bytes remaining on current independent overflow track.
D27	126-127	number of independent overflow tracks (RORG2) 2 bytes, binary	this field contains the number of tracks remaining in independent overflow area.
D28	128-129	overflow record count 2 bytes, binary	this field contains a count of the number of records in the overflow area.
D29*	130-131	cylinder overflow area count (RORG1) 2 bytes, binary	this field contains the number of full cylinder overflow areas.
D30	132-134	reserved - 3 bytes	this field is reserved for future use.

* These fields are not supported by the Model 20 Disk Programming System.

Appendix G. Standard Disk File Label, Format 3

The Format 3 disk file label is used to describe extra extent segments on the volume if these cannot be described in the Format 1 (and Format 2 if it exists) file label. This file label is preceded by a Format 1 or another Format 3 file label.

Field	Bytes	Name and Length	Description
1	0-3	key identification 4 bytes	each byte of this field contains the Hex Code 03 in order to avoid conflict with a data file name.
2-17	4-43	extents 40 bytes	four groups of fields identical in format to fields 21-24 in the Format 1 label.
18	44	format identifier 1 byte, EBCDIC	3 = Format 3
19-51	15-124	additional extents 80 bytes	eight groups of fields identical in format to fields 21-24 in the Format 1 label.
52	125-134	reserved 10 bytes	reserved for future use.

Appendix H. Standard Disk File Label, Format 4.

The Format 4 disk file label is used to describe the Volume Table of Contents and is always the first file label in the VTOC. There must be one and only one of these Format 4 file labels per volume.

Field	Bytes	Name and Length	Description
1	0-43	key field 44 bytes	each byte of this field contains the Hex Code 04 in order to provide a unique key.
2	44	format identifier 1 byte, EBCDIC	4 = Format 4
3	45-49	last active format 1 5 bytes discontinuous binary	Not supported by the Model 20 Disk Programming System.
4	50-51	available file label records 2 bytes, binary	Not supported by the Model 20 Disk Programming System.
5	52-55	highest alternate track 4 bytes discontinuous binary	contains the address (in the form cchh) of the next available track of a block of tracks set aside as alternates for bad tracks.
6	56-57	number of alternate tracks 2 bytes, binary	contains the number of alternate tracks available.
7	58	VTOC indicators 1 byte	Not supported by the Model 20 Disk Programming System.
8A	59	extent count 1 byte	contains a count of the number of extents in the Format 4 label.
8B	60-61	reserved - 2 bytes	reserved for future use.
9A	62-63	last cylinder initialized 2 bytes	contains the binary equivalent of the decimal values 102 or 202, as appropriate.
9B	64-70	reserved 5 bytes	reserved for future use.
9C	71	ERASE option 1 byte	Bit 4, if on, indicates ERASE option used; when not on indicates ERASE option not used.
9D	72-104	reserved 35 bytes	reserved for future use.
10-13	105-114	VTOC extent 10 bytes	these fields describe the extent of the VTOC, and are identical in format to fields 21-24 of the Format 1 file label. Extent type is 01 (prime data area).
14	115-124	LIA extent 10 bytes	these fields describe the extent of the label information area (LIA), and are identical in format with fields 21-24 of the Format 1 file label. Extent type is 01.
15	125-134	reserved - 10 bytes	to be left blank.

Appendix I. Model 20 DPS Program and Phase Names

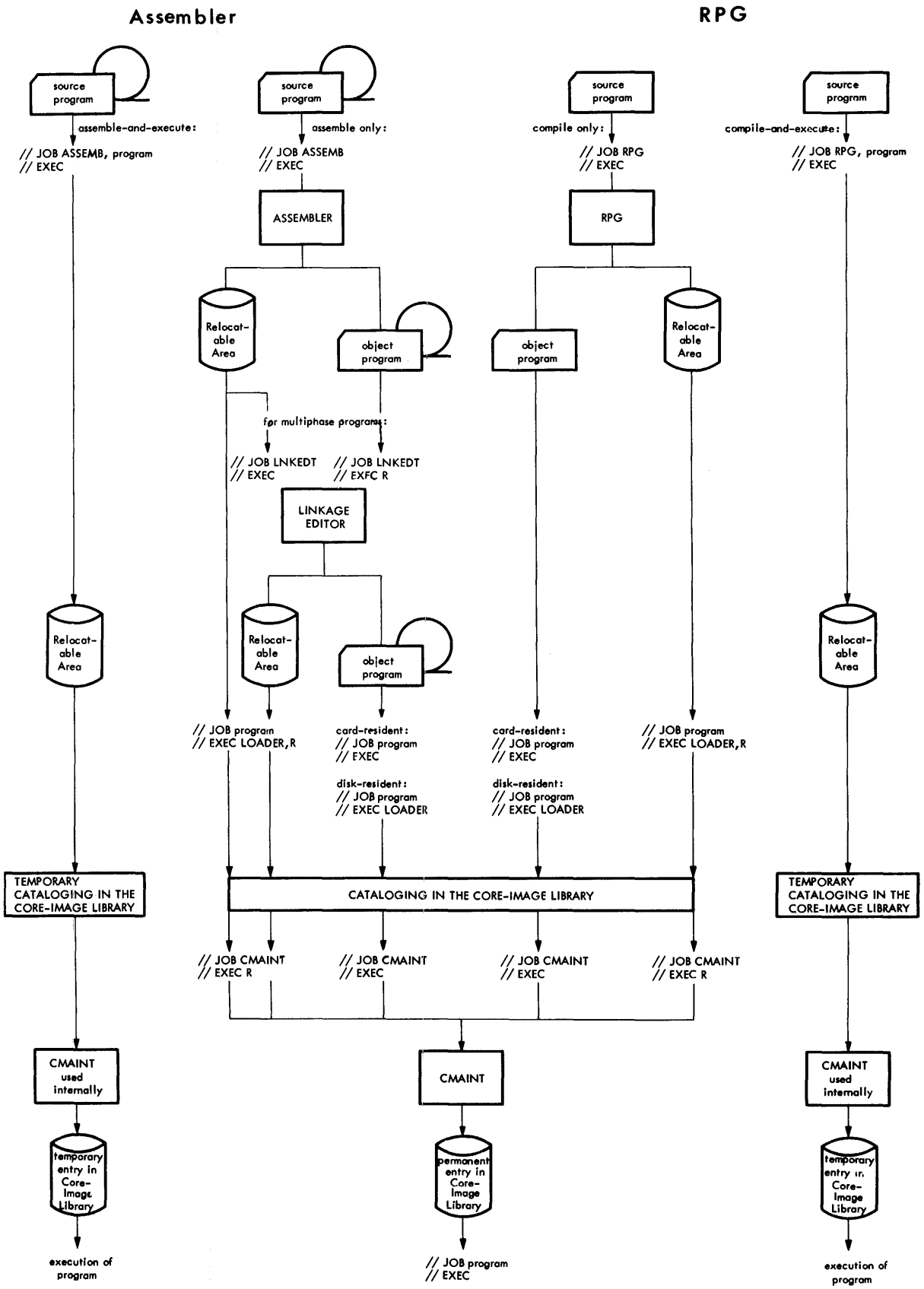
The table in Figure 42 lists the phase names of the Model 20 DPS programs. The names given must not be used as phase names for user programs.

PHASE NAME
$$$$$A, $$$CMA
AORGZ, AORGZ1, ASSEMB, ASSEMC, ASSEMD, ASSEME, ASSEMF, ASSEMG, ASSEMH, ASSEMI, ASSEMJ, ASSEMK, ASSEML, ASSEMM, ASSEMN, ASSEMO, ASSEMP, ASSEMQ, ASSEMR, ASSEMS, ASSEMT, ASSEMU, ASSEMV, ASSEMW, ATASGN
BACKUP, BACKUR, BACKUS, BACKUT, BACKUU, BACKU1, BACKU2, BACKU3, BACKU4, BACKU5, BACKU6
CARDSK, CARD01, CARD02, CARD03, CARTAP, CART01, CART02, CART03, CART04, CART05, CLRDSK, CMAINT, CMAIN1, CMAIN2, CMAIN3, CMAIN4, COPSYS, CSERV, CSERV1, CSERV2, CSERV3, CSERV4
DDUMP, DSERV, DSKCAR, DSKC01, DSKC02, DSKDSK, DSKD01, DSKD02, DSKD03, DSKPRT, DSKP01, DSKP02, DSKTAP, DSKT01, DSKT02, DSKT03
IAE, INITTP, INTDSK
LDSYS, LNKEDT, LNKED2, LNKED3, LNKED4, LNKED5
MMAINA, MMAINB, MMAINC, MMAINT, MMAIN1, MMAIN2, MMAIN3, MMAIN4, MMAIN5, MMAIN6, MMAIN7, MMAIN8, MMAIN9, MSERV, MSERV1, MSERV2, MSERV3, MSERV4, MSERV5
PL1, PSERV, PUNCH
RESTOR, RPG, RPG#AD, RPG#AE, RPG#AF, RPG#AG, RPG#AK, RPG#AM, RPG#AZ, RPG#BA, RPG#BD, RPG#BG, RPG#BK, RPG#CA, RPG#CC, RPG#CD, RPG#CE, RPG#CF, RPG#CI, RPG#CM, RPG#CN, RPG#CP, RPG#CR, RPG#CS, RPG#CT, RPG#CU, RPG#CX, RPG#DB, RPG#DC, RPG#DE, RPG#DG, RPG#DI, RPG#DS, RPG#EB, RPG#EE, RPG#EH, RPG#EL, RPG#EP, RPG#ES, RPG#EW, RPG#EY, RPG#FB, RPG#FE, RPG#FH, RPG#FL, RPG#FP, RPG#FS, RPG#FW, RPG#FY, RPG#GB, RPG#GE, RPG#GR, RPG#HF, RPG#HG, RPG#HP, RPG#HR, RPG#IB, RPG#IC, RPG#IF, RPG#IG, RPG#IH, RPG#II, RPG#IK, RPG#IL, RPG#IN, RPG#IO, RPG#IP, RPG#IR, RPG#IU, RPG#IW, RPG#KF, RPG#KK, RPG#KP, RPG#KU, RPG#LB, RPG#LF, RPG#LK, RPG#LP, RPG#LU, RPG#ME, RPG#MI, RPG#MO, RPG#NA, RPG#WB, RPG#WC, RPG#WD, RPG#WE, RPG#WF, RPG#WG, RPG#WH, RPG#WI, RPG#WK, RPG#WL, RPG#WM, RPG#WN, RPG#ZA, RPG#ZB
SORT, SORT02, SORT04, SORT06, SORT08, SORT10, SORT12, SORT14, SORT16, SORT18, SORT20, SORT22, SORT24, SORT26, SORT28, SORT30, SORT32, SORT34, SORT36, SORT38, SORT40, SORT42, SORT44, SYSEND, SYSEOJ
TAPCAR, TAPC01, TAPC02, TAPC03, TAPC04, TAPC05, TAPDSK, TAPD01, TAPD02, TAPD03, TAPD04, TAPD05, TAPPRT, TAPP01, TAPP02, TAPP03, TAPP04, TAPP05, TAPP06, TAPSRT, TAPS01, TAPS02, TAPS03, TAPS04, TAPS05, TAPS06, TAPS07, TAPS08, TAPS09, TAPS10, TAPS11, TAPS12, TAPS13, TAPTAP, TAPT01, TAPT02, TAPT03, TAPT04, TAPT05

Figure 42. Model 20 DPS Program Names

Dotted line flags planning information

Appendix J. Methods of Using the Disk Programming System



Index

IBM System/360 Printed in U.S.A. C24-9006-4

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

IBM System/360 Model 20
Disk Programming System
Control and Service Programs

Form C24-9006-4

- How did you use this publication?

  As a reference source ☐
  As a classroom text ☐
  As a self-study text ☐

- Based on your own experience, rate this publication . . .

	Very Good	Good	Fair	Poor	Very Poor
As a reference source:					
As a text:					

- What is your occupation? .....

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS, PLEASE . . .

This SRL 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.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold

Fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation

112 East Post Road

White Plains, N. Y. 10601

Attention: Department 813 BP

Fold

Fold

CUT ALONG THIS LINE

IBM System/360 Printed in U.S.A. C24-9006-4

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)