File Number S360-20
Form C24-3372-6

BOS

Systems Reference Library

IBM System/360
Basic Operating System
Programmer's Guide

This reference publication describes the IBM System/360 Basic Operating System. The system is a set of control programs and processing programs provided for smaller configurations of the IBM System/360. Utilizing IBM 2311 Disk Storage for on-line program residence, IBM System/360 Basic Operating System provides stacked-job processing capability, controls all input/output, and provides for continuous operation of all programs run in its environment.

This Programmer's Guide includes descriptions of the control programs, service programs, and system facilities supported by IBM. A comprehensive introduction gives an over-all picture of the entire system. Detailed information is given on these major topics:

1. Operation with the System Control Programs

2. Using the System Service Programs

3. Data Management

The prerequisite for a thorough understanding of this manual is a basic knowledge of System/360 machine concepts and instructions.

For titles and abstracts of other associated publications see the IBM System/360 Bibliography, Form A22-6822.

Seventh Edition, September 1967

This edition, Form C24-3372-6, is a major revision of, and obsoletes, Form C24-3372-5, and Technical Newsletters N24-5262 and N24-5316. The major changes in this publication include updated machine requirements and support of an intermediate storage size (24K) for System/360 Model 30.

Changes are indicated by a vertical line to the left of affected text and to the left of affected parts of figures. A dot (•) next to a figure title or page number indicates that the entire figure or page should be reviewed.

Specifications contained herein are subject to change from time to time. Any such change will be reported in subsequent revisions or Technical Newsletters.

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

A form is provided at the back of this publication for readers' comments. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, Endicott, New York 13760.

Part 1 of this publication, under the heading Introduction, gives an over-all discussion of the IBM System/360 Basic Operating System. It describes the various components of the system and shows how their functions are related to the total system. The Introduction begins with a quick summary of the components of the system and the system configuration required. Persons with experience in operating-system concepts may find this part of the introduction sufficient for presenting a general discussion of the system. They may wish to skim the rest of the Introduction to get into the detail of the later sections.

Part 2, under the heading System Control Programs, describes the Supervisor and Job Control Programs. This section is of interest to anyone using the system, including system analysts, programmers, and machine operators. The functions provided by these programs are discussed and the detailed Job Control control card formats are given. Note, however, that the macro instructions used to communicate with the Supervisor are discussed fully in the publication Assembler with Input/Output Macros, Form C24-3361.

Part 3, System Service Programs, is of particular interest to the persons responsible for generating and maintaining the resident systems. This section describes the Librarian and Linkage Editor programs.

Part 4, Data Management, discusses the organization and maintenance of data files, processing methods, disk storage concepts, label processing capabilities, etc. As with the Supervisor macros, the file processing macros (Input/Output Control System--IOCS) are discussed fully in the Assembler with I/O Macros publication.

Note: Programming specifications for the 1287 Optical Reader, binary synchronous communication, and remote job entry may be used for planning purposes only.

This planning information is listed separately in the Index.

REFERENCE PUBLICATIONS

Publications which are related to this one are:

1. IBM System/360 Principles of Operation, Form A22-6821.

2. IBM System/360 Basic Programming Support, Basic Tape System Programmer's Guide, Form C24-3354.

3. IBM System/360, Basic Operating System, Assembler with Input/Output Macros Specifications, Form C24-3361.

4. IBM System/360 Basic Operating System and IBM System/360 Basic Programming Support: Macro Definition Language, Form C24-3364.

5. IBM System/360 Basic Operating System, System Generation and Maintenance, Form C24-5060.

6. IBM System/360 Operating System: Remote Job Entry, Form C30-2006.

7. IBM System/360 Component Descriptions, A26-5988.

Another publication that may be useful is the Glossary for Information Processing, C20-8089.

The IBM System/360 Basic Operating System is designed to provide operating system capabilities for 8K and larger System/360 configurations that include one or more IBM 2311 Disk Storage Drives. Systems above 8K that do not require the expanded functions provided in the larger operating system packages offered by IBM may benefit from this 8K package. The system is disk resident, using IBM 2311 disk storage for on-line storage of all programs. Depending on the requirements of the particular application, the system can be expanded to include all processing programs used to perform the various jobs of a particular installation or it can be cut down to a minimum system to control a single program.

The Basic Operating System (BOS) consists of the following components:

Control Programs

The control programs constitute the framework of the Basic Operating System. They prepare and control the execution of all other programs executed. The control programs are:

1. Supervisor This program handles all input/output operations, interrupt conditions, and other functions for all problem programs. Part of the Supervisor resides in main storage at all times. Processing time alternates between the Supervisor and the program being executed. This is true for the user's programs as well as the other IBM-supplied components of the system. Certain functions of the Supervisor are provided by routines that remain in disk storage until needed and which are then loaded into a special area of main storage for execution.

2. Job Control This program runs between jobs and prepares the system for execution of all other programs. Job Control is loaded by the Supervisor from disk storage when needed.

3. IPL (Initial Program Load) Loader This program loads the Supervisor into main storage when system operation is initiated. The IPL Loader is loaded from disk storage simply by selecting the address of the disk drive in the load-unit switches on the system console and pressing the load key.

System Service Programs

The system service programs provide the functions of generating the initial operating systems, regenerating specialized systems, creating and maintaining the library sections, and loading and editing programs into disk residence before execution. Once a system has been built with complete program-assembly capability, other minimum systems can be built that do not include the system service programs. Such minimum systems still require disk residence.

The system service programs are:

1. Linkage Editor All user programs must be read from cards or tape or from the relocatable library and edited into the core image library of the resident disk pack by this program. These programs can be permanently placed in the system, requiring only control cards to call them for execution, or they can be stored temporarily, executed, and then overlaid in the core image library by new programs.

2. Librarian This is actually a group of programs, used for maintaining and reorganizing the disk library areas and providing printed and punched output from the libraries. Three libraries are used:

   a. Core Image library required. All programs (IBM-supplied and user programs) are loaded from this library by the System Loader routine of the Supervisor.

   b. Macro library optional. This library is used to store IBM-supplied and user-defined macro routines in resident packs built to provide program-assembly capability.

   c. Relocatable library optional. This library is required for assemble-and-execute operations and for Autotest. It can be used to store assembled object modules for subsequent linkage with other program sections when editing programs into the core image library.

3. Load System Program Operating as an independent program (loaded from cards, with its own IPL program, Supervisor and Job Control program), the Load

System program builds a disk resident system from cards. This program can be used to build minimum systems for specialized applications. If two disk drives are available, the librarian can be used instead of the Load System program to build specialized systems.

Processing Programs

All user programs are run within the Pasic Operating System environment, making full use of the power of the control programs. A system may include all of the user's programs and the following IBM-supplied service programs:

1. Language Translators: Assembler and Report Program Generator (RPG)
2. Autotest
3. Disk Sort/Merge
4. Utilities
5. Remote Job Entry

A special job-tailored disk resident system may consist of:

The control programs (Supervisor and Job Control) and one or more user programs, or

The control programs and the Linkage Editor with an area for temporary storage of user programs.

SYSTEM CONFIGURATION

This section presents the minimum machine requirements for system generation and operation of the Basic Operating System, the features in addition to the minimum that can be supported, and the disk residence requirements. The system control programs must always be present to execute any other programs, and therefore, they establish the absolute minimum configuration possible. The minimum features always required are listed under the control programs, in the section Minimum Machine Requirements: System Operation. For the other IBM-supplied programs, the required features listed are in addition to the basic minimum established by the control programs. Other features are mentioned for special uses. A complete listing of the features supported by the Basic Operating System is provided.

Note: The IBM-supplied programs assume the availability of a problem program area of at least 4096 bytes. It is possible to generate a Supervisor that does not leave this much room in an 8K system. This can occur if a large number of code-producing options are specified in the Supervisor macro instructions. See Appendix H for a discussion of the assembled Supervisor sizes.

MINIMUM MACHINE REQUIREMENTS: SYSTEM GENERATION

The minimum machine configuration for generating an installation-tailored Basic Operating System is:

- 8K bytes of main storage.
- One 2311 Disk Storage Drive.
- One Card Reader.
- One Printer.
- One Card Punch.
- One 1052 Printer-Keyboard (optional, but advisable for efficient system operation).

Magnetic tape units can be substituted for the card reader and card punch.

MINIMUM MACHINE REQUIREMENTS: SYSTEM OPERATION

Control Programs

The minimum features always required are:

- 8K bytes of main storage.
- Standard Instruction Set.
- One I/O channel (either multiplexor or selector).
- One Reader: 1442, 2520, or 2540 Card Read-Punch or 2501 Card Reader.
- One 2311 Disk Storage Drive (see Residence Requirements).

Librarian

Additional features required for library service are:

- One 1442, 2520, or 2540 Card Read-Punch (if a card deck is desired). This may be the same unit used for input.
- One 1403, 1404, or 1443 Printer (if a listing is desired).

This feature is required for the CORGZ function in library organization:

- One additional 2311 Disk Storage Drive.

Linkage Editor

This device is supported, if desired, for input to the linkage editor:

- One unit other than the control card reader (an additional card reader or 2400-series Magnetic Tape Unit).

Assembler

To perform an assembly, the Assembler program requires a System/360 with the following features, in addition to the minimum established under Control Programs:

- One 1403, 1404 (continuous forms operation only), or 1443 Printer, if the program listing is to be printed.
- One 1442, 2520, or 2540 Card Read-Punch, if the Assembler output deck is to be punched. This device may be the same I/O unit used for reading the source deck.
- If the size of the Supervisor exceeds 4096 bytes of main storage, a 16K system is required.
- One 2400-series Magnetic Tape Unit is required under any of the following conditions:

  1. If the source program is to be read from tape.
  2. If the output deck from the Assembler is to be written on tape.
  3. If the program listing is to be written on tape.

  This tape unit may be used for source input or object program output only if sufficient main storage is available (see Appendix H for requirements).
- If both the output deck and the program listing are to be written on tape, a second tape unit is required. The tape unit may be a 7-track unit with the data conversion feature, or a 9-track unit.
- For a split work area, an additional 2311 Disk Storage Drive.

To execute object programs, these features must be added to the minimum configuration:

- I/O units, as required by the object program (see the devices listed under Features Supported).
- The Data Conversion Special Feature, if a 7-track tape prepared with the data conversion feature is used.
- For BSC applications, a minimum of 16K of main storage.
- If in STR (Synchronous Transmitter Receiver) mode, one 2701 Data Adapter, Type I, is required. Also, if in STR mode, 16K bytes of storage are required for most applications. However, utility type applications, requiring minimum processing and code conversion, are supported in an 8K environment.

Report Program Generator (RPG)

Additional features required to compile a source program:

- One 1403, 1404 (continuous forms operation only), or 1443 Printer, if the program listing is to be printed.
- Decimal Arithmetic Feature.
- For source program input only, one 2400-series Magnetic Tape Unit, 7-track (with the Data Conversion Special feature) or 9-track, is supported.

Additional features required for object program execution:

- Input/output units as required by the object program (see devices listed under Features Supported).
- Decimal Arithmetic Feature.
- For extra input or output, one 2311 Disk Storage Drive.

- For program generation, one tape unit is supported, if desired, for object program input. The Data Conversion Special feature is required if a 7-track unit is used.

Autotest

In addition to the machine configuration required for the control programs, Autotest requires:

- 16K bytes of main storage. This program operates as a part of the 8K disk resident Basic Operating System and tests problem programs designed to run in the 8K system. However, at least 16K bytes of main storage must be available for the Autotest function.
- One 1403, 1404, or 1443 Printer.
- An additional 2311 Disk Storage Drive, if desired for use as a work file.

Disk Sort/Merge

The system used to run the sort/merge program must have:

- One 1403, 1404, 1443 Printer or 1052 Printer-Keyboard.

Utilities

Additional features required for program operation are:

- The I/O units used by the particular utility program.
- For logging and error messages, one 1403, 1404, 1443 Printer or 1052 Printer-Keyboard.

Remote Job Entry

Additional features required are:

- A minimum of 16K bytes of main storage, if user-written routines are not included. If user-written routines are included, at least 24K bytes of main storage are required.
- 2701 Data Adapter Unit with a Synchronous Data Adapter - Type II, for EBCDIC code, connected over point-to-point leased or switched lines, and equipped with the transparency feature.
- One 1052 Printer-Keyboard.
- One 1442, 2520, or 2540 Card Punch for punched card output.
- One 1403, 1404, or 1443 Printer for printed output.
- With the 2701 Data Adapter Unit, the Autocell feature and Dual Communication Interface are available, as desired.

FEATURES SUPPORTED

- Interval Timer.
- Simultaneous Read-while-Write Tape.
- Any channel configuration up to one multiplexor channel and two selector channels.
- One 1052 Printer-Keyboard for operator communication. With the Assembler program, the 1052 may be used for output of special diagnostic messages.
- Additional main storage with the following restrictions:

  1. All control programs (Supervisor, including the transient routines, and Job Control) and all system service programs (Linkage Editor, Librarian, Load System) cannot be located beyond 32K.
  2. When the user's program is executed, the portion of the problem that communicates with the control programs must be located in the first 64K of main storage. This includes DTF routines, channel command words (CCWs), command control blocks (CCBs), and the entry to certain problem program routines such as 1052 operator communications routine, program check routine, and interval timer routine. User imperative macros (GET, PUT, FETCH, etc.) can be origined beyond 64K if desired.
  3. When using logical IOCS, the OPEN, CLOSE, and end-of-volume routines checkpoint the first 2500 bytes of the program area. Therefore, the user routines, entered in the event of interruption from the 1052, or

the timer must begin at least 2500 bytes beyond the end of the Supervisor area.

- Dual-Density Feature.
- Problem programs produced by the Assembler or compiled by RPG can request I/O operations on the following devices:

1. 1442 Card Read-Punch.

2. 2501 Card Reader.

3. 2520 Card Read-Punch and 2520 Card Punch.

4. 2540 Card Read-Punch (also punch-feed-read).

5. 1403 Printer.

6. 1404 Printer (for continuous forms only).

7. 1443 Printer.

8. 1445 Printer.

9. 1052 Printer-Keyboard. Only one 1052 is supported. It is attached to the multiplexor channel. When this device is on the system, it is used for operator communication.

10. 2311 Disk Storage Drive. For the sort/merge program a maximum of eight, including the system drive, two or four of which can be used for intermediate storage, are supported.

11. 2400-series Magnetic Tape Units. If the 800/1600 bpi dual density feature is to be used, the tape unit must be one of the following:

    a. 2401 or 2402 Magnetic Tape Unit, Model 4, 5, or 6 and 2803 or 2804 Tape Control, Model 2.

    b. 2403 Magnetic Tape Unit and Control, Model 4, 5, or 6.

    c. 2415 Magnetic Tape Unit and Control, Model 4, 5, or 6.

    If variable length records are to be read or written on 7-track tape, the Data Conversion Special Feature is required.

    With the sort/merge programs, one to four tape units are supported for input/output to a sort operation, and one to six tape units for input/output to a merge operation.

    Both IBM-supplied system programs and user problem programs can use magnetic tape. However, the main storage required for physical and logical IOCS for both tape and disk will probably make this unfeasible in systems with less than 16K bytes of main storage.

12. 2671 Paper Tape Reader (Assembler programs only).

13. 1285 Optical Reader. (Assembler programs only.) A maximum of eight is supported.

14. 1287 Optical Reader* (Assembler programs only). A maximum of eight is supported.

15. STR (Synchronous Transmitter Receiver) devices, connected by leased or dial lines through an IBM Synchronous Data Adapter - Type I, on an IBM 2701 Data Adapter Unit (Assembler programs only). The following devices are supported:

    a. 1009 Data Transmission Unit.

    b. 1013 Card Transmission Terminal.

    c. 1974 II Data Transmission Terminal.

    d. 1978 Print, Read, Punch Terminal.

    e. System/360, Model 30, 40, 50, 65, or 75, with a 2701 Data Adapter attached.

    f. System/360, Model 20 with a Communications Adapter.

    g. 7701, 7702 Magnetic Tape Transmission Terminal.

    h. 7711 Data Communication Unit.

    Note: Most STR applications require 16K bytes of main storage; however, utility type applications, requiring minimum processing and code conversion, are supported in an 8K environment.

16. BSC (Binary Synchronous Communication) IBM 2701 Data Adapter Unit, equipped with an IBM Synchronous Data Adapter Type II, connected by leased or dial line to a remote IBM System/360, Model 30,

40, 50, 65, or 75, and equipped with an IBM 2701 or 2703 Data Adapter Unit with an SDA II.* (Assembler programs only.)

Note: BSC applications require a minimum of 16K bytes of main storage.

*Programming specifications for using the 1287 Optical Reader and BSC may be used for planning purposes only. Source programs must not contain instructions for these devices until the IOCS routines include the appropriate programming. An MNOTE stating IMPROPER DEVICE will appear if coding for these devices is included in a source program.

RESIDENCE REQUIREMENTS

The Basic Operating System requires that there always be a disk pack on-line when programs are being executed and that the disk pack must always contain certain resident programs and areas. There are two basic approaches to a minimum resident system:

1. Fixed-Job Systems A version of BOS can be built to handle specific jobs, with no system modification facility. Such a minimum system must have the fixed-assignment system features on tracks 0-5, a core image directory, and a core image library with the following programs:

   a. Supervisor
   b. Job Control
   c. The user's problem programs to be run in the system.

2. Variable-Job System A version of the system can be built to load and execute any number of problem programs. Each program is read from cards or tape, edited into the core image library, and then loaded into main storage for execution. Successive programs overlay the previous program in the core image library. Such a system must have the fixed assignment system features on tracks 0-5, a core image directory, and a core image library with the following programs:

   a. Supervisor
   b. Job Control
   c. Linkage Editor
   d. Enough room for the largest problem program that is to be run in the system.

In both of the above minimum configurations, the disk pack must include a label area (Volume Table of Contents-VTOC) and a track for storing label control card information. To gain residence flexibility, either of the minimum configurations can be expanded to include the core image library maintenance and organization routines.

Minimum Residence Size Estimates

The following size estimates are provided to assist in system planning. The system described is a minimum variable-job system. More complete residence requirements will be given at a later date. Main storage requirements and processing times are given in Appendix H.

Number of Tracks	Contents	
6	Fixed-assignment system features on tracks 0-5	Note 1
1	Core image directory	Note 2
50	Core image library	Note 3
1	Label control card area	Note 4

Note 1: System residence always begins with track 00 of cylinder 00. Before creating a resident pack, the disk pack must be prepared with the Initialize Disk Utility program. This program writes record zero on every track with an eight-byte data area and no key field. This program also preformats the disk file label area (Volume Table of Contents-VTOC). The size of the area required for the VTOC depends on the number of files on the pack and the number of separately defined extents for each file. One track is usually sufficient for the VTOC; it may never exceed one cylinder.

Note 2: The core image directory is written with entries for 124 phases on each track. The estimate given above (1 track) for the minimum system allows approximately 60 user problem program phases.

Note 3: Program phases are written in the core image library in fixed-length records 824 bytes long (no key fields). There are four records per track. A phase always begins in a new record but may begin in any record location on a track. The estimate given above (50 tracks) for the minimum system assumes the following programs requiring 40 tracks:

Supervisor
Job Control
Linkage Editor

This leaves 10 tracks for user problem programs. These 10 tracks can hold approximately eight 4K phases.

Note 4: A single track for the Label Control Card Area allows enough room for most jobs. This area is written with fixed-length records, each containing a seven-byte key (file name for VOL card) and an 85-byte data area. Each DLAB, TPLAB, and XTENT card is written as a separate record. Each track can hold up to 20 records.

Single-Drive System Limitations

System/360 installations with a single IBM 2311 Disk Storage Drive must create system residence on every disk pack used. Note that system residence need contain only those programs required for the files on the disk pack. Separate disk resident systems can be built to facilitate assembling or compiling programs. The same disk pack used to assemble or compile can also be used for program testing.

Special consideration must be given to jobs that process files that are stored on more than one disk pack. The programs used on the file must be contained in the core image library of each pack. Since the Basic Operating System uses system residence to store phase directory and the control card label information, if disk or tape labels are to be processed, the following restrictions must be met:

Single-Phase Program No disk or tape labels processed: The only restriction here is that each pack must have identical control programs. The problem program need be entered in the core image library of only the first pack. The Supervisor and Job Control, however, must be on each pack. This type of operation can be performed only when using the physical I/O macros, since logical IOCS requires disk label processing.

Single-Phase Program With disk or tape label processing, or Multi-Phase Program with or without label processing:

1. Each pack must have the same, identical control programs in the core image library.

2. The problem program must be prepared as a separate job for each pack. The job control cards, including any label cards, and the problem program deck (unless the program is permanently cataloged in the system) must be submitted for each pack.

Preparing the program as a separate job for each pack (step 2 above) can be done in various ways. The most straight-forward is to run the program to end of job for each pack, beginning the next pack by going through the complete IPL (initial program load) procedure for each pack. However, this procedure prevents passing information (totals, etc) from pack to pack unless provided for in the problem program (punched cards, to be reread for the next pack).

The second technique is to set up each pack of the file before execution begins, starting with the last pack of the file and ending with the first pack of the file. The job control cards for each pack except the last (the first pack of the file) include a PAUSE card before the EXEC card. This allows Job Control to enter the program in core image library, construct the phase directory, and process the label information control cards for each pack. Instead of pressing the interrupt key to begin execution, place the next pack on the drive (and replace the card deck in the hopper) and initiate the IPL procedure to repeat the operation on that pack. Program execution begins after the last pack has been set up. When the program completes processing the file on the first pack, a message (MSG) macro can be issued to place the system in the wait state and tell the operator to place the next pack on the drive. Since each pack has system residence identical to the last, processing can continue to the end of the file.

A third technique is to run the program as a separate job for each pack, with pack-to-pack information saved in an area of core storage reserved above the Supervisor. When the program completes processing the first pack, Job Control is called (EOJ macro) to reprocess the control cards and the problem program deck (unless the program is permanently cataloged in the core image library) for the next pack. The job control cards for all except the first pack are preceded by a PAUSE card, allowing the operator to replace the pack. Job Control and the user's problem program can

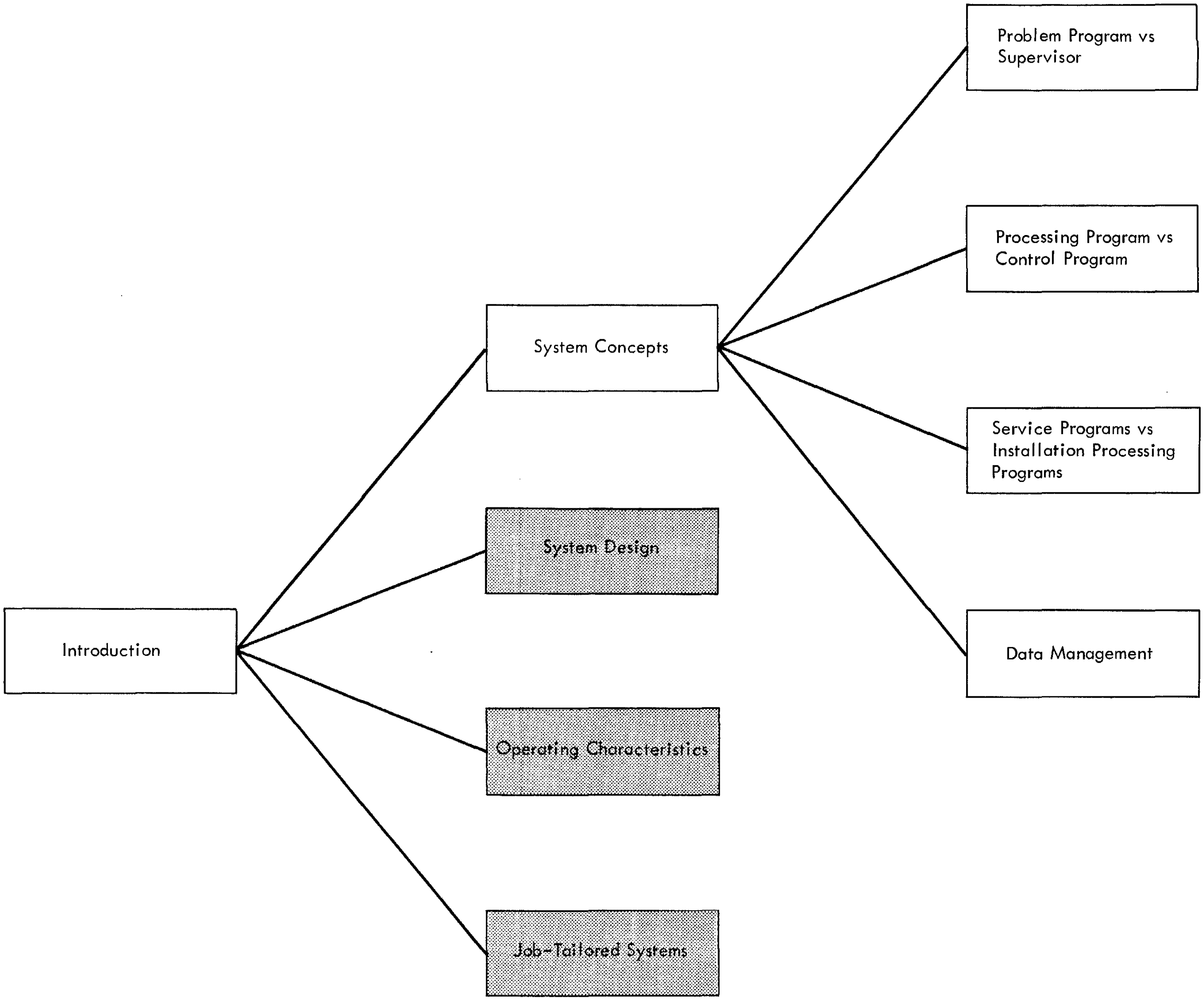

Introduction
System Concepts
System Design
Operating Characteristics
Job-Tailored Systems
Problem Program vs Supervisor
Processing Program vs Control Program
Service Programs vs Installation Processing Programs
Data Management

Figure 1. Introduction: System Concepts

be origined at some point above the Supervisor to leave space for pack-to-pack information. The problem program can test this area for blanks to detect whether it is beginning the first pack or was recalled to process a continuation pack.

SYSTEM CONCEPTS

The contents of this section are illustrated by Figure 1. BOS is designed to make programming easy and job execution efficient. To make full use of the system, the programmer should be familiar with the function of each part of the system, its position in the resident structure, and its relationship to other parts of the system in terms of sequence of operation.

The following sections present the concepts behind the various catagories of programs within BOS. The concepts discussed are:

1. Problem Program vs Supervisor

2. Processing Program vs Control Program

3. Service Programs vs Installation Processing Programs

4. Data Management

PROBLEM PROGRAM VS SUPERVISOR

A distinction can be made between two kinds of program functions in all data processing operations. One of these kinds is the common, required functions found in all programs, such as:

1. Handling of input/output devices
2. Error detection and recovery
3. Program loading
4. Communication between the program and the operator.

The other function in any program is the actual processing where information or data is operated on in some way to produce new or updated information. Both of these functions can be handled more easily and efficiently by separate programs. In BOS, the Supervisor is provided to handle the common, required functions. This leaves the System/360 programmer free to concentrate on the problem-solving aspects of his program. The user's program is referred to as the problem program.

The Supervisor and the problem program exist as two distinct programs in main storage. Processing alternates between the two, with the Supervisor receiving control either through a programmed interrupt to perform a requested operation or through an automatic (machine-caused) interrupt to handle a machine condition. The Supervisor returns control to the proper point in the problem program upon completion of the operation. Two outstanding characteristics of the System/360 are designed to facilitate this capability:

1. The processing unit operates in either of two distinct states, the problem state or the supervisor state, depending on which program is currently being executed. When in the supervisor state, certain privileged instructions that are available only to the Supervisor can be executed to handle those functions that are unique to its operation.

2. Five kinds of automatic program interrupt allow virtual nonstop processing. This means that machine conditions requiring program action can be brought to the immediate attention of the Supervisor instead of waiting for the program to test the condition. The most important advantage in this respect is the ease with which maximum I/O speeds can be maintained while utilizing overlapped process time.

Programmed interrupts are used by the problem program to request Supervisor operations. I/O requests, operator communication, and the loading of successive program overlays are among the operations handled by the Supervisor in response to these programmed interrupts.

SUPERVISOR

The Supervisor provides the following functions for all other programs in BOS including both IBM-supplied programs and the user's problem programs.

Physical I/O Control. The physical I/O control routines are the largest single element of the Supervisor. These routines handle the scheduling and supervised execution of channel programs. The problem programs, or Logical IOCS within the problem program (see Data Management), supplies a list of channel command words (CCWs) and issues physical I/O macros to request its execution. One of the physical I/O control routines, the Channel Scheduler, starts the I/O operation and returns control to the problem program. If the I/O channel or unit is busy when the request is made, the Channel Scheduler places the request in a list, or queue, and returns control to the problem program. The operation then begins as soon as the channel and unit are available. When the operation is completed, an I/O interrupt returns control to the Supervisor to check for and handle all device error conditions. If another request is pending for the device, it is then started. The Supervisor includes error routines for all I/O devices on the system.

Program Loading. All programs are loaded into main storage from disk storage by a routine of the Supervisor called the System Loader. (Programs are placed in disk storage from cards or tape by the Linkage Editor, discussed under Service Programs vs Installation Processing Programs.) The System Loader is used to load successive phases, or overlays, of a program by issuing a FETCH macro instruction. A problem program can be written with the main body of the program designed to remain in main storage throughout execution of the job and with special subroutines that are fetched when needed, each overlaying the last. Since the entire program is in disk storage, subroutines can be called and recalled as often as needed.

Operator Communication. All communication between the Supervisor or problem program and the machine operator is handled by the Operator Communication routines of the

Supervisor. Coded messages are made available to the operator through the system control panel and are printed on the 1052 printer-keyboard when it is available and assigned to SYSLOG. The operator can reply to a message or he can initiate communication with the problem program or the Supervisor.

Checkpoint/Restart. These routines provide a means of stopping a program at some point other than at end of job and restarting the program again from that point. In response to a macro instruction in the problem program, the Checkpoint routine writes the problem program out in disk storage or on magnetic tape, along with the other information needed to restart the program. The restart routine can then be used to reload the program and restart at the proper point.

Interruption Handling. All interruptions turn control over to the Supervisor. There are five kinds of interruption. The following lists the action taken for each:

1. Supervisor Call This interruption is caused by a request from the problem program (SVC instruction, normally assembled from a macro). The SVC interruption routine examines the interruption code supplied by the instruction and transfers control to the proper routine to handle the request.

2. Program Check If the user has supplied the address of a program check routine, control is transfered to it. If not, the job is terminated either immediately or after printing the contents of the problem program area of main storage.

3. Machine Check The system is placed in the wait state with all interruptions masked.

4. I/O Interruption All I/O interruptions are handled by the Channel Scheduler.

5. External Interruption External signal interruptions (available with the Direct Control special feature) are ignored by the Supervisor; control passes directly back to the point of interrupt. Interruptions caused by pressing the interrupt key on the console are handled by the Operator Communication routines. Timer interruptions are turned over to a routine specified by the user. If none is specified, the interruption is ignored.

End of Job. The Supervisor transfers control of the system to Job Control (discussed in the next section) at end of job to provide automatic job-to-job transition.

Storage Print. This routine can provide a print-out of main storage in the event of an abnormal end-of-job condition. The operation can be initiated by program check conditions, by a macro instruction in the problem program, or at the request of the operator.

Some of the routines of the Supervisor are loaded into main storage during system initialization (see IPL Loader in the next section). These routines are never overlaid and remain in main storage throughout the execution of any number of jobs. Other routines of the Supervisor are called into main storage from disk storage only when their particular function is needed. These are called transient routines. They are loaded into an area of main storage called the transient area. When a routine is loaded into the transient area, it overlays the previous routine in the area. This allows several Supervisor functions to be provided while using a minimum amount of main storage. Although the transient area of the Supervisor is not available for the problem program, a similar area can be set up in the problem program region to provide the same facility.

PROCESSING PROGRAM VS CONTROL PROGRAM

The concept of separation of function between problem program and Supervisor can be extended to apply to the entire collection of programs necessary to perform the many jobs of a data processing system. An efficiently run installation makes use of some kind of automatic control over the total operation. Without such control, the system frequently is left idle, requiring the intervention of an operator to set up and load successive programs or program sections. Automatic control of the total system can be accomplished by using special control programs that provide continuous operation of the system. The control programs must be able to perform all of the functions necessary to provide automatic transition from phase to phase within a job, and from job to job within the total processing environment. Thus, just as we distinguish between problem program and Supervisor, we can distinguish between processing programs and control programs.

The combination of a group of processing programs with the control routines necessary to maintain continuous operation of those programs is called an operating system. An operating system is self-contained and requires operator intervention only under exceptional conditions.

BOS control programs actually tie together the various processing programs into an operating system. In a sense, they are the operating system, with the processing programs simply running in the over-all environment created by them. In the minimum case, an operating system can be produced that consists only of the control programs, with the processing programs entered temporarily into the core image library for loading and then overlayed by the next program.

All processing programs, or "jobs", are loaded by and controlled by the system control programs. This distinction is helpful in understanding the system:

1. All processing programs are run as jobs, with the name of the program entered in a JOB control card.

2. System control programs are never run as a job, but instead, are entered automatically as needed to load and supervise jobs. Note that the Linkage Editor (discussed in the next section) is never run as a distinct job and in this respect is similar to the control programs. However, the Linkage Editor operates primarily as a function of the Librarian and, therefore, is classed with the system service programs.

The BOS control programs are:

1. Supervisor (discussed in the preceding section)
2. Job Control
3. IPL Loader

JOB CONTROL

The Job Control program runs between jobs and provides transition from job to job. This includes the following operations.

Control Card Processing

The sequence of jobs run in the basic operating system is determined by job control cards read from the system card reader. The job control cards are used to:

1. Provide the name of the program or phase to be run.

2. Assign actual device addresses to the symbolic unit names used in all problem programs.

3. Enter control information in the communications region. The information includes:

   a. Today's date
   b. The User Program Switch Indicators (UPSI)
   c. The object machine configuration (CONFG).

4. Enter information required to process disk and tape labels. This information is stored in the resident pack and used during job execution by the OPEN, CLOSE, and end-of-volume routines.

5. Inform Job Control that the operator desires that processing be suspended.

6. Inform Job Control that the Linkage Editor is or is not needed to edit the program into the resident pack.

7. Signal Job Control to begin executing the job.

8. Signal Job Control to begin or stop logging (printing) job control cards.

9. Prepare for restarting previously checkpointed jobs.

I/O Device Assignment

All problem programs refer to I/O devices by symbolic unit names. The Supervisor has a table associated with these symbolic unit names, each with a physical device address normally assigned during system generation. Job Control can reassign these addresses, if necessary.

Phase Directory Construction

Before starting execution of a job (and after the Linkage Editor, if used,) Job Control constructs a phase directory for the job. The System Loader routine of the Supervisor uses the directory to retrieve successive phases of the program as they are needed.

IPL LOADER

System initialization is provided by the IPL Loader. This program loads the Supervisor from the resident pack into the lower part of main storage and calls for the Job Control program. Once this is done, any number of jobs can be executed without reinitializing the system.

The IPL Loader itself is read into main storage through the automatic IPL procedure provided in the System/360. The operator places the resident disk pack on a disk drive and selects the address of that drive in the load-unit switches on the console. Pressing the load key causes the IPL Loader to be read into main storage from the resident pack.

SERVICE PROGRAMS VS INSTALLATION PROCESSING PROGRAMS

An operating system consists of two kinds of programs: the control programs and the processing programs. It is convenient, however, to distinguish between two types of processing programs.

The first type is called service programs, including the IBM-supplied programs that provide functions such as:

1. System Generation and Maintenance
2. Program Assembly and Compilation
3. Program Testing
4. Utility Programs
5. Sort/Merge Programs.

The second type is the installation processing programs that do the actual data processing for which the operating system is designed. For example, inventory control, payroll, accounts receivable update, etc. These user programs are incorporated as integral parts of BOS and take full advantage of the power of the control programs.

The IBM-supplied service programs available in BOS are:

System Service Programs: provide for the creation and maintenance of the system. These programs include:

1. Linkage Editor - edits the output of the language translators to produce executable program phases in the core image library of the resident pack. Programs are never loaded directly from cards into main storage. Instead, they are first edited into the core image library, and then loaded into main storage by the System Loader routine of the Supervisor. Input to the Linkage Editor can be from cards, tape, or the relocatable library. The programs can be entered temporarily, executed, and then overlaid in the library by the next program, or they can be entered as permanent components of the system.

2. Librarian - consists of a group of programs that maintain, service, and organize the system libraries.

3. Load System - creates the initial resident system from cards (this program operates as an independent, non-resident program). The system created may be used to generate other, specialized systems, or the Load System program itself can be used to produce specialized systems.

Language Translators: translate source programs into machine language object programs. Two language translators are provided:

1. Report Program Generator - compiles programs written in the problem-oriented RPG language.

2. Assembler - produces problem programs from source programs written in the one-for-one machine oriented Assembler language. Includes macro generation capability with a complete set of macros provided by IBM for record processing (IOCS) and for communication with the Supervisor. An easily used macro definition language allows the user to define his own macros for inclusion in the system. See Macro Library in the section on System Design for a description of the purpose and use of macros.

Autotest: exercises control over problem programs to provide the following functions:

1. Automatic inclusion of patch cards.

2. Print-out of the status of the system at any point in the program.

3. Main storage print-out in the event of program failure.

Disk Sort/Merge: sorts data files in disk storage. This program can accept input from cards or tape.

Utility Programs: provide for file-to-file transition for data files of almost any format. These generalized programs are tailored by control-card information to fit specific data files.

Remote Job Entry: provides facilities for submitting jobs and job control information to, and receiving output from, a central computing system. The Remote Job Entry Work Station program operates in conjunction with the Operating System Remote Job Entry program which resides at the central system. Jobs to be run under Operating System are transmitted to the central system and executed there. The output of these jobs is returned to the remote computer at the direction of the user submitting the job. Job output may also be specified for processing by the central system's output writers. Remote Job Entry (RJE) supports card input and punched or printed output. An exit is provided allowing a user written routine to write output to other available devices.

The Remote Job Entry Work Station program is generated from the RJE macro-instruction. The macro instruction parameters describe the System/360 configuration and the transmission line characteristics. This allows the Remote Job Entry Work Station program to be tailored to each work station configuration and the user's needs. All the resources of the central system and all the facilities of Operating System are made available to users at the remote work station.

Refer to the Remote Job Entry publication, listed in the preface of this manual, for a more detailed description.

DATA MANAGEMENT

BOS provides a number of routines to facilitate handling input and output data files. These facilities are collectively referred to as Data Management. Although data management does not correspond to any single component of the operating system, distinct from other programs or routines, it does describe a functional capability that can and should be treated as a separate subject. The system features that provide the data management functions are described below under the topics:

Physical IOCS
Logical IOCS
Label Processing.

PHYSICAL IOCS

Certain routines of the Supervisor (and the macro instructions provided to communicate with them) are referred to as Physical IOCS. These routines handle the scheduling and supervise the execution of channel programs. The problem program (or logical IOCS within the problem program) supplies the channel program (a list of CCWs) and issues physical I/O macros to request its execution.

The Supervisor starts the I/O operation and returns control to the problem program. When the operation is completed, the Supervisor checks for and handles all device error conditions. The user's program need not contain any I/O device error routines.

LOGICAL IOCS

A comprehensive set of macro routines is provided to handle the creation, retrieval, and maintenance of data files. Three basic sets of routines are available, each designed to handle files organized in a specific way. These routines are generated in response to descriptive macro instructions (DTFSR, DTFDA, DTFIS) as a part of the user's problem program. The routines are assembled immediately preceding the portion of the problem program coded by the user. They occupy an area of main storage between the user's program and the Supervisor. Imperative macro instructions issued by the programmer cause a branch to the proper point in the logical IOCS routines, and the requested operation is performed. The logical IOCS routines in turn request physical I/O operations to be performed by the physical IOCS routines. The functions provided by logical IOCS include:

- Request physical I/O operations by issuing the physical IOCS macro instruction otherwise required in the problem program. The necessary channel programs (CCWs) are provided by logical IOCS.
- Supply logical input records to, or accept logical output records from, the problem program. This includes blocking and deblocking logical data records from larger physical blocks. (Logical record refers to the individual unit of a data file. Physical record refers to the block of logical records read or written as a single string of information.)
- Switch between two I/O areas to provide usable time for processing while records are being read or written.
- Handle end-of-file and end-of-volume conditions.

- Construct and maintain disk file organization structures. This includes additions and deletions to sequential disk files and construction and use of index tables for random processing of sequentially organized disk files.

The DTF macro instructions from which logical IOCS routines are generated must be assembled along with all portions of the problem program that issue imperative macro instructions for these DTF routines. The Linkage Editor cannot combine DTF routines with separately assembled program sections if those program sections must communicate with the DTF routines.

LABEL PROCESSING

Disk and tape label processing capabilities are included in BOS to provide:

1. Assurance that the correct editions of disk and tape data files are provided for input and (in the case of multi-pack or multi-reel files) that this input is provided in the correct sequence.

2. Assurance that areas of disk storage or tape reels designated for output contain no current information. If usable, a new label is written for the output area or reel.

The actual label processing is performed by transient routines edited as part of the Supervisor during initial system generation. These routines are loaded into the transient area of the Supervisor and executed in response to OPEN and CLOSE macro instructions in the problem program. The actual request for the label routines is made to the System Loader either by the logical routines mentioned above (DTFSR, DTFDA, or DTFIS) or by another special routine assembled as part of the problem program (DTFPH).

SYSTEM DESIGN

OPERATING SYSTEM RESIDENCE

The contents of this section are shown in Figure 2. An overriding requirement of any operating system is that all its parts be immediately available to the system. It would be impractical, if not impossible, to keep all of these programs in main storage at all times. Therefore, it is necessary to have constant access to some external storage space. This storage space is called the residence of the system; the storage device is called the resident device.

BOS uses IBM 2311 Disk Storage for system residence. This means that there must be a disk pack on-line for execution of all programs, both IBM supplied and user programs. The amount of disk storage space required for a particular system ranges from a few cylinders to an entire pack, depending on the types of operations handled by the system. The various types of systems that can be developed are discussed under Job-Tailored Systems.

The resident system always occupies the first section of the disk pack, starting with track zero on cylinder zero. The first six tracks are always used for:

- IPL Loader

- Volume Labels (identify the disk pack)

- Directories indicating the location of various components of the system

- Work area used by the system service programs.

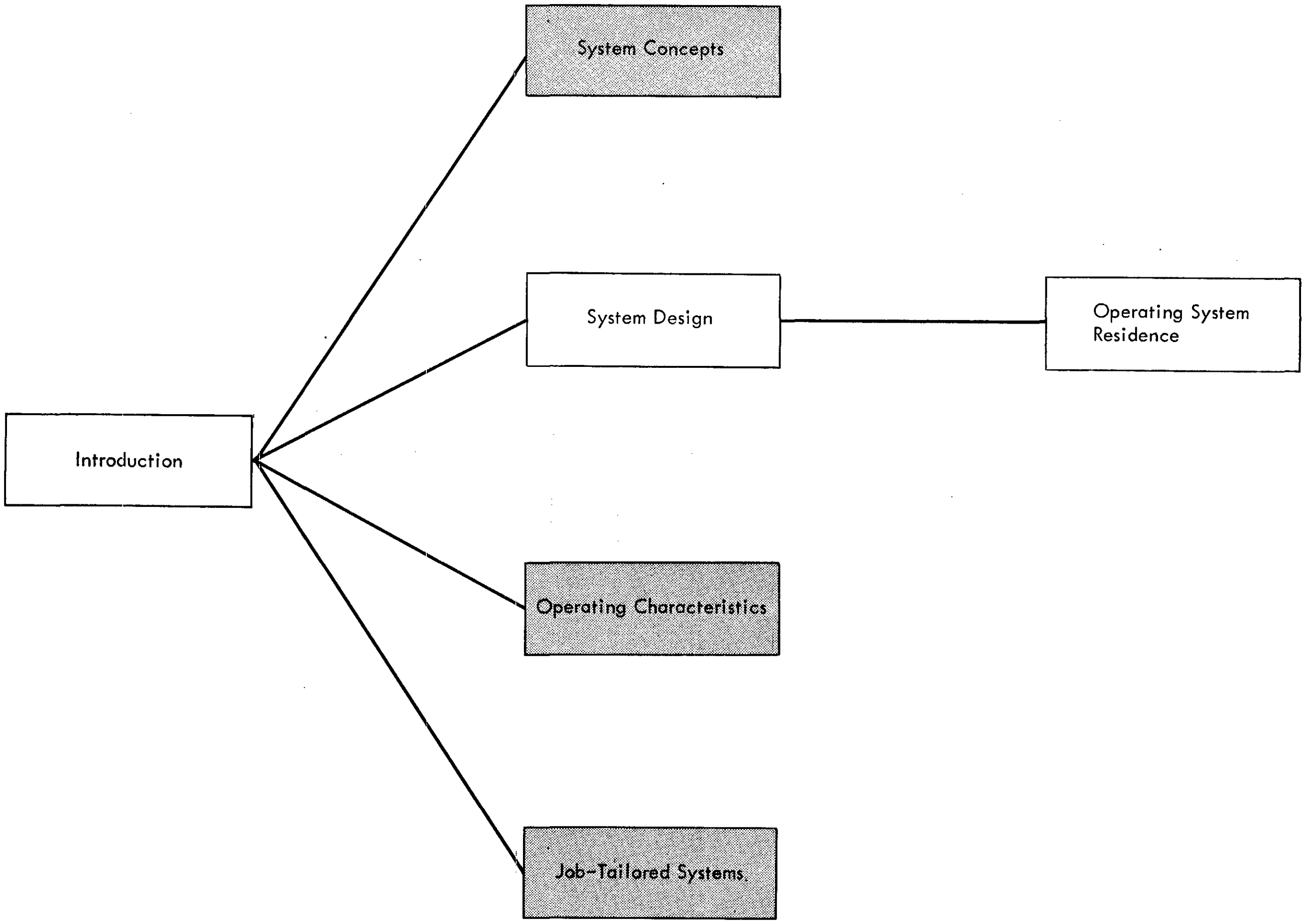


Figure 2. Introduction: System Design

LIBRARY AREAS

Core Image Library: The first six tracks are always followed by a core image program library area. With the exception of the IPL Loader, all programs (both processing programs and control programs) executed as part of the BOS are loaded from this core image library.

Programs are placed in this library either by the Load System program (when creating a system from cards) or by the Linkage Editor. The user's problem programs can be cataloged as permanent entries in the library or they can be entered temporarily, executed, and then overlaid by the next program. The control programs are always cataloged as permanent entries.

Two other library areas can be included in the system (following the core image library) if the purposes for which the system is built require them. They are:

Macro Library: Used to store the macro definitions processed by the Assembler during program assembly. A macro-definition language is provided as an extension of the assembler language. The macro-definition language provides the programmer with a convenient way of defining a sequence of assembler language statements that can be used by many different programs. The macro-definition is written only once and then cataloged in the macro library. Once placed in the macro library, any Assembler source program can include the sequence of instructions by issuing a single statement: a macro-instruction statement. The macro instruction can supply variable parameters to be placed in the assembled instruction sequence and to direct the Assembler to include or delete specific statements from the defined sequence.

A comprehensive set of macro definitions supplied by IBM provides the following functions:

1. Communicate service requests to the control programs

2. Logical record blocking and deblocking and I/O.

3. Generation of a Supervisor tailored to the specific requirements of the installation.

Relocatable Library: Used to store object modules (the output of the language translators) in relocatable format. The object modules stored in this library can be combined with other object modules (in the relocatable library or read from card or tape) by the Linkage Editor when editing a program into the core image library.

This library is designed so that an installation can maintain frequently used subroutines in the resident pack and link them into the problem program as it is edited into the core image library.

This library is not required in the minimum system. Separately assembled program sections can be combined when both are read from either cards or tape. The Assemble-and-Execute option uses this library.

LABEL CONTROL CARD AREA

The next area set aside in the system residence pack is the Label Control Card Area. Job Control stores control card information in this area for use by the disk and tape label processing routine (during program execution). Disk labels are required in BOS (see Volume Table of Contents below). When using the logical IOCS functions for processing disk files, these labels must be processed for each data file and this area must be available. At least one disk track must be allocated for this area on any system residence pack.

The use of this area allows label information to be entered before job execution.

CHECKPOINT AREA

The BOS Supervisor includes routines to checkpoint and restart programs. The checkpoint information can be written on either disk or magnetic tape. If the checkpoints are to be written on disk, this area is set aside during system generation. The checkpoint area follows the label control card area. The amount of space required depends on the size of the problem program. Successive checkpoints are each written over the last.

VOLUME TABLE OF CONTENTS

Every disk pack processed by programs in BOS must have an area set aside for file labels. This area is called the Volume Table of Contents (VTOC). File labels are required for certain areas even if logical IOCS is not used by the user's problem programs. See Disk Labels in Part 4 for a complete description of the use and format of the VTOC.

WORK AREAS

The areas discussed in the proceding paragraphs are all considered part of the residence structure. It should be pointed out, however, that some service programs supplied as part of the system require additional disk storage space for work areas. If more than one disk drive is available, these work areas can be partly or totally allocated to packs other than the resident pack. The service programs that require disk work areas in addition to system residence are:

Assembler	Autotest
RPG	Disk Sort/Merge

DATA FILE AREAS

All of the disk area in the resident pack not required for system residence or work areas can be used for data files. Data files can be stored in a resident disk pack with only the control programs and those processing programs used on the particular files. Other packs can be built for programming services only and never used for data file storage. See Job-Tailored Systems for a more complete discussion of the kinds of systems that can be produced.

OPERATING CHARACTERISTICS

The following discussion illustrates the functional relationship of the various components of BOS (Figure 3). The discussion assumes the existence of a complete operating system (i.e., the user has already performed system generation using the procedures described in the System Generation manual listed on the front cover of this publication). Note that steps 1-2 below are performed only to get the system started. Steps 3 and 4 are, in effect, a loop that continues until there are no jobs to be run.

Step 1. The machine operator places the resident disk pack on an IBM 2311 Disk Storage Drive and selects the address of that drive in the load-unit switches on the console. Pressing the load key causes the IPL Loader to be read from cylinder zero, track zero, of the resident pack.

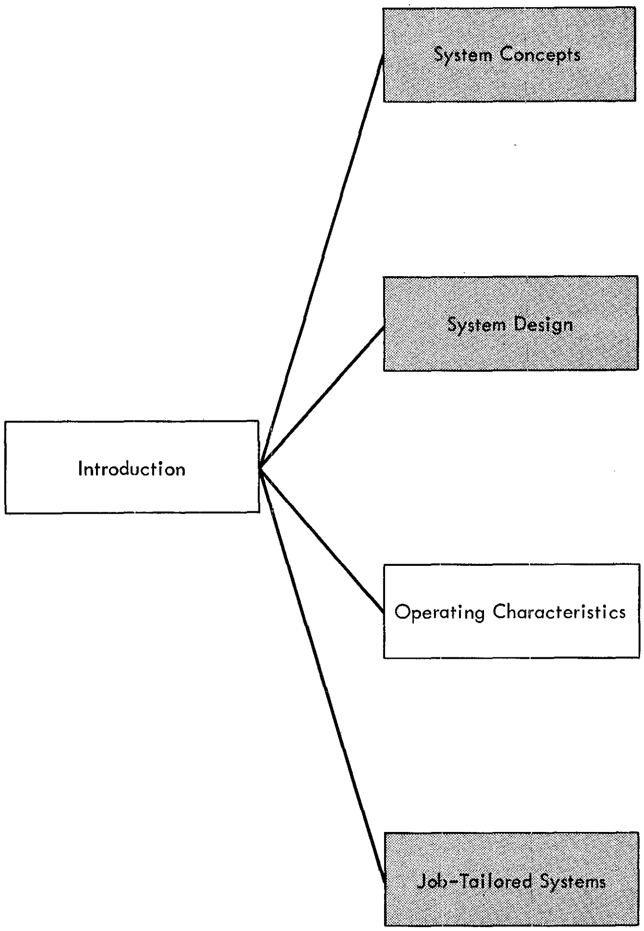


Figure 3. Introduction: Operating Characteristics

Step 2. The IPL Loader clears main storage and the general registers to zero. It then loads the Supervisor from the core image library into the lower part of main storage. Before turning control over to the Supervisor, certain other initializing functions are performed. The address of the resident disk drive is taken from the IPL PSW in location 0-3 and placed in a table for the Supervisor. Control is then passed to the Supervisor, which uses the System Loader routine to call Job Control. (Operation of the System Loader is described under step 4.)

Step 3. Job Control reads the job control cards defining the job to be run from the system card reader. Information from these cards is used to:

1. Assign actual I/O device addresses to the symbolic unit names used by all programs.

2. Store tape and disk label information in the Label Control Card Area of the resident disk pack for subsequent use by the routines that process the labels.

3. Place information in the communication region within the Supervisor area, including today's date, machine configuration, and user program switches. If the Linkage Editor is required, it is fetched from the core image library by the System Loader. The Linkage Editor reads program phases from the relocatable library and/or the system input unit and edits the program into the core image library. After completing the linkage of the entire program and placing it in the core image library, Job Control is brought back in.

The final function performed by Job Control (regardless of whether the Linkage Editor is required) is to construct a phase directory of the program to be run and write this on track 5 of the resident pack. The System Loader uses this directory to locate successive phases of the problem program in the core image library.

Step 4. The system Loader loads the first phase of the problem program in response to a request by Job Control. During the running of the job, control alternates between the problem program and the Supervisor. The problem program requests the System Loader to load successive program phases through the FETCH macro. The Supervisor uses the same method to retrieve the transient routines. At end of job, the problem program issues an EOJ macro (or, in a BSC environment, an ERRPT macro and an EOJ macro). At this point, Job Control is called back in to read the control cards for the next job. This puts us back to step 3. The control cards for the last job to be run are followed by a PAUSE card that causes processing to be suspended.

JOB-TAILORED SYSTEMS

The contents of this section are shown in Figure 4. Although an operating system provides a number of distinct advantages in terms of performance, utility, and application, it also requires space in both main storage and disk residence to perform its functions. Many of the services furnished by an operating system are basic (program loading, input/output control, etc). These would require storage space and time to be performed whether or not an operating system were used to perform them. Other services provided by an operating system yield better performance, wider application, or a reduction in manpower and training costs. The value of an operating system depends to a large extent on the specific requirements of an installation and how closely the services provided by the system meet those requirements. If a facility provided by an operating system is not required for a particular application, it should not take up storage space. Therefore, BOS is designed to enable individual facilities to be selected on the basis of whether they are required at a particular installation or for a particular application within an installation.

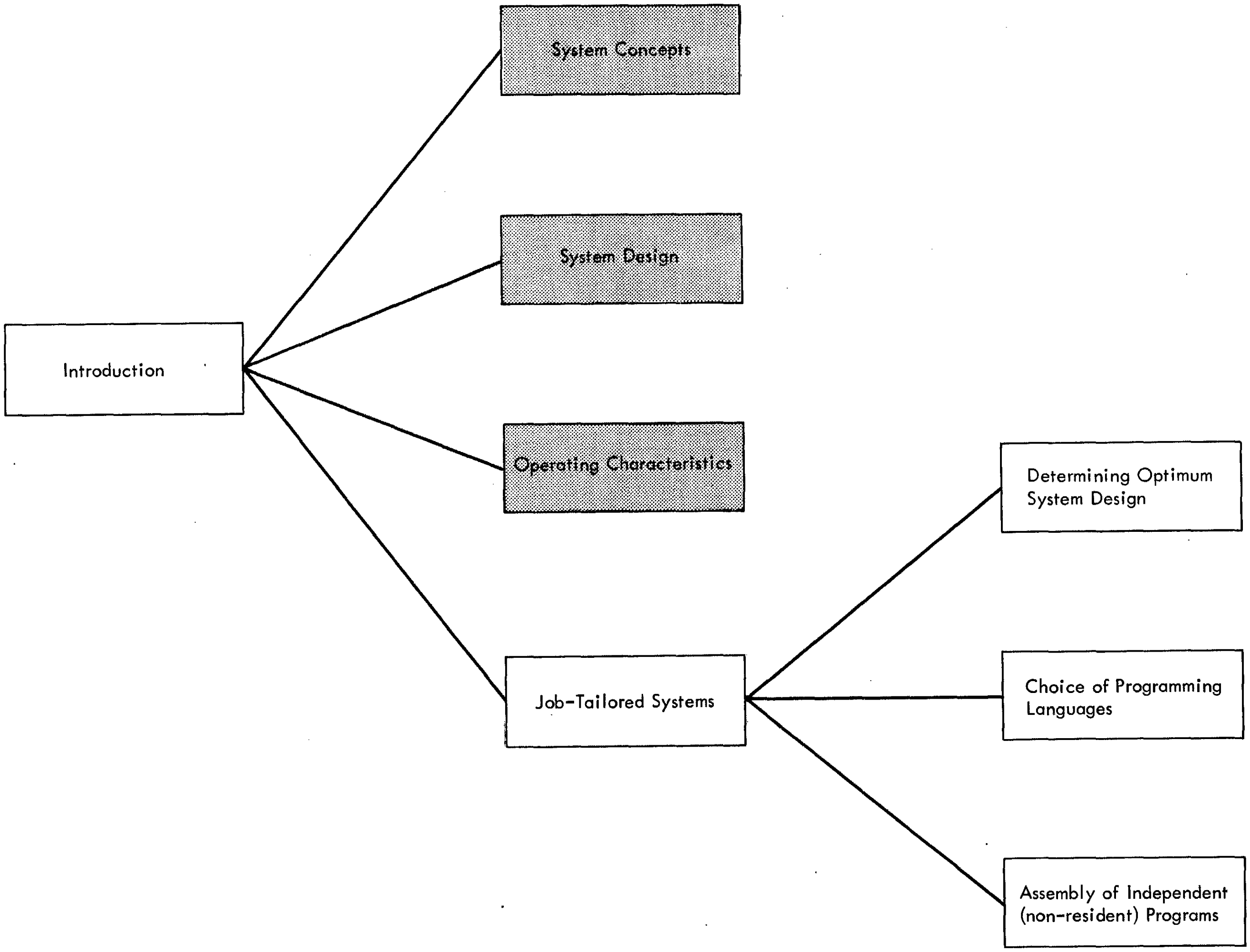

System Concepts
System Design
Introduction
Operating Characteristics
Job-Tailored Systems
Determining Optimum System Design
Choice of Programming Languages
Assembly of Independent (non-resident) Programs

Figure 4. Introduction: Job-Tailored Systems

DETERMINING OPTIMUM SYSTEM DESIGN

The optimum operating system design for a given application or group of applications must be determined by balancing many factors. Foremost among the factors affecting the decision are the possibilities and constraints of the machine system configuration. The two extreme cases are:

1. A single disk-drive system with disk data files that must share the resident pack with the programs that process them. A system of this kind dictates that special, minimum systems be generated for file processing.
2. Systems with a disk drive used specifically for operating system residence and containing no data files. A system of this kind allows basic operating system configurations to be expanded to include a large number or, in some cases, all of the installation's processing programs.

The following factors are discussed as they relate to determining optimum system design:

Disk-storage space requirements
System programming requirements
Operator handling considerations.

DISK-STORAGE SPACE REQUIREMENTS

Every BOS configuration must include the control programs (Supervisor and Job Control) in the core image library. In addition to these, a minimum system must include one of the following in the core image library (Figure 5):

1. The Linkage Editor and enough space to hold all the phases of the largest job that will ever be run in the system. A

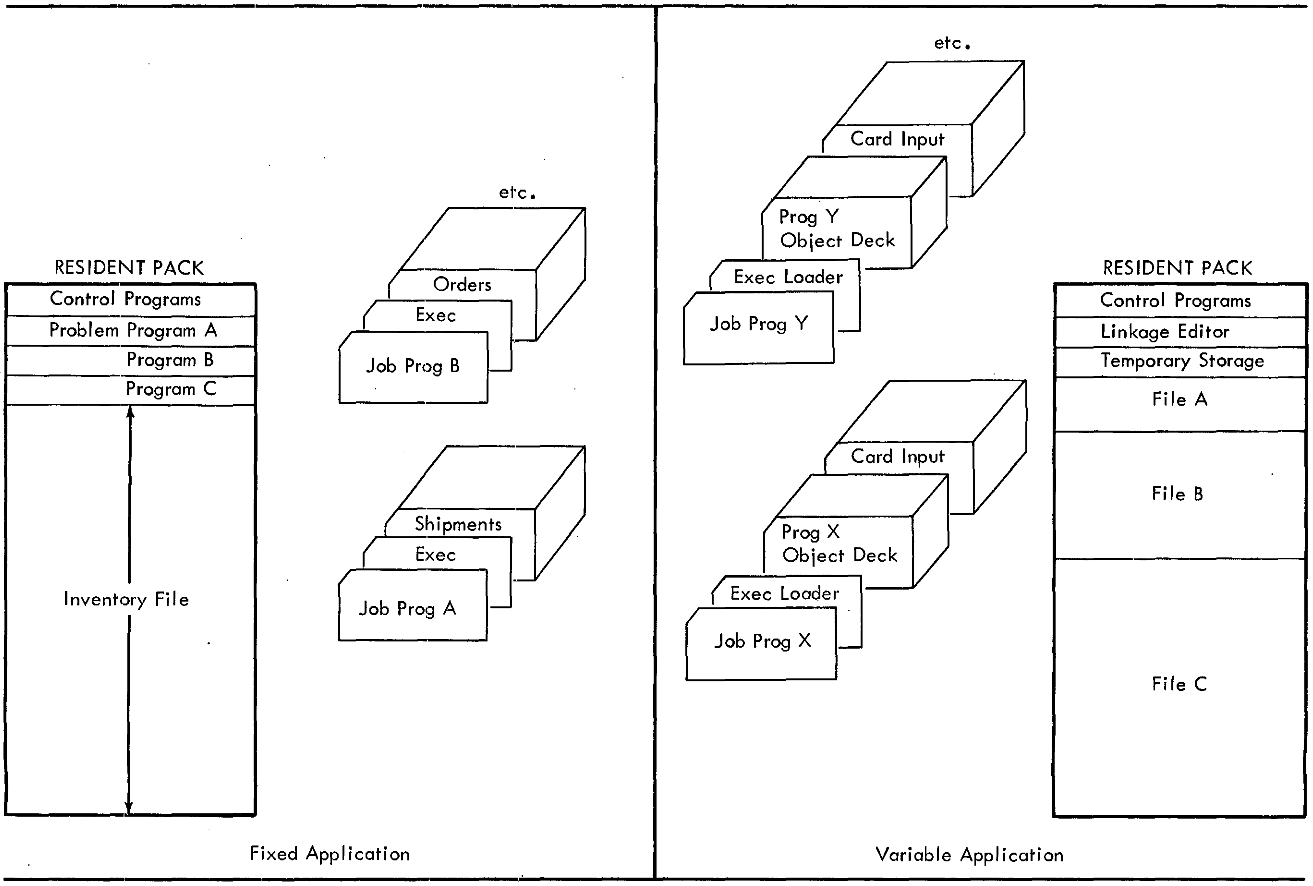


Figure 5. Two Approaches to Minimum Systems

system of this kind can load programs from card or tape, placing the phases for each job temporarily in the library for retrieval by the System Loader. The phases for each successive job are written over those for the previous job.

2. All problem programs ever run in the system. A system of this type is designed to execute specific programs in response to a JOB control card. All of the program phases used are entered at the time the system is built.

Either of these approaches can be used to create a minimum operating system. Both approaches are extreme cases by definition, however, and therefore are seldom the optimum configurations. Note that neither of these minimum systems can modify themselves. It is usually desirable to include at least the core image library maintenance and organization routines of the Librarian. These routines provide the flexibility of being able to reallocate or condense library space and to catalog program phases as permanent entries in the system.

SYSTEM PROGRAMMING REQUIREMENTS

BOS configurations built to provide programming services normally are not used to execute installation processing programs except as they are being tested and debugged. A single pack can include the Assembler, RPG, and Autotest, and thus be able to provide the programming services for the installation. Such a pack would include all of the following:

1. In the core image library:

   a. Supervisor

   b. Job Control

   c. Linkage Editor

   d. All of the routines of the Librarian

   e. Assembler

   f. RPG

   g. Autotest

   h. Enough room to temporarily load and execute problem programs. Once completely checked out, these problem programs can be copied into a separate pack along with the necessary control programs to make up a specialized system.

2. In the macro library:

   a. The IBM-supplied macro definitions
   b. Any user-written macro definitions

3. In the relocatable library:

   a. Any previously assembled user routines that are to be combined with routines from other assemblies.
   b. Enough space for Autotest to load any programs to be tested.

4. Label control card area (volume area).

5. A disk work area that can be used by all three of the programs.

Systems built to handle one or more specific processing programs, on the other hand, normally do not require any programming facilities. Thus, the relocatable and macro libraries would not be required. Furthermore, none of the following programs would be required in the core image library:

Assembler

RPG

Autotest

Librarian routines for macro library

Librarian routines for relocatable library.

Report Program Generator: This program never uses the macro library facilities of the BOS. The relocatable library and programs that maintain it are not required for RPG. Although programs produced by RPG are always single-phase (no overlays), they can be combined with other program sections (exit routines) by the Linkage Editor. The relocatable library is often useful for this purpose but is not required.

Assembler: The assembler uses the macro library to process all macro instructions encountered in source programs. Since all control program requests are made through the use of macro instructions, it can be assumed that the macro library is one of the minimum requirements of the system. The relocatable library, on the other hand, is not required and in some cases the facilities that it provides do not warrant its inclusion in the system. In other cases it can be used to fullest advantage and more than justifies the disk storage space required. Assemble and Execute operations will also use the relocatable library.

Autotest: The Autotest Control Program resides in main storage along with the Supervisor and the user's problem program. For this reason the machine requirements for using Autotest to test programs include a minimum of 16K bytes of main storage. The system residence requirements include the relocatable library and the system service programs used to maintain it.

OPERATOR HANDLING CONSIDERATIONS

An important consideration in choosing the optimum system design to be used in an installation is the way in which the system must be handled by machine-room personnel. There are obvious advantages to the fixed-application approach shown in Figure 5 when the disk storage space requirements of the application allow it. The operator does not concern himself with loading object programs. This approach saves job set-up time as well as eliminating the Linkage Editor operation.

CHOICE OF PROGRAMMING LANGUAGE

Report Program Generator (RPG)

The RPG compiler produces object programs to do simple listings, perform numerous calculations, use multiple files, search tables, and update files. Thus, it is possible to produce reports ranging from simple listings from cards to complete jobs such as payroll, accounts receivable, etc. Special coding sheets are provided to describe the job to be performed, and the kind of output necessary. The RPG language used on these sheets is problem oriented and does not require detailed knowledge of machine functions.

RPG compiles directly to object code. The final output is in the same form as that produced by the Assembler. This means that the main body of a program can be written in RPG, and separately assembled routines written in the Assembler language can be combined with it by the Linkage Editor. Both parts of the program can reference symbols defined in the other.

The object program produced by the RPG compiler may be punched into cards or written on disk in the relocatable library. The library entry can be permanent or temporary. If the object program is to be a permanent entry, duplicate entries must be deleted before cataloging this entry.

The object program produced by the BOS RPG compiler does not have the capability of running with a non-resident Supervisor.

Assembler

The Assembler language provides a convenient means of solving problems by offering the full flexibility of the powerful System/360 instruction set. This easy-to-use symbolic language is machine oriented and applicable to both commercial and scientific problems. The Assembler Language includes a complete set of macro instructions for all I/O and other Supervisor functions. Also, the user can define frequently used routines as macros. The actual problem coding is done with symbolic instructions that are translated, one for one, to machine instructions. All storage references can be made through symbolic names. Data constants can be defined in several different ways, either as explicit constants or as literals coded directly into the operand of an instruction. The problem programmer is able to produce the most efficient program possible for each specific and unique problem. Increased processing speed and reduced main storage requirements can usually be attained by a good programmer using the one-for-one assembler language instead of the more generalized RPG language. This often more than justifies the extra programming effort required.

ASSEMBLY OF INDEPENDENT (NON-RESIDENT) PROGRAMS

The programming services provided by BOS include assembling a Supervisor and a Job Control program for controlling problem programs executed in a non-resident environment. Disk files cannot be processed by these non-resident programs. The Supervisor and the Job Control program produced for non-resident use are loaded from cards (or loadable tape), and they require no resident storage.

Note: Object programs produced by the BOS RPG compiler are not capable of execution in a non-resident environment.

Both the non-resident Supervisor and the non-resident Job Control program (and Restart phase) can be generated from the macro definitions that are provided in either the IBM System/360 Basic Programming Support, Basic Tape System, or BOS.

A description of the non-resident Supervisor and Job Control program, and the operating characteristics of the non-resident environment are presented in the Basic Programming Support Programmer's Guide and Assembler with Input/Output Macros publications, listed in the Preface of this manual.

SUPERVISOR

The contents of this section are shown in Figure 6.

The Supervisor is the control program that operates with problem programs. Part of the Supervisor always resides in main storage. Certain other routines are kept in the core image library in the resident disk pack and are called into the transient area when needed. The functions performed by the Supervisor are:

1. Interruption handling
2. Channel scheduling
3. Device error recovery
4. Operator communication
5. Program retrieval
6. End-of-job handling
7. Checkpoint and restart
8. Label processing.

All functions except interruption handling are available to the problem program by issuing macros. The programmer is not concerned with machine interruption conditions since these are handled automatically by the Supervisor.

The Supervisor also contains a communication region for holding information useful to problem programs and to the Supervisor itself.

The Supervisor is generated from a macro library routine through an Assembler run. Normally this is done as part of the initial basic operating system generation. If desired, a Supervisor can be assembled for tape-only configurations to run outside of the BOS environment. This is a special case and is discussed under Assembly of Independent (Non-resident) Programs.

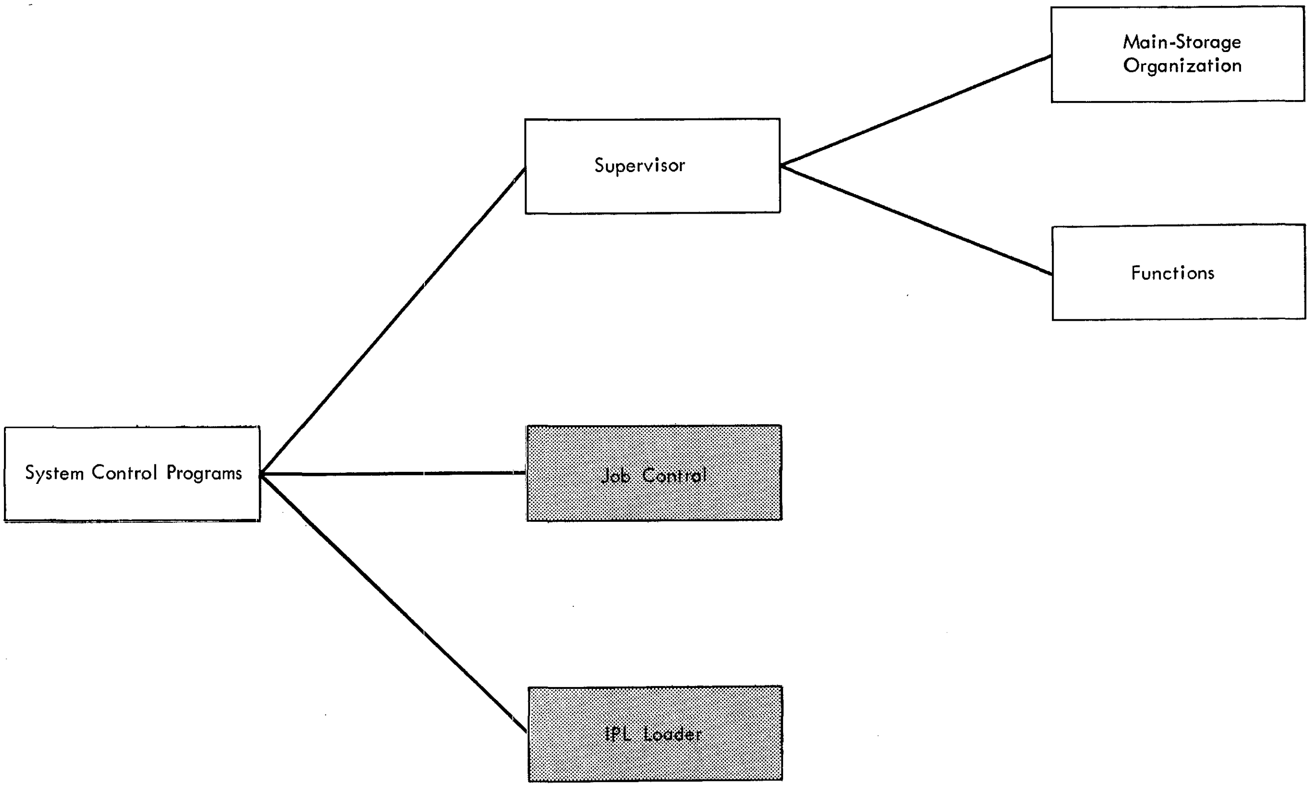


Figure 6. System Control Programs: Supervisor

MAIN STORAGE ORGANIZATION

The Supervisor occupies the lower area of main storage. The area occupied by the problem program begins just past the transient area. The main storage map in Figure 7 shows the relationship between the Supervisor and the problem programs. The transient routines are called into the transient area (overlaying the previous routine in the area) and executed when needed. The maximum length of a nonresident Supervisor is 1800 hexadecimal (6144 decimal), because IPL is loaded at 1800 hexadecimal. The disk IPL is relocatable and is loaded at the end of main storage minus 6E0 hexadecimal.

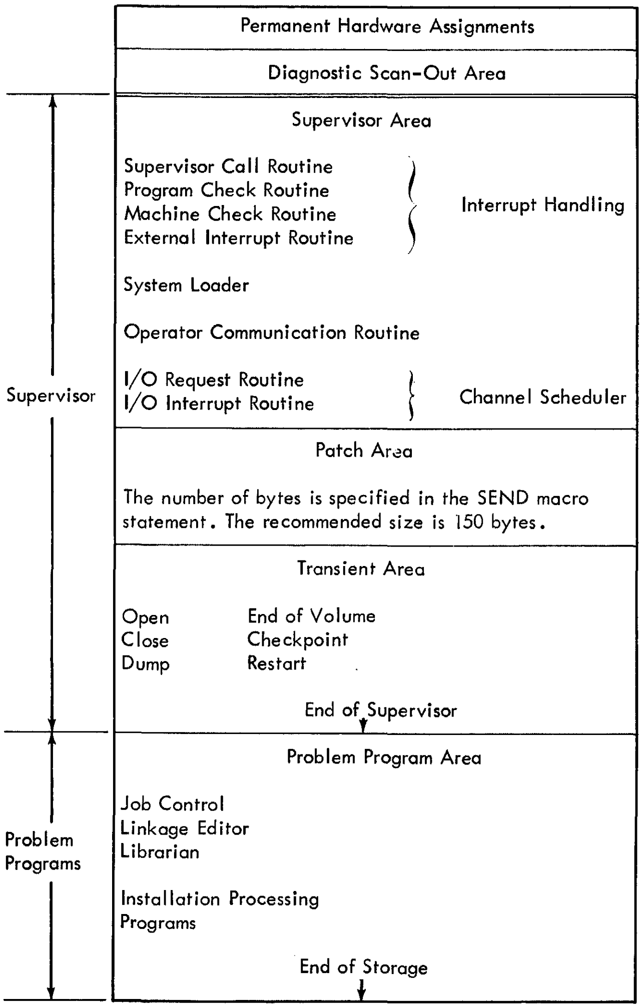


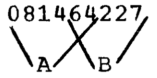
Figure 7. Main Storage Organization

COMMUNICATION REGION

The communication region is a 46-byte storage area within the Supervisor region for use by the supervisor and problem programs. Certain macros are available to allow access to the information contained in this region. Fields in the communication region are addressed relative to the first byte of the region.

The layout of the communication region in the Supervisor is shown in Figure 8.

9 bytes Calendar date, in unpacked decimal format. For example:



The first six bytes (A) are in the form: month, day, year. This is convenient for dating reports. The last five bytes (B) are the year and the day of the year, used by the label routines and also available for dating reports.

1 byte System configuration.

2 bytes Address of the end of the Supervisor.

8 bytes User area (for inter- or intra-program communications). These bytes are not changed by Job Control, unless Assemble and Execute or Compile and Execute or Load and Go is used.

3 bytes User area (for intraprogram communication). These three bytes are reset by Job Control.

1 byte UPSI (user program switch indicators). This byte is reset by Job Control.

6 bytes Program name. The first six bytes of the name on the JOB card.

10 bytes Area used by the Supervisor to retain information applicable to the problem program being executed (addresses of user-supplied routines specified by the STXIT macro).

6 bytes Phase name, the name of the last phase requested. This information is used by the Supervisor.

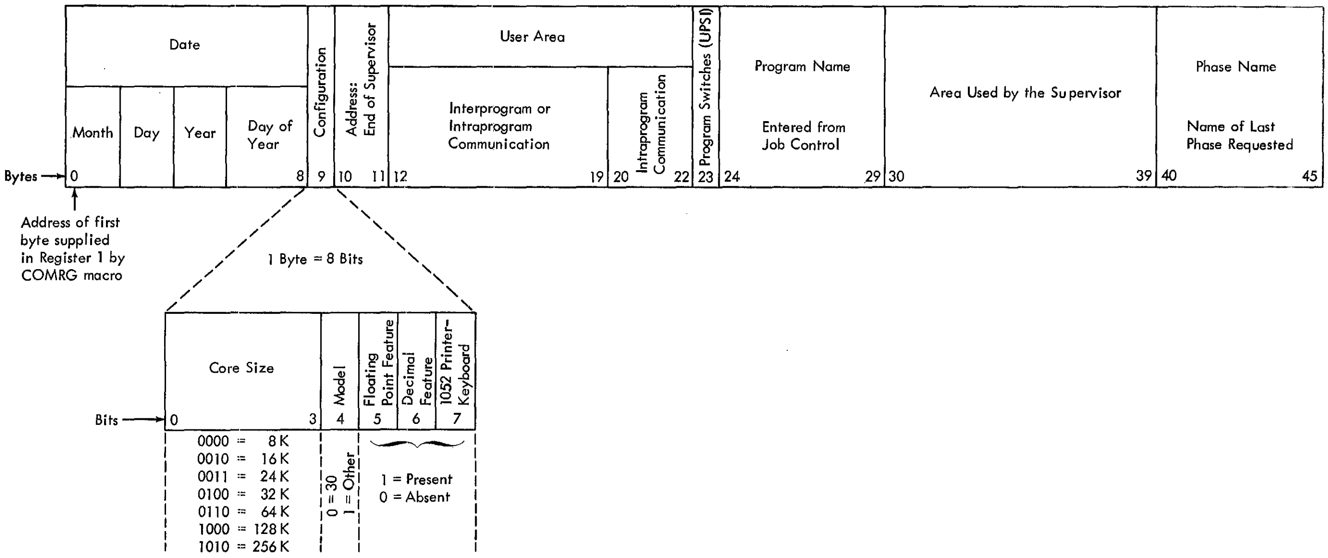


Figure 8. Communication Region (in Supervisor)

During job-to-job transition, Job Control deactivates any user-supplied program check routines. Job Control sets the program check option to dump and abort. The program name is updated by Job Control. The 8-byte user area (for interprogram or intraprogram communications) is not changed by Job Control unless assemble or compile and execute, or load-and-go modes of operation are used. Job Control always resets the 3-byte user area (for intraprogram communication), and the UPSI byte.

Communication Region Macros

Macros are provided to allow the problem program to communicate with the Supervisor and the communication region. A brief discussion of them follows. Details are found in the Assembler with Input/Output Macros publication, listed in the Preface of this manual.

COMRG Communication region. (COMRG) allows the problem program to address information stored in the communication region (obtain date, test switches, etc.). The address of the first byte of the region is placed in general register 1.

STXIT Set exit. (STXIT n,pc,it,oc) Activates problem program routines for program check, internal timer, and operator communication. The operands are:

n The number of a general register that can be used to provide linkage from one routine to another. (Register 0 should not be specified in this macro. Registers 12 and 13 also should not be specified in this macro unless SUPVR SAVEREG=YES was specified when the supervisor was assembled. See Appendix J.)

pc The name of the first instruction of the program check routine, or ABORT or DUMP.

it The name of the first instruction of the interval timer routine or CLOSE.

oc The name of the first instruction of the operator communication routine or CLOSE.

The value, or address, of the names included is placed in the proper places in the communication region.

MVCOM Move to communication region. (MVCOM a,b,c) allows the problem program to modify information in the user portion of the communication region (bytes 12 to 23). The following must be specified in the operand of the macro statement:

a. The relative address of the first byte to be modified in the region,
b. The number of bytes to be modified, and
c. The address or register that will contain the address of the first byte to be moved into the region.

INTERRUPTION HANDLING

An interruption is an automatic transfer of control from any storage location to a predetermined storage location. It can be caused by either a program instruction or a machine condition. The Supervisor automatically handles all interruptions so that the programmer need not be directly concerned with them. In most cases after an interruption is handled, control is returned to the point of interruption as if no break had occurred in the instruction sequence. Five kinds of interruptions are:

1. Supervisor call
2. External (timer or interrupt key)
3. Program check
4. Machine check
5. Input/output.

Figure 9 illustrates the flow of control between the Supervisor and a problem program during an interruption. Control is in the problem program initially. An interruption occurs, transferring control to the Supervisor. The status of the program is saved in the Program Old PSW. Depending on the type and reason for the interruption, control is given to an appropriate handling routine. Upon completion of the routine, the program is restored to its original condition (via the old PSW). Control is normally given back to the problem program at the point where it was interrupted.

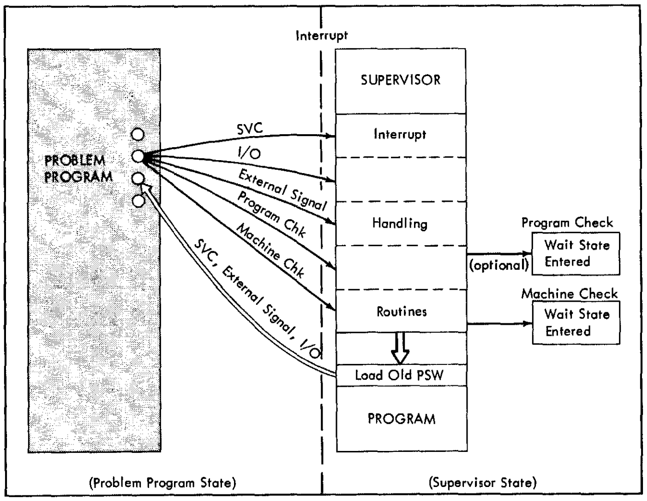


Figure 9. Flow of Control between Supervisor and Problem Program during an Interruption.

Supervisor Call

The supervisor call interruption is caused when the SVC instruction is executed. Certain macros use registers 0 and 1, and the SVC (supervisor call) instruction that is available to provide communication between the problem program and the Supervisor. The SVC in each macro has a certain interruption code which indicates to the Supervisor which routine is to be executed. If the SVC code is not recognized by the Supervisor, it forces a program check condition. The macros that allow problem programs to have access to supervisor functions via an SVC instruction are:

FETCH To load a program or program phase from the core image library into storage for execution.

MSG (Message) To provide for communication between the operator and the problem program.

EXCP (Execute channel program) To request an I/O operation to be performed when using physical IOCS.

EXIT To return to the user's main program after the user's handling of a timer interruption or operator-initiated communication.

EOJ (End of Job) To call Job Control to prepare the next job to be run.

Each macro generates a supervisor call interrupt with a specific code. The interrupt routine analyzes the code and gives control to another routine for the actual handling of the interruption.

External Interruption

An external interruption can be caused by the timer going from a positive to a negative value or by the operator pressing the interrupt key. (Note that External Signal interruptions, part of the Direct Control feature, are not recognized by the routine. Control is passed directly back to the problem program at the point of interruption.)

Timer When the interval timer has caused an interruption, control is given to a user-supplied timer routine to handle. If there is no timer routine, the interruption is ignored. (The timer routine is activated by the STXIT macro). When a Supervisor is assembled, the SUPVR macro must contain the parameter TR=YES, if an interval-timer routine is to be included in problem programs, using the Supervisor.

The program status of the interrupted mainline program is saved by the Supervisor. Also, the contents of registers 10 and 11 are saved so that they are available to the user's timer routine. At the completion of the timer routine, issuing the EXIT macro (with TR in the operand) causes a return to the problem program (registers 10 and 11 are restored). The EXIT macro must be the last instruction in the timer routine. It transfers control to the Supervisor, which then branches to the point of interruption. Figure 10 illustrates the sequence of events following the interruption.

Note that the programmer must be aware that a timer interruption occurring during execution of the problem program's interval-timer routine can cause an endless loop to develop within the timer routine. A sufficient time interval can be set in the timer to ensure that the timer does not cause an interruption or the programmer could code the timer routine to test for and handle any timer interruption occurring within the timer routine.

Job Control does not deactivate an established

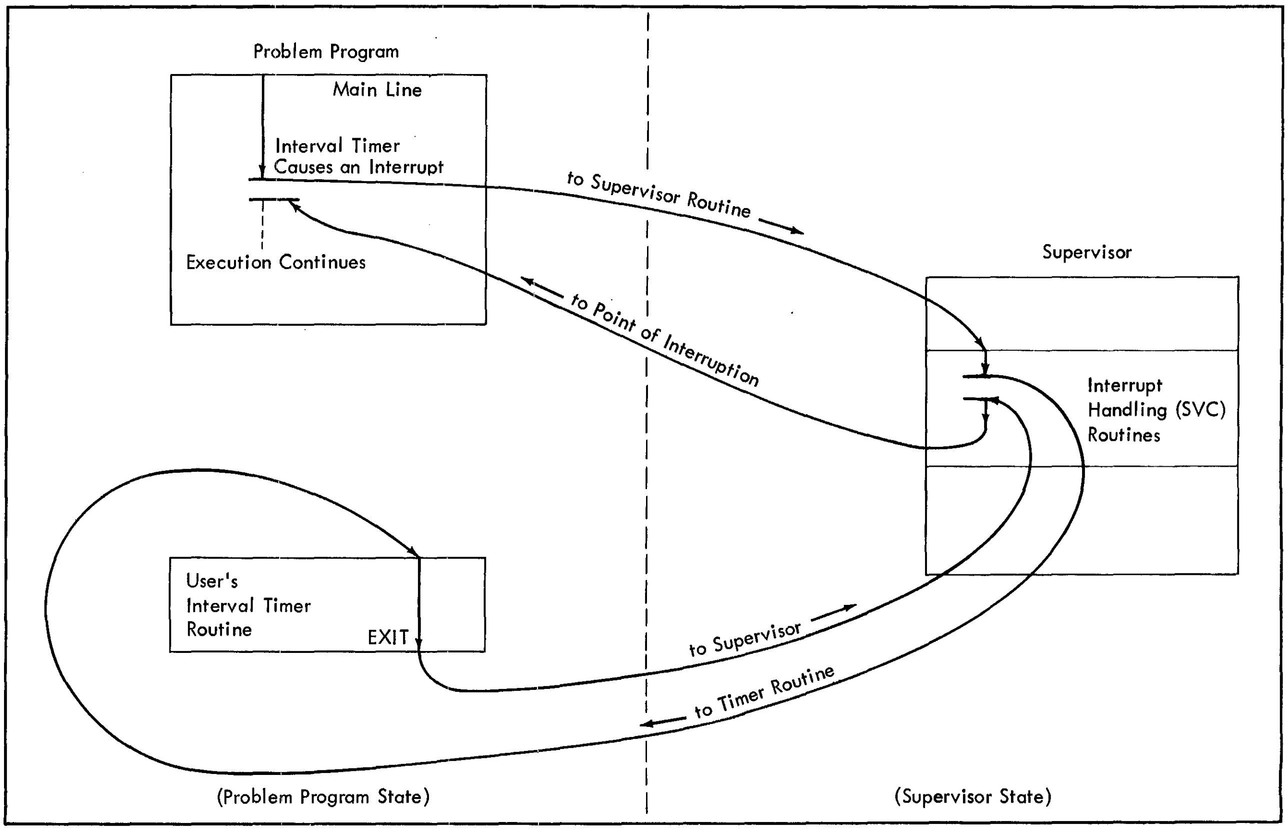


Figure 10. Sequence of Events after a Timer Interrupt

timer routine; therefore, one timer routine can be used with several jobs in a stacked-job stream.

Interrupt Key This key is used for communication from the operator to the Supervisor. It can indicate:

1. A reply to a message issued to the operator.

2. A message issued to the Supervisor by the operator. This includes indicating the completion of a desired operator action.

Program Check

Each program can select (with the STXIT macro) which of the following options is to be taken in the event of a program check:

1. Abort - The job being executed is terminated and the operator is informed of the cause of the termination.

2. Dump and Abort - The contents of the general registers, the Program Old PSW, the communication region, and entire problem program are printed, then the job is terminated.

3. Transfer to User Routine - The address of a subroutine supplied by the user can be placed in the communication region. The program check interrupt routine will branch to that subroutine when an interrupt occurs. The user subroutine can determine where the interrupt occurred by examining the address in the Program Check Old PSW.

Before starting each job, Job Control always selects the dump and abort option. This option will then be used unless the user changes the code in the communication region. The code can be changed at any point. For example, before performing a calculation where there is a possibility of a fixed-point overflow, the program might change the code to branch to a routine that inserts a maximum value anytime the interrupt occurs. The (STXIT) macro is provided to change the code.

Note: When a program-check interruption occurs during execution of a user's program-check routine, an endless loop may occur. Therefore, it is recommended that the user include code within his routine to prevent this occurrence, especially during the debugging stages of his programs.

Machine Check

A machine check interruption results from a machine malfunction. When the error is detected, certain diagnostic information is automatically placed in the diagnostic scan-out area beginning in location 128. The machine check interruption places the system in the wait state with an identifying code in the main storage address register on the console. The system can be restarted only through an IPL procedure. A card deck containing a diagnostic routine called the System Environment Recording, Edit, and Print program (SEREP) provides a diagnostic print out of the system.

A machine check interruption is simulated by the I/O device error routines by loading the machine check PSW if an error condition is detected that requires exceptional handling by the operator or an IBM Customer Engineer.

Input/Output Interruptions

An input/output interruption can be caused by:

1. I/O completion - End of transfer of data into or out of main storage or completion of a control operation.

2. I/O attention - Results from pressing the request key of the 1052.

When either of these conditions is detected, control transfers to the channel scheduler. (Note that a program-controlled interruption occurring as the result of the PCI flag in a CCW is ignored by the channel scheduler.)

CHANNEL SCHEDULER

Complete control of data input/output is accomplished through two kinds of routines: physical and logical. Supervisor input/output control consists of only physical routines (the actual transfer of data between main storage and some I/O device). These routines make up the Channel Scheduler. The functions it Channel Scheduler.

The functions it performs are:

1. Schedule I/O requests on each channel (queueing).
2. Start input/output operations.
3. Handle I/O interruptions (normal completion of data transfer, error detection, end-of-file detection, attention on the 1052).

All I/O devices in the System/360 are attached to channels rather than attached directly to the CPU. A channel provides a path for data transfer between main storage and the I/O device and in this system allows I/O operations to be overlapped with CPU processing. That is, instructions can be executed simultaneously with data movement in the channel or in several channels. For instance, at a given point in time, one channel may be reading data from a disk, another channel may be writing data on a printer, and a previously read record may be being processed. This is referred to as read/write/compute overlap.

Two kinds of channels are provided in this system: the selector channel and the multiplexor channel. The selector channel allows all I/O operations for all devices in this channel to be overlapped with CPU processing. On the multiplexor channel, tape and disk I/O operations cannot be overlapped with CPU processing. Card and printer I/O devices can be overlapped with CPU processing. Thus, greater throughput can be achieved if high-speed devices (tape and disk) are attached to a selector channel, and low-speed devices (card and printer) are attached to the multiplexor channel.

Overlapping I/O operations with CPU processing is inherent in the design of the machine and the channel scheduler program. However, achieving maximum overlapping is also partially dependent on the problem program. For instance, if overlap is desired in tape or disk operations, the problem program should provide for two I/O areas (or buffers). This allows data to be read into, or written from, one I/O area while records are being processed in the other area. Certain devices, however, have buffers built into the device (1403) and require only one I/O area in main storage to achieve overlap. The use of multiple I/O areas and separate work areas is discussed more fully under Processing Overlapped with Input/Output in Part 4.

All requests for I/O operations are handled by the channel scheduler. When a request is received and the affected channel and device are not busy, the requested operation starts and control passes back to the problem program. If the channel or device is busy, the request is placed at the end of a list (or queue) if I/O requests and the operation is performed as soon as all previous requests have been handled. Each selector channel has a single queue for all of its attached I/O devices. I/O device requests are scheduled on a first-in first-out basis per channel. The multiplexor channel has 12 request queues, if the 1285 optical reader is specified at assembly time. In this case, I/O operations can be interleaved, if no burst mode devices are assigned to the multiplexor channel. If the 1285 optical reader is not specified, the multiplexor channel is treated as a selector channel.

The channel scheduler also handles all I/O interrupts. If the interrupt indicates the normal end of an I/O operation (channel end and no errors) the channel scheduler examines the queue for the affected channel. If the queue is empty, control is returned to the problem program at the point of interruption. If instead, a request is pending, the channel scheduler starts the I/O operation and then returns to the problem program. However, if the request is for the same device, but device end has not been received, any pending I/O operation for that queue can not be started at this time, and the channel scheduler will return to the problem program. For example, for a 1403, channel end is received as soon as the buffer has been completely loaded, but device end is not received until completion of the print operation. (See Note for handling the channel end, device end condition when using BSC support).

The channel scheduler detects the following specific status conditions:

1. Wrong Length Record (WLR)
2. End of File (EOF)
3. Error

Upon detection of the WLR condition, the channel scheduler sets an indicator on and supplies the residual count to the problem program. The WLR condition is handled separately for BSC applications. (See Note.) At this point, the channel scheduler will examine the queue.

Upon detection of an EOF condition, the channel scheduler sets an indicator on for the problem program and, as above, examines the queue. The EOF condition is handled separately for BSC applications. (See Note.)

If an error is detected, all interrupts are masked and the channel scheduler passes control to the appropriate device error recovery routine. The channel scheduler remains in this condition until the error is resolved.

For the 1052, an I/O operation can be initiated by the operator. To do this, the operator presses the request key on the device. When the channel scheduler detects an Attention status condition, it passes control to the appropriate user routine for the 1052, if one is present. If the user has not supplied a routine, the interrupt is ignored. In the user routine, unrestricted reading and writing data to the device is allowed.

Note: For BSC I/O operations, upon detection of normal completion (channel end and device end), WLR or EOF, the channel scheduler goes to device error routines for further message format or line control response checking.

A problem program can perform I/O operations in two ways:

1. The problem program can issue physical I/O macros directly.

2. The problem program can use logical IOCS, which in turn issues the physical I/O macros. See Logical IOCS in Part 4.

Physical I/O Macros

The physical I/O macros are:

1. EXCP (execute channel program) This macro communicates directly with the channel scheduler to request that an I/O operation be started. When the EXCP macro is used, the problem program must supply the appropriate CCWs .

2. WAIT This macro suspends program operation until an I/O operation (referenced in the WAIT macro) is complete. The problem program must use this macro (or WAITM) at the point where processing cannot proceed until the I/O operation is complete. For instance, a problem program may issue the EXCP macro to read a disk block. At the point where the program needs the disk record for processing, a WAIT macro must be issued. The instructions generated from this macro simply loop, testing a program switch to see if the operation has been completed. The completion of the operation causes an I/O interrupt to the channel scheduler. Before control is returned to the loop, the switch is set to show the completion. Thus, the next time it is tested, the loop is broken and processing continues.

   For a discussion of the WAITM macro instruction, see Processing with STR Devices.

3. CCB (Command Control Block) This declarative macro generates a command control block for each list of CCWs to be executed. The command control block contains information required by the Channel Scheduler to execute the EXCP and WAIT macros. The block is also used to pass information between the problem program and the Channel Scheduler, such as status of the operation, action to be taken in the event of an error, etc.

4. CHNG (Change Channel) This macro is used to change the channel assignment for tape units that are attached to two selector channels through a tape control unit with simultaneous read-while-write capability.

A complete description of these macros is supplied in the Assembler with Input/Output Macros publication, listed in the Preface of this manual.

DEVICE ERROR RECOVERY

Each I/O device or class of I/O devices has a unique device error recovery routine. The appropriate routine is entered from the channel scheduler upon detection of an error. All of these routines have one function in common. That is, an attempt is made to recover from the error. This may be by programming (re-reading tape) or by operator action (2540 Not Ready).

If recovery is not possible, the following choices are available where applicable.

1. The record in error can be bypassed.

2. An error on an input record can be ignored.

3. The problem program can take action (an exit to a user routine is allowed).

4. Follow the machine-check procedure (terminate the job).

Depending on the type of device and on whether logical IOCS is used, some or all of the above options are available. In the absence of any other options, only choice 4 is available.

SYSTEM/OPERATOR COMMUNICATION

Communication between the IBM System/360 and the machine operator is of two types:

1. Communication from the system to the operator, and

2. Communication to the system, initiated by the operator.

Communication from the system to the operator is required for efficient operation and control of the system. Coded messages are either displayed on the system control panel (console) or printed on an IBM 1052 Printer-Keyboard, if one is available (and assigned to SYSLOG).

Communication initiated by the operator may be used to process inquiries and also for system control. The operator can communicate only to the Supervisor via the system control panel switches and keys. He can communicate to either the Supervisor or the problem program when a 1052 printer-keyboard is available (and assigned to SYSLOG).

In some cases, communication between the system and the operator may use a printer. A printer can give instructions and messages to the operator.

Communications from the System to the Operator

Coded Messages (MSG Macro): Communication to the operator from IBM-supplied programs and the user's problem programs can be made through use of a program instruction known as the MSG (Message) macro. The MSG macro places coded messages in positions 0-3 of main storage. These bytes can be displayed on the system control panel (console) by the operator. When an IBM 1052 Printer-Keyboard is available and assigned to SYSLOG, the coded message from the MSG macro will be printed on the 1052 printer. These messages may be a numerical code, or some set of meaningful characters, for example "ISEQ" for input sequence error. IBM-supplied programs usually use an alphameric code. The first character of the message identifies its source, as follows:

Identifying Code

Printed on 1052 Printer-Keyboard	Displayed on Console from Byte 0	Message Issued by
0	11110000	IPL or Supervisor.
1	11110001	Job Control.
2	11110010	Linkage Editor.
3	11110011	Other IBM Programs.
4	11110100	Logical IOCS.

Since these codes apply to specific IBM-supplied programs, a message issued by the problem program should not have a 0, 1, 2, 3, or 4 as the first character.

Some messages require a reply from the operator to continue processing. When a reply is required, the letter "A" is the fifth character of the message and it is printed on the 1052 following the four-character message. If a 1052 printer-keyboard is not available, the letter "A" can be displayed on the console from byte 4 of main storage. When a reply is not required, the fifth character of the message is a blank. A reply to a message is always a single character; any character may be a valid reply when properly defined by the problem program.

Valid reply codes via 1052 and System Control Panel (Byte 5-Hexadecimal) are described as follows:

Via 1052	Via System Control Panel	Meaning
blank	40	The request key was pressed in error; continue processing.
0	00 or F0	Terminate the job.
1	01 or F1	Dump the program and terminate the job.
2	02 or F2	Turn on program switch 7 in the UPSI byte of the communication region. Return to the program that initiated the message.
3	03 or F3	Turn off program switch 7 in the UPSI byte of the communication region. Return to the program that initiated the message.
4*	04	Ignore the indicated I/O error. Continue processing if physical IOCS issued the message. Otherwise return to the program that initiated the message.
5	05	Retry the indicated I/O operation. Continue processing if physical IOCS initiated the message. Otherwise return to the program that initiated the message.
6*	06	Disable STR lines (when using DTFSN), DUMP the program and terminate the job.
8**	08	Disable the BSC line (when using DTFBS), print out the error statistics and transmission counts, dump the program and terminate the job.
Other	Other	Return to the program.

*If STR routines are included in the assembled supervisor, a reply of 4 will set on the lost data and end-of-file bits in the expanded STR CCB. If STR devices are being used in the problem program, a reply of 6 must be used to ensure that the STR lines are properly disabled when the job is terminated. If the STR routines are not included in the assembled supervisor, code 6 is invalid and will be ignored.

**If BSC support is being used in the problem program a reply of 8 must be used to ensure that the BSC line is properly disabled when the job is terminated. If BSC routines are not included in the assembled supervisor, code 8 is invalid and will be ignored.

All reply codes are communicated to the program that initiated the message and (except for reply codes 0 and 1) can be tested by the program by addressing "name +7". "name" is the symbolic name used for the MSG macro instruction.

Operator Reply via the 1052 Printer-Keyboard: When a message to the operator is received via the 1052, the proceed light will be turned on if a reply is required, and the system will enter the wait state until a reply is sent. If no reply is required, the message will be printed and processing will continue uninterrupted. If an input/output error occurs on the 1052 during receipt of a message, the system will enter the wait state (wait light turns on) and the operator must use the system control panel to display the messages.

When a message requiring a reply is received, the operator:

1. Types the appropriate reply from a prepared list of messages and required responses.

2. Types an end-of-block character (holds the alternate-code key down and types a 5).

The system then resumes processing.

If an input/output error occurs during the operator response, the system will enter the wait state (wait light turns on). The operator must use the system control panel to reply to the message.

Operator Reply via the System Control Panel: When a message to the operator is issued, and a 1052 is not available, the system enters the wait state (wait light turns on) and the operator:

1. Presses the stop key.

2. Displays bytes 0-4 of main storage and determines whether a response is required (an "A" in byte 4). He then stores the appropriate reply (if required) in byte 5 and checks it in the console lights.

3. Presses the start and interrupt keys.

The system will resume processing.

Other Messages: Some messages may be received from the problem program by means other than the MSG macro. These messages may be in sentence form, giving instructions to the operator or information about the status of the system or program. These messages may appear on the 1052 or on the 1403 or 1443 printers of the system. Some examples are:

```
PHASE A COMPLETED
I/O DATA CHECK ERROR
UNIT UNAVAIL
```

Communication Initiated by the Operator

Communication initiated by the operator is of two types: to the Supervisor, and to the problem program. If a 1052 printer-keyboard is available, communication is possible to both the Supervisor and the problem program. Without a 1052, communication is possible only with the Supervisor.

Communication to the Supervisor via the 1052 Printer-Keyboard: When a 1052 printer-keyboard is available and assigned to SYSLOG, the operator can initiate a communication to the Supervisor by:

1. Pressing the request key. The system will acknowledge the request with a message on the 1052.

2. Typing the appropriate one-character code, 0-3, 6, or 8 (see chart), when the proceed light turns on. The system will analyze the code and continue processing accordingly.

Communication to the Supervisor via the System Control Panel: The operator initiates a communication by:

1. Pressing the stop key.

2. Storing the appropriate code (see chart) in byte 5 of main storage.

3. Pressing the start and interrupt keys.

The system will analyze the code and continue processing accordingly.

The following codes are used in IBM-supplied Supervisors.

Code Entered		
Via 1052	Via Console (Byte 5) Hexadecimal	Meaning
0	00 or F0	Terminate the job.
1	01 or F1	Dump the program and terminate the job.
2	02 or F2	Set UPSI bit 7 on and continue processing.
3	03 or F3	Turn UPSI bit 7 off and continue processing.
6	06 or F6	Disable STR lines (when using DTFSN), dump the program and terminate the job.
8	08 or F8	Disable BSC line (when using DTFBS), print out the error statistics and transmission counts, dump the program and terminate the job.

Any other code entered will be ignored, unless CR=YES has been specified in the Supervisor assembly and the communication routine has been activated by the STXIT macro. In this case, the code will be interpreted as one being used by the problem program. If the STR routines are not included in the assembled supervisor, code 6 is invalid and will be ignored. The same is true of code 8 if BSC routines are not included in the assembled supervisor.

Communication to the Problem Program: Three conditions must be met for the operator to communicate with the problem program:

1. The 1052 printer-keyboard must be available and assigned to SYSLOG.

2. The parameter CR=YES must have been specified in the SUPVR macro statement when the Supervisor was assembled.

3. A user-written routine to process the communication must be present in main storage, and its address must have been specified in a STXIT macro.

When these three conditions are present, the operator:

1. Presses the request key.

2. Receives a message acknowledging the request.

3. Types any code except 0 through 3, 6, or 8, when the proceed light turns on.

When the Supervisor recognizes that this is not one of the defined control program codes (0-3, 6, or 8), it performs the following steps.

1. Stores the I/O interruption old PSW.

2. Stores the contents of registers 10 and 11, making them available to the user's communication routine.

3. Transfers control to the user's operator communication routine. The user's routine may use the IOCS macro GET and PUT to control input/output operations, if registers 14-15 and 0-1 are saved and restored by the user. When the communication routine is being executed, no other operator-initiated communication are accepted. The MSG macro can still be used to request a response from the operator. Upon completion of the communication routine, an EXIT macro with the CR operand must be issued to cause normal processing to be resumed. The Supervisor restores registers 10 and 11 before returning to the mainline program.

Job Control does not deactivate an established operator communication routine; therefore, a communication routine can be used with several jobs in a stacked-job stream.

SYSTEM LOADER

The System Loader is a permanently main-storage resident routine in the Supervisor. It loads all programs and program phases run in a BOS environment with the exception of the main-storage resident Supervisor itself.

All programs are loaded into main storage from the core image library. (Programs can be read from cards, temporarily placed in the core image library and executed. See the section describing the Linkage Editor.) The system loader is entered through an SVC instruction (even when entered from some other routine of the Supervisor). The unique interruption code of this System Loader SVC results from several problem program macro instructions (OPEN, CLOSE, EOJ, CHKPT, DUMP, and FETCH). The FETCH macro explicitly names the phase to be loaded. The others have implied phase names supplied from the macro library during assembly of the problem program. Job Control issues the fetch SVC to retrieve the Linkage Editor, the first phase of each program, and the restart routine.

The System Loader examines the name of the requested program or phase. If the first three bytes contain the letters SYS, the transient directory is read and checked for a corresponding entry. If the first three characters are not SYS, the first four characters are compared to the name of the current program (in the communication region). If equal, the phase directory is read and checked for a corresponding entry. If neither of the above conditions is met, the core image directory is read and checked for the entry.

When the name is found in one of the directories, the System Loader reads the phase into main storage and transfers control to the indicated entry point.

FETCH Macro

In the source language, this macro has the format:

```
FETCH xxxxxx
```

The operand (xxxxxx) is the name of the program or phase to be loaded. The macro is assembled into an SVC instruction followed by the name defined as a constant. The supervisor call interrupt routine transfers control to the System Loader. Physical IOCS is used to perform the necessary disk operations. In this case, the I/O request is held until all pending requests are complete.

CHECKPOINT/RESTART

When a problem program is expected to run for an extended period of time, provision should be made for taking checkpoint records periodically during the run. The records contain the status of the job and system at the time the records are written. Thus, they provide a means of restarting at some point other than the beginning of the entire job, if processing must be terminated for any reason prior to the normal end of job. For example, a job of higher priority may require immediate processing, or some malfunction such as a power failure may occur and cause such an interruption.

If checkpoint records are written periodically, operation can be restarted using the last set of checkpoint records prior to the interruption. Therefore the records must contain everything needed to re-initialize the system when processing is restarted.

BOS includes routines to take checkpoint records and to restart a job at a given checkpoint after an interruption. The checkpoint and restart routines are included in the core image library when the basic operating system is initially generated. The routines are considered part of the Supervisor and are executed in the transient area. The checkpoint routine is called in response to a CHKPT macro in the problem program. The restart routine is called by Job Control when it reads a RSTRT card.

Checkpoint records can be written in a previously defined area of the resident disk pack or written on magnetic tape. When on disk, each successive set of checkpoint records is written directly over the last. Thus, there is never more than one set of checkpoint records on disk. When written on tape, however, each checkpoint is written following the previous checkpoint (or interspersed with data records, if written on a normal data-file tape) and each is uniquely identified. When restarting from tape, the RSTRT card specifies which checkpoint is to be loaded.

Note: Checkpoints must not be written on a normal data-file tape, if the tape is to be processed by IBM System/360 Operating System.

CHKPT Macro

There can be only one checkpoint macro in an assembly. It can be executed as often as desired. In the source language, the CHKPT macro has the format:

CHKPT n_1,label,SYSxxx,DISK

where:

- n_1 is the number of tapes to be repositioned by the restart routine.
- Label is the symbolic name of the instruction to which control is to be passed by the restart routine.
- SYSxxx is the symbolic name of the unit on which the checkpoint is to be taken (SYSRES if disk, SYS000-SYS254 if tape).
- DISK indicates that the checkpoint records are to be written by a problem program running in a BOS environment in the resident disk pack. This parameter is left out if they are to be written by a problem program running as an independent program, not in a BOS environment.

The restart facility is described in the section on Job Control. See the Assembler with Input/Output Macros publication listed in the Preface of this manual, for more detail on the checkpoint macro.

LABEL CHECKING

All label checking (both disk and tape labels) is performed by transient routines of the Supervisor. These routines are called into the transient area and executed in response to macro instructions issued by the problem program. Since these routines do not occupy space that can be used by the problem program, there is no need to handle OPEN and CLOSE operations as overlays in special phases.

The label processing routines are called in response to OPEN and CLOSE macro instructions in the user's program. These macros cause a branch to either the logical IOCS routines (DTFSR, DTFDA, or DTFIS) or to the special, physical IOCS routine (DTFPH) that is assembled as part of the problem program. The DTF routine issues the actual FETCH for the label-processing routine.

NORMAL AND ABNORMAL END-OF-JOB HANDLING

When a program reaches the normal end of job, issuing the EOJ macro causes the Supervisor to fetch Job Control to begin processing the control cards for the next job.

In a BSC environment the ERRPT macro (see Binary Synchronous Communication) must be issued preceding the EOJ macro.

A special routine is provided as part of the Supervisor to provide a print-out of main storage in the event of some abnormal end-of-job situation. This routine is fetched into the transient area in response to the DUMP macro instruction. The DUMP routine prints the contents of the fixed-assignment locations (0-127), the general purpose registers, and the entire problem program area.

JOB CONTROL

The contents of this section are shown in Figure 11. The Job Control program provides job-to-job transition within BOS. It is called into main storage to prepare each job to be run. It performs its functions between jobs and is not present while a problem program is being executed. Job Control is called by:

1. The IPL Loader, to process the first job after an IPL procedure.

2. The Supervisor, after an abort or dump operation.

3. By the problem program, at normal end-of-job.

A macro instruction, EOJ, is provided to make this call when programming in the Assembler language. This assembles into a FETCH for the Job Control program.

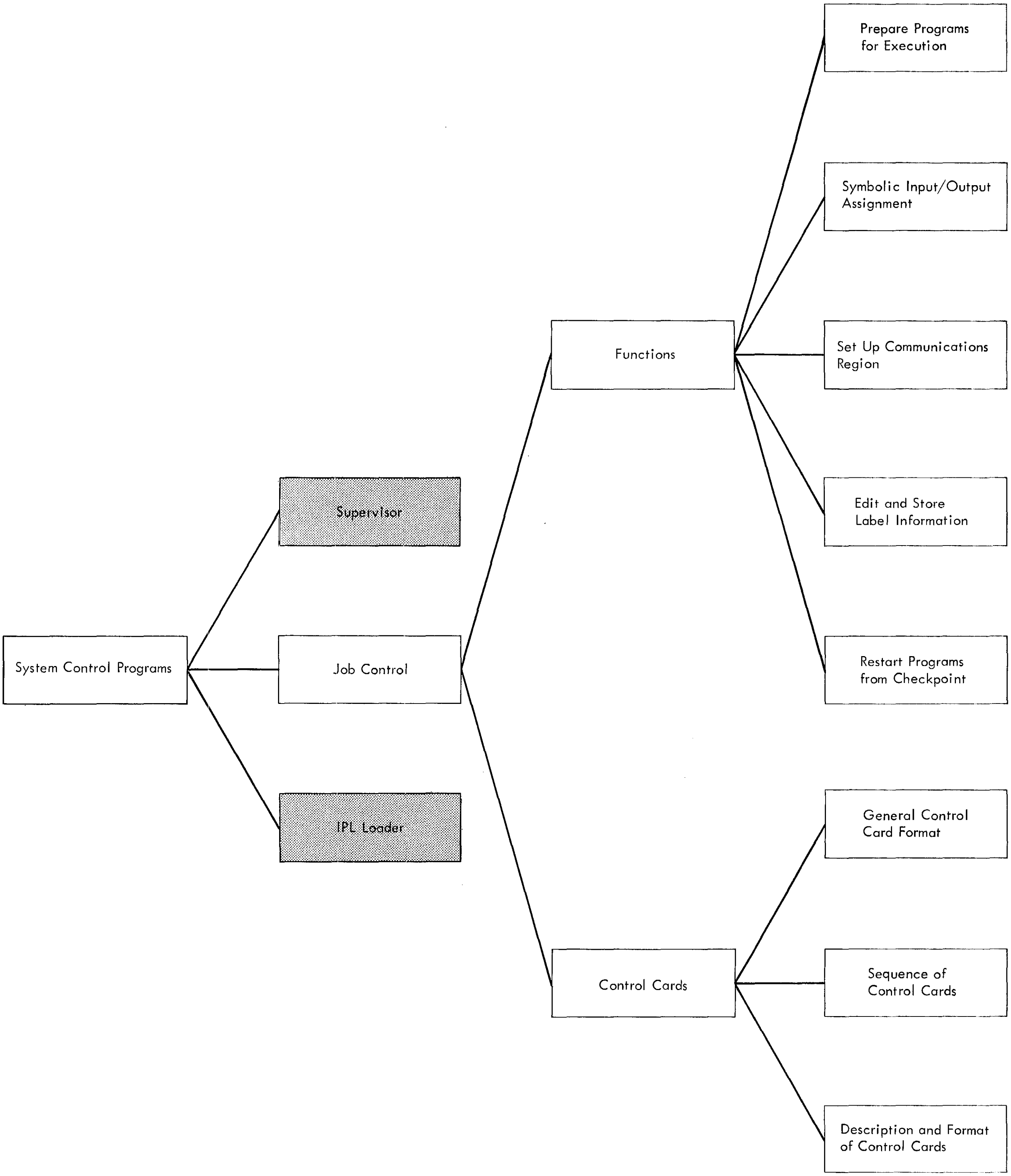

System Control Programs
Supervisor
Job Control
IPL Loader
Functions
Control Cards
Prepare Programs for Execution
Symbolic Input/Output Assignment
Set Up Communications Region
Edit and Store Label Information
Restart Programs from Checkpoint
General Control Card Format
Sequence of Control Cards
Description and Format of Control Cards

Figure 11. System Control Program: Job Control

FUNCTIONS

Job Control performs various functions on the basis of information provided in job control cards. These functions, and the cards used to provide the necessary information are:

1. Prepare programs for execution.

Card Identification	Information Provided
// JOB	Program name. If a program is to be run under control of Autotest, an additional parameter (T) is punched. If a program is to be assembled and run as a single job, the card first names the Assembler, then the problem program.

Card Identification	Information Provided
// EXEC	Source of program to be run

2. Assign device addresses to symbolic names.

Card Identification	Information Provided
// ASSGN	Actual device addresses

3. Set up fields in the communication region.

Card Identification	Information Provided
// JOB	Program name
// DATE	Date
// UPSI	User program switch indicators
// CONFG	Machine configuration

4. Edit and store volume and file label information.

Card Identification	Information Provided
// VOL	Volume label information
// DLAB	Disk file-label information
// XTENT	Lower and upper limits of disk file area
// TLAB	Tape file-label information

5. Prepare for restarting of checkpointed programs.

Card Identification	Information Provided
// RSTRT	Location of checkpoint records
// FILES	Tape drives to be positioned

6. Place programs under Autotest control.

Card Identification	Information Provided
// JOB	Indication of a program to be tested
// ATEOF	Indication of the end of a group of programs involved in a single test

Three additional cards are used to provide explicit commands to Job Control. These are:

Card Identification	Meaning of Command
// LOG	Begin listing job control cards
// NOLOG	Stop listing job control cards
// PAUSE	Place system in wait state

PREPARE PROGRAMS FOR EXECUTION

All programs run in the basic operating system are loaded from the core image library for execution. If a program has been previously cataloged (see Librarian) as a permanent entry in the core image library, Job Control has only to construct a phase directory of that program and then transfer to the system loader to load it for execution. On the other hand, if a program is to be placed temporarily in the core image library for execution, Job Control first calls the Linkage Editor to prepare the program in the library. The Linkage Editor reads the program from either or both the relocatable library and card (or tape, in card image). After performing its task, the Linkage Editor recalls Job Control to construct the phase directory and fetch the first phase.

The phase directory includes an entry for each program phase in the core image library whose name has the same first four characters as the name in the JOB card. The use of this directory is discussed under System Loader in the section on the Supervisor.

SYMBOLIC INPUT/OUTPUT ASSIGNMENT

Job Control is responsible for assigning input/output device addresses. Programs do not reference I/O devices by their actual physical addresses, but rather by a symbolic name. The physical addresses are assigned to the symbolic names at job execution time by Job Control. The ability to reference an I/O device by a symbolic name rather than a physical address provides advantages to both programmers and machine operators. The symbolic name of a device is chosen by the programmer. He can write a program that is dependent only on the device type and not on the actual device address. At execution time, the operator determines the actual physical device to be assigned to a given symbolic name. He communicates this to Job Control by a control card (ASSGN). Job Control associates the actual physical device with the symbolic name by which the programmer references it.

The assignment of the system resident unit (SYSRES) is determined during the IPL procedure. The operator specifies the device address with the load unit switches. At the completion of the IPL procedure, this address is in the IPL PSW. The IPL Loader places this address in the proper location in the device table (Figure 12). SYSRES <u>cannot</u> be assigned by an ASSGN card.

A fixed set of symbolic names (symbolic units) is used to reference I/O devices. No other names can be used. They are:

SYSRES	System residence disk drive
SYSRDR	Card reader used for Job Control cards
SYSLST	Printer for output listings from system programs
SYSIPT	Card reader or magnetic tape unit used as the input unit for system programs
SYSOPT	Card punch used as the main output unit for system service programs
SYSLOG	Printer or printer-keyboard used to log Job Control cards
SYS000-SYS254	All other units in system.

The first six of the above names are used by the system control programs and service programs. SYSRDR and SYSIPT can both refer to the same device. Any additional devices in the system are referred to by names ranging from SYS000 to SYS254.

Physical Unit Block (PUB)

At the time the Supervisor is assembled, a device table is set up with an entry for each of the symbolic units that will ever be used in the system. Each entry is called a <u>physical unit block</u> (PUB). The format of the device table is shown in Figure 12. The length of the table depends on the number of devices specified in the SYMUN (symbolic unit) macro. The first six PUB's are always present. Each additional device specified, starting with SYS000, is entered in the next PUB.

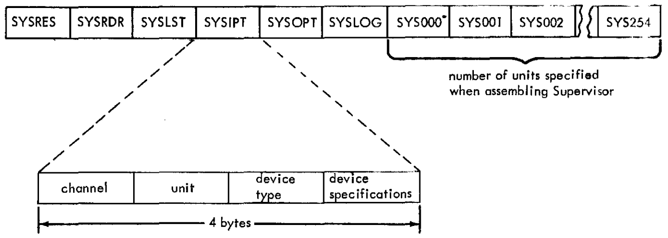


Figure 12. Sequence and Format of PUBs in Device Table

Normally, each symbolic unit specified is assigned a physical device address at the time the Superivsor is assembled. In some cases, a single device may be assigned to two or more symbolic units. The extra main storage required (for bytes for each additional PUB) may be justified by eliminating the necessity of using the ASSGN cards except in exceptional cases. An installation can make specific assignments at system generation time and establish these as conventions to be followed by all programmers. By following the conventions, most job decks can be submitted for execution with no ASSGN cards. When one or more unit assignments must be changed for a job, this can be flagged as an exceptional situation in the setup instructions to the operator. When the exceptional job is completed, the operator can re-establish the conventional assignments for the next job. Figure 13, for example, shows a typical system configuration. <u>The following conventions might be established for the installation's own programs</u>. Note that some of the assignments might have to be changed for certain of the IBM-supplied programs.

1. Card input is always read from SYSRDR. This is one symbolic unit that is never assigned to any other unit. This same device is assigned to SYSIPT because most of the system programs (language translators, Linkage Editor, etc) read from SYSIPT.

2. Card output is always punched on SYSOPT.

3. Printed output is always on SYS000. By using SYS000 instead of SYSLST, programs can operate when SYSLST is unassigned. This might be necessary when special forms are being used in the printer. By using the "unassign" option in the ASSGN card, the control programs can be prevented from printing operator messages. (When SYSLST is unassigned, messages will be printed on the 1052.)

4. All programs address the 1052 as SYSLOG. Since the control card logging option is specified in the LOG and NOLOG control cards, there is no reason to unassign this device.

5. The two disk drives are addressed as SYS001 and SYS002 when referring to data files. When specifically addressing the system residence pack, use SYSRES.

6. The two tape drives are addressed as SYS003 and SYS004.

The initial device assignments present after each IPL procedure are those made when assembling the Supervisor or those established by the device-table service program (PSERV). Once the Supervisor is loaded into main storage, however, any reassignments made are carried from job to job.

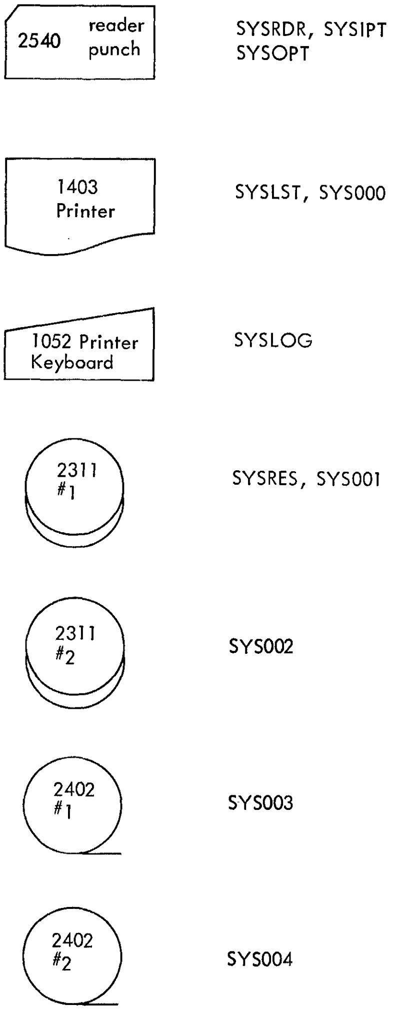


Figure 13. Example of Symbolic Device Assignment

SET UP COMMUNICATION REGION

Job Control takes the following information from control cards and places it in the communication region.

1. Program Name Taken from JOB card. This field is used by the System Loader in program retrieval.

2. Date Taken from the DATE card. This field is used by the OPEN routine for label checking. It can be used by the user's problem program to date output reports.

3. User Program Switch Indicators Taken from the UPSI card. The bit pattern in this byte can be used as switch indicators to specify program options.

4. Machine Configuration Taken from the CONFG card. This information can be used by programs that are written to run on various machine configurations, modifying themselves to take advantage of optional features, additional core storage, etc. The information contained in this byte is shown in Figure 8.

EDIT AND STORE LABEL INFORMATION

All volume and file label processing is done during problem program execution. However, label information to be checked against is read from cards by Job Control and stored in the resident pack for subsequent processing. This eliminates the problem of inserting label information cards within program decks or within a card input file. The formats of the label information cards are discussed in this section. See Disk and Tape Labels in the section on Data Management for a complete discussion of volume and file labels.

RESTARTING PROGRAMS FROM CHECKPOINT

Job Control prepares the system for restarting from a checkpoint. It reads the job control cards for the job, repositions tape drives, reassigns I/O device addresses, reinitializes the communication region, reconstructs the phase directory, stores the information from the RSTRT card, and issues a FETCH for the restart program. The restart program handles the actual restarting of the problem program. If the FILES card is included, Job Control will reposition any magnetic tape drives to a specified tape mark. Note that while tape positioning is normally a restart procedure, it can also be done when initially starting a program. Job Control does not provide any label checking on the tape files to be positioned.

JOB CONTROL CARDS

GENERAL CONTROL CARD FORMAT

Certain rules must be followed when filling out control cards. All job control cards conform to these rules.

1. Identifier Two slashes identify the card as a control card (//). They must be in columns 1 and 2. At least one blank immediately follows the second slash.

2. Operation This describes the type of control card (the operation to be performed). It can be up to five characters long. At least one blank follows its last character.

3. Operand This may be blank or may contain one or more entries separated by commas. The last term must be followed by a blank.

All control cards are essentially free form. Information starts in column 1 and cannot extend past column 71. Exception: For the file label cards (TPLAB and DLAB) or the ASSGN card, information can be punched in more than one card. A character is punched in column 72 (any non-blank character). This specifies that information is continued on the following card (continuation card). Information on the continuation card begins in column 16; columns 1-15 must be blank. Certain control cards have operands that are specified as character strings within single quotes (8-5 punch). Hexadecimal constants are punched within quotes, and the first quote is preceded by the letter X.

Continuation cards are used only for the label cards and ASSGN cards (other cards can not be continued).

Job Control reads from the card reader identified by the symbolic unit SYSRDR. The following cards are recognized:

Operation	Meaning
JOB	Job
ASSGN	I/O assignment
FILES	Files
VOL	Volume Information
TPLAB	Tape file label information
DLAB	Disk file label information
XTENT	Disk file extent
DATE	Date
UPSI	User Program Switch Indicators
CONFG	Configuration
PAUSE	Pause
EXEC	Execute
RSTRT	Restart
LOG	Begin logging
NOLOG	Stop logging
ATEOF	End of Autotest job input

Any control card other than these is recognized as an error. A message is issued to the operator so that he can correct the card in error. Some of the errors recognized are:

1. Invalid symbolic unit name.
2. No space reserved in device table for a symbolic unit name.
3. Invalid device-type.
4. Invalid length of hexadecimal field.
5. Invalid hexadecimal character.
6. Invalid date.
7. More than one JOB card encountered prior to an EXEC card.
8. No date in communication region.
9. A label card (TPLAB or DLAB) does not immediately follow its associated volume (VOL) card.
10. An extent card (XTENT) does not immediately follow its associated disk label (DLAB) card.

All non-control cards appearing before a JOB card are automatically bypassed by Job Control.

SEQUENCE OF CONTROL CARDS

The job control cards for a specific job always begin with a JOB card and end with an EXEC card. The only limitation on the sequence of cards between JOB and EXEC is that discussed below for the label information cards. The following cards are placed between JOB and EXEC.

ASSGN
VOL
TPLAB
DLAB
XTENT
DATE
UPSI
CONFG
RSTRT
FILES

The label cards must be in the order:

```
VOL      or    VOL
TPLAB .        DLAB
               XTENT (one for each
                area of file in volume)
```

The PAUSE, LOG, and NOLOG cards can be placed before or after a JOB card. Any other control cards placed before a JOB card will be diagnosed as errors.

PAUSE When outside of the JOB-EXEC set, a message is immediately displayed and processing is suspended. When within the JOB-EXEC set, all control cards are processed, a message is displayed and processing is suspended just before FETCH is issued for the first phase of the problem program. In either case, to resume processing the operator must reply to the message. A PAUSE card should follow the EXEC card for the last job of a batch.

LOG All following job control cards are logged.

NOLOG Logging stops immediately.

The ATEOF card is placed after the EXEC card of the last job in a given test group.

DESCRIPTION AND FORMAT OF JOB CONTROL CARDS

JOB Card

This card indicates the beginning of control information for a job. It contains the name of the program to be run. It may appear in three formats, indicating one of the following modes of operation:

Single-Job Format

```
// JOB progname
```

A single job is to be executed. The source of the program (core image library, relocatable library, or SYSIPT) is indicated in the EXEC card.

progname The name of the first phase of the user's program. This is the same name used in the PHASE card discussed in the section on the Linkage Editor. The name is not restricted in length; however, only the first six characters are processed by Job Control. These six characters must be unique for each phase in the core image library. Successive phases of a multi-phase program are retrieved more quickly if the first four characters in each phase name are identical. Job Control constructs a phase directory consisting of an entry for each phase in the core image library whose name has the same first four characters as the name in the JOB card. A FETCH is then issued using the six-character name taken from the JOB card.

Compile or Assemble and Execute Format

```
// JOB ASSEMBLER, progname
// JOB RPG, progname
```

A source program is to be compiled or assembled, written out in the relocatable library, and then edited into the core image library and executed.

ASSEMBLER or RPG — The name of the language translator to be run. This is the main job to be run and this determines the information punched in the other control cards. The EXEC card has no operand. The I/O device assignments must be the same for the problem program as for the language translator.

progname — The name of the first phase of the user's source program. Same as in single-job format.

Execute Under Autotest Control Format

```
// JOB progname,T
```

A program is to be executed under control of Autotest. This card identifies the particular program to be tested with a test set. The job control cards for the program to be tested come between those for the Autotest program itself and before the ATEOF card.

progname — The name of the first phase of the program to be tested. Same as in single-job format.

The letter T identifies this as the program to be tested, as distinct from other programs run as part of the same test set (such as utilities) but not under direct Autotest control.

Examples of JOB Cards

```
// JOB PAYR01

// JOB INVENTORY (only INVENT is used)

// JOB ASSEMBLER,PAYR01 (A source
          program whose first phase is
          named PAYR01 is to be
          assembled and executed.  The
          Assembler is cataloged as
          ASSEMB.)

// JOB PAYR01,T (an object program
          whose first phase is named
          PAYR01 is to be executed
          under Autotest control.)
```

ASSGN Card

When programs are assembled, they use symbolic unit names to reference I/O devices. At execution time this card is used to assign a specific device address to the symbolic unit used. It contains the symbolic unit and various parameters to describe the physical device.
The format is:

```
// ASSGN SYSxxx,X'cuu',dd,X'ss'
```

SYSxxx — The symbolic unit. It may be one of the following:

SYSRDR	SYSLST
SYSIPT	SYSLOG
SYSOPT	SYS000-SYS254

X'cuu' — Channel and unit number (in hexadecimal).

c = 0 for a multiplexor channel
    1 for selector channel 1
    2 for selector channel 2
uu = 00 to FF (0 to 255 in hexadecimal)

dd — Device type. The following two character codes are valid:

C1	1052 Printer-Keyboard
D1	2311 Disk Storage Drive
L1	1403 or 1404 Printer
L2	1443 or 1445 Printer
P1	2540 Used as a card punch
P2	1442 Used as a card punch
P3	2520 Used as a card punch
R0	2671 Paper Tape Reader
R1	2540 Used as a card reader
R2	2540 Using Punch Feed Read feature
R3	1442 Used as a card reader and punch

R4	2501 Card Reader
R5	2520 Used as a card reader or card reader and punch for combined files
RR	1285 Optical Reader or 1287 Optical Reader (in journal tape mode)
RD	1287 Optical Reader (in document mode)
ST	2701 Data Adapter Unit with SDA I (STR)
T1	2400 seven-track tape
T2	2400 nine-track tape
BS	2701 Data Adapter Unit with SDA II (BSC)

X'ss' Device Specifications (in hexadecimal) required for seven-track tape, and optional for nine-track tape and for the 1403 printer with the Universal Character Set (UCS) special feature. Omit the field if it is not needed. The specifications for seven-track tape are:

	Bytes per		Trans-	Convert
ss	Inch	Parity	late	Feature
10	200	odd	off	on
20	200	even	off	off
28	200	even	on	off
30	200	odd	off	off
38	200	odd	on	off
50	556	odd	off	on
60	556	even	off	off
68	556	even	on	off
70	556	odd	off	off
78	556	odd	on	off
90	800	odd	off	on
A0	800	even	off	off
A8	800	even	on	off
B0	800	odd	off	off
B8	800	odd	on	off

The specifications for 9-track tape are:

ss	Bytes per Inch
C8	800
C0	1600

If this operand is omitted, a density of 800 bytes/inch will be assumed.

For a 1403 printer with the UCS feature:

Printer Specifications

ss	Meaning
73	Ignore data check
7B	Accept data check

Note: To include printer specifications 73 or 7B in the ASSGN card, the 2821 printer control unit must have been updated through Engineering Change Number 125632 or Request for Engineering Action Number 0100037.

The device specifications byte is also used internally as a queue pointer for the channel scheduler. Users should not utilize this byte for any purpose other than device specifications.

When Job Control encounters an ASSGN card, it:

1. Verifies that a physical unit block (PUB) was reserved in the device table for the symbolic unit used.

2. Places the information contained in the card in the device table for use by the physical I/O macros. If, however, the fourth operand is a command, 73 or 7B, that is to be issued to a 1403 printer with the UCS (Universal Character Set) special feature, only the information for the first three bytes of the device table will be obtained from the card. The fourth byte will be X'00'.

3. Issues a command to the printer control unit if the fourth operand is 73 or 7B to enable the control unit to accept, or ignore, data checks. If the fourth operand of 73 or 7B is erroneously issued to a printer without the UCS feature, a command reject occurs and the system enters the wait state.

If a fourth operand of 73 or 7B is encountered by the SYSTG program, it is placed in the device table, but a command to the printer is not issued.

A special format of this card can be used to "unassign" a device. When this format is used, the previously assigned physical unit address is deleted from the physical unit block for the specified name. No new address is substituted in its place. For example, the unit assigned to SYSLST might be dropped to bypass the optional listing output of the Librarian. The special format of the ASSGN card is:

```
// ASSGN SYSxxx,UA
```

To facilitate the use of symbolic I/O assignment, a second format is available for the ASSGN card. Two cards are actually used. The first card contains the symbolic unit; the second card contains the device address. Their formats are:

```
                                   col.72
First card:   // ASSGN SYSxxx,       X
```

```
Second card:       col.16
                      X'cuu',dd,X"ss'
```

The second card may also contain UA, in columns 16 and 17, to unassign a symbolic unit. Using the two-card format simplifies specification of assignment. A few prepunched cards can be made available to the operator. When a particular symbolic unit is specified, a card containing that unit, with another containing the actual device address, is placed in the Job Control input.

FILES Card

This card can be used to position tape files. If there is an ASSGN card for the same symbolic unit, the FILES card must follow it. This card may be used for any job. It must be used when restarting a previously checkpointed job, if any tape file was indicated as requiring reposition. Its format is:

```
// FILES SYSxxx,n
```

SYSxxx The symbolic unit used to refer to a tape file.

n The number of tape marks to be skipped to reposition the tape file. A maximum of four positions can be used (1-9999).

VOL Card

The volume card is used when checking or writing standard labels for a disk or tape file. A VOL card must be used for each file on a multi-file volume. Its format is:

```
// VOL SYSxxx,filename
```

SYSxxx Symbolic unit.

filename File name. This can be one to seven characters and is identical to the name used in the program to identify the file (if an ISFMS file, use only five characters). If an eight-character name is put in a card, it will be considered an error and will be indicated as such to the operator.

DLAB Card

The disk-label card (completed in a continuation card) contains file label information for disk label checking and creation. This card must immediately follow the volume (VOL) card. The DLAB card and the continuation card have the following format:

```
// DLAB 'label fields 1-3',

              xxxx,yyddd,yyddd,'system code'
```

'label fields 1-3', The first three fields of the Format 1 disk file label are punched just as they appear in the label. This is a 51-byte character string, punched within single quotes (8-5 punch), and followed by a comma. The entire 51-byte field must be contained in the first of the two cards. Column 72 must contain a continuation punch (any character). The Format 1 label is shown in Appendix B. Fields 1-3 are:

File Name 44-byte alphameric, including file ID and, if used, generation number and version number of generation.

Format Identifier 1-byte, EBCDIC 1.

File Serial Number 6-byte alphameric, must be the same as the volume serial number in the volume label of the first or only pack of the file.

c Continuation punch in column 72.

xxxx Volume Sequence Number. This 4-digit EBCDIC number is the EBCDIC equivalent of the 2-byte binary volume sequence number in Field 4 of the Format 1 label. This number must begin in column 16 of the continuation card. Columns 1-15 are blank.

The File Creation Date, followed by the File Expiration Date. These two 5-digit numbers are the EBCDIC equivalent of the 3-byte discontinuous binary dates in Fields 5 and 6 of the Format 1 label. yy is the year (00-99), and ddd is the day of the year (001-366).

'system code' System Code is punched as a 13-byte character string, within single quotes (8-5 punch). If punched for an output file, it is written in Field 8 of the Format 1 label. It is ignored when used for an input file. This field is not verified by the BOS label processing routines.

XTENT Card

The extent card defines each area, or extent, of a disk file. One or more XTENT cards must follow each DLAB card. The XTENT card has the following format:

```
 // XTENT type,sequence,lower,upper,
          'serial no.',SYSxxx
```

type Extent Type 1 column, contains a 1,2 or 4, indicating:

1 = data area
2 = independent overflow area (for indexed sequential file)
4 = index area (for indexed sequential file)

sequence Extent Sequence Number 3 columns, contains a 3-digit number from 000 to 255, indicating the sequence number of this extent within a multi-extent file. Extent sequence numbers for all files begin with 000 except for indexed sequential files that do not have a master index. The first extent sequence number for an indexed sequential file that does not have a Master Index is 001. In the case of an indexed sequential file, an extent card submitted for an independent overflow area must be the last card in the sequence of extent cards for that file.

lower Lower Limit of Extent 7 columns, contains the lowest address of the extent in the form CCCCHHH, where:

CCCC = cylinder number (0000 to 0202)
HHH = head number (000 to 009)

A lower limit of seven zeros, CCCC = 0000 and HHH = 000, is not permitted.

upper Upper Limit of Extent 7 columns, contains the highest address of the extent, in the same form as the lower limit.

'serial no.' Volume Serial Number This is a 6-byte alphameric character string, punched within quotes (8-5 punch). The number is the same as in the volume label (volume serial number) and the Format 1 label (file serial number).

SYSxxx This is the symbolic address of the disk drive.

TPLAB Card

The TPLAB card contains an image of a portion of the standard tape file label. The format and content of this label are presented in Appendix G. Label fields 3-10 are always punched just as they appear in the label. These are the only fields used for label checking. The additional fields (11-13) can be included, if desired. If punched for an output file, they are written in the corresponding fields of the output label. They are ignored when used for an input file. These fields are never used by BOS label-processing routines. The TPLAB card may have either of the following two formats:

1. // TPLAB 'label fields 3-10'

2. // TPLAB 'label fields 3-13'

'label fields 3-10' This is a 49-byte character string, punched within quotes (8-5 punch), identical to positions 5-53 of the tape file label.

These fields can be punched in one card.

'label fields 3-13' This is a 69-byte character string, punched within quotes (8-5 punch), identical to positions 5-73 of the tape file label. These fields are too long to be punched in a single card. The character string must extend into column 71, a continuation punch (any character) is punched in column 72, and the character string is completed in a continuation card. The continuation card is punched beginning in column 16.

DATE Card

This card contains a date which is put in the communication region. Its format is:

```
// DATE yyddd
```

yy Two-digit year.

ddd Three-digit day of the year.

For example, March 15, 1965 would be 65074. Job Control will convert this date and store it in the communication region in the form 031565074. The problem program can use the first six bytes for dating reports; the label checking routines use the last five bytes.

The DATE card must be entered only once after the Supervisor is loaded. Job Control will ensure that a date card has been entered before executing a problem program.

UPSI Card

This card (User Program Switch Indicators) allows the user to set program switches that can be tested much the same as sense switches or lights are used on other machines. Its format is:

```
// UPSI nnnnnnnn
```

n Each position is punched with a 0 or 1. Note that any punch other than a 1 will be interpreted as a zero.

Job Control clears the UPSI byte to zeros before reading control cards for each job. When Job Control reads the UPSI card, it stores the information in the UPSI byte in the communication region. Left to right in the UPSI card, the digits correspond to bits 0 to 7 in the UPSI byte. Each of the eight bits may be tested by problem programs at execution time. For example, a bit can be set to indicate that the problem program should print only total lines on a report. Or it can indicate an end-of-the-month condition so that certain routines in the program will be used.

Bit positions to the right of the last 1-bit are not required. All bits may be set by a single card or by multiple cards. For example, to set bits 1, 5, and 7 on, two methods can be used.

```
Method 1:  // UPSI 01000101

Method 2:  // UPSI 01       Set switch 1 on
           // UPSI 000001   Set switch 5 on
           // UPSI 00000001 Set switch 7 on
```

Therefore, for operational efficiency, only eight cards (one for each bit switch) need be kept by the operator to set any combination of switches.

CONFG Card

The configuration contains information about the machine size, model, and features. It can be used to override the configuration specified at the time the Supervisor was assembled. Its format is:

```
// CONFG nnnnnnnn
```

The operand (nnnnnnnn) consists of up to eight 0 or 1 punches. Note that any punch other than a 1 will be interpreted as a zero. Bit positions to the right of the last 1-bit are not required. The eight bits indicate:

0-3 Binary representation of machine size

8K	0000
16K	0010
24K	0011
32K	0100
64K	0110
128K	1000
256K	1010

4 Model (for diagnostic scan-out area)

0	Model 30
1	Other models

5 Floating-point feature } 0 = not present

6 Decimal feature } 1 = present

7 1052 Printer-Keyboard }

For example, // CONFG 00100010 represents a 16K Model 30 with the decimal feature.

Note that changing the machine configuration in this manner only affects programs that utilize this information at execution time. For instance, if the machine size is increased from 8K to 16K, use of CONFG will effectively inform the Assembler processor that it has more main storage to work with. At Supervisor assembly time the designation for model is used to reserve a 12-byte diagnostic scan-out area for a Model 30 or a 256-byte area for all other models. The Supervisor can not adapt itself to any changes at object time; it must be assembled again.

EXEC Card

Execute. This must be the last card processed before a job is executed. It indicates the end of control cards for a job and that execution is to begin. If the program to be run is not cataloged in the core image library, an operand LOADER is punched to indicate that the Linkage Editor is to be used to read the program from SYSIPT. Programs are permanently cataloged in the core image library through a Librarian run (JOB SYSCMAINT).

If the entire program is to be read from the relocatable library and edited into the core image library, the operand is written as LOADER,R. When this operand is used, a switch is set to instruct the Linkage Editor to go directly to the relocatable library for the program instead of first going to the unit assigned to SYSIPT. The name of the first object module is the name punched in the JOB card. Thus it is not necessary to have the INCLUDE Linkage Loader control card in the system input unit. When the LOADER,R format is used, the program in the relocatable library must be complete, preceded by a PHASE card image and followed by an ENTRY card image. The name in the PHASE card image for the first phase to be executed must be the same as the name in the JOB card.

The three possible formats of the EXEC card are:

```
// EXEC
// EXEC LOADER
// EXEC LOADER,R
```

LOG and NOLOG Cards

Job Control will list all control cards on either a 1052 Printer Keyboard or a printer assigned to SYSLOG. Control cards are not logged until a LOG card is encountered. Once a LOG card is read, logging continues from job-to-job until the control card NOLOG is recognized (or until the system is stopped, requiring an IPL procedure to restart). Both the LOG and NOLOG cards can be placed before a JOB card or between JOB and EXEC cards. The format of these cards is:

```
// LOG (any operand is treated as a
        comment)
// NOLOG (any operand is treated as a
        comment)
```

PAUSE Card

This card can be used to allow for operator setup between jobs. Its format is:

```
// PAUSE (any operand is treated as a
          comment)
```

If a PAUSE card precedes the JOB card (outside of the JOB-EXEC set), a pause message is immediately displayed, and processing is suspended. If the PAUSE card is between a JOB card and an EXEC card, all control cards are processed, a pause message is displayed, and processing is suspended just before the execution of the new job (after Linkage Editor function, if required). In both cases, the operator replies to the message to resume processing.

RESTART Card

A restart facility is available for checkpointed programs as a second phase of Job Control. A programmer can use the CHKPT in his program to cause checkpoint records to be written either on the checkpoint area of the resident disk pack or on a magnetic tape. This allows sufficient information to be stored so that program execution can be restarted from specified points. The checkpointed information includes the general registers (not floating-point registers), part of the communication region, tape-positioning information, the problem program area, and a restart address. (Not all of the Supervisor program is included in the checkpointed records.)

The restart facility allows the operator to continue execution of an interrupted job at a point other than the beginning. The procedure is to submit a group of job control cards including a restart (RSTRT) card.

A FILES card is also required if any tape units are to be repositioned. The FILES card has been explained in the previous section. There are two formats of the RSTRT card:

// RSTRT (if checkpoint is in resident disk pack)

// RSTRT SYSxxx,kkkk (if checkpoint is on magnetic tape)

SYSxxx Symbolic unit name of the tape unit on which the checkpoint records are stored.

kkkk Identification of the checkpoint record to be used for restarting. This can be any four characters. It corresponds to the checkpoint identification used when the checkpoint was taken.

When checkpoints are taken on tape, the problem program must supply a unique code for the checkpointed records. This code can be any four characters. The first checkpoint record could be CK01; the second could be CK02, etc. It is the user's responsibility to update this code before issuing the CHKPT macro.

The checkpoint records also include a count of the actual data blocks read or written on each tape. The block count does not include checkpoint records interspersed within the data file. If logical IOCS is used to process the tape files, the block counts are accumulated by the logical IOCS routines, except when an unlabeled file is to be read backwards or the file has nonstandard labels. The user must supply the block count for the two exceptions mentioned in the preceding sentence, and for a file processed by physical IOCS. The block count that the user supplies must be relative to the beginning of the tape file, regardless of whether the file is read forward or backward (see CHKPT Macro in the Assembler with Input/Output Macros publication, listed in the Preface of this manual).

When a checkpoint is taken on tape, it is helpful to punch a card or print a message indicating this code so that the operator knows the identification of the last checkpoint record taken. However, restarting can be done from any checkpoint record, not just the last.

The job control cards required for restarting are:

// JOB progname	Name of checkpointed program
// FILES SYSxxx,n	(if any tapes are to be repositioned)
// RSTRT or	(if checkpoint on disk)
// RSTRT SYSxxx,kkkk	(if checkpoint on tape)
// EXEC	

VOL and TPLAB and/or VOL, DLAB, and XTENT cards are necessary if files are to be opened or closed by the program being restarted.

Assignment of input/output devices to symbolic units may vary from the initial assignments. Assignments are made for restarting jobs in the same manner as assignments are made for normal jobs.

The FILES cards are used to position the tapes to the correct file. The restart program then uses the block counts contained in the checkpoint records to position the tapes correctly within particular files.

The RSTRT card can be put anywhere between the JOB and EXEC cards. Because this information is contained in the checkpoint records, the UPSI card is ignored if resubmitted. The DATE and CONFG information is not saved and should be resubmitted, if required.

If any one or more of the following conditions exist, no operand is used in the EXEC card.

1. The checkpointed program consists of a single phase, or overlay, and, therefore, is entirely contained in the checkpoint area.
2. The checkpointed program is permanently cataloged in the core image library.
3. The checkpointed program, although not permanently cataloged, is still in the core image library. This implies that the core image library has not been modified since the checkpoint occurred.

If a multiple-phase program is checkpointed, only the phase currently in core is written in the checkpoint area. If the program is not permanently cataloged in the core image library or if the library has been modified since the checkpoint, the EXEC card must be either:

```
        // EXEC LOADER      or,

        // EXEC LOADER,R
```

INITIAL PROGRAM LOADING

Operation of BOS is initiated through an IPL procedure from the resident disk pack. The operator has only to place the resident disk pack on a drive, select the address of that drive in the load unit switches, and press the load key. This causes record 1 on track 0 to be read into main storage bytes 0-23. The information read in consists of an IPL PSW and two CCWs , which in turn cause the reading and loading of the IPL Loader.

Operating in the supervisor state, the IPL Loader clears the general registers and all main storage except the area occupied or used by the IPL Loader. It then reads the Supervisor entry from the transient directory on track 4. This entry provides the location of the Supervisor within the core image library, and other information required for loading. Using this information, the IPL Loader reads the Supervisor into main storage.

Before releasing control, the IPL Loader performs these operations:

1. Stores the channel and unit number of the resident drive in the PUB that corresponds to SYSRES in the device table in the Supervisor. (See Physical Unit Block (PUB) in the section on Job Control.)

2. Initializes the timer word in main storage locations 80-83 to minus one.

3. Places the processing unit in the wait state with all interrupts masked except the External Interrupt. The operator can now enter into main storage location 0-3: the channel, unit number, and device type of the card reader assigned to SYSRDR. Normally, this device will have been assigned during system generation and the operator has only to press the interrupt key.

4. When the interrupt occurs, the IPL Loader tests bytes 0-2 to determine whether the operator entered an assignment for SYSRDR. If so, this number is placed in the corresponding PUB in the device table.

After completing these operations, the IPL Loader issues a FETCH for the Job Control program. The System Loader routine of the Supervisor loads Job Control and transfers control to it to begin processing the control cards for the first job.

LINKAGE EDITOR

The contents of this section are illustrated by Figure 14.

All programs executed in a BOS environment must be edited into the core image library by the Linkage Editor. Operating primarily as a function of the Librarian, the Linkage Editor reads the relocatable output of the language translators and edits it into executable, non-relocatable phases in the core image library. Once a program has been checked out and cataloged as a permanent entry in the core image library, the Linkage Editor is no longer required for that program. The program can then be run as a distinct job and loaded directly from the resident pack by the System Loader.

The extent of the editing function performed depends on the structure of the input program. The simplest case is that of a single-control-section program that was assembled with the proper main-storage origin point. Such a program has no external linkages to resolve and does not require relocation. The Linkage Editor has only to edit the program into the core image library and create a single phase entry in the core image directory. This corresponds to the first diagram in Figure 15.

In more complex situations, the operation may involve linking together and relocating multiple-control sections from separate assemblies to produce a number of separate phases in the core image library (see the last diagram in Figure 15). The Linkage Editor resolves all linkages between segments of the program and relocates the phases to specified main-storage locations.

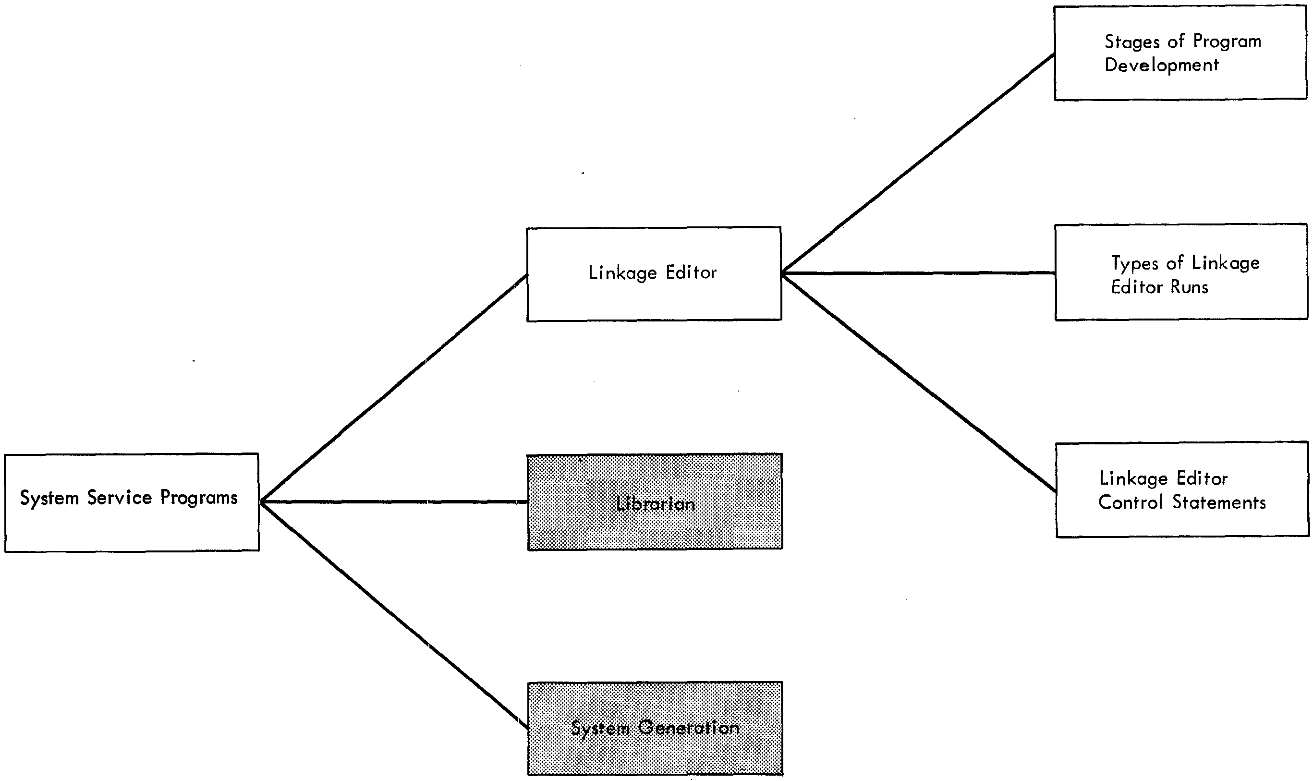

Stages of Program Development
Linkage Editor
Types of Linkage Editor Runs
Linkage Editor Control Statements
System Service Programs
Librarian
System Generation

Figure 14. System Service Programs: Linkage Editor

To facilitate writing and testing large programs, assembled program sections in the relocatable library can be combined with other sections from SYSIPT (card or tape). It is advantageous to handle some kinds of subroutines in this way instead of as macros.

STAGES OF PROGRAM DEVELOPMENT

In BOS, the term program could be confused with several things. The programmer codes sets of source statements that may be a complete program or part of a program. These source statements are then compiled or assembled into a machine language program which, in turn, must be edited into an executable program, and may be combined with other programs. Consequently, it is convenient to refer to each stage of program development by a particular name.

A set of source statements that is processed by a language translator (Assembler or RPG) is referred to as a source module.

The output of a language translator is referred to as an object module. All object modules must be further processed by the Linkage Editor before they can be executed in BOS.

Note: Each entry in the relocatable library is called a module. These relocatable modules normally consist of a single object module. If desired, however, more than one object module can be cataloged as a single entry in the relocatable library.

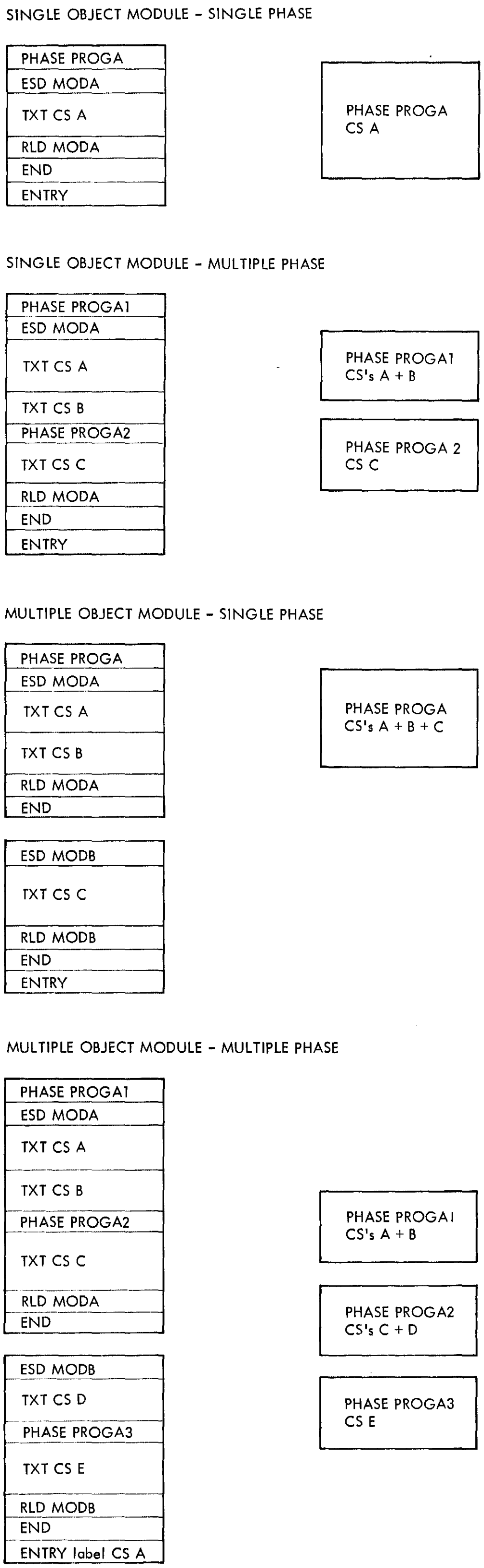


Figure 15. Linkage Editor Input and Output

The output of the Linkage Editor consists of one or more program phases in the core image library. A phase is in executable, non-relocatable, core image form. Each separate phase is loaded by the System Loader of the Supervisor in response to a FETCH macro.

STRUCTURE OF A PROGRAM

SOURCE MODULE: A source module is input to the language translator and consists of definitions for one or more control sections. When the source module is translated, the output (object module) consists of one or more defined control sections. To the language translators, a source module consists of one or more control sections. Each control section is a block of code assigned to contiguous main storage locations. The input for building a phase (a section of a program loaded as a single overlay) must consist of one or more complete control sections. Phases can be defined in the source module by using the Assembler REPRO instruction followed by a PHASE statement.

OBJECT MODULE: An object module is the output of a single, complete Assembler or RPG run. It consists of control dictionaries and text of one or more control sections. The control dictionaries contain the information necessary for the Linkage Editor to resolve cross references between different object modules. The text is the actual instructions and data fields of the object module. The program cards produced by the language processors (as distinct from the Linkage Editor control statements to be discussed later) have an identifier field in columns 2-4 that indicates the content of the card. The following card types are produced by the language translators:

Identification	Contents or Meaning
ESD	External Symbol Dictionary Item
TXT	Text
XFR	Transfer Address
RLD	Relocation Dictionary Item
END	End of a Module

All of these cards except XFR must be present in an object module in the indicated order. The XFR card can appear anywhere between the last ESD item and the first RLD item.

PROGRAM PHASE: A program phase, the output of the Linkage Editor, is that section of a program that is loaded as a single overlay with a single FETCH by the system loader. Each entry in the core image library is a single phase. Programs may consist of many phases, the first fetched by Job Control and each of the rest, by a preceding phase. Successive phases of a multi-phase program are often called overlays.

The input for building a single phase consists of the text from one or more complete control sections. When building a phase, the Linkage Editor constructs a composite ESD and a composite RLD from the control dictionaries of each of the modules that make up the phase. These composite dictionaries are used to resolve all linkages between different control sections as if they had been assembled as one module. Each control section within the phase is relocated as necessary, and the entire phase is assigned a contiguous area of main storage. All relocatable address constants are modified to contain the relocated value of their symbols. The Linkage Editor always ensures that each phase or control section begins on a double word boundary.

Each phase is constructed by building the text in a work area and then writing it in blocks in the core image library. Thus, a phase may consist of one or more blocks of contiguous core image locations. Although it is not necessary for the input program to use contiguous main storage assignment, this will provide faster operation of the Linkage Editor.

TYPES OF LINKAGE EDITOR RUNS

The Linkage Editor is never run as a distinct job. In this respect, its relationship to the programmer is like that of the control programs. It is meaningful, however, to classify it as one of the system service programs along with the Librarian. The Linkage Editor is called as an intermediate step in three kinds of jobs (Figure 16).

1. Catalog Phases in Core Image Library The Linkage Editor is called as an interim step in the Librarian operation (SYSCMAINT) that catalogs (CATAL operation) program phases as permanent entries in the core image library. The sequence of events when this operation is performed is shown in the first line of Figure 16. Note that the input to the SYSCMAINT job could include modules from the relocatable library instead of, or in addition to, the card or tape

unit assigned to SYSIPT. If the input is entirely from the relocatable library, the CATAL card shown would include the operand: modulename,R

2. Load and Go The Linkage Editor is called by Job Control when the EXEC card for a job has the operand LOADER. The program is edited into the core image library, and then Job Control is recalled to construct the phase directory and begin execution of the program. The name on the JOB card is the name of the first phase that is to be executed. The sequence of events when this operation is performed is shown in the second line of Figure 16. Just as with the catalog operation, the input can consist of object modules from the relocatable library instead of, or in addition to, the card or tape unit assigned to SYSIPT. If the input is entirely from the relocatable library, the EXEC card shown would include the operand: LOADER,R. The name on the JOB card is both the name of the module in the relocatable library and the name of the first phase to be executed.

3. Compile or Assemble and Execute Source modules can be assembled or compiled and then executed as a single job. In order to do this, the Assembler and RPG can be directed to output the object module directly into the relocatable library. Upon completion of this output operation, the language translator automatically calls the Linkage Editor to edit the program into the core image library. The Linkage Editor then fetches Job Control to construct the phase directory and begin execution. The sequence of events when this operation is performed is shown in line three of Figure 16. Note that object modules previously cataloged in the relocatable library could be linked with the module being assembled through the use of the INCLUDE statement.

The Linkage Editor uses the system directory to determine the first available locations in the core image library and the core image directory. At the completion of the Linkage Editor run, if no errors have been detected, entries for each of the phases written are placed in the core image directory.

These entries contain:

1. Phase name
2. Library location
3. Number of blocks
4. Length of last block
5. Main-storage assignment
6. Transfer address

If the program is being cataloged by the SYSCMAINT routine of the Librarian, the entries in the system directory that indicate the next available location in the

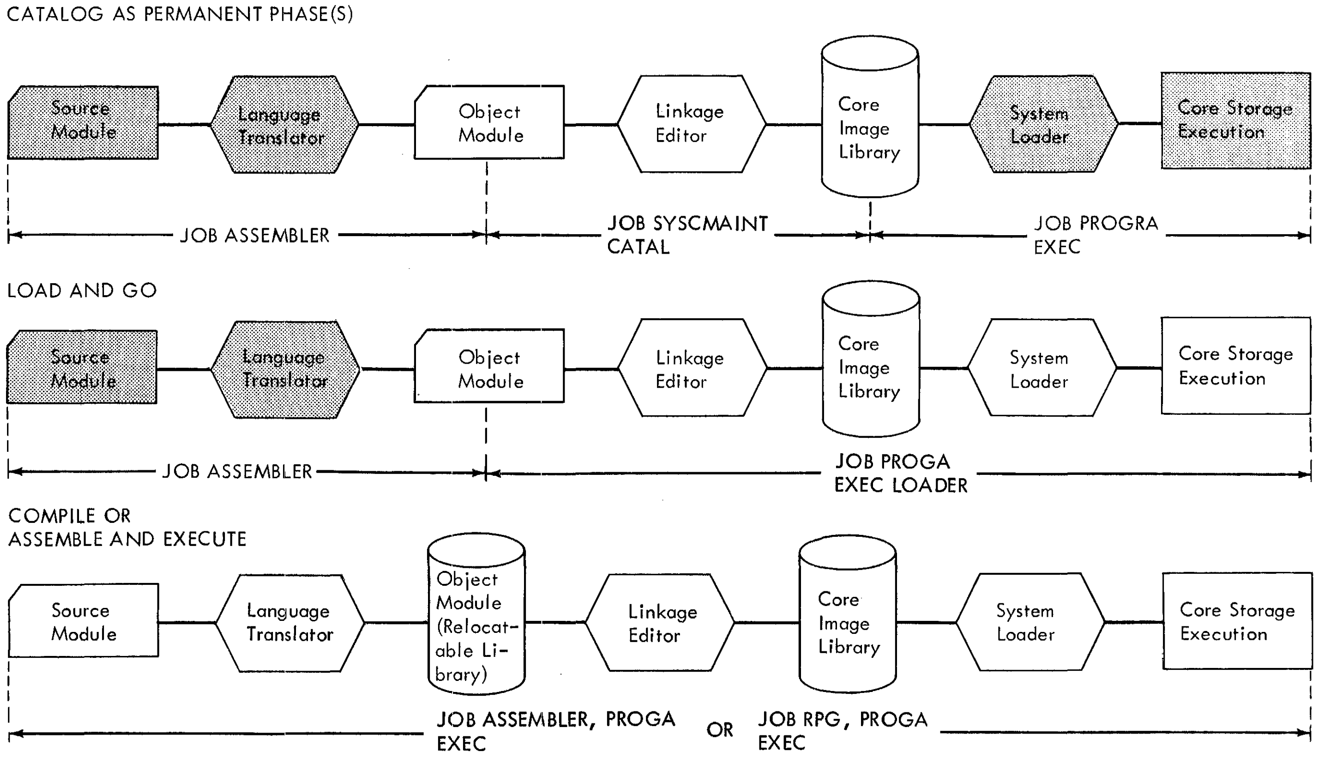


Figure 16. Three Types of Linkage Editor Operations

core image library and in the core image directory are modified to reflect the newly added phases. Thus, the next phase is loaded in the next available part of the library. When a phase is temporarily loaded (Load and Go, Compile and Execute, or Assemble and Execute), these entries in the system directory are not changed. Therefore, the next time the Linkage Editor is run, the new phases are written over the temporary phases, and the previous entry in the core image directory is also written over.

LANGUAGE TRANSLATOR CARDS

The five cards described in this section (ESD, TXT, RLD, END, XFR) are generated output of the Assembler or RPG.

ESD (EXTERNAL SYMBOL DICTIONARY): The external symbol dictionary contains all the symbol and storage assignments for a program section. For instance, it contains all symbols defined in this section that are referred to by some other section (ENTRYs).

It can contain all symbols referred to by this program section that are defined in some other section (EXTRNs).

ESD cards give information needed to link a module with others to prepare an operating program.

The four classifications of the ESD recognized by the Linkage Editor are:

1. SD (Section Definition) Generated by the use of START or CSECT pseudo Op in the source program. This type specifies the symbolic name, the assembled origin, and the length of the control section.

2. PC (Private Code) The same as SD except that no symbolic name is provided. Multiple PC items are not error conditions.

3. LD (Label Definition) Generated by the use of an ENTRY pseudo Op in the source program. It defines a symbolic label that may be used by any other program section as an entry point, data label, etc.

4. ER (External Reference) Generated by the use of an EXTRN pseudo Op in the source program. It indicates a symbolic label that is used in this program section to refer to a point in some other separately assembled program section.

The format of the ESD card is in Appendix I.

TXT (TEXT): The program that is eventually loaded into main storage for execution is contained within the TXT cards. The text card contains the assembled origin of the instructions or data included in the card, and also the count of the number of bytes contained on the card. This card includes a reference to the control section in which this information occurs and allows the relocation factor involved to be derived. TXT cards will be modified as required by RLD information. Formats of Assembler or RPG output cards are in Appendix I.

RLD (RELOCATION DICTIONARY): The relocation dictionary cards identify portions of the TXT card that must be modified due to relocation. They provide the information necessary to perform the relocation. The format of the RLD is in Appendix I.

END: The END card indicates end of module to the Linkage Editor. The END card can supply a transfer address that follows the rules of the transfer address supplied on an XFR card. Its format is in Appendix I.

XFR (TRANSFER): The transfer card provides a transfer to the point of entry of the problem program phase. In the case of multiple XFR cards or END cards being encountered during any phase, the symbolic label supplied in the first encountered XFR or END of a phase will be accepted as the entry point.

The transfer address in the XFR card or END card need not necessarily be an entry point defined in the program section, but may be one defined within the program section by an EXTRN and referring to an entry point within another program section. In the latter case, the entry cannot be adjusted by a displacement relative to a label. The XFR card will be generated by the XFR pseudo Op. The format of the XFR card is in Appendix I.

USER REPLACE CARD

When it becomes expedient to change text in an already-assembled (or compiled) object module, the programmer can use the facilities provided by the REP (replace text) card. The REP card is described in this section.

REP (REPLACE TEXT): The REP card allows the programmer to replace previously assembled or compiled text with new text. The format to be used for the REP card is in Appendix I. Each REP card must contain the assembled address (in hexadecimal) of the first byte to be replaced. The external symbol identification (ESID) must also be punched (in hexadecimal) into the card. The card format provides for 2 to 22 bytes of text, to be punched as 1 to 11 four-digit hexadecimal fields separated by commas (but no blanks).

REP cards must be placed after the last TXT card of the module that it is to modify. If the module represents input for more than one phase, the Linkage Editor will replace the text in the correct phase. The text punched in the REP card replaces the original text, byte for byte, beginning at the address specified in the REP card.

LINKAGE EDITOR CONTROL STATEMENTS

In addition to the program cards previously listed, object modules used as input to the Linkage Editor include Linkage Editor control statements. There are four kinds of these control statements: ACTION, PHASE, INCLUDE, and ENTRY. Their functions are:

ACTION 1. To inhibit printing of diagnostic messages and the main storage map.
2. To clear extraneous data from DS reserved areas.

PHASE To indicate beginning of a phase. Gives the name of the phase and the main-storage address where it is to be loaded.

INCLUDE To signal the Linkage Editor to include a module from the relocatable library. Gives the name of the module as it appears in the relocatable library directory.

ENTRY To signal the end of the last input object module and, if necessary, specify an overriding transfer point in the first phase processed.

The first (or only) object module input to the Linkage Editor must include a PHASE control statement before the first ESD item. The last (or only) object module in a job must be followed by an ENTRY control statement. The rules governing placement of INCLUDE and other PHASE control statements are discussed under Control Statement Placement.

SOURCES OF INPUT

Input to the Linkage Editor can be:

1. Entirely from the card or tape unit assigned to SYSIPT.

2. From SYSIPT, except when directed to the relocatable library by an INCLUDE statement.

3. Entirely from the relocatable library.

In the second case listed, it is possible for all input to be from the relocatable library except for the first INCLUDE statement and the ENTRY statement. In the third case, however, nothing is read from SYSIPT. This case is indicated in the following ways:

1. The presence of the ,R operand in a CATAL or EXEC card. For example:

```
a.  // JOB SYSCMAINT
    // EXEC
    // CATAL modulename,R
b.  // JOB progname
    // EXEC LOADER,R
```

   The modulename in example a is the name of the module in the relocatable library to be input to the Linkage Editor. The progname in example b is the name of the module in the relocatable library to be input to the Linkage Editor, and it also provides the name of the first phase that is to be executed.

2. All assemble and execute operations. For example:

```
        // JOB ASSEMBLER,progname
```

The rules stated under Control Statement Placement governing the placement of the Linkage Editor control statements apply equally to input from SYSIPT and from the relocatable library.

Note: When SYSIPT is assigned to a tape unit, the Linkage Editor assumes that the tape is positioned to the first control statement. The tape unit is not rewound by the Linkage Editor at the completion of processing.

GENERAL CONTROL STATEMENT FORMAT

The Linkage Editor control statements are similar in format to statements processed by the Assembler. No special symbols are required preceding the operation field. The operation field must begin to the right of column 1 (column 1 must be blank) and must be separated from the operand field by at least one blank position. The operand field is terminated by the first blank position. It cannot extend past column 71.

CONTROL STATEMENT PLACEMENT

The ENTRY card can be automatically produced at the end of each assembly, immediately following the END card, if specified in an Assembler control card (AOPTN ENTRY). When editing multiple-object modules in a single Linkage Editor run, the ENTRY card must follow the last, and only the last, object module.

The ACTION, PHASE, and INCLUDE cards can be reproduced from the source module by using the Assembler REPRO instruction. At the place in the source module where the Linkage Editor control statement will be needed, the programmer adds the REPRO card, followed by the ACTION, PHASE, or INCLUDE card. This will cause the control statement to appear in the output object module at the correct point.

Figure 17 shows the possible placement of the PHASE and INCLUDE statements. All statements can be hand inserted. Statements between the END card and the ENTRY card cannot be reproduced by the Assembler. They must be hand inserted. The REP card must also be hand inserted.

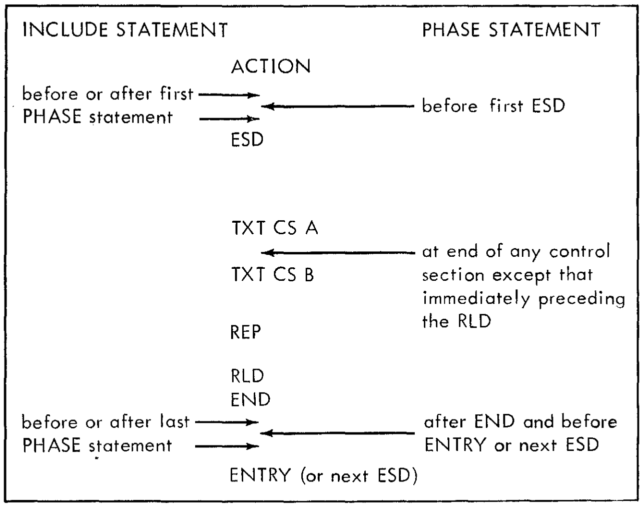

INCLUDE STATEMENT
PHASE STATEMENT
ACTION
before or after first PHASE statement
before first ESD
ESD
TXT CS A
at end of any control section except that immediately preceding the RLD
TXT CS B
REP
RLD
END
before or after last PHASE statement
after END and before ENTRY or next ESD
ENTRY (or next ESD)

Figure 17. Placement of PHASE and INCLUDE Statements

ACTION CARD

The ACTION card can be used to inhibit the printing of diagnostic messages and the main storage map. The ACTION card can also be used to specify that an area in the core image library be cleared prior to editing a new phase into the area.

When the ACTION card(s) is used, it must be placed before the first PHASE card in the input stream.

The ACTION card can have one of three operands: NOMAP, MAP, and CLEAR.

The Linkage Editor automatically prints diagnostic messages and a main storage map. If the messages and the map are not desired, the user can use an ACTION NOMAP card to prevent them from printing. If an ACTION MAP card is used, the Linkage Editor will ignore the card.

Name	Op	Operand
blank	ACTION	NOMAP
blank	ACTION	MAP

Diagnostic messages are printed, on the printer assigned to SYSLST, whenever an error occurs on a card in the input stream. The messages printed when an error occurs have the following format.

```
ERROR xx ppp...p
```

xx is a code used to identify the error. This code is printed in EBCDIC.

ppp...p is a print-out of the contents of all eighty columns of the card in error. If the card is a control card, the eighty columns will be printed in EBCDIC. If the card is not a control card, columns 6-8 (assembled origin) are printed in hexadecimal and the other columns in EBCDIC.

The main storage map describes the location of all portions of the edited program. This map is printed on the printer assigned to SYSLST after the input has been edited. The printed map will consist of:

1. PHASE usage--name of the phase, loading address, last-byte address used for the phase, disk address in the core image library, and transfer (execute) address.
2. ESD card items defined within each phase. These include SD, PC, and LD definitions. Unreferenced entry points and control sections are also shown.
3. Unresolved ERs (resulting from EXTRNs that have not been defined as entry points) after all phases and points within have been listed.
4. The number of unresolved RLD items encountered.

The ACTION card with the operand CLEAR can be used to clear the area that is available in the core image library for editing new phases.

Name	Op	Operand
blank	ACTION	CLEAR

The programmer can use this statement when he wants DS reserved areas to be cleared of extraneous data. Note that the Linkage Editor will not clear areas reserved by DS statements unless the ACTION CLEAR control card is used, since a DS is intended only to reserve the area. This function can be time consuming, because it clears the entire unused portion of the core image library each time it is used. Therefore, it is recommended that the user plan his programs in such a way that the ACTION CLEAR function is used only when debugging programs or when a number of temporary phases are being tested.

PHASE CARD

The PHASE card can be the first card of the first object module processed by the Linkage Editor and, in the case of multiple phases, is the first card of each succeeding phase also. Under no circumstances can a PHASE card occur within a control section. This is an undetectable error condition that will result in erroneous program loading. There can be several control sections within a phase and several phases within a module.

This card provides the Linkage Editor with a phase name and an origin point for the phase. The phase name is used to catalog the phase in the core image library. This name is used in a FETCH macro to load the phase for execution. The origin point for the phase can be relative to the end of the Supervisor, the last location of main storage, or a previously defined symbol. It can also be an absolute machine address.

Op	Operand
PHASE	phasename,f,displacement,symbol

The operands in the PHASE card are:

phasename

Symbolic name of the phase. One to six characters are used as the phase name. If more than six characters are used, those in excess of six are truncated. The first four characters of the phase name for each phase of a multi-phase program should be the same. This allows faster retrieval by the system loader. The first three characters can be SYS, only when cataloging a phase with the SYSCMAINT program.

A system can have only 124 phase names beginning with SYS. These three characters are used to begin the phase names for transient routines such as OPEN and CLOSE. Therefore, the user should not use phase names beginning with SYS. Other phase names that should not be used are listed in Appendix K.

f-Flag

Indicates the origin point for loading the phase. (The main storage loading address is relative to this point.

S Address of the last location in the Supervisor

C Address of the last location of main storage (core size is taken from the CONFG byte in the communication region)

L A symbolic label defined in a previous phase

A Absolute address

displacement

Allows the origin point (loading address) to be modified by a set amount. A positive displacement has no sign and is allowed to be 1 to 6 decimal digits. If the displacement is negative, a minus sign (-) precedes 1-5 decimal digits. A plus sign (+) preceding the displacement is an error and can be treated as part of the address. Displacement relative to S, C, L, or A will result in a double word boundary.

The displacement can also be expressed in hexadecimal. There can be from one to six hexadecimal characters enclosed in single quotes (5-8 punch). If a negative displacement is required, a minus sign must precede the expression; for example, X'hhhhhh' or -X'hhhhhh'

If the flag (f) is A, this value is treated as an absolute address instead of a displacement.

symbol

Symbolic label. May be 1 to 8 characters. It is predefined by having appeared in a previous phase. When included, it allows this phase to origin at the address the label represents.

Some examples of PHASE cards follow:

```
PHASE PHNAME,C,-504
```

This causes loading to start 504 bytes below the last byte of main storage.

Note: If displacement is a value such that it causes loading beyond the end of main storage, the condition will be handled by the Supervisor and indicated as an error when the System Loader attempts to load the phase.

```
PHASE PHNAME,L,20,POINT3
```

This card causes the phase to be loaded at an address of POINT3+20. POINT3 has appeared in a previous control section and must be the name of a START or CSECT statement or the operand of an ENTRY statement.

```
PHASE PHNAME,L,,POINT3
```

This would cause loading to begin at the address identified by POINT3.

```
PHASE PHNAME,A,4800
```

Loading begins at 4800, an absolute address specified in the phase card.

```
PHASE PHNAME,S
```

Loading begins at the immediate end of the Supervisor area.

Note: In each of the preceding examples, if the origin address supplied is not a double word boundary, the Linkage Editor will automatically increment to the next double word boundary.

INCLUDE CARD

This card is used to indicate to the Linkage Editor that an object module in the relocatable library is to be included in the program. The card has a single operand, which is the name of the module as it is cataloged in the relocatable library. By using this feature of the Linkage Editor, the programmer can include standard subroutines in his program at linkage time instead of defining them as macros.

The placement of the INCLUDE card determines the position of the module in the program. An included module (in the relocatable library) can be preceded or followed by one or more additional INCLUDE cards.

Name	Op	Operand
blank	INCLUDE	modulename

The INCLUDE card has a single operand.

Modulename Symbolic name of the module, as cataloged in the relocatable directory. One to six characters are used. If more than six characters are punched, the name is truncated to six.

An object module in the relocatable library can have any number of INCLUDE statements, only if it was not itself included by a module in the relocatable library. This is illustrated in Figure 18. Modules included by statements in the input from SYSIPT are referred to as being in the first level. Modules included by statements in the first level are at the second level. Thus, modules in the first level can include others in the second level. Those in the second level may not have INCLUDE statements.

ENTRY CARD

The ENTRY card signals the Linkage Editor that the end of the program has been reached. A transfer label can be included in this card to override the previous transfer point in the first phase. Every program, as input to the Linkage Editor, must be terminated by an ENTRY card.

Name	Op	Operand
blank	ENTRY	entrypoint

The ENTRY card has a single, optional parameter:

Entrypoint Symbolic name of an entry point. This parameter is optional. If used, it must be a symbol defined in the program with the Assembler ENTRY statement or it must be the name of a START or CSECT statement. It overrides the previous transfer point in the first phase.

PHASE ENTRY POINT

The Linkage Editor establishes the transfer point (entry point) for each program phase as follows:

1. If an entry point for a phase is not specified in either an XFR card or the END card, the origin of the phase is designated the entry point.

2. If the phase contains XFR and/or END cards that specify entry points, the first such card encountered designates the entry point for the phase.

3. If the ENTRY card for a program specifies an entry point, this entry point will override a previously established entry point for the first (or only) phase of the program.

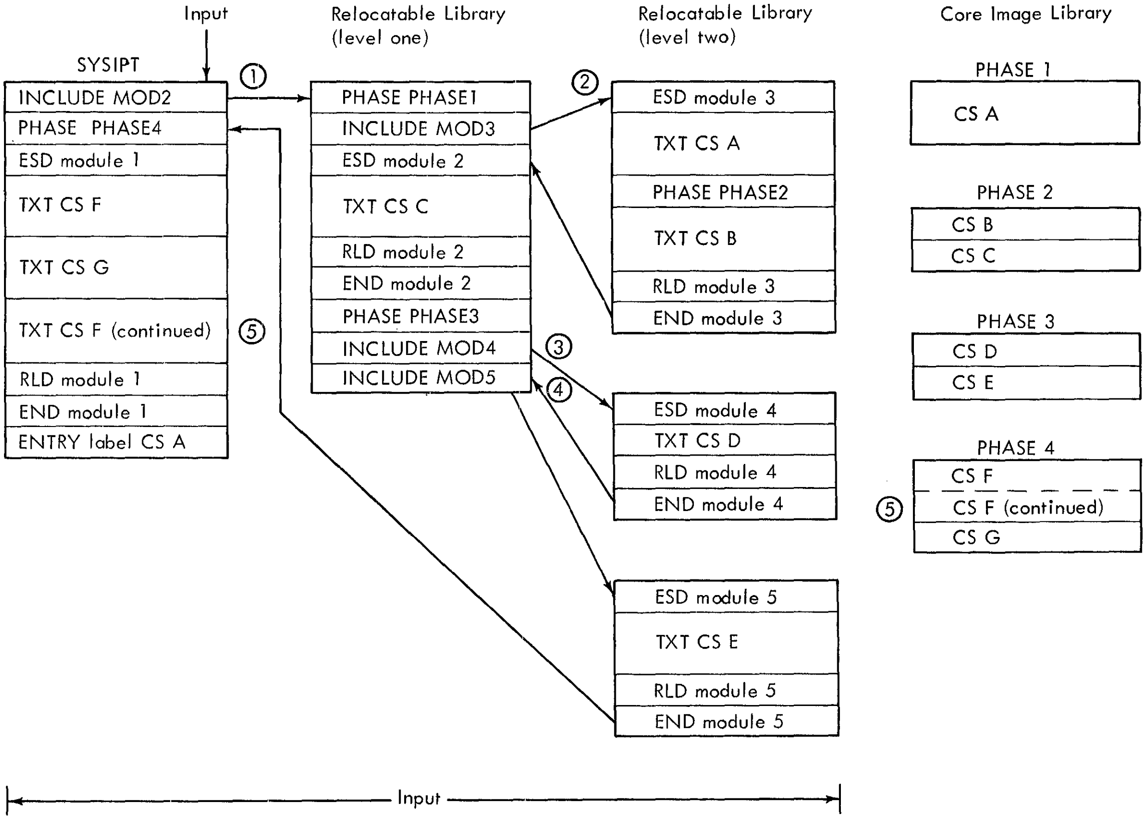


Figure 18. Example of Linkage Editor Input and Output

EXAMPLE OF LINKAGE EDITOR INPUT AND OUTPUT

The program shown in Figure 18 illustrates the rules governing input to the Linkage Editor and shows the output obtained. Though this example is more complex than the normal program, by following the flow of the input one can find practically every situation that may arise.

The left-most block (module 1) is shown as being read from SYSIPT, which we can assume to be a card reader. The two columns in the center are modules that have been cataloged into the relocatable library. The right-most column shows the output phases as they appear in the core image library.

1. The first card sends the Linkage Editor to the relocatable library. Therefore, the first entry in module 2 must be either a PHASE or another INCLUDE.
2. Because the phase has been named, module 3 can begin with an ESD.
3. The input for a phase can consist entirely of INCLUDE modules.
4. INCLUDE cards can be grouped.
5. This split control section (CS F) is assigned a contiguous area of main storage.

RESTRICTIONS

The number of phases and references to other phases is limited. Each phase, control section, label definition, or external reference represents an entry. Matching entries (LD of SYMBOL and ER of SYMBOL), however, are combined into one entry. In an 8K system, about eighty entries can be handled. In a 16K or larger system, up to 256 entries can be handled. The maximum size of a phase, which can be link edited, is 255 blocks or 210,120 bytes.

LIBRARIAN

The contents of this section are illustrated by Figure 19. This section describes the set of programs that maintain, service and organize the libraries of BOS. This set of programs is collectively referred to as the Librarian.

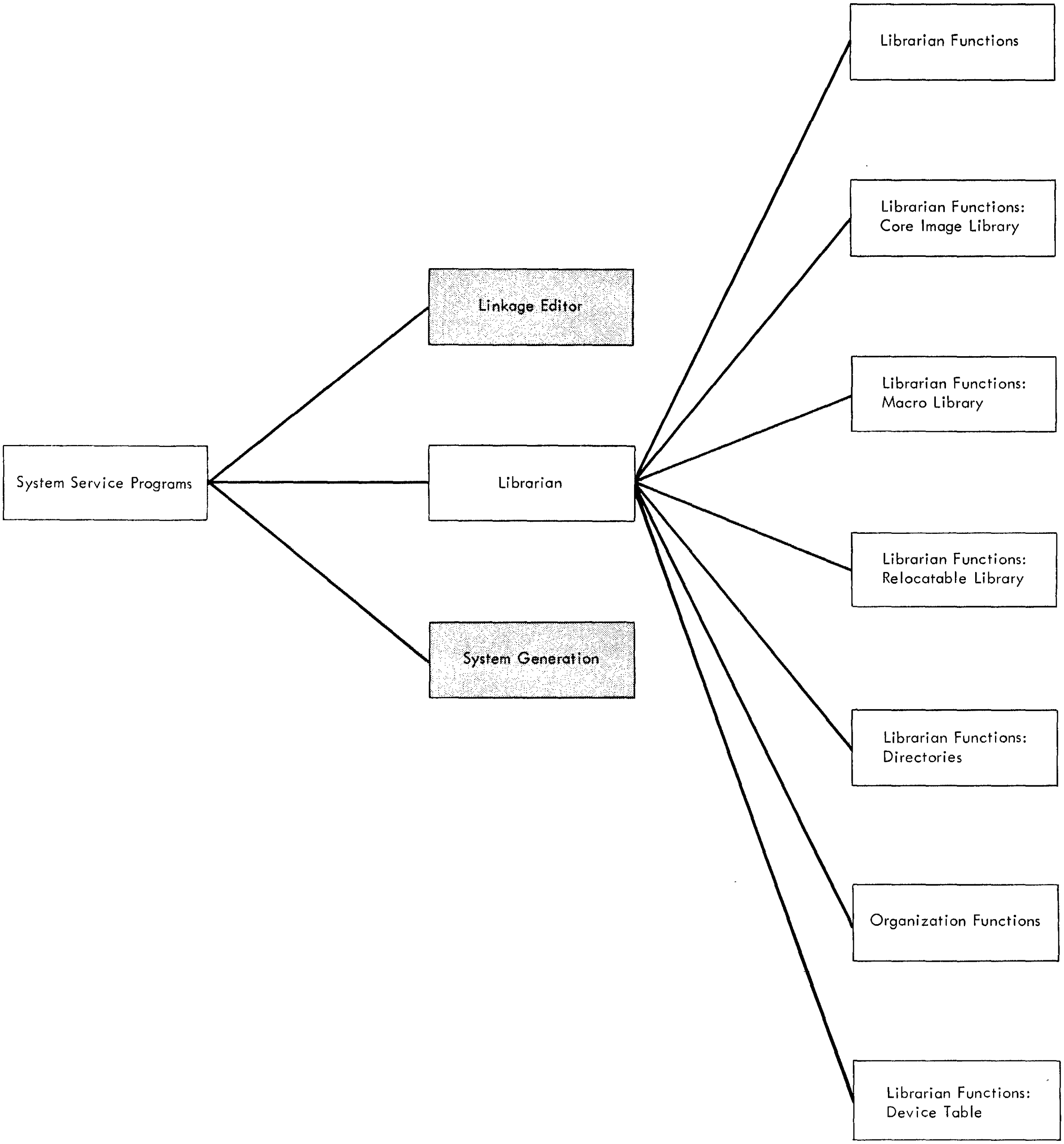

System Service Programs
Linkage Editor
Librarian
System Generation
Librarian Functions
Librarian Functions: Core Image Library
Librarian Functions: Macro Library
Librarian Functions: Relocatable Library
Librarian Functions: Directories
Organization Functions
Librarian Functions: Device Table

Figure 19. System Service Programs: Librarian

The BOS residence can contain three separate and distinct libraries. They are:

1. Core image library
2. Macro library
3. Relocatable library

The core image library is required for each disk resident system. The other two libraries, the macro library and the relocatable library, are not required for operating a system.

CORE IMAGE LIBRARY

The core image library contains any number of phases. Each phase is either a complete program or a program overlay. All programs in the core image library are edited to run with the resident supervisor. Each phase is assigned a fixed location in main storage.

Some of the programs in the core image library are permanent, while others are temporary. See the section entitled Linkage Editor for a description of how programs can be present in the core image library temporarily. All programs that are executed by the system are loaded from the core image library. The Linkage Editor converts programs from loader text to core image format and places them in the core image library. The programs in the core image library include system programs, the librarian programs, IBM programs such as Assembler, RPG, and sort programs, and user programs.

Associated with the core image library is a core image directory. The core image directory contains a unique descriptive entry for each phase in the core image library. Each entry includes the name of the phase, the starting disk address of the phase in the core image library, the number of blocks needed to contain the phase, the length of the last block, the starting address in main storage where the phase will be loaded, and the transfer address. The entries in the core image directory are used to locate and retrieve phases from the core image library.

Phases in the core image library and entries in the core image directory can be in any order.

MACRO LIBRARY

The macro library contains macro definitions that are used by the Assembler to expand macro instructions encountered in the source problem program. The macro definitions can be Supervisor macros, logical IOCS macros, and user-supplied macros.

Each macro contains a macro instruction prototype statement, one or more model statements, and a macro definition trailer statement. Each time the macro is used in a source deck for a problem program, it is transformed into a number of machine and/or assembler language statements.

Associated with the macro library is a macro directory. The macro directory contains a unique descriptive entry for each macro in the macro library. Each entry is composed of the macro name, the starting disk address of the macro in the macro library, and the number of blocks needed to contain the macro. The entries in the macro directory are used to locate and retrieve macros in the macro library.

Macros in the macro library and entries in the macro directory can be in any order.

RELOCATABLE LIBRARY

The relocatable library contains a number of modules. Each module can be one or more complete assemblies in the relocatable format. The purpose of the relocatable library is to allow the user to maintain frequently used routines in residence and combine them with other modules without requiring reassembly. The routines are edited from the relocatable library to the core image library by the Linkage Editor.

Associated with the relocatable library is a relocatable directory. The relocatable directory contains a unique descriptive entry for each module in the relocatable library. Each entry is composed of the module name, the starting disk address of the module in the relocatable library, and the number of blocks needed to contain the module. The entries in the relocatable directory are used to locate and retrieve modules in the relocatable library.

Modules in the relocatable library and entries in the relocatable directory can be in any order.

DISK STORAGE SPACE REQUIRED FOR LIBRARIES AND DIRECTORIES

The relative location of each of the library and directory areas is fixed. The number of tracks assigned to each is determined by the user. Each area consists of one or more complete disk tracks. The core image directory always begins on track 6, immediately following the fixed-assignment tracks (0-5). Beginning with the core image directory, the sequences of areas are:

1. Core image directory
2. Core image library

   required

3. Macro directory
4. Macro library

   optional

5. Relocatable directory
6. Relocatable library

   optional

If the macro library is not used, the relocatable directory immediately follows the core image library. If neither the macro nor relocatable libraries are used, the label control card area (volume information area) immediately follows the core image library. See Appendixes K and L for the sizes of IBM-supplied items.

Core Image Directory Size

Each track allocated to the core image directory can contain entries for 125 phases, except the last track, which can contain only 124 entries. Thus, the number of tracks (T) required for the core image directory equals:

$$T = \frac{P+1}{125}$$

Where: P = total number of phases in the core image library. The value of T is rounded off to the next-highest integer if a remainder results.

Core Image Library Size

Each track allocated to the core image library contains four fixed-length blocks. Each block contains a maximum of 824 bytes of instructions or data. Each phase can be up to 255 blocks long or 210,120 bytes. The core image library contains exactly the same information as is loaded into main storage for execution, and no more. Each phase is written beginning in a new block. The number of tracks required for the core image library can be calculated as follows:

1. Determine the number of blocks (B_x) required for a phase:

   $$B_x = \frac{L}{824}$$

   Where: L = total number of bytes in the phase. The value of B_x is rounded off to the next-highest integer.

2. Determine the total number of blocks (B_t) required for all phases in the core image library:

   $$B_t = B_1 + B_2 + B_3 \ldots\ldots + B_n$$

3. Determine the number of tracks (T) required to hold all phases in the core image library:

   $$T = \frac{B_t}{4}$$

   The value of T is rounded off to the next-highest integer if a remainder results.

Macro Directory Size

Each track allocated to the macro directory can contain entries for 216 macro definitions, except for the last track, which can contain only 215 entries. Thus, the number of tracks (T) required for the macro directory equals:

$$T = \frac{M + 1}{216}$$

Where: M = total number of macro definitions in the macro library. The value of T is rounded off to the next-highest integer.

Macro Library Size

Each track allocated to the macro library contains eight fixed-length blocks. Each block contains a maximum of 378 bytes of macro definition information. The macro-definition statements coded by the user are compressed before writing them out in the macro library. This compression is performed by eliminating all unnecessary blanks in each macro-definition statement. A five-byte field is then added to each statement before writing it out in the macro library. The number of tracks required for the macro library can be calculated as follows.

1. Determine the number of statements (N) used to define a macro.

2. Determine the average compressed statement length (L_s) in this macro. The compressed statement length equals the sum of:

L_n + 1 where L_n = bytes in name field

L_o + 1 where L_o = bytes in operation field

L_p + 1 where L_p = bytes in operand field

L_c + 1 where L_c = bytes in comments field

$\underline{+\ 5}$
$=C_1$ where C_1 =total bytes in compressed statement

Determine the average (L_s) of these to continue.

3. Determine the number of blocks (B_x) needed to hold the macro:

$$B_x = \frac{N(L_s)}{378}$$

The value of B_x is rounded off to the next-highest integer if a remainder results.

4. The total number of blocks (B_t) required to hold all of the macros in the library:

$$B_t = B_1 + B_2 + B_3 \ldots + B_n$$

5. The number of tracks (T) required to hold all of the macros in the macro library:

$$T = \frac{B_t}{8}$$

The value of T is rounded off to the next-highest integer if a remainder results.

Relocatable Directory Size

Each track allocated to the relocatable directory can contain 160 entries, except the last, which can contain only 159 entries. Thus, the number of tracks required for the relocatable directory equals:

$$T = \frac{M + 1}{160}$$

Where: M = total number of modules in the relocatable library. The value of T is rounded off to the next-highest integer if a remainder results.

Relocatable Library Size

Each track allocated to the relocatable library contains 16 fixed-length blocks. Each block is 160 bytes long. A number of factors affect the packing of information in these blocks. The factors include the following variables:

- The number of separate control sections.

- The use of DS (define storage) statements, which reserve storage that may or may not be utilized for data constants defined in the program.

- Alteration of the location counter during assembly (use of ORG statements).

The following calculations provide a fairly accurate approximation of the library area required for typical programs.

1. Determine the number of blocks (B_c) required for all cards or statements except the actual program text. Assume a separate block for each card of the following types:

$$\left.\begin{array}{l}\text{ACTION}\\ \text{END}\\ \text{ENTRY}\\ \text{ESD}\\ \text{INCLUDE}\\ \text{PHASE}\\ \text{REP}\\ \text{RLD}\\ \text{SYM}\\ \text{XFR}\end{array}\right\}\ \text{Let } B_c\text{=total number of cards}$$

2. Determine the number of blocks (B_i) required for the actual instructions or data in the TXT cards. Assume an average of 100 bytes of text in each block. (The maximum per block, for contiguously assigned text, is 132 bytes per block.) Thus,

$$B_i = \frac{\text{total bytes of text in TXT cards}}{100}$$

3. Determine the total number of blocks (B_x) required for a module in the relocatable library:

$$B_x = B_c + B_i$$

4. Determine the total number of blocks (B_t) required to hold all of the modules in the library:

$$B_t = B_1 + B_2 + B_3 ... B_n$$

5. Determine the number of tracks (T) required for the relocatable library:

$$T = \frac{B_t}{16}$$

The value of T is rounded off to the next-highest integer if a remainder results.

LIBRARIAN FUNCTIONS

The librarian consists of a set of programs that performs three major functions. They are:

1. Maintenance
2. Service
3. Organization.

Maintenance functions are used to add, delete, or rename components of the three libraries. The SYSCMAINT program is the maintenance program for the core image library. The RMAINT program is the maintenance program for the relocatable library. The MMAINT program is the maintenance program for the macro library.

Service functions are used to translate information from a particular library to printed (displayed) or punched output. Information in a library directory can also be displayed. The CSERV program is the service program for the core image library. The RSERV program is the service program for the relocatable library. The MSERV program is the service program for the macro library. The DSERV program is the service program for the directories.

Organization functions are used to reallocate library and directory areas, to condense libraries, and to copy, either completely or selectively, the disk on which the system resides. The AORGZ program is the organization program for the reallocation function. The LORGZ program is the organization program for the condense function. The CORGZ program is the organization program for the copy function.

Librarian functions are performed through the use of control cards. The control cards are:

1. A JOB control card requesting a particular librarian program.
2. A number of ASSGN control cards that may be required to change the assignment of actual input/output device.
3. An EXEC control card.
4. Librarian specification cards describing various functions to be performed.
5. An END control card.

The ASSGN and EXEC control cards are the same as those described in the Job Control section. The operand field of the JOB control cards is described in this section. The other cards pertain to the Librarian and are described in this section. All cards used by the Librarian conform to the rules of syntax for control cards as stated in the section on Job Control.

Librarian functions can be performed separately, or in certain combinations as described in the following sections. All control card information is read from the device assigned (in the ASSGN cards) to SYSRDR. All input data is read from the device assigned to SYSIPT. SYSIPT and SYSRDR can be assigned to the same physical input/output device. The status report of a library is printed on the device assigned to SYSLST. If SYSLST is unassigned, the printed status report is suppressed, but the librarian function is performed.

Figure 20 is a table of all maintenance functions. Figure 21 is a table of all service functions. Figure 22 is a table of all organization functions.

MAINTENANCE FUNCTIONS

The set of maintenance functions contains three subsets. They are:

1. Catalog
2. Delete
3. Rename

The catalog function adds phase(s) to the core image library, adds a module to the relocatable library, or adds a macro to the macro library. The input data for the catalog function is read from the device assigned to SYSIPT. The input device can be a card reader or a tape unit. Input for the catalog function for the core image library can also be a module on the relocatable library. A module can either be the entire input for the catalog function for the core image library, or modules can form part of the input for the phase. When the latter case is true, an INCLUDE card, discussed in detail in the section entitled <u>Linkage Editor</u>, can be used in the stream of input data for incorporating the module as part of the phase. The INCLUDE card can be present on SYSIPT or within a module in the relocatable library.

FUNCTION	UNIT	ELEMENT	CONTROL CARDS REQUIRED

Catalogue	Core-Image Library	Phase	// JOB SYSCMAINT
			// EXEC
			// CATAL modulename,R
			or
			// CATAL
			// END

	Macro Library	Macro	// JOB MMAINT
			// EXEC
			// CATAL
			// END

	Relocatable Library	Module	// JOB RMAINT
			// EXEC
			// CATAL modulename
			// END

Delete	Core-Image Library	Phase	// JOB SYSCMAINT
			// EXEC
			// DELET phasename
			// END

	Macro Library	Macro	// JOB MMAINT
			// EXEC
			// DELET macroname
			// END

		Library	// JOB MMAINT
			// EXEC
			// DELET ALL
			// END

	Relocatable Library	Module	// JOB RMAINT
			// EXEC
			// DELET modulename
			// END

		Library	// JOB RMAINT
			// EXEC
			// DELET ALL
			// END

Rename	Core-Image Library	Phase	// JOB SYSCMAINT
			// EXEC
			// RENAM oldname,newname
			// END

	Macro Library	Macro	// JOB MMAINT
			// EXEC
			// RENAM oldname,newname
			// END

	Relocatable Library	Module	// JOB RMAINT
			// EXEC
			// RENAM oldname,newname
			// END

Figure 20. Maintenance Functions

The delete function deletes an entry from a directory that corresponds to a phase, module, or macro in a library. The phase, module, or macro in the appropriate library is not deleted; however, as far as the system is concerned, the phase, module, or macro no longer exists. When a macro library or a relocatable library and its directory are to be removed from system residence, the delete function is required. The reallocation function should be used to completely remove the library and its directory.

The rename function is used to rename an existing phase, module, or macro in the appropriate library and directory.

If SYSLST is assigned, each job requesting a maintenance function will have the status of the system printed at the completion of the run.

SERVICE FUNCTIONS

The set of service functions contains three subsets. They are:

1. Display
2. Punch
3. Display and punch

The disk resident system can display and/or punch phases in the core image library, modules in the relocatable library, and macros in the macro library. In addition, the core image directory, the relocatable directory, and the macro directory can be displayed.

Whenever a requested service function provides punched-card output, a card punch must have been assigned to SYSOPT. Whenever a requested service function provides printed output, a printer must have been assigned to SYSLST.

FUNCTION	UNIT	ELEMENT	CONTROL CARDS REQUIRED

Display	Core-Image Library	Phase	// JOB CSERV
			// EXEC
			// DSPLY phase1,phase2,
			...,phasen
			// END

		Library	// JOB CSERV
			// EXEC
			// DSPLY ALL
			// END

		Directory	// JOB DSERV
			// EXEC
			// DSPLY CD
			// END

	Macro Library	Macro	// JOB MSERV
			// EXEC
			// DSPLY macro1,macro2,
			...,macron
			// END

		Library	// JOB MSERV
			// EXEC
			// DSPLY ALL
			// END

		Directory	// JOB DSERV
			// EXEC
			// DSPLY MD
			// END

	Relocatable Library	Module	// JOB RSERV
			// EXEC
			// DSPLY mod1,mod2,
			...,modn
			// END

		Library	// JOB RSERV
			// EXEC
			// DSPLY ALL
			// END

		Directory	// JOB DSERV
			// EXEC
			// DSPLY RD
			// END

	System Directory	Directory	// JOB DSERV
			// EXEC
			// DSPLY SD
			// END

	Transient Directory	Directory	// JOB DSERV
			// EXEC
			// DSPLY TD
			// END

	Directories	All	// JOB DSERV
			// EXEC
			// DSPLY CD,RD,MD,SD,TD
			or
			// DSPLY ALL
			// END

Figure 21. Service Functions (Part 1 of 2)

FUNCTION	UNIT	ELEMENT	CONTROL CARDS REQUIRED

Punch	Core-Image Library	Phase	// JOB CSERV
			// EXEC
			// PUNCH phase1,phase2, ...,phasen
			// END

		Library	// JOB CSERV
			// EXEC
			// PUNCH ALL
			// END

	Macro Library	Macro	// JOB MSERV
			// EXEC
			// PUNCH macro1,macro2, ...,macron
			// END

		Library	// JOB MSERV
			// EXEC
			// PUNCH ALL
			// END

	Relocatable Library	Module	// JOB RSERV
			// EXEC
			// PUNCH mod1,mod2, ...,modn
			// END

		Library	// JOB RSERV
			// EXEC
			// PUNCH ALL
			// END

Display and Punch	Core-Image Library	Phase	// JOB CSERV
			// EXEC
			// DSPCH phase1,phase2, ...,phasen
			// END

		Library	// JOB CSERV
			// EXEC
			// DSPCH ALL
			// END

	Macro Library	Macro	// JOB MSERV
			// EXEC
			// DSPCH macro1,macro2, ...,macron
			// END

		Library	// JOB MSERV
			// EXEC
			// DSPCH ALL
			// END

	Relocatable Library	Module	// JOB RSERV
			// EXEC
			// DSPCH mod1,mod2, ...,modn
			// END

		Library	// JOB RSERV
			// EXEC
			// DSPCH ALL
			// END

Figure 21. Service Functions (Part 2 of 2)

ORGANIZATION FUNCTIONS

The set of organization functions contains three subsets. They are:

1. Reallocation
2. Library condense
3. Copy system.

Function	Control Cards Required	Remarks

Reallocation	// JOB AORGZ
	// EXEC
	// TRCKS id,no,id,..., id,no
	// END
		Values to be substituted for id:
		CL - Core Image Library
		RL - Relecatable Library
		ML - Macro Library
		CD - Core Image Directory
		RD - Relocatable Directory
		MD - Macro Directory
		CP - Checkpoint Area
		VL - Volume Area
		Values to be substituted for no:
		Any interger

Library Condense	// JOB LORGZ
	// EXEC
	// CONDS lib, lib, lib
	or
	// CONDS ALL
	// END
		Values to be substituted for lib:
		CL - Core Image Library
		RL - Relocatable Library
		ML - Macro Library

Copy System	// JOB CORGZ
	// EXEC
	// TRCKS card (if required)
	// COPYS id,name1, namen
	or
	// COPYS ALL
	or
	// COPYS id,ALL
	or
	// COPYS id,name1, namen
	or
	// COPYS id,name2-namen
	// END
		TRCKS card is required if new library limits are to be established. If used, the number of tracks to be allocated for each library, directory, or area must be specified. The values for id and no are the same as shown for the TRCKS card for the AORGZ function.
		In the COPYS card, namen is the name of the phase, module, or macro to be copied. The value for id is:
		CL - Core Image Library
		RL - Relocatable Library
		ML - Macro Library

Figure 22. Organization Functions

The reallocation function is used to redefine the sizes of the libraries, the library directories, the checkpoint area, and the volume area. The reallocation function can be used to increase, decrease, or eliminate specific areas of the disk resident system. Each library reallocated is automatically condensed. Any number of areas can be reallocated within a single JOB run. When the reallocation function is completed, a status report of the system is provided if a printer is assigned to SYSLST. The printed output consists of the status of each definable item on the system resident pack.

The library condense function is used to minimize vacancies between elements in a library. One or more of the three libraries can be condensed within a single JOB run. The condense function is used whenever a number of vacancies have accumulated within a library, thus affecting the ability to catalog a new element to a library.

The copy-system function is used to copy, selectively or completely, the disk resident system. A complete copy can be used to obtain back-up if the original system is inadvertently destroyed. A selective copy can be used for reducing a complete basic operating system to a system that is designed to perform a specific purpose.

LIBRARIAN FUNCTIONS: CORE IMAGE LIBRARY

This section describes the maintenance and service functions that relate to the core image library. Organization functions for the core image library are discussed in the section entitled Organization Functions.

MAINTENANCE FUNCTIONS

To request a maintenance function for the core image library, use the following JOB control card.

```
// JOB SYSCMAINT
       or
// JOB SYSCMA
```

One or more of the three maintenance functions can be requested within a single JOB run. Any number of phases within the core image library can be acted upon in this run.

Catalog

The catalog function adds a phase to the core image library. The CATAL control card is required to add a phase or a group of phases to the core image library. The CATAL control card is read from the device assigned to SYSRDR. The CATAL control card is in one of the following formats.

```
// CATAL
// CATAL modulename,R
```

The first format is used if the input for the phase(s) to be cataloged is from the device assigned to SYSIPT. The operation field, separated from the two slashes by at least one blank, contains CATAL.

The second format is used if the entire input for the phase(s) to be cataloged is from the relocatable library. The operation field, separated from the two slashes by at least one blank, contains CATAL. The operand field, separated from the operation field by at least one blank, contains the name of the module that is to be the input. Immediately following the name of the module is a comma and an R. The operand,R, indicates that the input for the phase is one or more modules in the relocatable library. Only the first six characters of the module name are used.

Each phase that is cataloged in the core image library derives its name from the PHASE control statement.

If a phase in the core image library is to be replaced by a new phase having the same name, only the catalog function need be used. The delete function is implied with each catalog function.

Only one CATAL card is required to catalog a multi-phase program. Any number of phases or multi-phase programs can be cataloged in the core image library within a single JOB run.

The cards that make up the input for a phase are described in the section entitled Linkage Editor. The cards are:

1. ACTION control statement
2. PHASE control statement
3. INCLUDE control statement
4. ESD cards
5. TXT cards
6. RLD card
7. REP card
8. END card
9. XFR card
10. ENTRY control statement.

These cards are read from the device assigned to SYSIPT or from the relocatable library.

For the catalog function for the core image library, SYSRDR must be assigned to a card reader, SYSIPT must be assigned to either a card reader or a tape unit, and SYSLST can be assigned to a printer. If SYSIPT is a tape unit, this program assumes that the operator has positioned the tape to the first input record. The tape unit is not rewound at the end of job.

Control card input for the catalog function that is read from the device assigned to SYSRDR is as follows.

1. The JOB control card (SYSCMAINT), followed by

2. The ASSGN control cards, if the current assignments are not those required. The ASSGN cards that can be used are SYSIPT, SYSRDR, and SYSLST, if a printed status report is desired. The ASSGN cards are followed by

3. The EXEC control card, followed by

4. The CATAL control card(s), followed by

5. The END card, which is the last control card of the job.

Cataloging a New Supervisor

When a new Supervisor is to be cataloged in the core image library to replace a user-assembled Supervisor on an existing system, special factors must be taken into account. These factors include:

1. Unique control cards required to perform the function.

2. Relative sizes of the old and new Supervisors.

These two factors are discussed in detail in the following subsections.

Control Cards: Cataloging a new Supervisor requires that a special control card immediately follow the EXEC card. This control card is punched in the following format.

```
// SPVSR
```

The operation field, separated from the two slashes by at least one blank, contains SPVSR. The operand field is blank.

Following the SPVSR card is the catalog control card, punched in one of two formats. If the new Supervisor is in the form of a card deck to be read from the device assigned to SYSIPT, use the standard CATAL control card. This control card is punched in the following format.

```
// CATAL
```

If the new Supervisor is stored in the relocatable library as a module, use the following control card.

```
// CATAL SYSSUP,R
```

The entry in the operation field, separated from the two slashes by at least one blank, is CATAL. The entry in the operand field, separated from the operation field by at least one blank, is SYSSUP,R.

Relative Sizes of Supervisors: When a new Supervisor is cataloged on the core image library, consideration must be given to the relative sizes of the existing Supervisor and its related programs and the new Supervisor.

If a Supervisor is ever to be used in a system with main storage greater than 32K, this fact must be specified at supervisor-assembly time because additional coding is required in the Supervisor. For this, the CONFG parameter of the SUPVR macro must specify 64K or greater. For program execution on a smaller configuration, this specification can be altered by a Job Control CONFG card. See section on CONFG Card.

An estimate of the size of a Supervisor to be assembled can be made by using the information provided in Appendix H. Core Sizes and Timings. After a Supervisor has been assembled, the assembly listing can be examined to determine the exact size of the Supervisor. The symbolic label SYSEND identifies a DC instruction that contains the actual end of Supervisor address.

If the Supervisor to be replaced has a patch area large enough, the new Supervisor can be made equal in size to the old Supervisor. The SEND macro statement can be used to adjust the patch area of the new Supervisor, so that the new Supervisor will fit into the same area used for the Supervisor being replaced.

If the two Supervisors are the same size, the programs in the core image library that were edited to run with the old Supervisor will run with the new Supervisor. Only the old Supervisor needs to be replaced.

If the new Supervisor is smaller than the old Supervisor, the programs that were edited to run with the old Supervisor will run with the new Supervisor. The transient routines (the routines that are loaded and executed in the transient area) should be re-edited at this time. The programs can be re-edited to run with the new Supervisor if the user so wishes. Re-editing could provide additional main storage for problem programs.

If the new Supervisor is larger than the old Supervisor, the programs that were edited to run with the old Supervisor might require re-editing to run with the new Supervisor. This would be true in the case where the related programs are dependent on the size of the Supervisor. Certain phases must be recataloged during the same JOB run that catalogs the Supervisor on the core image library. These phases are the key system programs and have names beginning with SYS. The key programs are listed in Appendix K. The remaining programs on the core image library need not be cataloged during the same JOB run in which the new Supervisor is cataloged. The user must ascertain that the core image library contains ample space to catalog the new Supervisor and the related key system programs that must be cataloged in the same JOB run. If the space is inadequate:

1. Perform a library-condense job to minimize vacancies in the library, or

2. Perform a reallocation job to enlarge the size of the core image library, or

3. Delete some of the phases that need not be cataloged at the same time as the Supervisor (provided back-up exists for the programs to be deleted).

Delete

The delete function is used to remove references to specific phases from the core image library. Any number of phases can be deleted during a single run. The phases are not physically deleted from the library; rather, the entry describing the phase in the core image directory is deleted.

The DELET control card is used to delete phases from the core image library. It is punched in the following format.

```
// DELET phasename
```

The entry in the operation field, separated from the two slashes by at least one blank, is DELET. phasename in the operand field, separated from the operation field by at least one blank, represents the name of the phase to be deleted. The name of the phase can be of any length; however, a maximum of the first six characters is used to locate and delete the phase.

Any number of DELET control cards can be used for the core image library within a single JOB run.

For the delete function, SYSRDR must be assigned to a card reader and SYSLST can be assigned to a printer.

Control card input for the delete function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (SYSCMAINT), followed by

2. The ASSGN control cards, if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR and SYSLST if a printed status report is desired. The ASSGN cards are followed by

3. The EXEC control card, followed by

4. The DELET control card(s), followed by

5. The END control card, which is the last control card of the job.

Rename

The rename function is used to change the name of a phase in the core image library to another name.

The RENAM control card is used to achieve the rename function. As soon as the card is processed, the system recognizes only the new phase name. The RENAM card is punched in the following format.

```
// RENAM oldname,newname
```

The operation field, separated from the two slashes by at least one blank, contains RENAM. The operand field entries, oldname and newname, represent the old phase name and the new phase name. The two entries in the operand field must be separated by a comma. The operation field is separated from the operand field by at least one blank. The names in the operand field can be of any length; however, only a maximum of the first six characters is used by the system to locate and rename the phase.

Any number of RENAM control cards can be used for the core image library within a single JOB run.

For the rename function, SYSRDR must be assigned to a card reader, and SYSLST can be assigned to a printer.

Control card input for the rename function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card, (SYSCMAINT), followed by
2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR and SYSLST if a printed status report is desired. The ASSGN cards are followed by
3. The EXEC control card, followed by
4. The RENAM control card(s), followed by
5. The END control card, which is the last control card of the job.

SERVICE FUNCTIONS

To request a service function for the core image library, use the following JOB control card.

```
// JOB CSERV
```

One or more of the three service functions can be requested within a single JOB run. Any number of phases can be acted upon in this run.

Display

The display function is used to get a printout of a phase in the core image library. Any number of phases can be displayed within a single JOB run. The printed output consists of a header and the phase.

Contained in the printed header is the phase name, the starting main-storage address, the transfer address, the number of blocks within the phase, the number of bytes in the last block, and the address of the phase in the core image library.

The printed output of the core image phase is represented in hexadecimal characters. Each line of printed output contains 106 characters, each line containing eight full words or 32 bytes of information. In the printed line, the main-storage address and each full word are separated by blanks.

The DSPLY control card is used to display phases in the core image library. It is punched in one of the following formats.

```
// DSPLY phase1,phase2,...,phasen

// DSPLY ALL
```

The first format is used if only particular phases are to be displayed. The entry in the operation field, separated from the two slashes by at least one blank, is DSPLY. The entry in the operand field, <u>phasen</u>, is separated from the operation field by at least one blank. It represents the name of the phase to be displayed. If more than one phase is to be displayed, the phase names are separated by a comma. The phase name can be of any length; however, only a maximum of the first six characters is used by the system to locate the phase. Continuation cards are not recognized.

The second format is used if the entire core image library is to be displayed. The entry in the operation field is DSPLY. The entry in the operand field is ALL.

For the display function, SYSRDR must be assigned to a card reader, and SYSLST must be assigned to a printer.

Control card input for the display function, read from the device assigned to SYSRDR, is:

1. The JOB control card (CSERV), followed by
2. The ASSGN control cards, if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR and SYSLST. The ASSGN cards are followed by
3. The EXEC control card, followed by
4. The DSPLY control card(s), followed by
5. The END control card, which is the last control card of the job.

Punch

The punch function is used to convert a phase in the core image library into an absolute (not relocatable) card deck.

The punched-card deck that results from the punch function is usable as input in the same manner as a relocatable card deck, except that it cannot be relinked or relocated. The information required to relink or relocate is no longer available to the phase.

The punched-card deck representing the phase can be used as input for any of the following:

LDSYS
SYSCMA
RMAINT
Load and Go

Any number of phases in the core image library can be punched within a single JOB run.

The card deck will contain CATAL cards, a set of cards for each phase, ENTRY cards, and a Librarian // END card. Each set of cards representing a phase will contain a PHASE card, an SD type of ESD card, TXT cards, and an Assembler END card, A CATAL card will precede the first PHASE card in each group of forty phases.

The PHASE card contains the phase name, the absolute flag, and the absolute main-storage address for loading. This card is always the first card of a phase.

The SD type of ESD card provides the length of the phase and the absolute loading address. This card is always the second card of the phase.

The TXT cards contain the actual data of the phase.

The Assembler END card provides the absolute transfer address. It is always the last card of a phase.

An ENTRY card follows the Assembler END card for the last phase in each group of forty phases and the last phase of the run.

A Librarian // END card is the last card in the deck of cards punched for the run.

The PUNCH control card is used to convert phases in the core image library to punched-card output. It is punched in one of the following formats.

```
// PUNCH phase1,phase2,...,phasen

// PUNCH ALL
```

The first format is used if only particular phases are to be punched. The entry in the operation field, separated from the two slashes by at least one blank, is PUNCH. The entry in the operand field, _phasen_, is separated from the operation field by at least one blank. It represents the name of the phase to be punched. If more than one phase is to be punched, the phase names are separated by a comma. The phase name can be of any length; however, only a maximum of the first six characters is used by the system to locate the phase. Continuation cards are not recognized.

The second format is used if the entire core image library is to be punched. The entry in the operation field is PUNCH. The entry in the operand field is ALL.

Any number of PUNCH cards can be used within a single JOB run.

For the punch function, SYSRDR must be assigned to a card reader, SYSOPT must be assigned to a card punch, and SYSLST can be assigned to a printer.

Control card input for the punch function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (CSERV), followed by
2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR, SYSOPT, and SYSLST. The ASSGN cards are followed by
3. The EXEC control card, followed by
4. The PUNCH control card(s), followed by
5. The END control card, which is the last control card of the job.

Display and Punch

The display and punch function is used to combine the separate operations of the display function and the punch function. The output of the display and punch function is identical to that described in the two preceding subsections. Any number of phases in the core image library can be displayed and punched within a single JOB run.

The DSPCH control card is used to convert phases in the core image library to printed and punched-card output. The DSPCH control card is punched in one of the following formats.

```
// DSPCH phase1,phase2,...,phasen

// DSPCH ALL
```

The first format is used if only particular phases are to be displayed and punched. The entry in the operand field, separated from the operation field by at least one blank, is phasen. It represents the name of the phase to be displayed and punched. If more than one phase is to be displayed and punched, the phase names are separated by a comma. The phase name can be of any length; however, only a maximum of the first six characters is used by the system to locate the phase. Continuation cards are not recognized.

The second format is used if the entire core image library is to be displayed and punched. The entry in the operation field is DSPCH. The entry in the operand field is ALL.

For the display and punch function, SYSRDR must be assigned to a card reader, SYSOPT must be assigned to a card punch, and SYSLST must be assigned to a printer.

Control card input for the display and punch function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (CSERV), followed by

2. The ASSGN control cards, if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR, SYSOPT, and SYSLST. The ASSGN cards are followed by

3. The EXEC control card, followed by

4 The DSPCH control cards(s), followed by

5. The END card, which is the last control card of the job.

LIBRARIAN FUNCTIONS: MACRO LIBRARY

This section describes the maintenance and service functions that relate to the macro library. Organization functions for the macro library are discussed in the section entitled Organization Functions.

MAINTENANCE FUNCTIONS

To request a maintenance function for the macro library, use the following JOB control card.

// JOB MMAINT

One or more of the three maintenance functions can be requested within a single JOB run. Any number of macros can be acted upon in this run.

Catalog

The catalog function adds a macro to the macro library. Each macro is a macro definition, composed of a macro-definition header statement, a macro-definition prototype statement, one or more model statements, and a macro-definition trailer statement. Card input for the catalog function is from the device assigned to SYSIPT. Macros to be cataloged to the macro library can be in any order. Any number of macros can be added within a single JOB run.

The CATAL control card is required to add a macro to the macro library. It is read from the device assigned to SYSRDR. The CATAL card is punched in the following format.

// CATAL

The operation field, separated from the two slashes by at least one blank, contains CATAL. The operand field is blank. The name of the macro to be cataloged is obtained from the operation field of the macro-definition prototype statement.

For the catalog function, SYSRDR must be assigned to a card reader, SYSIPT must be assigned to a card reader or a tape unit, and SYSLST can be assigned to a printer. If SYSIPT is a tape unit, this program assumes that the tape is positioned to the first input record. The tape is not rewound at end of job.

Control card input for the catalog function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (MMAINT), followed by

2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSIPT, SYSRDR, and SYSLST if a printed status report is desired. The ASSGN cards are followed by

3. The EXEC control card, followed by

4. The CATAL control card(s), followed by

5. The END control card, which is the last control card of the job.

Card input, read from the device assigned to SYSIPT, is as follows.

1. Macro-definition header statement (MACRO), followed by

2. Macro-definition prototype statement, followed by

3. Macro instructions, followed by

4. Macro-definition trailer statement (MEND).

If SYSIPT and SYSRDR are assigned to the same device, the macro card input immediately follows the CATAL control card.

Delete

The delete function is used to delete references to specific macros from the macro library. Any number of macros can be deleted during a single run. The macros are not physically deleted from the library; rather, the entry describing the macro in the macro directory is deleted.

The DELET control card is used to delete macros from the macro library. The DELET control card is punched in one of the following formats.

```
// DELET macroname

// DELET ALL
```

The first format is used when a specific macro is to be deleted. The entry in the operation field, separated from the two slashes by at least one blank, is DELET. The entry, macroname, in the operand field is separated from the operation field by at least one blank. It represents the name of the macro to be deleted. The name of the macro has a maximum length of five characters, the first of which must be alphabetic.

The second format is used if the entire library is to be deleted. The entry in the operand field is ALL.

Any number of DELET control cards can be used for the macro library within a single JOB run.

For the delete function, SYSRDR must be assigned to a card reader, and SYSLST can be assigned to a printer.

Control card input for the delete function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (MMAINT), followed by

2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR and SYSLST if a printed status report is desired. The ASSGN cards are followed by

3. The EXEC control card, followed by

4. The DELET control card(s), followed by

5. The END control card, which is the last control card of the job.

Rename

The rename function is used to change the name of a macro in the macro library. This is achieved by changing the entry in the macro directory.

The RENAM control card is used for the rename function. After the card is processed, the system recognizes only the new macro name. The RENAM card, read from the device assigned to SYSRDR, is punched in the following format.

```
// RENAM oldname,newname
```

The entry in the operation field, separated from the two slashes by at least one blank, is RENAM. The operand field entries, oldname and newname, are separated from the operation field by at least one blank. They represent the old macro name and the new macro name. The two entries in the operand field must be separated by a comma. The name of the macro has a maximum length of five characters, the first of which must be alphabetic.

Any number of RENAM control cards can be used for the macro library within a single JOB run.

For the rename function, SYSRDR must be assigned to a card reader, and SYSLST can be assigned to a printer.

Control card input for the rename function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (MMAINT), followed by

2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR, and SYSLST if a printed status report is desired. The ASSGN cards are followed by

3. The EXEC control card, followed by

4. The RENAM control card(s), followed by

5. The END control card, which is the last control card of the job.

SERVICE FUNCTIONS

To request a service function for the macro library, use the following JOB control card.

// JOB MSERV

One or more of the three service functions can be requested within a single JOB run. Any number of macros can be acted upon in this run.

Display

The display function is used to get a printout of a macro in the macro library. Any number of macros can be displayed within a single JOB run.

When displaying a macro from the macro library, the printed output is in EBCDIC. The first line contains the macro name and the name field, if used, of the prototype statement. The next group of lines contains the parameters specified in the prototype statement. The succeeding lines contain the macro itself.

The DSPLY control card is used to display macros in the macro library. It is punched in one of the following formats.

```
// DSPLY macro1,macro2,...,macron

// DSPLY ALL
```

The first format is used if only particular macros are to be displayed. The entry in the operation field, separated from the two slashes by at least one blank, is DSPLY. The entry in the operand field, _macron_, is separated from the operation field by at least one blank. It represents the name of the macro to be displayed. If more than one macro is to be displayed, the macro names are separated by a comma. The name of the macro has a maximum length of five characters, the first of which must be alphabetic. Continuation cards are not recognized.

The second format is used if the entire macro library is to be displayed. The entry in the operation field is DSPLY. The entry in the operand field is ALL.

For the display function, SYSRDR must be assigned to a card reader, and SYSLST must be assigned to a printer.

Control card input for the display function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (MSERV), followed by

2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR and SYSLST. The ASSGN cards are followed by

3. The EXEC control card, followed by

4. The DSPLY control card(s), followed by

5. The END control card, which is the last control card of the job.

Punch

The punch function is used to convert a macro in the macro library into a punched-card output deck. The punched-card deck consists of a CATAL card, a macro-definition header statement, a macro-definition prototype statement, one or more model statements, and a macro-definition trailer statement. The deck is exactly like the input card deck that was used when the macro was cataloged, except that additional blanks that were suppressed when the macro was cataloged are not reinserted, and the deck is reidentified and resequenced.

Any number of macros in the macro library can be punched within a single JOB run.

The PUNCH control card is used to convert macros in the macro library to punched-card output. The PUNCH control card is punched in one of the following formats.

```
// PUNCH macro1,macro2,...,macron

// PUNCH ALL
```

The first format is used if only particular macros are to be punched. The entry in the operation field, separated from the two slashes by at least one blank, is PUNCH. The entry in the operand field, _macron_, is separated from the operation field by at least one blank. It represents the name of the macro to be punched. If more than one macro is to be punched, the macro names are separated by a comma. The name of the macro has a maximum length of five characters, the first of which must be alphabetic. Continuation cards are not recognized.

The second format is used if the entire macro library is to be punched. The entry in the operation field is PUNCH. The entry in the operand field is ALL.

For the punch function, SYSRDR must be assigned to a card reader, SYSOPT must be assigned to a card punch, and SYSLST can be assigned to a printer.

Control card input for the punch function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (MSERV), followed by
2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR, SYSOPT and SYSLST. The ASSGN cards are followed by
3. The EXEC control card, followed by
4. The PUNCH control card(s), followed by
5. The END control card, which is the last control card of the job.

Display and Punch

The display and punch function is used to combine the separate operations of the display function and the punch function. The output of the display and punch function is identical to that described in the two preceding subsections. Any number of macros in the macro library can be displayed and punched within a single JOB run.

The DSPCH control card is used to convert macros in the macro library to printed and punched-card output. The DSPCH card is punched in one of the following formats.

```
// DSPCH macro1,macro2,...,macron

// DSPCH ALL
```

The first format is used if only particular macros are to be displayed and punched. The entry in the operation field, separated from the two slashes by at least one blank, is DSPCH. The entry in the operand field, macron, is separated from the operation field by at least one blank. It represents the name of the macro that is to be displayed and punched. If more than one macro is to be displayed and punched, the macro names are separated by a comma. The name of the macro has a maximum length of five characters, the first of which must be alphabetic. Continuation cards are not recognized.

The second format is used if the entire macro library is to be displayed and punched. The entry in the operation field is DSPCH. The entry in the operand field is ALL.

For the display and punch function, SYSRDR must be assigned to a card reader, SYSOPT must be assigned to a card punch, and SYSLST must be assigned to a printer.

Control card input for the display and punch function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (MSERV), followed by
2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR, SYSOPT, and SYSLST. The ASSGN cards are followed by
3. The EXEC control card, followed by
4. The DSPCH control card(s), followed by
5. The END control card, which is the last control card of the job.

LIBRARIAN FUNCTIONS: RELOCATABLE LIBRARY

This section describes the maintenance and service functions that relate to the relocatable library. Organization functions for the relocatable library are discussed in the section entitled Organization Functions.

MAINTENANCE FUNCTIONS

To request a maintenance function for the relocatable library, use the follcwing JOB control card.

```
// JOB RMAINT
```

One or more of the three maintenance functions can be requested within a single JOB run. Any number of modules can be acted upon in this run.

Catalog

The catalog function adds a module to the relocatable library. If a module exists in the relocatable library with the same name as a module to be cataloged, the module in the relocatable library is deleted. Input for the catalog function is from the device assigned to SYSIPT. A module in the relocatable library consists of input for a phase, a part of a phase or several phases, and is the output of one or more complete assembler runs.

Modules can be added to the relocatable library in an additional way. The assembler, through the use of the AFILE card, provides the capability of adding permanent or temporary modules to the relocatable library.

A module added to the relocatable library by way of the catalog function or the assembler permanent option can be removed by using the delete function.

A module entered as temporary by the Assembler is always simply overlaid by the next module entered into the relocatable library.

The CATAL control card is required to add a module to the relocatable library. The CATAL control card is read from the device assigned to SYSRDR and is punched in the following format.

```
// CATAL modulename
```

The operation field, separated from the two slashes by at least one blank, contains CATAL. The operand field contains one entry that is separated from the operation field by at least one blank. The entry, modulename, is the name by which the module will be known to the control system. The module name can be of any length; however, only a maximum of the first six characters is used by the system to assign the name to the module. Each module to be cataloged in the relocatable library must be preceded by a CATAL control card that supplies the name of the module.

The input for a relocatable-library module must be one or more complete object modules. The cards that may be included in the object module are described in the section entitled Linkage Editor. The cards are:

1. ACTION control statement
2. PHASE control statement
3. INCLUDE control statement
4. ESD card
5. TXT card
6. REP card
7. RLD card
8. END card
9. XFR card
10. ENTRY control statement or REND.

All these cards are read from the device assigned to SYSIPT and are written out in the relocatable library. If the user does not wish an ENTRY statement to follow the module in the library, the ENTRY card can be replaced by a special Librarian control statement with the operation code REND. The REND card signals the end of the module to the RMAINT program but is not written in the library. It has no name-field entry, the operation field (REND) begins at least one position to the right of column 1 (column 1 must be blank) and there is no operand.

For the catalog function, SYSRDR must be assigned to a card reader, SYSIPT must be assigned to a card reader or a tape unit, and SYSLST can be assigned to a printer. If SYSIPT is a tape unit, this program assumes that the tape is positioned to the first input record. The tape is not rewound at end of job.

Control card input for the catalog function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (RMAINT), followed by
2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSIPT, SYSRDR, and SYSLST if a printed status report is desired. The ASSGN cards are followed by
3. The EXEC control card, followed by
4. The CATAL control card(s) followed by
5. The END control card, which is the last control card of the job.

Delete

The delete function is used to delete references to specific modules on the relocatable library. Any number of modules can be deleted during a single JOB run. The modules are not physically deleted from the library; rather, the entry describing the module in the relocatable directory is deleted.

RPG programmers should make use of this function to avoid duplicate entries in the relocatable library. When recompiling an RPG object program which already exists as a relocatable library entry, the previous entry must be deleted prior to recompilation.

The DELET control card is used to delete modules from the relocatable library. The DELET control card is punched in one of the following formats.

```
// DELET modulename

// DELET ALL
```

The first format is used when a specific module is to be deleted. The entry in the operation field, separated from the two slashes by at least one blank, is DELET. The entry, modulename, in the operand field is separated from the operation field by at least one blank. It represents the name of the module to be deleted. The module name can be of any length; however, only a maximum of the first six characters is used by the system to locate the module.

The second format is used if the entire library is to be deleted. The entry in the operation field is DELET. The entry in the operand field is ALL.

Any number of DELET control cards can be used for the relocatable library within a single JOB run.

For the delete function, SYSRDR must be assigned to a card reader, and SYSLST can be assigned to a printer.

Control card input for the delete function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (RMAINT), followed by

2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR and SYSLST if a printed status report is desired. The ASSGN cards are followed by

3. The EXEC control card, followed by

4. The DELET control card(s), followed by

5. The END control card, which is the last control card of the job.

Rename

The rename function is used to rename a module in the relocatable library.

The RENAM control card is used for the rename function. After the card is processed, the system recognizes only the new module name. The RENAM card, read from the device assigned to SYSRDR, is punched in the following format.

```
// RENAM oldname,newname
```

The operation field, separated from the two slashes by at least one blank, contains RENAM. The operand field entries, oldname and newname, are separated from the operation field by at least one blank. They represent the old module name and the new module name. The two entries in the operand field must be separated by a comma. Both module names can be of any length; however, only a maximum of the first six characters is used by the system to locate and rename the module.

Any number of RENAM control cards can be used for the relocatable library within a single JOB run.

For the rename function, SYSRDR must be assigned to a card reader, and SYSLST can be assigned to a printer.

Control card input for the rename function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (RMAINT), followed by

2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR and SYSLST if a printed status report is desired. The ASSGN cards are followed by

3. The EXEC control card, followed by

4. The RENAM control card(s), followed by

5. The END control card, which is the last control card of the job.

SERVICE FUNCTIONS

To request a service function for the relocatable library, use the following JOB control card.

```
// JOB RSERV
```

One or more of the three service functions can be requested with a single JOB run. Any number of modules can be acted upon in this run.

Display

The display function is used to get a printout of a module in the relocatable library. Any number of modules can be displayed within a single JOB run. The printed output consists of a header and the module.

Contained in the printed header is the module name, the number of blocks needed to contain the module, and the address of the module in the relocatable library.

The printed output of the module is represented by hexadecimal characters and EBCDIC, depending on the type of record and the information contained within the record. The field identifying the type of loader card and the symbolic labels within the ESD type of record are printed in EBCDIC.

The DSPLY control card is used to display modules in the relocatable library. It is punched in one of the following formats.

```
// DSPLY mod1,mod2,...,modn

// DSPLY ALL
```

The first format is used if only particular modules are to be displayed. The entry in the operation field, separated from the two slashes by at least one blank, is DSPLY.

The entry in the operand field, modn, is separated from the operation field by at least one blank. It represents the name of the module to be displayed. If more than one module is to be displayed, the module names are separated by a comma. The module name can be of any length; however, only a maximum of the first six characters is used by the system to locate the module. Continuation cards are not recognized.

The second format is used if the entire relocatable library is to be displayed. The entry in the operation field is DSPLY. The entry in the operand field is ALL.

For the display function, SYSRDR must be assigned to a card reader, and SYSLST must be assigned to a printer.

Control card input for the display function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (RSERV), followed by
2. The ASSGN control cards, if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR and SYSLST. The ASSGN cards are followed by
3. The EXEC control card, followed by
4. The DSPLY control card(s), followed by
5. The END card, which is the last control card of the job.

Punch

The punch function is used to convert a module in the relocatable library into a punched-card output deck.

Any number of modules in the relocatable library can be punched within a single JOB run. The punched-card output is acceptable to every function that uses relocatable modules as input. Each module will be punched with a CATAL card (containing the name of the module) as the first card of the module. The last card of the module will be either an ENTRY card or a REND card.

The PUNCH control card is used to convert modules in the relocatable library to punched-card output. It is punched in one of the following formats.

```
// PUNCH mod1,mod2,...,modn

// PUNCH ALL
```

The first format is used if only specific modules are to be punched. The entry in the operation field, separated from the two slashes by at least one blank, is PUNCH. The entry in the operand field, modn, is separated from the operation field by at least one blank. It represents the name of the module to be punched. If more than one module is to be punched, the module names are separated by a comma. The module names can be of any length; however, only a maximum of the first six characters is used to locate the module. Continuation cards are not recognized.

The second format is used if the entire relocatable library is to be punched. The entry in the operation field is PUNCH. The entry in the operand field is ALL.

Any number of PUNCH cards can be used within a single JOB run.

For the punch function, SYSRDR must be assigned to a card reader, SYSOPT must be assigned to a card punch, and SYSLST can be assigned to a printer.

Control card input for the punch function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (RSERV), followed by
2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR, SYSOPT, and SYSLST. The ASSGN cards are followed by
3. The EXEC control card, followed by
4. The PUNCH control card(s), followed by
5. The END control card, which is the last control card of the job.

Display and Punch

The display and punch function is used to combine the separate operations of the display function and the punch function. The output of the display and punch function is identical to that described in the two preceding subsections. Any number of modules in the relocatable library can be displayed and punched within a single JOB run.

The DSPCH control card is used to convert modules in the relocatable library to printed and punched-card output. The DSPCH card is punched in one of the following formats.

```
// DSPCH mod1,mod2,...,modn

// DSPCH ALL
```

The first format is used if only particular modules are to be displayed and punched. The entry in the operation field, separated from the two slashes by at least one blank, is DSPCH. The entry in the operand field, modn, is separated from the operation field by at least one blank. It represents the name of the module that is to be displayed and punched. If more than one module is to be displayed and punched, the module names are separated by a comma. The module names can be of any length; however, only a maximum of the first six characters is used by the system to locate the module. Continuation cards are not recognized.

The second format is used if the entire relocatable library is to be displayed and punched. The entry in the operation field is DSPCH. The entry in the operand field is ALL.

For the display and punch function, SYSRDR must be assigned to a card reader, SYSOPT must be assigned to a card punch, and SYSLST must be assigned to a printer.

Control card input for the display and punch function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (RSERV), followed by
2. The ASSGN control cards, if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR, SYSOPT, and SYSLST. The ASSGN cards are followed by
3. The EXEC control card, followed by
4. The DSPCH control card(s), followed by
5. The END card, which is the last control card of the job.

LIBRARIAN FUNCTIONS: DIRECTORIES

This section describes the service function, display, that relates to five directories. The organization functions for these directories are discussed in the section entitled Organization Functions.

To request the display service function for a directory (core image directory, relocatable directory, macro directory, system directory, or transient directory), use the following control card.

```
// JOB DSERV
```

The display function is used to print the status of the directories defined for the system. Any number of directories can be displayed within a single JOB run.

The printed display for each directory will appear as follows.

1. The name of the directory.
2. A heading line with field headings that describe the contents of the directory.
3. The contents of the directory. If any entry in the core image or relocatable library is temporary, an asterisk is printed to the left of the phase, or module name.

The DSPLY control card is used to display specific directories or all directories.

It is punched in one of the following formats.

```
    // DSPLY dir1,dir2,...,dirn

    // DSPLY ALL
```

The first format is used if only specific directories are to be displayed. The entry in the operation field, separated from the two slashes by at least one blank, is DSPLY. The entry in the operand field, dirn, is separated from the operation field by at least one blank. It represents the name of the directory to be displayed. It can be:

CD	for the core image directory
RD	for the relocatable directory
MD	for the macro directory
SD	for the system directory
TD	for the transient directory

If more than one directory is to be displayed, the symbols for the directories must be separated by a comma and can be in any order.

The second format is used if all five directories are to be displayed. The entry in the operation field is DSPLY. The entry in the operand field is ALL.

For the display function, SYSRDR must be assigned to a card reader, and SYSLST must be assigned to a printer.

Control card input for the display function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (DSERV), followed by
2. The ASSGN control cards, if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR and SYSLST. The ASSGN cards are followed by
3. The EXEC control card, followed by
4. The DSPLY control card(s), followed by
5. The END control card, which is the last control card of the job.

ORGANIZATION FUNCTIONS

An organization function is requested by the use of a JOB card. The operand field of the JOB card contains one of three possible operands, depending on the kind of organization function required.

The kinds of organization functions are:

1. Reallocation
2. Library condense
3. Copy system.

The reallocation function is used to redefine the number of tracks a library, directory, or area is to contain.

The library condense function is used to reorganize a library to minimize vacancies between phases, modules, or macros in that library.

The copy system function is used to copy either selectively or completely , the system residence.

REALLOCATION

The reallocation function is used to redefine the number of tracks allotted to the libraries, directories, checkpoint area, and the volume (label control card) area on a disk resident system. Any number of these areas can be reallocated within a single JOB run.

The reallocation function can be used to increase, decrease, or eliminate specified areas in the disk resident system. Each library and directory on system residence is condensed. The JOB control card required to perform a reallocation function is punched in the following format.

```
          // JOB AORGZ
```

The control cards, VOL, DLAB, and XTENT are required for the reallocation job. The VOL control card is punched in the following format.

```
          // VOL SYSRES,SYSRES
```

The DLAB control card is punched in the following format.

```
// DLAB 'BOS 8K DISK (33 blanks) lssssss', c

              0001,yyddd,yyddd,'0000000000000'
```

The operand ssssss is the volume serial number of the disk pack. The operands yyddd,yyddd are the creation date and expiration date of the "BOS 8K DISK" file.

The XTENT card is punched in the following format.

```
// XTENT 1,000,0000001,cccchhh,'ssssss',SYSRES
```

The operand cccchhh is the highest address of the extent to be reserved for system residence. The operand ssssss is the same volume serial number used in the DLAB card.

Associated with the reallocation job is the TRCKS control card. The TRCKS control card is punched in the following format.

```
// TRCKS id,no,id,no,...
```

The entry in the operation field, separated from the two slashes by at least one blank, is TRCKS. The entries in the operand field are separated from the operation field by at least one blank. The operand field can contain any even number of operands, up to sixteen. The entry, id, represents the library, directory, or area to be reallocated and can be any of the following.

CL	for the core image library
RL	for the relocatable library
ML	for the macro library
CD	for the core image directory
RD	for the relocatable directory
MD	for the macro directory
CP	for the checkpoint area
VL	for the volume (label control card) area

no in the operand field is an integer that represents the number of tracks to be allocated.

The operands can appear in any order as long as each pair of operands is related. Operands are separated by a comma. When reallocation is being performed, only the areas being changed need appear on the control card. Consider this example.

```
// TRCKS ML,5,CL,5,RL,3,RD,1
```

In the example, the macro library would contain five tracks; the core image library, five tracks; the relocatable library, three tracks; and the relocatable directory, one track.

For the reallocation function, SYSRDR must be assigned to a card reader, and SYSLST can be assigned to a printer.

Control card input for the reallocation function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (AORGZ), followed by
2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR and SYSLST if the printed status report is desired. The ASSGN cards are followed by
3. The VOL, DLAB, and XTENT control cards for SYSRES, followed by
4. The EXEC control card, followed by
5. The TRCKS control card, followed by
6. The END control card, which is the last control card of the job.

LIBRARY CONDENSE

The library condense function is used to reorganize a library to minimize vacancies between phases, modules, or macros in their respective libraries. One or more of the three libraries can be condensed within a single JOB run. Use the library condense function whenever a number of vacancies have accumulated within a library, thus affecting the ability to catalog a new phase, module, or macro into the particular library.

The JOB control card required to perform a library condense function for any library is punched in the following format.

```
// JOB LORGZ
```

Associated with the library condense JOB control card is the CONDS control card. The CONDS control card is punched in one of the following formats.

```
// CONDS ALL

// CONDS lib,lib,lib
```

The first format is used if all three libraries are to be condensed. The entry in the operation field, separated from the two slashes by at least one blank, contains CONDS. The entry in the operand field, separated from the operation field by at least one blank, contains ALL.

The second format is used if only specific libraries are to be condensed. The operation field, separated from the two slashes by at least one blank, contains CONDS. The operand field is separated from the operation field by at least one blank. It can contain one, two, or three operands, separated by commas. The entry, lib, in the operand field represents one of the following values.

CL	for the core image library
RL	for the relocatable library
ML	for the macro library.

Entries in the operand field can be in any order. Any number of the three libraries of the disk resident system can be condensed within a single JOB run.

Consider the following examples.

```
// CONDS CL

// CONDS ML,RL

// CONDS RL,CL,ML
```

In the first example, only the core image library will be condensed. In the second example, both the macro library and the relocatable library will be condensed. In the third example, all three libraries will be condensed.

For the library condense function, SYSRDR must be assigned to a card reader, and SYSLST can be assigned to a printer.

Control card input for the library condense function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (LORGZ), followed by
2. The ASSGN control cards, if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR and SYSLST. The ASSGN cards are followed by
3. The EXEC control card, followed by
4. The CONDS control card(s), followed by
5. The END control card, which is the last control card of the job.

COPY SYSTEM

The copy system function is used to copy, either selectively or completely, the directories and libraries of the system residence. Along with the actual copying, the number of tracks allocated to libraries, directories, label control card area, and checkpoint area of the new system pack can be redefined.

The device number of the disk pack on which the disk resident system is to be copied is assigned to SYS000. The ASSGN card precedes the EXEC card in the device assigned to SYSRDR. SYSLST can be used to print additional diagnostics.

By using the TRCKS card, described in the subsection entitled Reallocation, the number of tracks allocated to the label control card area, the checkpoint area, libraries, and the directories can be respecified, if required, for the new system. The TRCKS card in no way affects the old system. If the number of tracks allocated to the new libraries and/or directories is not to differ from the old system, no TRCKS card is required. If a TRCKS card is used, it must include the allocation for each area in the new system, regardless of whether it differs from the old system. The new system must contain at least the three areas: core image directory, core image library, and label control card area.

A complete copy consists of copying each directory and library on the resident pack. The checkpoint and label control card areas are not copied, but are allocated. The selective copy consists of copying only particular libraries, or specific phases, modules, or macros within libraries (see Appendix K and Appendix L for names). When either kind of copy (complete or selective) is performed, all libraries are automatically condensed.

The JOB card required for a copy-system function is punched in the following format.

```
// JOB CORGZ
```

The control cards, VOL, DLAB, and XTENT are required for the copy job. The VOL control card is punched in the following format.

```
// VOL SYS000,SYSRES
```

The DLAB control card is punched in the following format.

```
// DLAB 'BOS 8K DISK (33 blanks) 1ssssss', c

            0001,yyddd,yyddd,'0000000000000'
```

The operand ssssss is the volume serial number of the disk pack. The operands yyddd,yyddd are the creation date and expiration date of the "BOS 8K DISK" file.

The XTENT card is punched in the following format.

```
// XTENT 1,000,0000001,cccchhh,'ssssss',SYS000
```

The operand cccchhh is the highest address of the extent to be reserved for system residence. The operand ssssss is the same volume serial number used in the DLAB card.

Associated with the copy system job is the COPYS control card. The COPYS control card is punched in one of the following formats:

```
// COPYS id,name1,name4-name7,...,namen

// COPYS ALL

// COPYS id,ALL
```

The first format is used when only selected elements of specific libraries are to be copied. The operation field, separated from the two slashes by at least one blank, contains COPYS. The first entry, id, in the operand field is separated from the operation field by at least one blank and can be one of the following:

CL	for the core image library
RL	for the relocatable library
ML	for the macro library.

The other entries, namen, in the operand field are the name(s) of the phases, modules, or macros that are to be copied. All entries in the operand field must be separated by commas. Several elements of a particular library can be specified on one control card.

If the order in which the elements are present in the particular library is known, then a series of elements can be copied by specifying the name of the first and last element to be copied and separating the two names by a hyphen (11-punch) instead of a comma. Continuation cards are valid if:

1. The operand entries are contiguous to column 71.
2. A continuation character is punched in column 72.
3. At least the first two operands appear on the first card.
4. The first character in the next card is punched in column 16.

For example, the first card could be:

```
                                  cols.
                                    7|7
                                    1|2
// COPYS CL,NAMEA,NAMEC,...,NAMX
```

and the second card could be:

```
col.
  1
  6
  EM,NAMEQ,NAMES-NAMEW,NAMEZ
```

Each library that is to be selectively copied requires a separate group of COPYS control cards. All elements to be selectively copied from one library must be specified before elements of another library are specified. The selective copy function will not go back and forth from one library to another. When specific elements in a library are to be selectively copied rather than the entire library, a limit of 351 items can be specified for each library. The number of items is calculated as follows.

Each specific operand on the COPYS card is counted as one item, except

1. The library identifier is not counted.
2. Two names separated by a hyphen (11-punch) are counted as three items.

Only permanent elements on a resident system can be copied. If a temporary item is specified by name, a not-found error will result.

The second format is used when the entire system is to be copied. The entry in the operation field, separated from the two slashes by at least one blank, is COPYS. The entry in the operand field, separated from the operation field by at least one blank, is ALL.

The third format is used when an entire library is to be copied. The entry in the operation field, separated from the two slashes by at least one blank, is COPYS. The entry in the operand field, id, separated from the operation field by at least one blank, can be CL, RL, or ML. The second entry in the operand field is ALL.

For the copy-system function, SYSRDR must be assigned to a card reader, SYS000 must be assigned to a disk pack, and SYSLST can be assigned to a printer.

Control card input for the copy-system function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card (CORGZ), followed by
2. The ASSGN control cards if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR, SYS000, and SYSLST. The ASSGN cards are followed by
3. The VOL, DLAB, and XTENT control cards for SYS000, followed by
4. The EXEC control card, followed by
5. The TRCKS control card (optional), followed by
6. The COPYS control card(s), followed by
7. The END control card, which is the last control card of the job.

LIBRARIAN FUNCTIONS--DEVICE TABLE

This section describes the service functions that relate to the device table of a Supervisor. The device table contains an entry called a Physical Unit Block (PUB) for each symbolic unit that will be used in the system. This device table is referred to as the PUB table. To request a service function for the PUB table use the following JOB control card:

```
// JOB PSERV
```

Three service functions are available through the use of the PSERV (PUB service) program.

- Display the PUB table (current device assignments) from main storage.
- Display the PUB table (permanent device assignments) from the core image library on disk.
- Assign symbolic units to the PUB table in the core image library.

Display

The DSPLY control card is used to display the contents of the PUB table. It is punched in one of the following formats.

```
// DSPLY source,SYSxxx,...,SYSxxx
```

or

```
// DSPLY source,ALL
```

The first format is used when specific physical unit blocks (PUBs) are to be displayed. The second format is used when the entire PUB table (all physical unit blocks) is to be displayed.

The entry in the operation field, DSPLY, is separated by at least one blank column from the two slashes in columns 1 and 2 of the card. The first entry in the operand field is separated from the operation field by at least one blank. When more than one entry is used in the operand field, the entries are separated by a comma. Blank columns are not permitted within entries or between entries in the operand field.

The first operand for both formats, source, indicates the location of the PUB table to be displayed. The first operand must be CORE, if the PUB table in main storage is to be displayed. The display shows the current symbolic unit assignments for the Supervisor that resides in main storage at the time the display is made.

If the PUB table in the core image library (on the system resident disk pack) is to be displayed, the first operand must be DISK. The display would show the symbolic unit assignments that were made when the Supervisor was assembled or when assignments were established by the assign function of the PSERV program. If both the main storage and the core image library PUB tables were displayed, any difference between the two displays would be due to assignments made after the IPL procedure had been executed.

The first format is used when specific physical unit blocks (PUBs) are to be displayed. The entries, SYSxxx, in the operand field following the source entry represent the specific symbolic units to be displayed. Any of the following are valid.

SYSRES

SYSRDR

SYSLST

SYSIPT

SYSOPT

SYSLOG

SYS000-SYS254

If more than one symbolic unit is to be displayed, the SYSxxx entries can be punched in any order. Continuation cards can be used when more than one card is necessary to contain the operand field.

The second format is used if the entire PUB table (all symbolic units) is to be displayed. The second operand is ALL.

The display function requires that a card reader be assigned to SYSRDR, and a printer be assigned to SYSLST.

Control card input for the display functions, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card PSERV, followed by
2. The ASSGN control cards, if the current assignments are not those required. The ASSGN cards that can be used are SYSRDR and SYSLST. The ASSGN cards are followed by
3. The EXEC control card, followed by
4. The DSPLY control card(s), followed by
5. The END control card, which is the last control card of the job.

Assign

The assign function is used to make permanent assignments to the PUB table in the Supervisor that resides in the core image library. The PUB table in main storage is not affected by the PSERV program assign function. Device assignments made by the assign function will have no effect on processing operations until the Supervisor (including the modified PUB table) is loaded into main storage via an IPL procedure.

The assign function permits changes to the core image library PUB table that would otherwise require reassembly of the Supervisor.

The ASSGN control card is used to assign a specific device adddress to the symbolic unit used. The ASSGN card is read from the device assigned to SYSRDR. It is punched in the following format.

```
// ASSGN SYSxxx,X'cuu',dd,X'ss'
```

Refer to the section entitled ASSGN Card for an explanation of the entries in the operand field. All ASSGN card formats acceptable to the Job Control program can be used with the assign function of the PSERV program.

For the assign function, SYSRDR must be assigned to a card reader, and SYSLST must be assigned to a printer.

Control card input for the assign function, read from the device assigned to SYSRDR, is as follows.

1. The JOB control card PSERV, followed by
2. The ASSGN control cards required for making the proper assignment to SYSRDR and SYSLST in the main storage PUB table. If the current assignments are those that are required, these cards are not necessary. The ASSGN cards are followed by
3. The EXEC control card, followed by
4. The ASSGN control cards for changing the symbolic unit assignments in the PUB table that resides in the core image library.
5. The END control card, which is the last control card of the job.

SYSTEM GENERATION

The disk-resident system is received on either a magnetic tape or a disk pack which contains the core image, relocatable, and macro libraries. If the disk system is received on tape, it must be copied onto a disk before the system generation procedure can begin.

The core image library contains the Supervisor, Job Control, Linkage Editor, Librarian, and Assembler. All programs in the core image library are edited to run with the IBM-supplied Supervisor.

During system generation, the IBM-supplied programs in the relocatable library can be cataloged and link-edited to run with a Supervisor that is adapted to the configuration of the individual installation. Among these programs are Job Control, Linkage Editor, and Librarian.

The system received by the user is capable of immediate operation. Most installations, however, generate Supervisors adapted to their configurations. Also, system libraries may be edited according to the needs of different installations. When this process is completed, the newly created system replaces the system that the user received.

Briefly, the system generation procedure is as follows: The user codes a set of Supervisor macro instructions describing his system configuration. The assembly of these macro instructions results in a new Supervisor. Subsequently, IBM-supplied programs in the relocatable library are cataloged to operate with the newly assembled Supervisor.

The complete system generation procedure is described in the System Generation and Maintenance manual as listed on the front cover of this publication.

INTRODUCTION

The Data Management section of this manual introduces the reader to the functions and concepts of data files and data organizations for tape and disk files. The organization of this section is shown in Figure 23. Included in this section is an introduction to the Input/Output Control System (IOCS) and basic data file processing techniques provided within the basic operating system IOCS library.

BOS provides the user with several automatic functions for management of external storage (card files, tape files, and the 2311 disk-storage files, etc.). The input/output operations associated with external storage files are included in the IOCS routines available with BOS.

Management of input/output operations and the related data files involves:

- File Organization
- Record Loading and Retrieval
- Blocking and Deblocking of Records

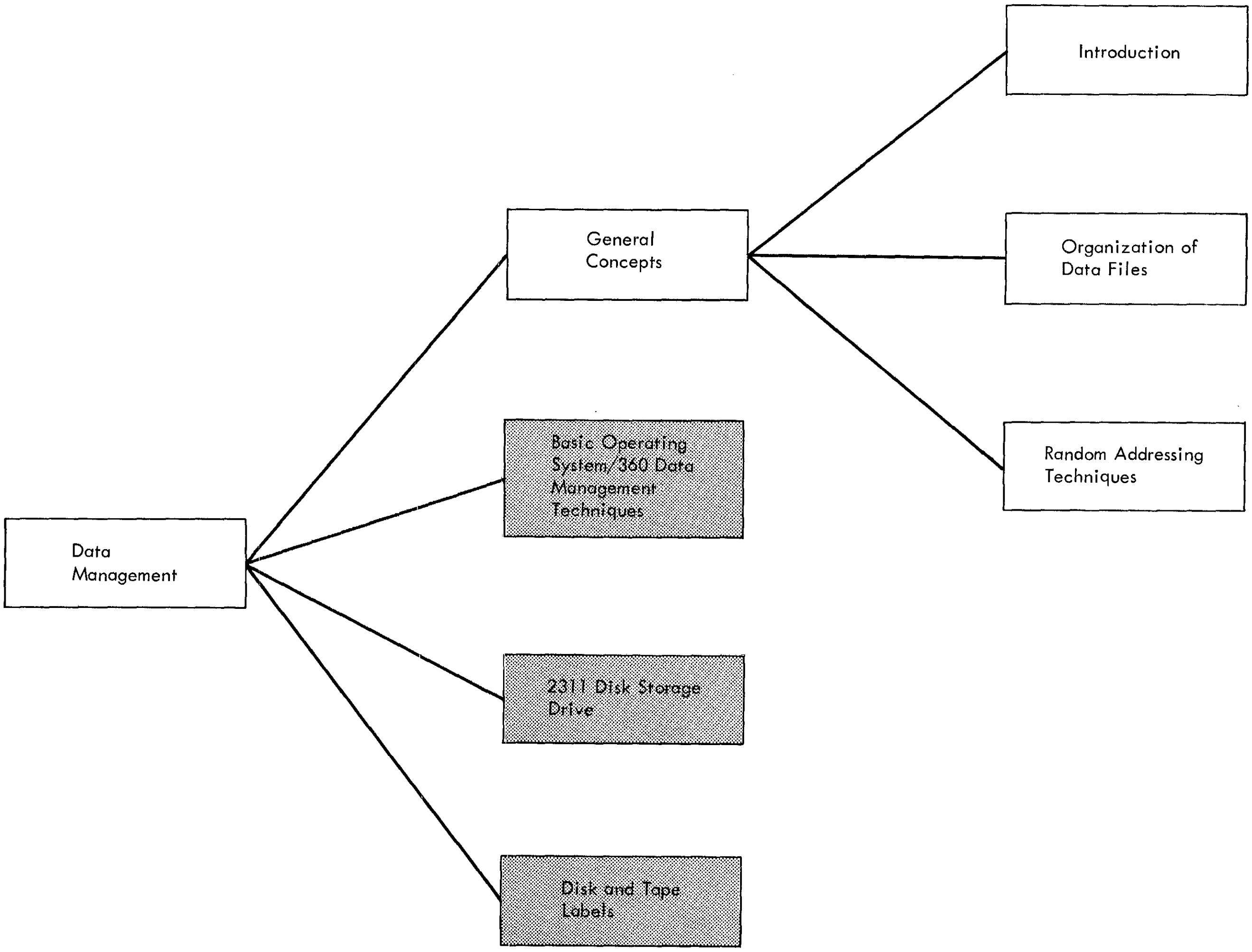

Data Management
General Concepts
Basic Operating System/360 Data Management Techniques
2311 Disk Storage Drive
Disk and Tape Labels
Introduction
Organization of Data Files
Random Addressing Techniques

Figure 23. Data Management: General Concepts

- Scheduling and Control of Channel I/O Program

- Handling of Input/Output Interruption and Error Conditions

- Label Writing and Checking

THE INPUT/OUTPUT CONTROL SYSTEM (IOCS)

The Input/Output Control System capability provided by BOS consists of two parts:

1. Physical IOCS - the physical I/O routines incorporated in the Channel Scheduler portion of the Supervisor.

2. Logical IOCS - the logical I/O routines assembled from macro instructions written in the problem program.

Physical IOCS controls the actual transfer of physical records between external storage devices (cards, tape, disk, etc) and main storage. A physical record is that amount of data actually read into main storage or written from main storage as the result of an input/output command. Physical IOCS routines perform the following functions.

- Scheduling I/O requests on each Channel (queueing)

- Starting I/O operations

- Handling interruptions associated with I/O operations

- Handling error conditions for devices supported.

The physical IOCS routines are incorporated in the Channel Scheduler portion of the Supervisor (Part 2: The System Control Programs has a discussion on the Channel Scheduler).

Logical IOCS controls those functions that a user would have to perform to locate a logical record for processing. A logical record is one unit of information in a file of similar units of data. For example, one employee's record in a master payroll file, or one part-number record in an inventory file, would be considered logical records. One or more logical records are contained within one physical record. The term logical IOCS refers to the routines that perform the following functions:

- Blocking and deblocking records

- Switching between I/O areas when two areas are specified for a file

- Handling end-of-file and end-of-volume conditions

- Label writing and checking.

Logical IOCS uses physical IOCS to execute I/O commands whenever it determines that a transfer of data is required. For example, if a file consists of blocked records and a block has been read into main storage (Figure 24), logical IOCS merely makes each record in succession available to the user, until the end of the block is reached. No physical IOCS is required. When logical IOCS determines that the last record in the block has been processed, however, it requests physical IOCS to start an I/O operation to transfer the next physical record into main storage. In the illustration, only logical IOCS (LIO) is required to make records 2 and 3 (and 5 and 6) available for processing. Physical IOCS (PIO) is required to transfer records 4, 5, and 6 as one block of records into main storage, then logical IOCS makes records 4, 5, and 6 available in succession to the problem program.

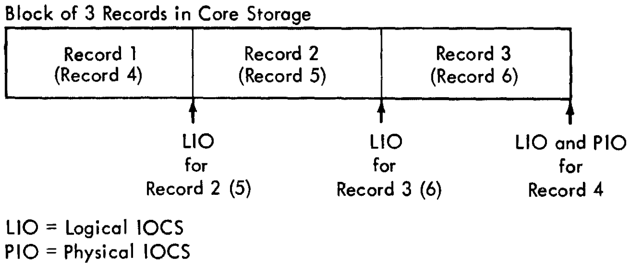


Figure 24. Physical IOCS vs Logical IOCS

Both logical IOCS macros (such as GET, PUT, READ, WRITE) and physical IOCS macros (such as EXCP and WAIT) are available to the programmer for handling records. The logical IOCS macro routines cause all the functions of both logical and physical IOCS to be performed for the programmer. The user can completely bypass the logical IOCS functions by using the physical IOCS macro instructions EXCP, WAIT and CCB in his problem program. Logical IOCS provides four sets of routines for processing records.

The consecutive-processing routines are used to read, process, and write successive records in a logical file. These routines apply to all files in serial-type I/O devices. The card readers, card punches, magnetic tape units, paper tape readers, and printers are serial-type devices. Also, these routines can be used for 2311 disk file records to be processed in a serial manner. The basic macros used in these routines are GET and PUT.

The Direct Access Method (DAM) provides READ, WRITE, and WAITF macro instructions for processing disk files only. The DAM macro instructions can be used to create and maintain logical files in random order. Although the DAM technique will usually be used for randomly organized files, in some situations serial-type files on disk can be processed with the DAM macros. There are options in this technique that permit sequentially processing serial files with Key Areas.

The Indexed Sequential File Management System (ISFMS) routines are provided to take advantage of the flexibility of the IBM 2311 Disk Storage Drive. These routines provide a comprehensive method for establishing and maintaining logical files in a manner which allows random and sequential processing in the same file. Quite often more than one type of processing is required on the same data file. For example, records in a file may require sequential processing for month-end reports, and these same records can be randomly processed for daily updating.

Macro routines are provided for sending and receiving data to and from a number of STR (Synchronous Transmitter Receiver) devices. STR is a specific mode of data transmission, and all STR devices use identical data transmission codes and line control procedures. The STR routines provide READ/WRITE level macro instructions to simplify use of these devices. There is also a macro instruction to convert to and from the standard STR transmission code [fixed count four-out-of-eight (4/8)].

The macro routines used for BSC (Binary Synchronous Communication) support provide for sending and receiving data in a CPU-to-CPU communications environment. Using strict line control procedures, the BSC macro routines provide READ/WRITE level macro instructions to simplify the processing of data in this teleprocessing environment.

Macro Instructions for I/O Routines

Two types of macros are considered when a program is being written to process records in a file.

1. The declarative file-definition macro instructions are written at the beginning of the problem program to describe the file, indicate the type of processing required, and specify main-storage areas needed for processing the file.
2. The imperative macro instructions are written into the problem program to provide linkage to the routines described by the declarative file-definition macro instructions.

The file-definition macro instructions are provided for the user to tailor his programs for his particular file processing needs. Five file-definition macros, DTFSR, DTFDA, DTFIS, DTFRF, and DTFSN are available for defining logical IOCS routines. One file-definition macro DTFPH is provided when the user wishes to use the physical IOCS macro instructions EXCP, WAIT, and CCB, and his files have standard labels to be processed.

At program-assembly time, logical IOCS routines are generated to conform to the specifications written by the programmer in his file-definition macro statements.

The imperative macro statements written by the programmer are assembled into linkage routines that communicate with routines already generated by the file-definition macros. These imperative macros, such as OPEN, CLOSE, GET, PUT, READ, WRITE, etc, are used to perform the following functions:

1. Activate files for processing; this includes label checking.
2. Deactivate files after processing is completed.
3. Make logical records available for processing (deblocking).
4. Assemble logical records for output (blocking).
5. Alternate I/O areas (when two I/O areas are used).
6. Transfer records from main storage to external files.
7. Perform control operations, such as, rewind tape, stacker-select cards, or seek disk file tracks.

For details on writing file-definition macro statements and imperative macro statements, refer to the BOS Assembler with Input/Output Macros publication, listed in the Preface of this manual.

Labels

To ensure that the correct logical file is used for each job, it is common practice to identify all logical files with labels recorded on the magnetic surface of a disk pack or a tape. This internal label is in addition to a printed label on the outside surface of the pack or on the tape reel. The internal labels provide a means for computer-controlled file protection and identification. Labels are required on all 1316 disk packs to be processed by a System/360.

BOS provides IOCS routines for label writing and checking. There are essentially two basic labels - the Volume label and the File label. The volume label uniquely identifies the volume (a reel of tape or a disk pack), and the file label contains information applicable to a given data file or portion of a data file stored on a particular volume.

DISK-FILE ORGANIZATION

There are many advantages to direct-access storage in a data processing system. The IBM 2311 Disk Storage Drive has greatly increased these advantages. The flexibility of this unit allows several kinds of processing within the same system. The advantages offered by one kind of processing are often quite different from those offered by another. For example, in a random-processing application, the advantages center around the ability to process only those records that have current transactions, rather than to process all of a file. In a second case, the advantage may be the ability to process one or more sequential files against a single sequenced input.

File organization is the key to effective use of disk storage. The objective of disk-file organization is the systematic storing of information in disk storage in such a manner that records can be retrieved in the quickest way possible, while maintaining the over-all processing objectives of the system.

The method of organization best suited to a particular file of disk records depends upon many factors. These factors must be analyzed for each file in any one particular application. Often, more than one organization scheme can be considered for the same file. In one application, records could be processed purely at random; in another, the same records could be processed in sequence by various control fields. For example, records within a file might be processed at random during an updating run and sequentially within certain groups, such as branch office or due date, when producing reports or billing. A file such as this would be analyzed to determine whether it should be organized:

1. Purely randomly, thus keeping process time at a minimum during one run but destroying the advantage of the sequential nature of the other.

2. Sequentially, thus minimizing the time required to produce reports but increasing updating time.

3. Randomly for updating, and then sorted into sequence for reports.

The decision would depend on the nature of the file. Questions such as the following might be asked:

1. Can transactions be batched and sorted before processing, or must they be processed as they occur?

2. Is the activity distributed throughout the file in such a manner as to warrant passing the entire file when updating?

3. Would the processing time saved by sorting warrant the time and effort required?

Questions of this kind must be asked of each file in an installation. In choosing organization methods, the over-all processing objectives of the system must be kept in mind at all times.

ORGANIZATION OF DATA FILES

This section is written for the reader who seeks a general understanding of the functions and concepts of direct-access storage devices and data organization. The discussion is general and informal. It does not define the rules for using the basic operating system. It does attempt to explain the basic concepts of data organization and define some of the terms that can be encountered in related literature.

The experienced programmer may find this section of interest, but it is written primarily for the reader whose experience with tape and disk files has been limited.

LOGICAL FILE VS PHYSICAL UNIT

To understand file organization, it is important to distinguish between a logical file and the physical unit used to store the file. A logical file is a group of related data records, such as a payroll file (one record for each employee, showing his rate of pay, deductions, etc), a customer file (one record for each customer, showing his address, credit limit, unpaid balance, etc), or an inventory file (one record for each inventory item, showing cost, selling price, number in stock, etc). A physical unit used for storage of data records could be an IBM 2400 Magnetic Tape Unit or an IBM 2311 Disk Storage Drive. Also, an IBM 2540 Card Reader can be considered as a physical unit when data records punched into cards are being read into the system.

DATA FILES AND RECORDS

Data files stored in such media as paper, cards, tapes, or disk storage devices, are encountered in practically every business activity. These files provide the basis for most manual, mechanical, and electronic data processing. Data files are composed of a number of individual records ranging from a few records up to thousands or millions of records.

A record can be defined as a collection of information comprised of alphameric and/or non-alphameric characters related to a common identifier. The common identifier is known as a record's control field or its key. Usually one of the prime information elements (fields) present within a record is used to identify the record. For example, man number could be used as the key or identifier for a payroll record, and policy number could be the key of an insurance policy file.

The size or length of records will vary from file to file, and the size can range from a single character up to thousands of characters.

A single record usually includes one or more logical data fields; a data field is a sequence of one or more characters treated as a processing unit of information. An individual data field is normally identified by its location within a record.

The logical structure of records and of fields within records has become increasingly important since the advent of computers and high-speed recording media such as magnetic tapes and disks. This logical structure is strongly affected by whether a record is of fixed or variable length.

Fixed-Length Records

In fixed-length record files, all records are allocated the same number of character storage positions. Identical data fields are present in every record, whether they are used or not. Each data field is identified by its position within the record. The control field (key) is usually the first field present in a record, but this is not a fixed rule. Fixed-length record files are the most straightforward files to process, and are most frequently encountered.

In many applications the use of fixed-length records would make inefficient use of file-storage space. For example, consider a file having a maximum record length of 850 positions. Assume that the average record length is 230 positions and the minimum length is only 100 positions. It is readily apparent that a fixed record length of 850 positions would cause many storage positions to be wasted. Situations such as this require the development of space-saving techniques based on varying the number of storage positions allocated to data records.

Variable-Length Records

Completely variable-length records are sometimes developed for more efficient use of storage. In this approach the data portion of the record may be of any length, but the key (control-field) size is constant. The length of a variable-length record is indicated by a record-length character-count field present in each record.

Note: If variable length records are to be read or written on 7-track tape, the data-conversion special feature is required.

Fixed-length and variable-length records can be processed on the System/360 tape or disk files; however, some of the data management methods do not provide for handling variable-length records. This will be discussed under Data Management Techniques.

Blocking Records

The length of individual data records will vary with the type of data and the application requiring such data. The design of the format of a data record is very significant to the efficient use of the various storage media available on the System/360. One very important element in the design of data records involves what is commonly called blocking and deblocking. Input/output units (storage media) are relatively inefficient when used to record short blocks of information. To increase the efficiency of input/output units, data records are assembled into blocks of records whose size is convenient and efficient for processing. Each physical record on either tape or disk requires interrecord gaps. These gaps are blank areas used to distinguish beginning and ending points of a record. If records are blocked prior to loading onto a tape or disk, many of these gaps can be eliminated. The average number of reads required to locate a record can usually be reduced by increasing the blocking factor (number of records per block). The greater the blocking factor, the greater the chance that the next record required will be in the same block. This is an important consideration when designing jobs that involve file-searching either on tape or disk. It is particularly important when using disk-storage techniques that develop overflow records. Overflow records occur when there are more items assigned to a disk track than can be stored on that track.

Blocked records normally require the use of more main storage than unblocked records. This results from the need for main storage to house the block of records being read from or written onto a storage device. Also more main storage is required to hold blocking-and-deblocking program instructions.

A given input/output device usually contains several times as much information in blocked records than in unblocked records. BOS IOCS macro instructions are designed to handle the blocking and deblocking of records so that the user need only design the most efficient blocking factor for his particular data file and equipment specifications.

Several types of records can be stored on tape or on direct access storage devices. The following record types are described in the section Types of Records.

- Fixed-length unblocked records
- Fixed-length blocked records
- Variable-length unblocked records
- Variable-length blocked records

Schematics of these types of records are shown in Figures 27, 28, 29, and 30.

FILE ORGANIZATION AND PROCESSING

Data records should be organized and stored in a manner that will facilitate subsequent processing. The relationship between file organization and the processing of data must be carefully considered to achieve the most efficient use of the System/360 components. This is particularly important when designing data files for storage in a direct-access storage device such as the IBM 2311.

File Organization

There are two basic types of file organization: Sequential and random.

Sequential File Organization - The logical sequence of records in a sequential file depends upon significant control information (the record key) appearing in the records. A sequentially organized file is established by arranging records into sequential order on the storage media used to contain the file.

There are two basic kinds of sequential file organization:

1. Sequential files organized in a serial manner;

2. Sequential files organized through the use of indexes.

The first kind of sequential file organization (serial) provides for records with successively higher record keys to have successively higher locations within the file. Cards and tape files are usually organized in this serial manner. This kind of file is usually considered as one continuous string of records in record key sequence and is usually processed consecutively. Disk records can also be organized in a serial manner and processed consecutively.

Additions to and deletions from a sequential file can be handled in several ways. In some cases it is possible to batch additions and deletions, and merge them into the file during a regular updating run. This is the method required when adding to and deleting from sequential files such as those stored on magnetic tape.

The second kind of sequential file organization involves the use of indexes to establish and process data on a direct access storage media. Because of the manner in which the indexes are used when adding records to an established file, it is not always necessary to store records in consecutive locations. The Indexed Sequential File Management System (ISFMS) provided by the basic operating system uses a method that permits additions without a reorganization of the established file. Special overflow references are set up in a track index record, and each record in the overflow area has a special sequence-link field used to maintain file sequence. This technique is described in the section on ISFMS.

Random Organization - Generally, in a random file organization, the records are not stored in the sequence of their control numbers (keys). A randomizing formula is used to convert the record key (control number) to a numerical address (physical address). The record is stored at the physical address developed by the randomizing formula.

File-Processing

There are three basic file-processing techniques: consecutive, random, and sequential.

Consecutive Processing - Every record in a file is examined, and each successive record in the file is processed in order. For example, card records are processed in the order cards are fed into the system. Usually only those files that have been organized in some logical sequence will be processed consecutively.

Random Processing - The sequence of processing has no relationship to the sequence of data stored in the file.

To find a record at random in an indexed file, an index (or series of indexes) is first scanned to localize the area of search. The index is a sequential list of the keys of selected data-file records with corresponding physical addresses. To find a record at random in a random file, the physical address is computed by the same randomizing formula used to load the file of records. A direct access of the record can be made, and no index tables are required.

Sequential Processing - The sequence of processing disk files is identical to the sequence of logical record storage as opposed to physical location. If the records are stored in man-number order, transactions affecting the records are pre-sorted into man-number order and processed in that sequence. Because the records are on a direct-access storage device, only records with transaction activity need be scanned.

The ISFMS technique provided by BOS can be used to organize a sequential file with indexes. Then, ISFMS can be used to process records in this file in both random and sequential order. This technique is described in the section on the Indexed Sequential File Management System.

SEQUENTIAL VS RANDOM ORGANIZATION

Time, storage space, and cost considerations are the most-often-used criteria in comparing file organizations. The significance of each factor must be weighed against the application and system-design goals. An advantageous sequential organization for one job may prove to be a disadvantage in another job. For example, the saving of several hundred positions of main storage is of small impact, if ample main storage is available. Similarly, the reduction of the time required to process a single transaction by 50 to 60 milliseconds means little if only a thousand transactions (approximately a minute) are handled daily.

The advantages and disadvantages of sequential and random file organization should be studied in a particular application environment before a worthwhile evaluation can be made.

Some of the advantages and disadvantages of sequential and random file organization are:

Sequential Advantages

1. Less direct access storage space is required in most cases, even considering that 2 to 10% of the space available is required for storage of the necessary indexes.
2. In most cases, both sequential and random transactions can be handled effectively.
3. Sorting is not required for output listings that are in the record-key sequence.
4. Visual checking of the stored records is facilitated because output is in a meaningful sequence.
5. Conversion to direct access storage is simplified because most files are already in sequence. Retention of this sequence avoids radical departure from the methods familiar to people outside the data processing activity.

6. A detailed study of the structure of the control information used to identify individual records is not required, if the existing sequence is retained.

7. Precision monitoring of the conversion is made feasible at installation time, if data loaded into direct access storage is maintained in the same sequence as the source data file.

Sequential Disadvantages

1. Indexes may require additional main storage.

2. When processing is being done randomly, processing time for a single transaction is greater than with random organization because index file processing requires additional computer time.

3. When processing is being done randomly, average access time is greater than with random organization; due to need for accessing one or more indexes before locating required record.

4. Whenever <u>any</u> type of processing is being done for an Indexed Sequential file, all packs in a multi-pack file must be on-line.

Random Organization Advantages

1. Will require less main storage in some applications.

2. Random transactions can be processed in less time because access time is faster--no index file accessing, and processing.

Random Organization Disadvantages

1. Dynamic files can require frequent file reorganization, especially when processing time is a major consideration.

2. Development of address conversion randomizing routine, along with record-key analysis, is required for implementation.

3. All packs in a multi-pack file must be on-line for random processing.

RANDOM-ADDRESSING TECHNIQUES

There are two basic elements involved with file addressing:

1. The file of records, which must be stored and retrieved in a data processing system.

2. The direct access storage device itself.

The data records that must be stored in a direct access storage device are usually identified by a control field, such as part number, and employee number. Normally the numbers or characters in the control field are unevenly distributed. For example, a seven-position control field may be used to identify 25,000 items in a parts master file. However, with a seven-position number, it is possible to identify ten million items. In this example, only 0.25% of the available numbers are used.

The direct access storage devices, on the other hand, are usually composed of physical locations that are identified by an evenly distributed set of numbers. The addressing problem is to convert an unevenly distributed set of numbers to an evenly distributed sequential set of numbers within the address limits of the direct access device. Many addressing techniques have been developed to accomplish this task. In choosing a technique for address conversion, it is important to remember that an ideal distribution of control field throughout the range of the control fields is a completely uniform one. Uniform distribution means that the difference between any pair of successive control fields taken in ascending order is constant.

The worst distribution of control fields is a random one. There is no way to transfer from random keys to addresses with better than random distribution. In practice, purely random control field sets and completely uniform ones are both rare. A data file is likely to have control fields that distribute in groups or clusters of irregular length and separation. This kind of grouping of numbers introduces a degree of <u>uniformity</u>. The irregular length and separation of the number groups implies a degree of <u>randomness</u>. A well chosen conversion technique produces an address set that reflects both elements and has a distribution intermediate between random and uniform. To be ideal for use in direct access storage devices, the conversion technique should produce a unique storage address for every record in a file. This is seldom possible. Most control-field conversion routines result in assigning some address to more than one record. These duplicate addresses are sometimes referred to as <u>synonyms</u>. The conversion routine selected should convert the control fields (keys) of the records in a data file to a series of addresses with a minimum number of synonyms and within the desired storage address range. The following sections discuss briefly the most successful conversion routine. After that discussion, the handling of synonyms will be discussed.

RANDOM-ADDRESSING FORMULA

The simplest method of file organization is that in which a unique disk address is obtained from the control data of each record. This is referred to as the random-addressing method. If the control numbers of a set of data records in a file are consecutive numbers without gaps, they may be converted to disk addresses by simple arithmetic. For example, if the account numbers for a customer file run from 10000 to 17563 (7564 account numbers), and ten account records can be stored on each disk track, 757 tracks are needed. By subtracting 10,000 from an account number and then dividing by 10, a numeric address in the range 000 to 756 is obtained. To place this file on an IBM 1316 Disk Pack, starting at track address 1200 (cylinder 120 head 0) a constant 1200 is added to the quotient and a constant (1) is added to the remainder. This constant (1) is required, because record zero (R0) of each track is reserved to facilitate the handling of defective recording areas that may occur during the life of the disk pack. Using this approach, a record containing the data for account number 16349 would be stored at track reference 1834 in record-reference ten, calculated as follows:

16349 - 10000 = 6349
6349 ÷ 10 = 634 with remainder 9
634 + 1200 = 1834 = track reference
9 (remainder + 1) = 10 = record reference

When processing this file randomly, any record can be found with a single seek. When it is possible to process sequentially, only one seek is needed per cylinder. Record retrieval time is thus at a minimum. This is an optimum situation, and it rarely occurs in actual practice.

Normally, the control data of a file of records can seldom be used directly as disk addresses. If a file has no control fields that can be used directly as disk addresses, it is sometimes possible to preassign addresses. For example, the item number 513XP could become 513XP-13472, which could then be converted to a track and record reference as shown above.

PRIME NUMBER DIVISION

If the control fields of a file of records are not consecutive or contain numerous unused numbers, as is usually the case, the random-addressing technique under the topic Random Addressing Formula makes inefficient use of the storage locations. All possible numbers are assigned locations, and those numbers not used leave empty record areas in the storage unit. Files established with control numbers composed of coded information usually have a much higher potential range of items than is required for storage. To handle this situation, an initial conversion is made on the control numbers to reduce the range to a practical size. This conversion is often referred to as randomizing.

Randomizing generally refers to the techniques developed to convert a set of control numbers with numerous unused numbers to a tightly packed set, to result in very few unused storage areas. There are many techniques used for this conversion of numbers: folding, extracting, squaring, and radix transformation are a few of the techniques in use. One method, sometimes called prime number division* or divide remainder, is adaptable and usually satisfactory for converting a file of numbers. Its merits have been established by mathematical analysis and by testing an actual file.

To illustrate the prime-number-division technique, suppose the customer file in the example under the topic Random Addressing Formula used a coded control number of ten digits. The first three could be a geographical code (branch office number), the next two could describe the nature of the business, the next one could be a size-of-customer code, and the final four could be sequentially assigned within class.

Thus, account number 139 457 0307 would be the 307th account assigned branch office 139. It would belong to a customer-of-size code 7 in industry class 45. Because this ten-digit number cannot be used efficiently to describe 7564 accounts, it is converted by dividing by the closest prime number to number of storage locations available. Assume 10,000 locations available, then divide by 9973. The remainder is used as the control number and a technique similar to the example under the topic Random Addressing Formula is used to calculate a track and record reference.

----

* A prime number is a number divisible only by itself, or one.

1394570307 ÷ 9973 = 139834

with a remainder of 5825.

This remainder is then operated on as in the example, under the topic Random Addressing Formula, assuming 10 records per track.

5825 ÷ 10 = 582 with a remainder of 5.

To load the file starting at track address 1100, add 1100 to 582 for a sum of 1682. The track reference for this record is 1682, and the record is the sixth record on the track (remainder of 5 + 1 = 6).

To summarize prime number division:

1. Select a divisor equal to or greater than the number of records to be stored (ten to twenty percent greater is recommended and explained later). The best divisors are primes. No even numbers or multiple of five should ever be used - divisors must end in 1, 3, 7, or 9. The divisor is called the range.

2. Divide the control number by the range and use the remainder to generate the track address.

Prime-number division will always work; that is, it will always convert control numbers into the desired range. This is because in division, the remainder is always less than the divisor and the highest valued remainder is the divisor -1. Dividing any number, no matter what size, by a desired range, always produces remainders in the desired range. Using a prime number as the divisor will usually result in relatively few duplicate remainders, and therefore relatively few address synonyms.

A prime number is not always the best choice of divisor for a given set of keys. Also, it is not necessarily true that all primes produce equally good results. Primes do, however, avoid serious maldistribution and may be safely used with little analysis of the control-field set of the data files.

DISCONTINUOUS BINARY NUMBER

The track and record reference field used to identify records on a 2311 disk drive is a discontinuous binary number. This number has the form CCHHR, where CC is the cylinder number, HH is the head number, and R is the record number. The track reference example calculated in the preceding section (1682 with a remainder of 5) must be developed in such a way that the cylinder, head, and record numbers can be placed in positions CC, HH, and R, respectively, as discontinuous binary numbers. The cylinder number (168 in this example) is placed in the cylinder, CC, field as a binary number, (hexadecimal 00A8). The head number (2 in this example) is placed in the head, HH, field as a binary number, (hexadecimal 0002). The record number (in this example the remainder 5 plus 1) is placed in the record, R, field as a binary number, (hexadecimal 06). Then the track and record reference field appears as:

CC		HH		R
00	A8	00	02	06

OVERFLOW RECORDS

The transformation of record control fields to direct access storage device addresses usually produces some overflow (synonyms) records.

The file organization used with a data file employing indirect addressing (addresses converted by a random addressing formula) must be able to accommodate the synonyms or duplicate addresses.

The first consideration in organizing the file is sometimes called the packing factor. The number of synonyms produced by a random-addressing conversion routine can be reduced by assigning more disk storage space than is actually required by the file. The percentage of the file area actually used for records is called the packing factor. The packing factor for an efficiently organized file can vary from 65% to 95%. A packing factor of 80% usually proves to be a good point to start. After all efforts have been made to design a file-conversion technique with few overflows, an approach to handling the remaining overflows must be chosen.

One such technique will be discussed as an example. This technique is often referred to as the chaining method. As each record is read into the computer for loading into a direct storage access device, its control field or key is converted to a physical address. These converted addresses are called home addresses.

Note: The home addresses discussed here are not directly related to the track Home Address (see IBM 2311 Disk Storage Drive) used to control the physical operation of the 2311 disk drives. These home addresses are related to the record identifier (ID) that is discussed in Reference Methods.

The first record converted to a particular address is stored in the home address location. The additional records converted to this address are stored in overflow locations. The address of the first overflow location is stored in the home address location. The address of the second overflow location is stored in the first overflow location, etc. Chaining requires that in the home address and all overflow locations, space be reserved for the address of the next location or link in the chain (Figure 25).

Retrieval of records is accomplished by converting the control information (record key) to the home address. The record in the home address location is read into main storage, and its control information is compared to that of the record being sought. If the control fields are not equal, the address of the first overflow record is extracted from the home record, and another read command is issued using this address. The process is repeated until the desired record is found.

Several additional techniques exist for handling overflow records in a random file organization. Also, there are many techniques that can be used to make the technique mentioned previously more efficient for a particular application.

Two common techniques used to solve the overflow problem are similar in concept to the chaining method but do not require a chaining field to be present in each record:

1. Preassigned tracks overflow technique. When there is no room to store a record in its home location, a specific preassigned overflow track is used. The overflow track(s) should be defined to be in the same cylinder containing the home location, to reduce the number of seeks required to locate a record.

2. Consecutive spill overflow technique. In this technique when overflow occurs, a sequential search is made starting at the next record within the cylinder until an empty record storage location if found. If the last track in a given cylinder overflows, a return is made to the first track. This technique does not require the use of a track overflow (chaining) field, nor is a seek required to locate the overflow record (must be on same cylinder).

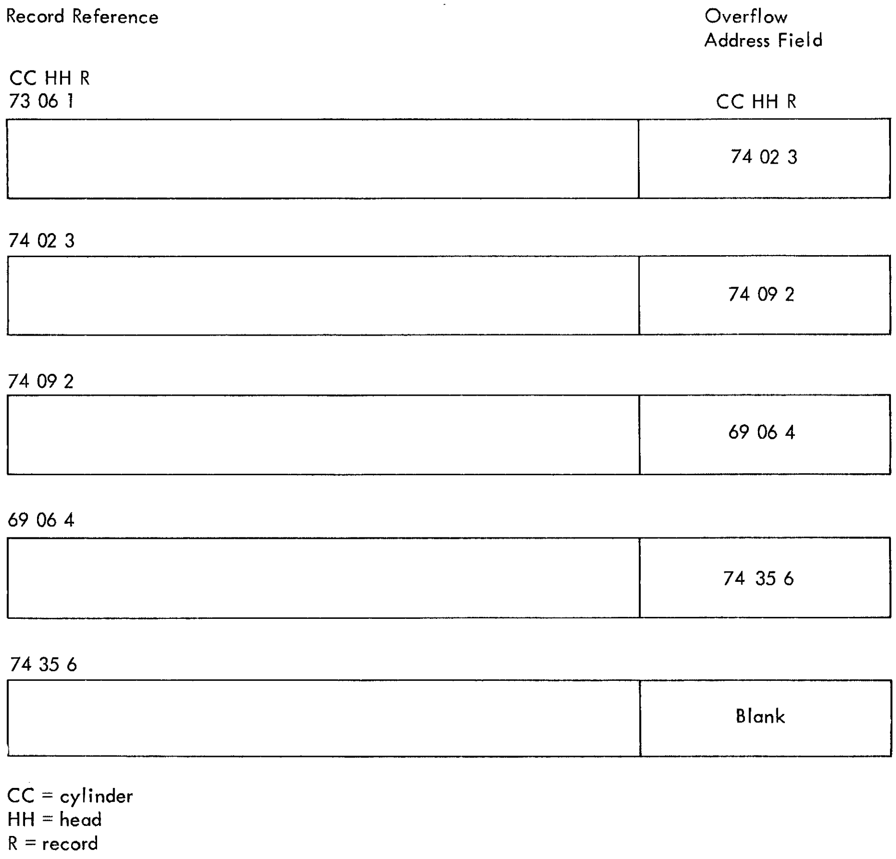


Figure 25. Direct Access Address Chaining

Another technique can be used in which overflow chaining fields are established only for each track (requires only one overflow address on home track). A version of this technique can be used with sequential files, when both random and sequential processing are appropriate to a particular application. A series of indexes is used, instead of attaching the chain linkage to the home track. The lowest level of index will have two entries for a specified group of records. This group of records is usually the records stored on one disk track. The index is then called a track index. The first entry on the track index refers to the first track of data records in the main (prime) section of the logical file. The term prime area is used to distinguish the main section from the area used for overflow records. The second entry on the track index is used to link the overflow records to the records on the prime track. When a request for a record is processed, the indexes are scanned to localize the area of search. The track index (or lowest level) will point directly to the proper track, either in the prime area or the overflow area. This technique can be programmed to handle additions and deletions effectively, with reorganization of the logical file kept to a minimum.

BOS provides a comprehensive IOCS file-management technique for organizing and processing records in a manner similar to the one just discussed.

This IOCS technique is described in the section entitled Indexed Sequential File Management System.

Trailer Records

In some applications, efficient file organization and processing requires that logical data files be separated into master and trailer records. Master records are the basic data records of the file. Trailer records are extensions of the master records and are maintained in a different area of disk storage as a separate file.

Trailer records may be utilized by user-supplied routines. They are not supported by any IBM subroutines or macros.

BASIC OPERATING SYSTEM DATA MANAGEMENT TECHNIQUES

The management of external storage devices (card files, tape files, and the 2311 disk storage device) has been described in general terms in preceding sections of this manual.

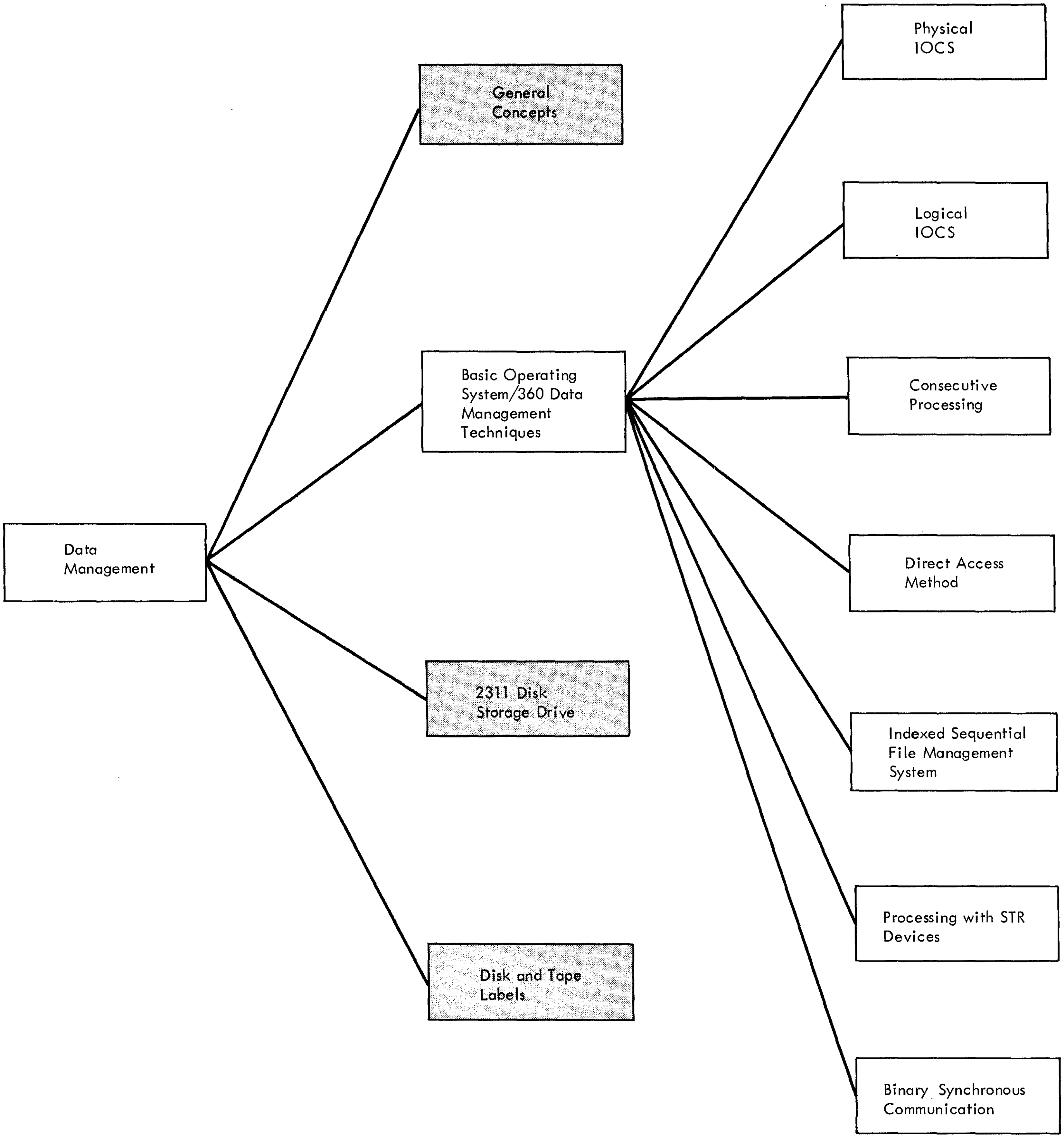

Data Management
General Concepts
Basic Operating System/360 Data Management Techniques
2311 Disk Storage Drive
Disk and Tape Labels
Physical IOCS
Logical IOCS
Consecutive Processing
Direct Access Method
Indexed Sequential File Management System
Processing with STR Devices
Binary Synchronous Communication

Figure 26. Data Management: Basic Operating System Data Management Techniques

This section contains fundamental information on how BOS can be used to manage data stored on external files. Figure 26 shows the organization of this section. This capability for data handling is provided by I/O macro instructions (IOCS) included with the Assembler. Any kind of file organization can be handled by one of the logical IOCS routines, or the physical IOCS macro instructions and user programming. For specific instructions on writing the necessary IOCS macro instructions, refer to the BOS Assembler with Input/Output Macros publication, listed in the Preface of this manual.

PHYSICAL IOCS

Physical IOCS consists of input/output (I/O) routines that handle the actual transfer of data records between external storage devices (cards, tape, disk, etc.) and main storage. It has the capability for handling any external devices supported by BOS. Other devices can be handled when the user provides the necessary error routines.

The user can tailor the physical IOCS routines to handle the complete I/O configuration used in his installation. All I/O operations for all programs to be processed in a particular installation can be handled by the physical IOCS routines contained in one program Supervisor. This facility is part of the capability provided by BOS to allow a system to operate in a stacked job environment. See Job Control and FETCH for other parts of the stacked job capability.

Physical IOCS Functions

Incorporated in the Channel Scheduler portion of the Supervisor, are program routines for handling the following functions.

- Starting an I/O operation when requested, or queueing the request for the appropriate channel, if the channel is busy.
- Processing any interruptions associated with an I/O operation.
- Starting an I/O operation for next request on the channel queue when the preceding I/O operation has been completed.
- Processing any I/O errors that occur, by first retrying the I/O operations that have the possibility of being corrected by a retry operation. If errors cannot be corrected in this manner, the problem program is notified by a flag, or the operator is informed by a message.
- Scheduling all I/O operations and handling events associated with I/O interruptions.

Processing with Physical IOCS

There are occasions when the user may need to bypass the logical IOCS routines in order to handle a particular I/O data file. Physical IOCS macro instructions allow the user unlimited processing flexibility. This capability allows the user the facility of requesting only those I/O commands that he needs for a particular situation. This permits the user to write his own logical IOCS for blocking and deblocking, his own channel command words for the channel program, and other user routines he may require.

To aid the programmer in using physical IOCS, an assembler instruction statement CCW - define channel command word, is provided. This CCW assembler instruction statement is a convenient means to define and generate the eight-byte channel command words needed for the channel program.

Four macro instructions are available to the programmer for direct communication with physical IOCS.

CCB can be used to create a command control block. The command control block provides information needed by the channel scheduler for execution of the EXCP and WAIT macros.

EXCP can be used to request physical IOCS to Start I/O (execute channel program).

WAIT can be used to have the problem program wait in a programmed waiting loop until an I/O operation is completed.

WAITM can be used to have the problem program wait in a programmed waiting loop until one of several specified I/O operations is completed.

The WAITM macro instruction is discussed under: Processing with STR Devices.

The file-definition macro DTFPH (Define The File for Physical IOCS) is provided for the user to define files that have standard labels to be checked. If the user chooses to have physical IOCS check his standard file labels, he uses the DTFPH for file definition, and the imperative macros OPEN and CLOSE and FEOV (magnetic tape only) to handle label checking and creation.

An I/O operation can be traced through physical IOCS in the following manner.

A request is made to physical IOCS to start an I/O operation by means of the EXCP macro instruction in the problem program. Physical IOCS determines from information in the CCB, the channel for which the request was made and places the request on a queue for that channel. If the channel is not busy, the I/O is started and control is returned to the problem program. If the channel is busy, control will be returned to the problem program, but the I/O request waits in the channel queue. When the request reaches the top of the channel queue, the I/O will then be started.

The problem program will be interrupted when the I/O operation is complete (all data transferred to or from main storage and the external device and no permanent errors have been detected). At this point the request is removed from the channel queue.

If an error was detected that could not be corrected by the device-error routines, the problem program or the computer operator would be notified. User error routines can be notified to handle conditions such as wrong-length record.

Physical IOCS always attempts to perform its function so that the time of executing an I/O operation is overlapped with the I/O operations on other channels and also allows the I/O operations to be overlapped with processing.

LOGICAL IOCS

Logical IOCS is provided to perform the functions of handling the logical data records that physical IOCS transfers to or from main storage and external storage devices.

When logical IOCS is specified in a problem program, the physical IOCS routines are controlled by logical IOCS. Logical IOCS routines always use physical-level macro instructions to communicate with the channel programs. Therefore the user does not need to be concerned with physical IOCS if he chooses to use logical IOCS to perform I/O operations.

Functions of Logical IOCS

The functions of logical IOCS can be categorized into the following topics.

Record Retrieval. Logical records can be retrieved randomly or sequentially from a data file. The retrieval method depends on file-organization and processing techniques used.

Record Output. IOCS routines are provided for creating new files of data. This function involves writing completely new records on an output media such as disk or tape. Records can also be punched into blank cards or printed.

Record Updating. Records can be retrieved from a file, changes made to the data within the records, and the records returned to the same locations from which they were retrieved. This capability is available for the 2311 disk storage drive and for card equipment that provides for reading and punching into the same card. The term combined files is used to refer to card files that are punched with additional information after the cards have been read.

Control Operations. Macro instructions are provided by logical IOCS to perform certain machine-control functions. For example, card stacker selection, printer line spacing, and form skipping, tape rewinds, disk seek operation, etc are performed by the control macro instructions.

End-of-Volume and End-of-File Procedures. Logical IOCS provides all the routines required for detecting and handling end-of-volume and end-of-file conditions. This includes for example, standard label checking and creating routines; provision for entering non-standard label routines provided by the user; provision for automatically switching to a new volume (tape reel or disk pack); and provision for entering an end-of-file routine provided by the user.

File Organization. A technique for complete file management is provided by logical IOCS. The technique, Indexed Sequential File Management System (ISFMS), provides both random and sequential processing capability.

Additional data file loading and processing capabilities are provided by the Consecutive Processing macro instructions and the Direct Access Method (DAM).

Types of Records

Logical IOCS provides techniques for handling records that are:

- Fixed-length. All records have the same number of bytes of data.
- Variable-length. Each record can have a different number of bytes of data.
- Unblocked. Only one logical record in each physical record.
- Blocked. Two or more logical records in one physical record.
- Undefined. The record characteristics are not specified to IOCS.

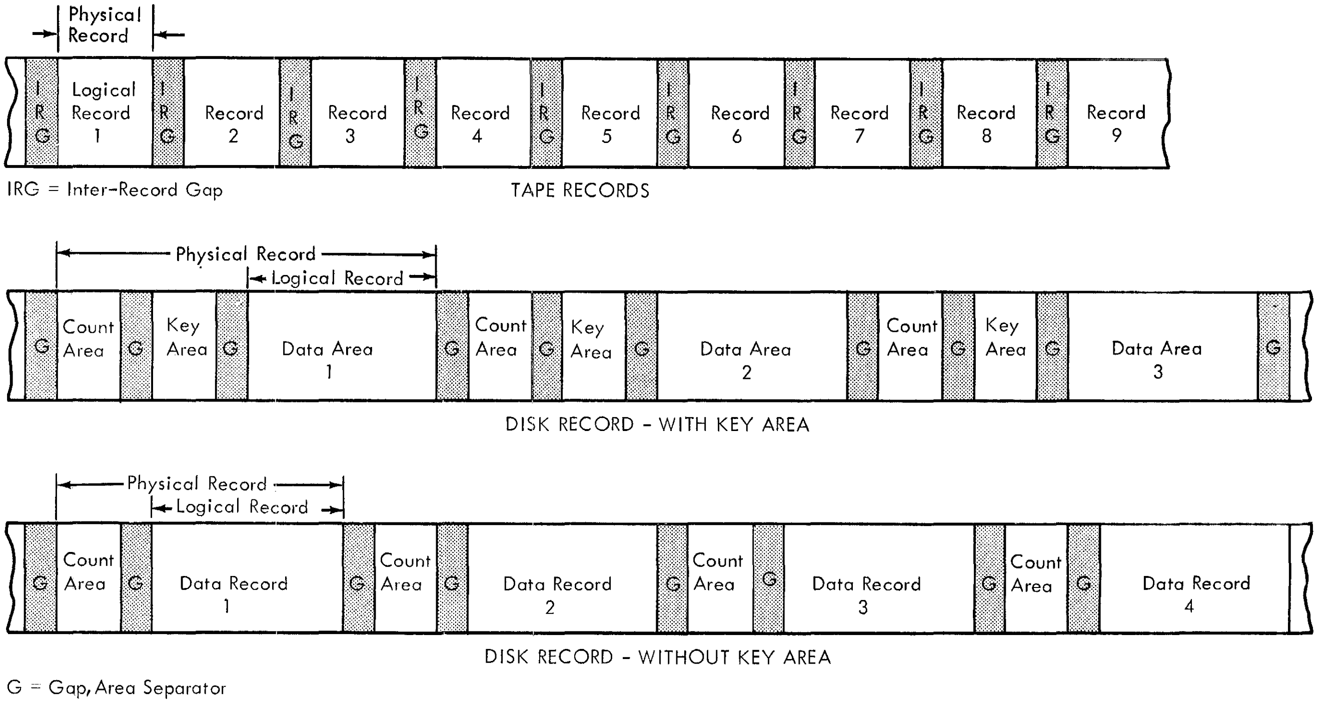


Figure 27. Fixed-Length Unblocked Record Format

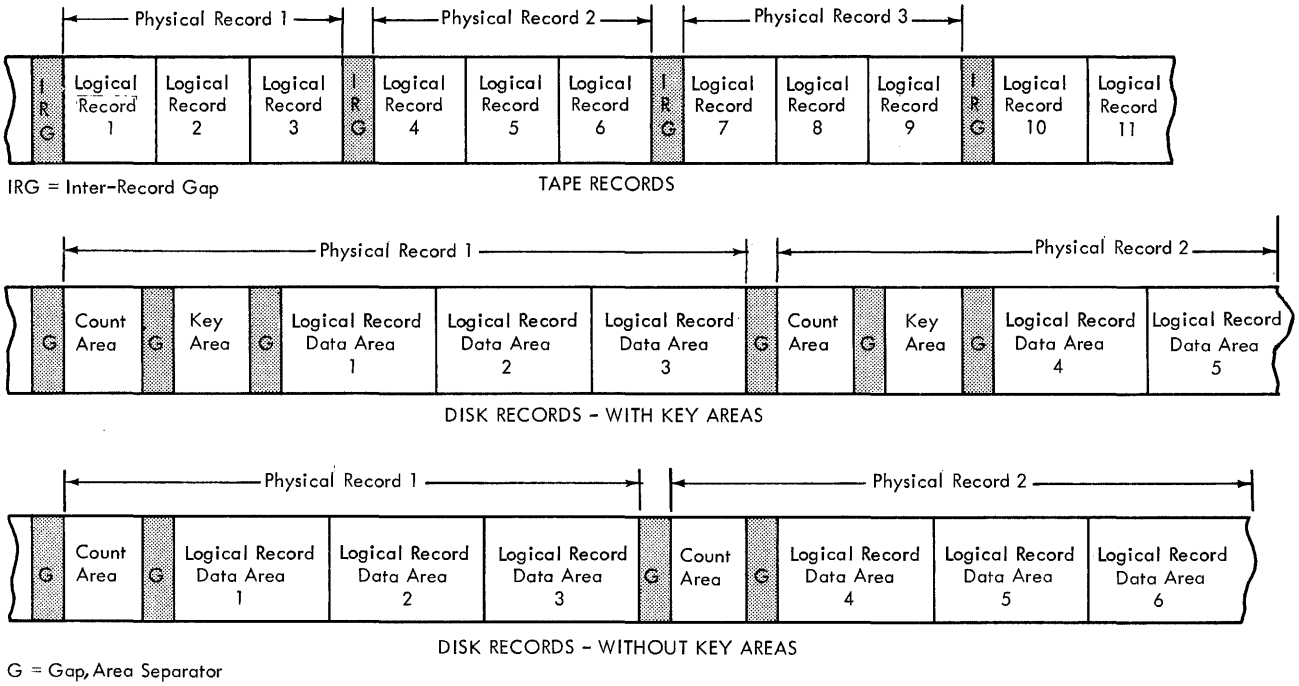


Figure 28. Fixed-Length Blocked Record Format

Fixed-length, unblocked (Figure 27). Each logical record is the same length as the physical record. The disk record formats, with key area and without key area, are discussed in the section in Direct Access Method.

Fixed-length, blocked (Figure 28). Blocked records are usually considered to be two or more logical records within one physical record. The number of records in each block (blocking factor) is usually kept constant. For example, the illustrations in Figure 28 show blocked records with a blocking factor of 3; there are three logical records within each block (physical record). However, within the basic operating system macro instructions, there is a TRUNC macro that permits writing a short block of records. This macro (TRUNC) can be used at the end of a category of records to write a block that has fewer records than normal. This short block may contain one logical record. See the TRUNC (truncate) macro instruction in the Assembler with Input/Output Macros publication, listed in the Preface of this manual. The TRUNC macro is supported only by consecutive processing and applies to both tape and disk blocked records.

Variable-length, unblocked (Figure 29). Each physical record contains one logical record, and the records can vary in length. Each record must contain both a block-length field (BL) and a record-length field (RL) to provide IOCS with the size of the block and the size of the logical record. The first two characters (XX) of the block-length field (BL) specify the actual block length in 16-bit binary form. The last two characters (indicated by bb) are blank. For variable-length unblocked records, BL will specify the logical record length plus 4 bytes (the size of BL).

The first four bytes following the block-length field must contain the record-length field (RL). The first two bytes (XX) specify the length of the logical record including the bytes used for RL field itself. The remaining two bytes (bb) are blanks. Variable-length records are supported only by consecutive processing macros.

Variable-length, blocked (Figure 30). One or more logical records are contained within each physical record. The first four bytes (block-length field) of each physical record (block) specifies to IOCS the total number of bytes in the block. The first two bytes (XX) specify the length of the block (including the four bytes for the block-length field itself). The remaining two bytes (bb) are blank. The size of each logical record must be placed in a record-length field (RL). The RL must be the first four bytes of the logical record. The first two bytes (XX) of RL specify the length of the logical record including the bytes used for the RL field. The remaining two bytes (bb) are blank. Variable-length records are supported only by consecutive processing macros.

These record types are also processed by Utility programs, Sort/Merge programs, and Report Program Generator programs, in addition to being processed by logical IOCS macros used in problem programs.

When variable-length blocked records are built directly in the output area, the TRUNC macro must be used to write a completed block of records. The TRUNC macro is discussed fully in the assembler/IOCS publication C24-3361.

Undefined. When file records do not conform to any of the four previous formats, they are classified as undefined. In this case, IOCS performs only the actual reading and writing of the records. Logical processing must be performed by the problem program.

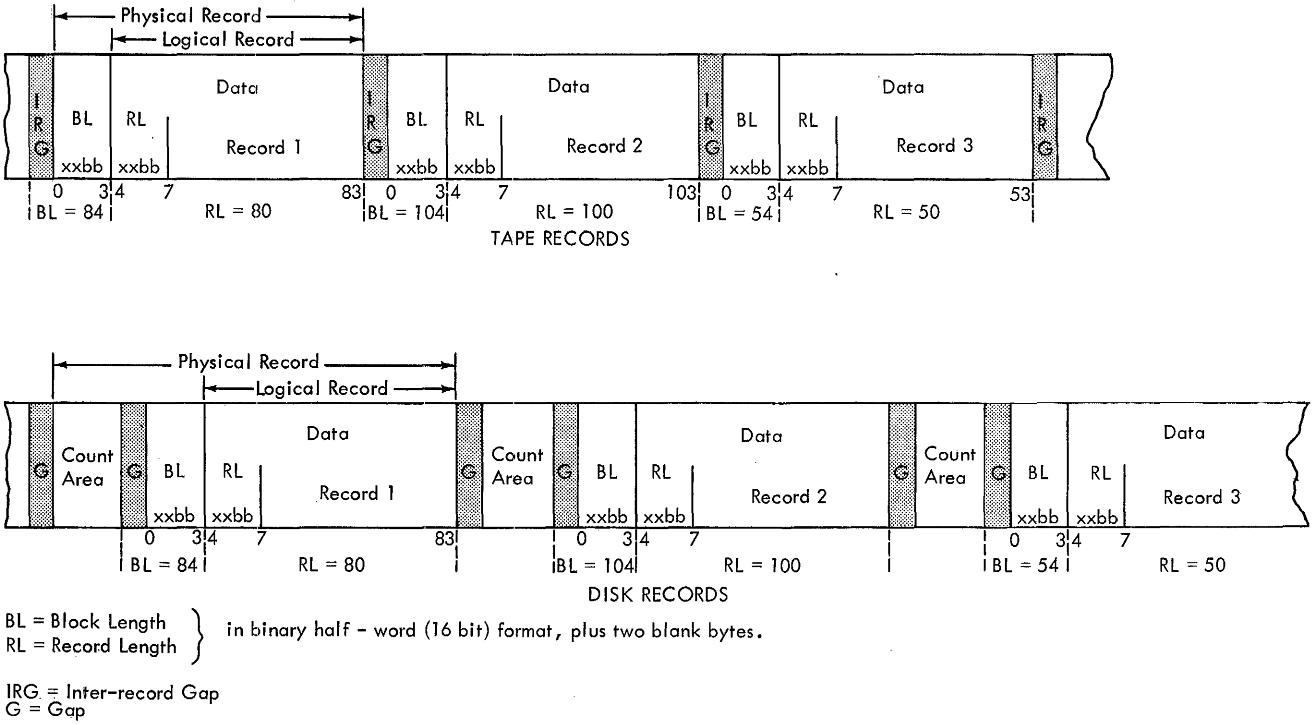


Figure 29. Variable-Length Unblocked Record Format (Consecutive Processing)

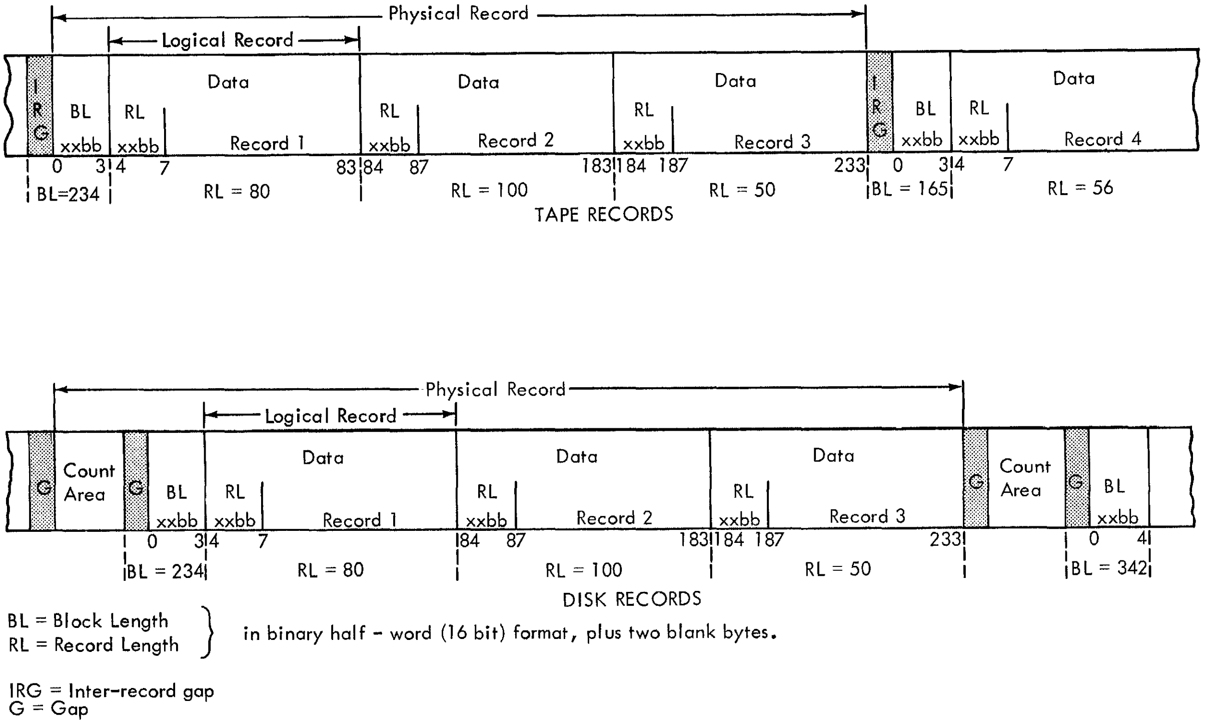


Figure 30. Variable-Length Blocked Record Format (Consecutive Processing)

CONSECUTIVE PROCESSING

Consecutive processing macro instructions read/write and process successive physical records in a logical data file. For example, card records are read and processed in the order the cards are fed; tape records are processed starting with the first record after the file labels and continuing consecutively through the records to the end-of-file trailer label: disk records are processed beginning at the first disk location addressed and continuing with each physical record on successive tracks (and possibly cylinders) to the ending address.

A consecutive file on disk is contained within one or more sets of limits. These file limits are specified by the user with job-control XTENT cards. If the logical data file consists of more than one set of limits, IOCS automatically processes each set as requested by the user. The records within each set must be adjacent and contained within one volume (disk pack). The sets need not be adjacent, and they may be on more than one volume.

In many cases files written on disk by the direct access method can also be processed consecutively.

The basic imperative macros used for consecutive processing are GET and PUT. These instructions overlap data transfer and processing as much as possible. The extent of overlap depends upon the user's I/O area assignment.

Whenever a file of records is to be processed in consecutive order, the logical file, the device used for the file, and the main-storage areas allotted to the file must be defined by the declarative macro DTFSR (Define The File for a SeRial-type Device). The detailed parameter entries for defining the logical data file are described in the BOS Assembler with Input/Output Macros publication, listed in the Preface of this manual.

IOCS can consecutively process all record types (see Types of Records) in the same program, but all records in a given file must be the same type. The record type, the block size, and usually the record size, are defined by the user when he writes the parameters for the DTFSR macro statement for each logical data file to be processed.

STORAGE AREAS

When logical IOCS macro instructions are used, each input record can be made available to the program for processing either in an input area or a work area. Similarly, on output, each record can be built in a work area or directly in an output area.

Input/output areas and work areas for a particular file can be specified and handled by IOCS in any of the following combinations:

one I/O area
one I/O area and one work area
two I/O areas
two I/O areas and one work area.

When blocked records are to be processed in one I/O area, a register must be specified in the DTFSR statement. This register is used by logical IOCS to point to the beginning of each logical record in an input area or it is used to point to the next available area for building a logical record in an output area.

If two I/O areas are used for processing either blocked or unblocked records, a register must be specified. This register identifies the next logical record to be processed in an input area, or it points to the next available output area. It contains the absolute base address of the currently available input record or currently available output area. The GET/PUT routines maintain the proper address in the register.

If variable-length records are blocked and are built in an output area(s), an additional register must be specified. This register provides the programmer with the remaining space in the output area each time a PUT instruction is executed.

If variable-length unblocked tape records are read backwards and processed directly in the input area, a register must be specified in the DTFSR statement IOREG.

When work areas are specified in a DTFSR statement, registers are not required to be specified. Instead, the work area must be named in each GET or PUT instruction used for processing the so-defined logical data file.

GET

The GET macro makes the next consecutive logical record from an input file available for processing in either an input area or a specified work area.

PUT

The PUT macro moves records from a specified work area to an output area and writes, punches, or displays the records from the output area when necessary.

If the records are being built directly in the output area rather than·in a work area, the PUT macro points to the next available position for building a logical record in the output area.

TWO INPUT/OUTPUT AREAS

Logical IOCS provides DTFSR statements for defining two input/output areas for a file. Two input or two output areas may be used to permit an overlap of data transfer and processing operations. Whenever two I/O areas are specified, the IOCS routines automatically alternate between each area at the end of a physical record. These routines completely handle this flip-flop so that the next consecutive logical input record is always available to the problem programs for processing. For output records the flip-flop control keeps the proper output-record area available to the program for the next consecutive output record.

When two input areas and unblocked records are specified, each GET makes the last record that was transferred to main storage available for processing. The same GET also starts the transfer of the following record to the other input area.

The reader is referred to the section in this manual on Processing Overlapped with Input/Output for a discussion on overlapping the physical transfer of data with processing.

UPDATING

A consecutive file on a disk pack, a card input file in a 1442, or a card file in the punch feed of a 2540 equipped with the punch-feed-read special features can be updated. That is, each disk or card record can be read, processed, and transferred back to the same disk location, or card, from which it was read. This function must be specified with a DTFSR file-definition statement. In the case of a card file, the file must be specified as a combined file (CMBND) in the DTFSR entry TYPEFLE.

The disk or card record is transferred to main storage by a GET instruction. After the record is processed, the next PUT instruction causes the updated record to be written in the same disk location, or punched in the same card, from which the record was read. PUT transfers the record to the disk or card file from the input area of main storage. If a work area is specified in the GET and PUT instructions, PUT first moves the updated record from the work area back to the input area and then transfers the record to this file.

A GET instruction must always precede a PUT instruction for a disk or card record, and only one PUT can be issued for each record. A PUT instruction may be omitted, except for the 2540, if a particular disk or card record does not require updating.

The programmer is referred to the BOS Assembler with Input/Output Macros publication, listed in the Preface of this manual, for detail instructions on writing DTFSR file-definition macro statements and the related imperative macro statements such as GET, PUT, RELSE, TRUNC, CNTRL, FEOV, CHANG, and PRTOV.

PROCESSING OVERLAPPED WITH INPUT/OUTPUT (CONSECUTIVE PROCESSING)

The problem program has the ability to retrieve and store records when consecutive processing macro instructions are used. In doing this, it makes use of both the Channel Scheduler and the logical I/O routines (GET, PUT macros) of IOCS. All of these routines are designed to provide for overlapping the physical transfer of data with processing. The amount of overlapping actually achieved (effective overlap) is governed by the problem program through the assignment of I/O areas and work areas. An I/O area is that area of main storage to or from which a block of data is physically transferred by the Channel Scheduler and physical IOCS routines. A work area is an area used for processing an individual record from the block of data.

There are certain combinations of I/O areas and work areas that are possible. These are:

1. One I/O area with no work area.

2. One I/O area with a work area.

3. Two I/O areas with no work area.

4. Two I/O areas with a work area.

Also, certain devices are buffered, increasing the amount of achievable overlap of processing and I/O. Tape and disk records can be blocked. Illustrations of these combinations for buffered devices, unbuffered devices, blocked tape, and blocked disk records follow.

The maximum achievable overlap in Figure 31 is the device time only. The transfer time between I/O area and buffer is not overlapped. If the next GET (or PUT) is issued prior to Device End, the data transfer between I/O area and buffer does not take place until Device End is reached.

The maximum achievable overlap in Figure 32 is the total data transfer time (the device time plus the time for data transfer between I/O area and buffer). If the next GET (or PUT) is issued after Channel End but before Device End, the transfer of data between the work area and the I/O area can take place (even though physical IOCS can not start the data transfer between the device and the buffer until Device End is reached). Control transfers to the problem program.

If the next GET (or PUT) is issued before Channel End, logical IOCS must wait until Channel End to transfer data between the work area and the I/O area.

The maximum achievable overlap (for logical IOCS and the problem program) in Figure 33 is the total data transfer time (the device time plus the time for data transfer between I/O area and buffer). If the next GET (or PUT) is issued after Channel End and before Device End, only I/O area switching occurs. Control transfers to the problem program but physical IOCS does not start the device/buffer transfer until Device End is reached.

If the next GET (or PUT) is issued before Channel End, logical IOCS must wait for Channel End before performing any action.

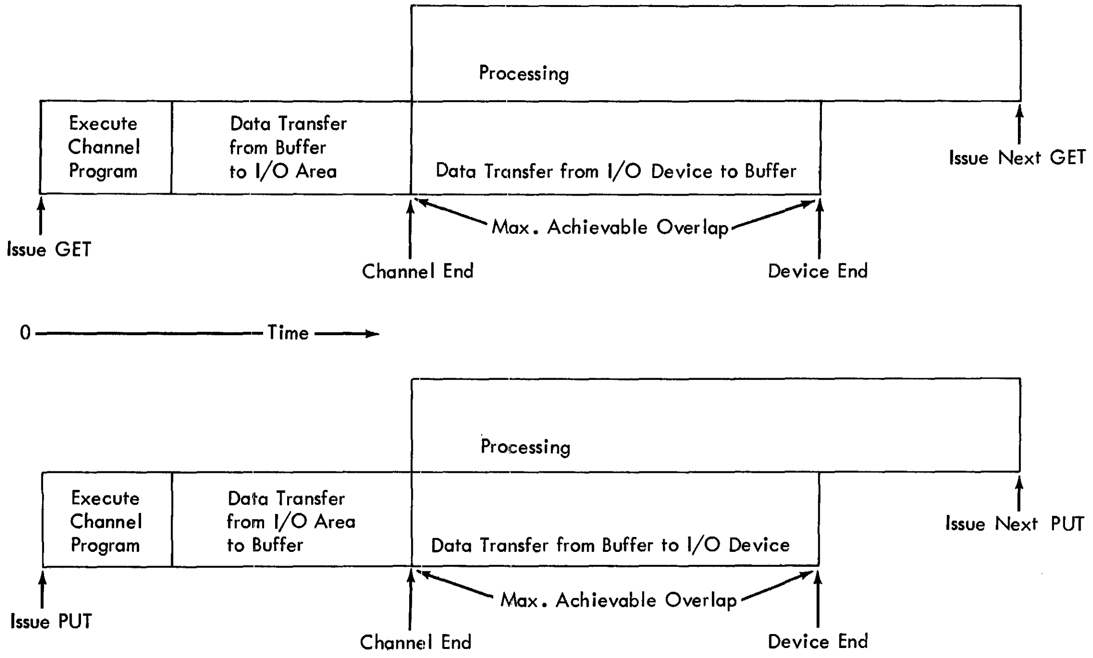


Figure 31. Overlap of Processing and I/O: One I/O Area and No Work Areas (Buffered I/O Device)

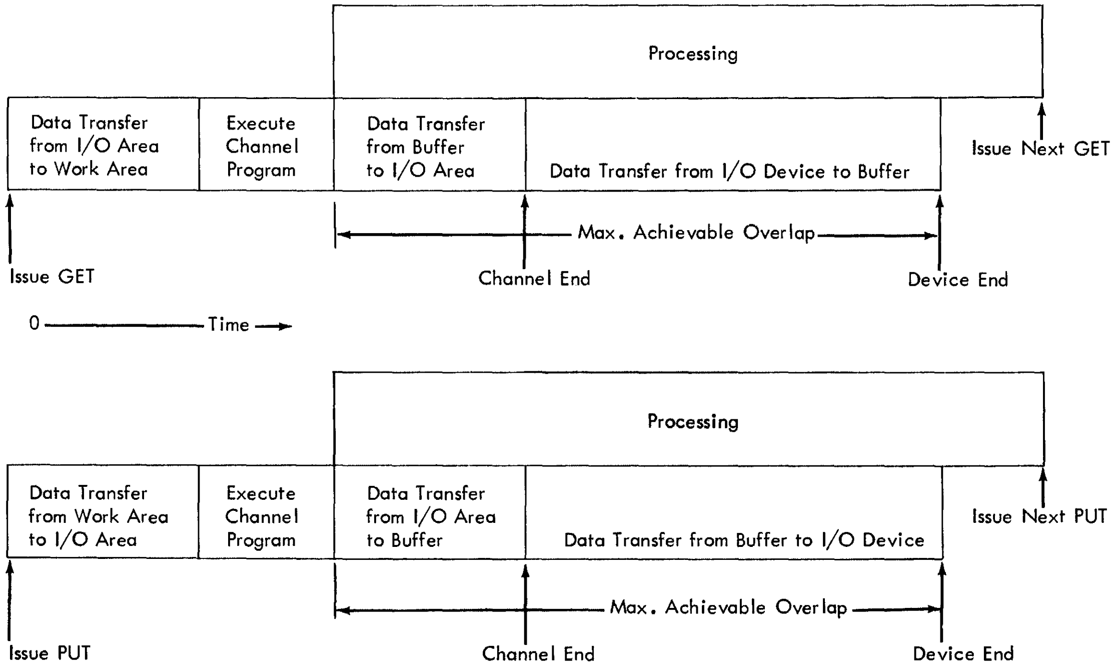

Processing
Data Transfer from I/O Area to Work Area
Execute Channel Program
Data Transfer from Buffer to I/O Area
Data Transfer from I/O Device to Buffer
Issue Next GET
Max. Achievable Overlap
Issue GET
Channel End
Device End
0
Time
Processing
Data Transfer from Work Area to I/O Area
Execute Channel Program
Data Transfer from I/O Area to Buffer
Data Transfer from Buffer to I/O Device
Issue Next PUT
Max. Achievable Overlap
Issue PUT
Channel End
Device End

Figure 32. Overlap of Processing and I/O: One I/O Area and One Work Area (Buffered I/O Device)

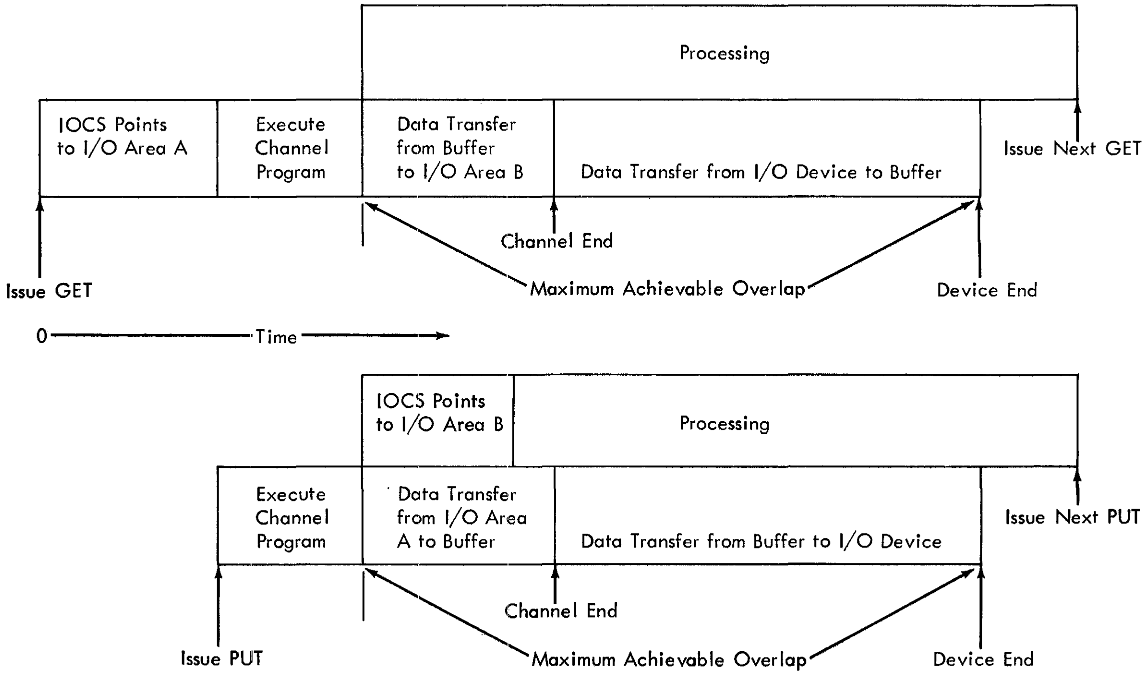

Processing
IOCS Points to I/O Area A
Execute Channel Program
Data Transfer from Buffer to I/O Area B
Data Transfer from I/O Device to Buffer
Issue Next GET
Channel End
Issue GET
Maximum Achievable Overlap
Device End
0 Time
IOCS Points to I/O Area B
Processing
Execute Channel Program
Data Transfer from I/O Area A to Buffer
Data Transfer from Buffer to I/O Device
Issue Next PUT
Channel End
Issue PUT
Maximum Achievable Overlap
Device End

Figure 33. Overlap of Processing and I/O: Two I/O Areas and No Work Area (Buffered I/O Device)

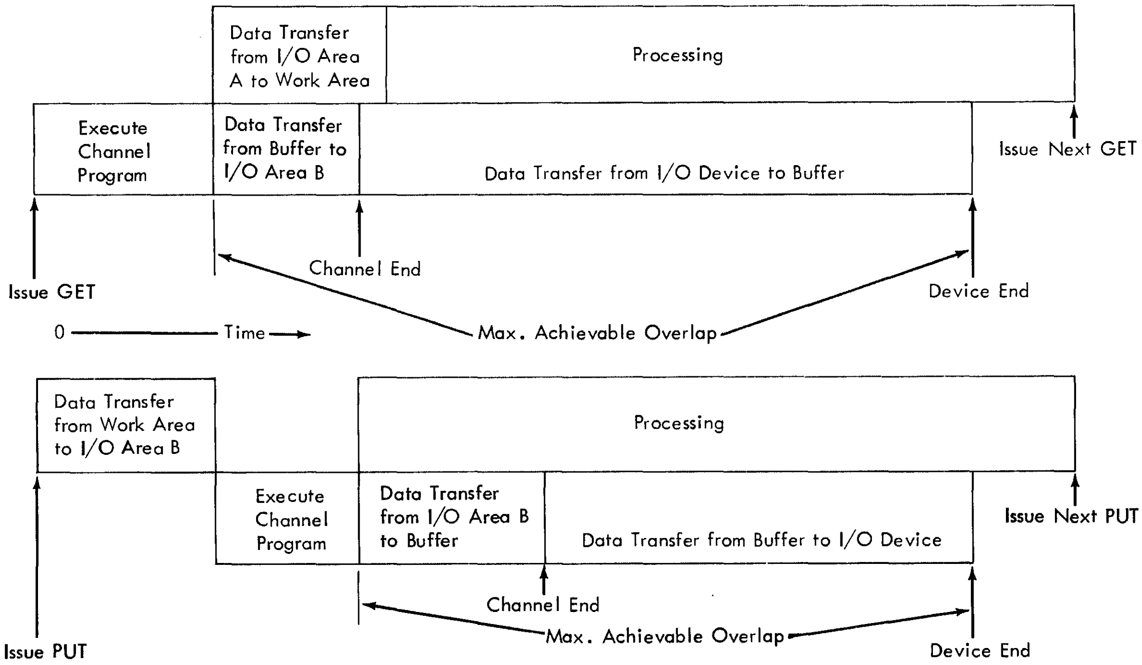

Data Transfer from I/O Area A to Work Area
Processing
Execute Channel Program
Data Transfer from Buffer to I/O Area B
Data Transfer from I/O Device to Buffer
Issue Next GET
Channel End
Issue GET
Device End
0 Time
Max. Achievable Overlap
Data Transfer from Work Area to I/O Area B
Processing
Execute Channel Program
Data Transfer from I/O Area B to Buffer
Data Transfer from Buffer to I/O Device
Issue Next PUT
Channel End
Max. Achievable Overlap
Issue PUT
Device End

Figure 34. Overlap of Processing and I/O: Two I/O Areas and a Work Area (Buffered I/O Device)

The maximum achievable overlap (for logical IOCS and the problem program) in Figure 34 (as in Figures 32 and 33) is the total data transfer time (the device time plus the time for data transfer between I/O area and buffer). However, there is a disadvantage to the combination illustrated (in comparison with Figures 32 and 33), because it requires extra main storage.

If the next GET (or PUT) is issued after Channel End but before Device End, the data transfer between the I/O area and the work area can take place. Control returns to the problem program (even though physical IOCS cannot start the device until Device End is reached).

If the next GET (or PUT) is issued before Channel End, logical IOCS must wait.

There is no overlap possible in the illustration in Figure 35.

The maximum achievable overlap in Figure 36 is the data transfer time between device and I/O area. If the next GET (or PUT) is issued prior to Channel End and Device End, logical IOCS must wait before performing any action.

The maximum achievable overlap (for logical IOCS and the problem program) in Figure 37 is the data transfer time between device and I/O area. If the next GET (or PUT) is issued before Channel End and Device End, logical IOCS must wait before performing any action.

The maximum achievable overlap (for logical IOCS and the problem program) in Figure 38 (as in Figures 36 and 37) is the data transfer time between device and I/O area. However, there is a disadvantage to the combination illustrated (in comparison with Figures 36 and 37), because it requires extra main storage. If the next GET (or PUT) is issued before Channel End and Device End, logical IOCS must wait before any action is performed.

The combination illustrated in Figure 39 has no overlap of processing with input/output. The input/output time per record depends on the blocking factor. With this combination, the I/O time per record can be reduced if the blocking factor is increased.

The maximum overlap achievable in Figure 40 is the time for data transfer between device and I/O area. The GET (or PUT) for all records, except the last in a block, involves only a transfer between work area and I/O area. For the last record in a block, the data transfer is followed by an overlap of device time and processing (control returns to the problem program). Channel End and Device End must occur before logical IOCS can process the first record.

The maximum overlap achievable in Figure 41 is the time for data transfer between device and I/O area. The GET for all but the first record of a block takes time only for pointing to the next record. The GET for the first record must wait for Channel End (and Device End) of the data transfer to the alternate area. Then pointing to the first record and returning control to the problem program is overlapped with the next device transfer. The PUT is the same as GET, except that the wait occurs with the last record of a block.

There is a disadvantage to the combination shown in Figure 42 over those in Figures 40 and 41, because it requires extra main storage.

A summary of the overlap of processing and I/O is shown in Figure 43.

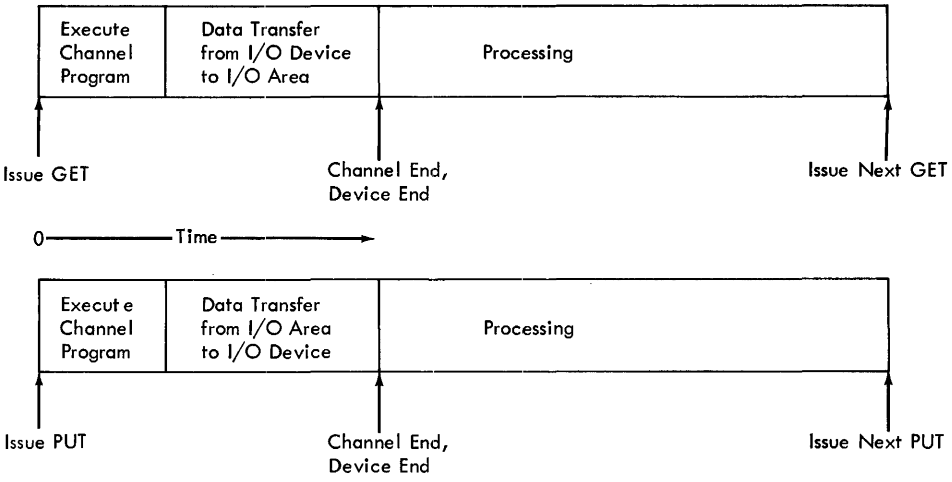


Figure 35. Overlap of Processing and I/O: One I/O Area and No Work Area (Unbuffered I/O Device)

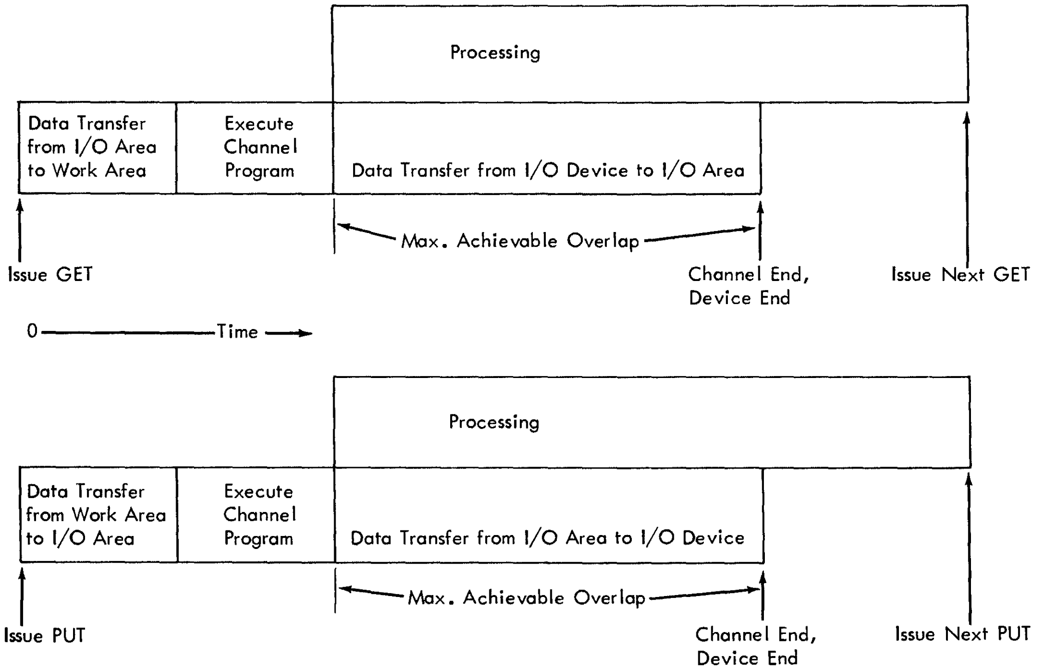


Figure 36. Overlap of Processing and I/O: One I/O Area and a Work Area (Unbuffered I/O Device)

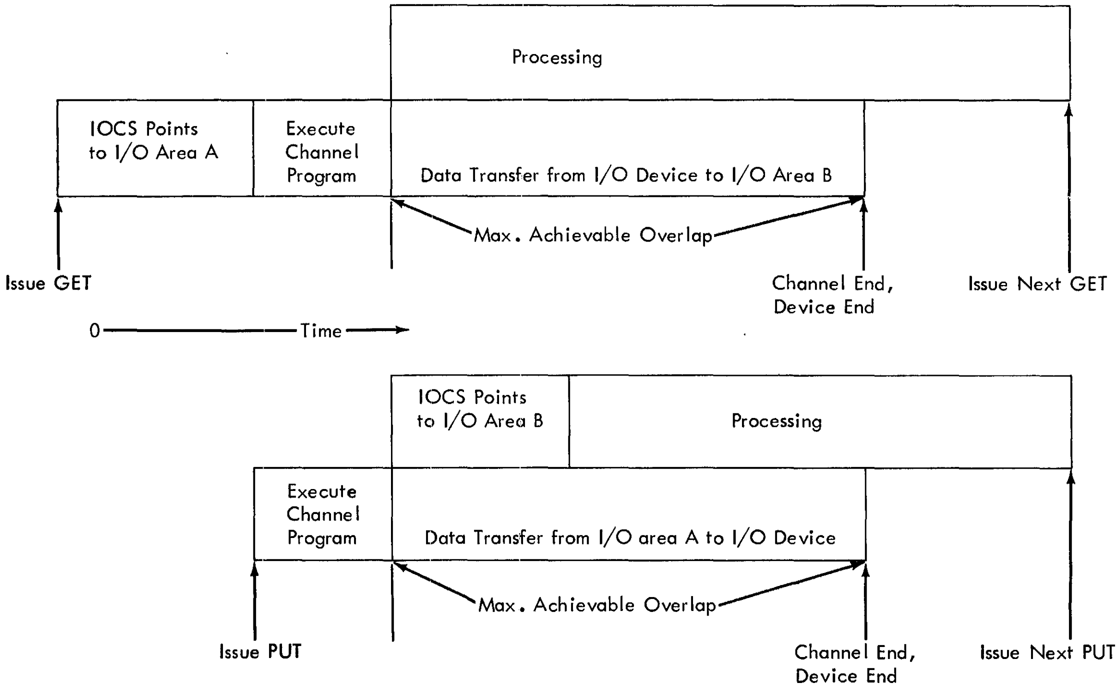


Figure 37. Overlap of Processing and I/O: Two I/O Areas and No Work Area (Unbuffered I/O Device)

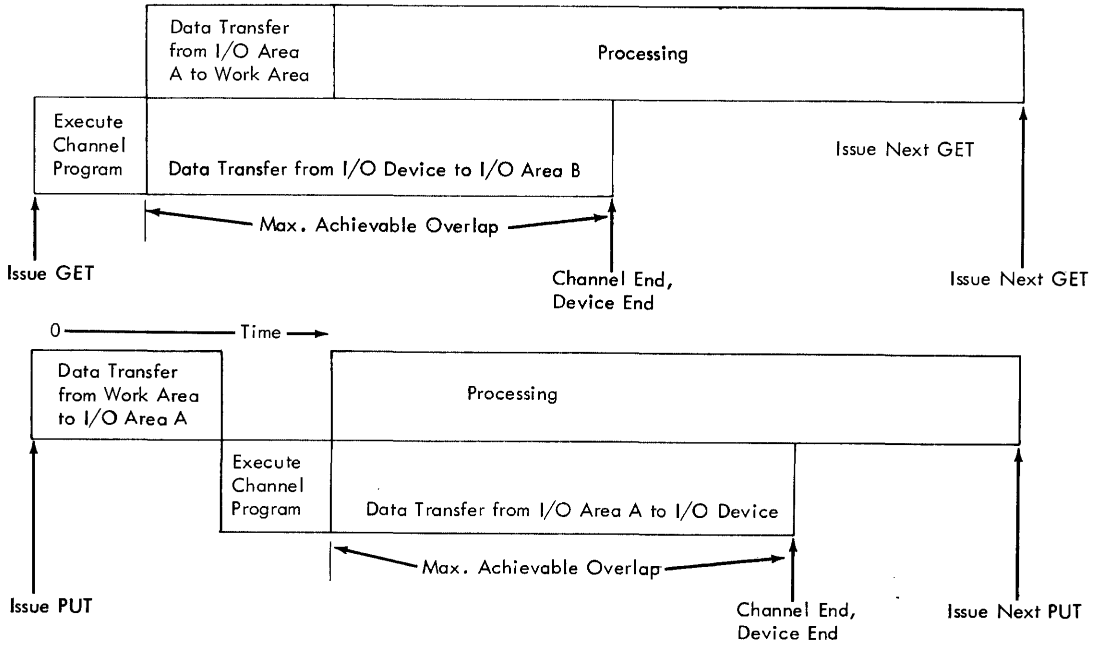


Figure 38. Overlap of Processing and I/O: Two I/O Areas and a Work Area (Unbuffered I/O Device)

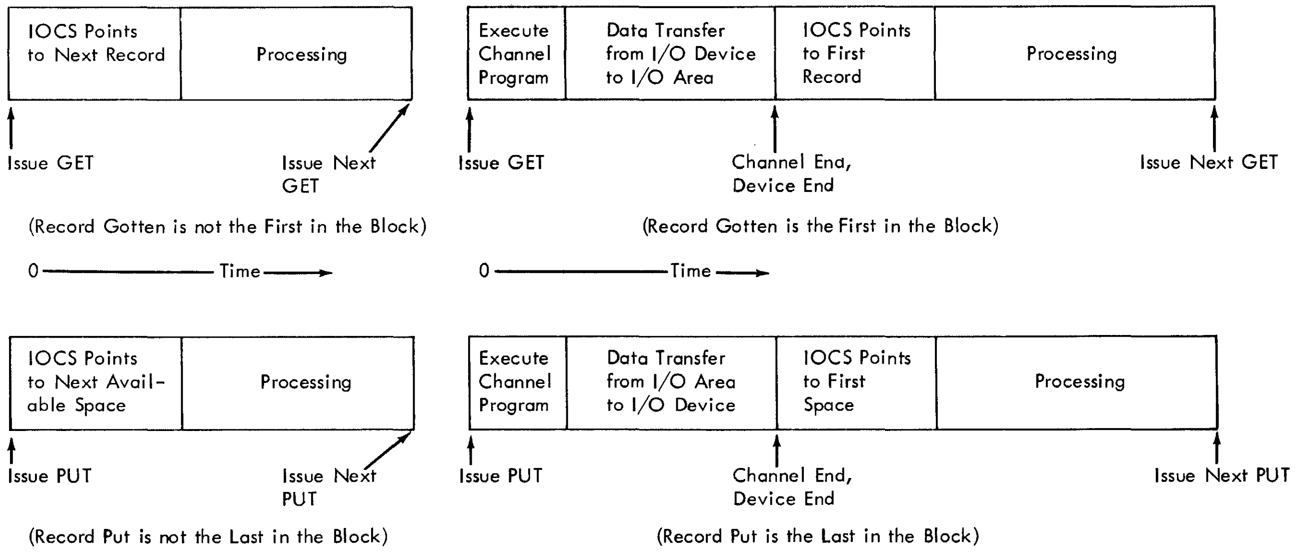
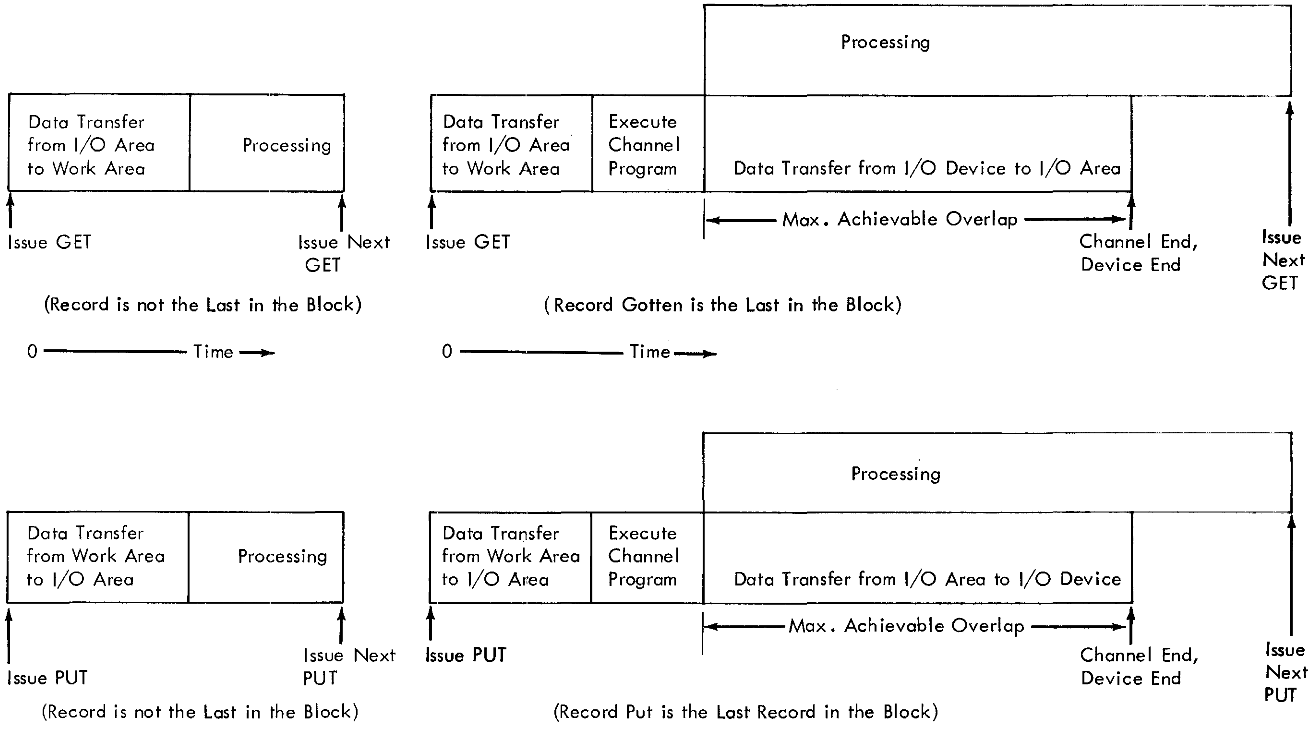

IOCS Points to Next Record
Processing
Issue GET
Issue Next GET
(Record Gotten is not the First in the Block)
Execute Channel Program
Data Transfer from I/O Device to I/O Area
IOCS Points to First Record
Processing
Issue GET
Channel End, Device End
Issue Next GET
(Record Gotten is the First in the Block)
0 Time
0 Time
IOCS Points to Next Available Space
Processing
Issue PUT
Issue Next PUT
(Record Put is not the Last in the Block)
Execute Channel Program
Data Transfer from I/O Area to I/O Device
IOCS Points to First Space
Processing
Issue PUT
Channel End, Device End
Issue Next PUT
(Record Put is the Last in the Block)

Figure 39. Overlap of Processing and I/O: One I/O Area and No Work Area (Blocked Records)


Data Transfer from I/O Area to Work Area
Processing
Issue GET
Issue Next GET
(Record is not the Last in the Block)
0 Time
Processing
Data Transfer from I/O Area to Work Area
Execute Channel Program
Data Transfer from I/O Device to I/O Area
Issue GET
Max. Achievable Overlap
Channel End, Device End
Issue Next GET
(Record Gotten is the Last in the Block)
0 Time
Data Transfer from Work Area to I/O Area
Processing
Issue PUT
Issue Next PUT
(Record is not the Last in the Block)
Processing
Data Transfer from Work Area to I/O Area
Execute Channel Program
Data Transfer from I/O Area to I/O Device
Issue PUT
Max. Achievable Overlap
Channel End, Device End
Issue Next PUT
(Record Put is the Last Record in the Block)

Note: For both GET and PUT, a channel end must have occurred before the data transfer between I/O area and work area can take place (for the first record in a block).

Figure 40. Overlap of Processing and I/O: One I/O Area and a Work Area (Blocked Records)

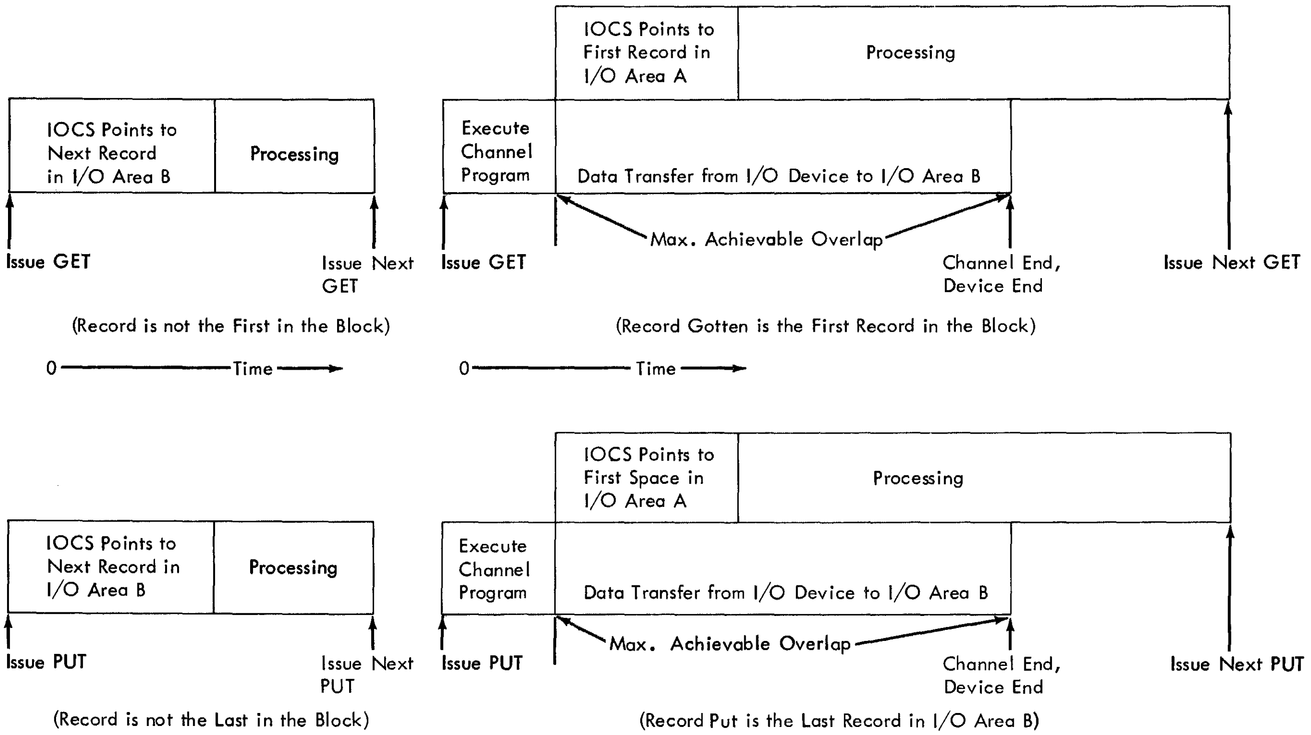

IOCS Points to First Record in I/O Area A
Processing
IOCS Points to Next Record in I/O Area B
Processing
Execute Channel Program
Data Transfer from I/O Device to I/O Area B
Max. Achievable Overlap
Issue GET
Issue Next GET
Issue GET
Channel End, Device End
Issue Next GET
(Record is not the First in the Block)
(Record Gotten is the First Record in the Block)
0 Time
0 Time
IOCS Points to First Space in I/O Area A
Processing
IOCS Points to Next Record in I/O Area B
Processing
Execute Channel Program
Data Transfer from I/O Device to I/O Area B
Max. Achievable Overlap
Issue PUT
Issue Next PUT
Issue PUT
Channel End, Device End
Issue Next PUT
(Record is not the Last in the Block)
(Record Put is the Last Record in I/O Area B)

Figure 41. Overlap of Processing and I/O: Two I/O Areas and No Work Area (Blocked Records)

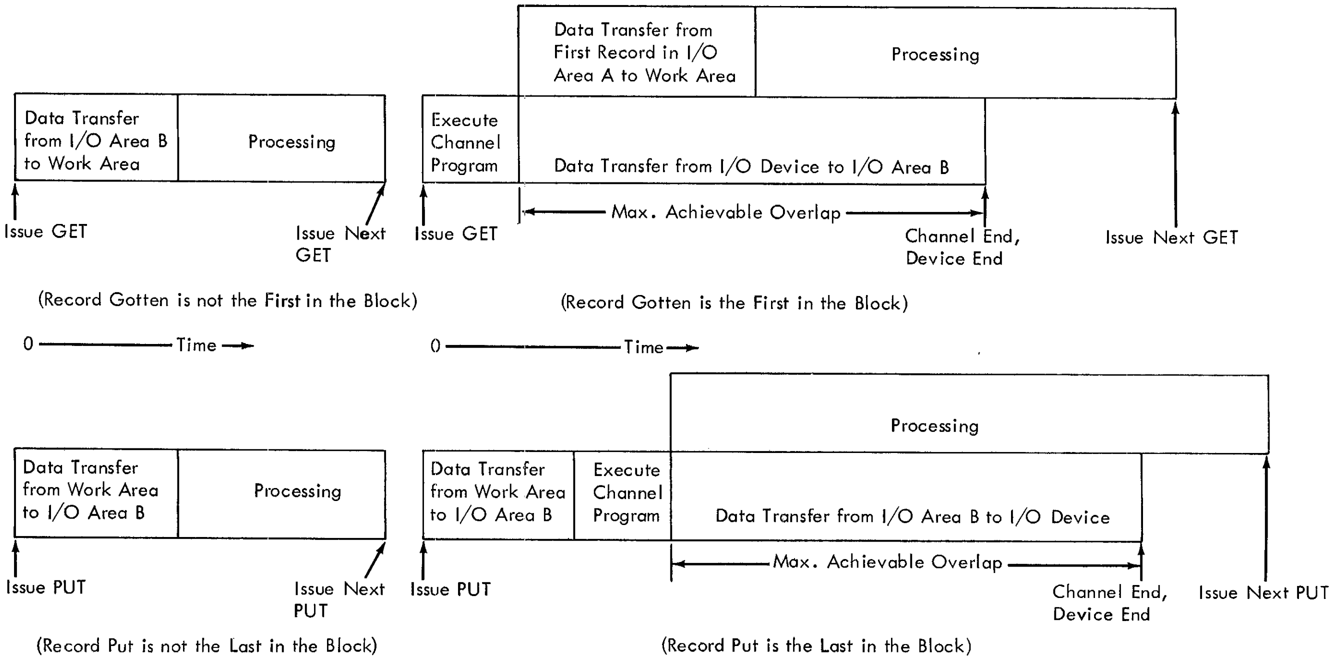

Data Transfer from I/O Area B to Work Area
Processing
Issue GET
Issue Next GET
(Record Gotten is not the First in the Block)
0 Time
Data Transfer from First Record in I/O Area A to Work Area
Processing
Execute Channel Program
Data Transfer from I/O Device to I/O Area B
Issue GET
Max. Achievable Overlap
Channel End, Device End
Issue Next GET
(Record Gotten is the First in the Block)
0 Time
Data Transfer from Work Area to I/O Area B
Processing
Issue PUT
Issue Next PUT
(Record Put is not the Last in the Block)
Processing
Data Transfer from Work Area to I/O Area B
Execute Channel Program
Data Transfer from I/O Area B to I/O Device
Issue PUT
Max. Achievable Overlap
Channel End, Device End
Issue Next PUT
(Record Put is the Last in the Block)

Figure 42. Overlap of Processing and I/O: Two I/O Areas and a Work Area (Blocked Records)

Record Format (Blocked or Unblocked)	Number of I/O Areas	Separate Work Area	Amount of Effective Overlap
Unblocked	1	no	Overlap of the device operation only for buffered devices such as 1403, 1443, 2540. No overlap of magnetic tape, 1015, 1052, 1442, 2311, 2671, 1285 (unbuffered at device).
		yes	Overlap processing of each record. (Record move required, except for 1015 and 1052).
	2	no	Overlap processing of each record. (No record move required).
		yes	Overlap processing of each record. (No advantage to a work area).
Blocked	1	no	No overlap.
		yes	Overlap processing of last record in each block.
	2	no	Overlap processing of full block.
		yes	Overlap processing of full block. (No advantage to a work area).

Note: Overlap given is the maximum achievable.

Figure 43. Summary of Achievable Overlap of Processing and Input/Output

DIRECT ACCESS METHOD

The Direct Access Method (DAM) provides a flexible set of macro instructions for creating and maintaining a data file on the IBM 2311 Disk Storage Drive. This technique applies primarily to records organized in a random order. The macro language offered by this data management technique permits the user to load, read, write, update, add or replace records in a 2311 disk storage file. This language gives the user great flexibility in data handling techniques, with the preparation of CCWs (channel command words) handled by the DAM macro routines.

Advantages of the Direct Access Method

- DAM is an IOCS processing method designed specifically for direct access storage devices.
- DAM can be used to process records organized in a random order.
- DAM can also be used to process, in key sequence, a file of records stored sequentially by record key.
- DAM provides READ/WRITE macro routines for handling input/output requirements.
- DAM will process records with or without key area.
- The capacity record feature is offered as an option.
- DAM offers a multiple-track searching feature to search beyond the specified track for a key argument.
- DAM offers two reference methods, either by Identifier (ID) area, or by control information set up in a key area.
- On certain READ/WRITE macros, the user can specify a location for IOCS to supply the record identifier (ID) of the current record or the next record after a READ or WRITE has been executed.
- DAM permits the user to have one or more user labels in addition to the standard volume and file labels.

DAM has these restrictions :

- DAM will not block or deblock records.
- DAM will process only records that are specified as fixed length or undefined.
- A track reference and a record reference must be provided to IOCS by the user for every record to be read or written.
- Only one I/O area can be specified for each file to be processed.
- A work area can not be specified in the DAM technique.

The basic means of communication between the problem program and the logical IOCS routines of the direct access method is through the use of READ/WRITE imperative macro instructions. These macro routines allow the user to read, write, update, add, or replace records on a disk file.

When the user issues a READ or WRITE macro instruction for a file, program control is transferred to a logical IOCS routine that selects the proper channel program to accomplish the command. The IOCS routine issues an execute channel program that will cause the I/O request to be started; or if the channel is busy, will place the request in a queue for later execution. IOCS then returns control to the problem program. A WAITF macro instruction must be written into the problem program by the user before the next READ or WRITE for this file. The WAITF macro establishes routines to test the status of the channel to ensure that the operation is complete. If the I/O operation is not complete, the routines branch into a programmed waiting loop. The WAITF macro routines will supply indications of exceptional conditions to the problem program. The symbolic name of a two byte field (error/status field) in which IOCS can store the error/status information must be supplied by the user in a DTF statement.

At the completion of the I/O operation, control is returned to the problem program. The user should test the condition codes before proceeding with his program.

The declarative macro DTFDA (Define The File for Direct Access) provides the necessary entries for defining the logical file to be processed and the type of processing to be done on the file. These entries are explained in detail in the BOS Assembler with Input/Output Macros publication, listed in the Preface of this manual.

RECORD TYPES

The DAM IOCS routines will not block or deblock records. If the user wants to use blocked records, he must provide his own blocking and deblocking routines. Records are considered to be either fixed length or undefined. Whenever undefined records are to be loaded, added, or written in a file, the user must determine the length of the data area of each record and load it in a register (specified by the DTFDA entry RECSIZE) before he issues the WRITE instruction for the record. IOCS adds the length of the key area when it is required. If the RECSIZE DTFDA entry is used and a READ macro is issued, IOCS will return the length of the data area of the record read to the user in this register.

Records to be processed by DAM can have either of two formats:

With Key Area

```
┌───────┐    ┌──────┐    ┌───────────┐
│ Count │    │ Key  │    │           │
│ Area  │    │ Area │    │ Data Area │
└───────┘    └──────┘    └───────────┘
```

Without Key Area

```
┌───────┐    ┌───────────┐
│ Count │    │           │
│ Area  │    │ Data Area │
└───────┘    └───────────┘
```

The Count Area includes the physical track and record address, the number of bytes (key length) in the key area, and the number of bytes (data length) of data stored in the data area.

The Key Area can be used to identify the data record by control information, (not necessarily related to physical location), such as part number, employee number, etc.

The Data Area contains the logical information stored in the file.

The user is referred to the section on Disk Record Format in this manual for further details (see Figures 58 and 59).

MAIN STORAGE I/O AREA

A DTFDA entry is required to specify the symbolic name of the I/O area to be used by the file. This area must be reserved by a DS instruction that uses the same symbolic name for the area.

Enough main storage must be reserved to contain the maximum number of bytes required by any READ-WRITE instruction issued for the file. The length of the I/O area is affected by the length of record data areas, and by the need for count and key areas as follows:

- When undefined records are specified in the DTFDA entry for type of record, the I/O area must provide for the largest data record that will be processed.

- If records with key areas are to be processed and the IOCS routines used require the use of the key area, the I/O area must provide room for the key area as well as the data area.

- When the problem program issues WRITE instructions that transfer the count area, eight bytes of main storage must be available at the beginning of the I/O area. IOCS will construct the count field, to be transferred to the disk, in these eight bytes. The I/O area requirements are illustrated in Figure 44.

REFERENCE METHODS

Each record to be read or written must be identified by providing the DAM IOCS routines with two references.

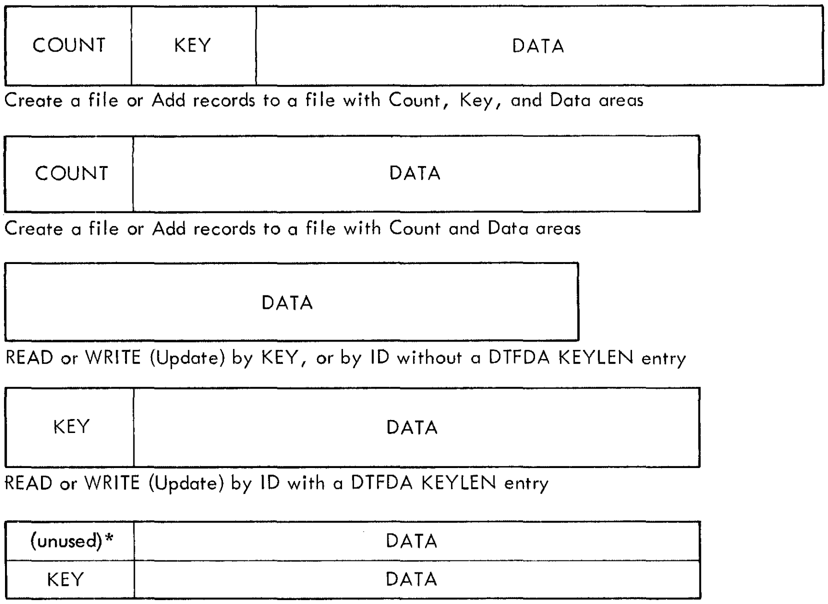
- Track Reference. This identifies the track that contains the record desired.

- Record Reference. The location of the record on the track can be identified by its control information, if the records contain key areas, or by its physical location on the track.

Track Reference

The track reference that the user furnishes to the DAM IOCS routines must be set up in an 8-byte field (Figure 45) in main storage. Before any read or write macro instruction is issued, the track reference field must contain the proper track identification (physical location of the track in the disk pack). Track identification is an 8-byte binary address of the form MBBCCHHR, where:

M, is the sequential pack number of the pack(s) that constitutes the logical file. It selects the channel and unit number on which the referenced track will be found. The numbers placed in M refer to the symbolic unit(s) assigned to the logical file. The symbolic unit numbers, SYS001 etc, must be sequential. A reference to a track on the first disk pack assigned to a logical file would require an M of zero. Assume, for example, that a logical file has been assigned to units SYS002, SYS003, and SYS004. Then, M=0 refers to SYS002;



Create a file or Add records to a file with Count, Key, and Data areas

Create a file or Add records to a file with Count and Data areas

READ or WRITE (Update) by KEY, or by ID without a DTFDA KEYLEN entry

READ or WRITE (Update) by ID with a DTFDA KEYLEN entry

READ or WRITE (Update) by KEY and by ID with a DTFDA KEYLEN entry

*The first bytes are unused when a READ (or WRITE) by Key is executed.

Figure 44. Schematic of DAM Input/Output Main Storage Requirements

M Pack Number	B,B Cell (2321)		C,C Cylinder		H,H Head		R Record	
0	1	2	3	4	5	6	7	← Byte Position
0-254	0	0	0	0-202	0	0-9	0-255	← Contents (Binary)

Figure 45. DAM Track Reference Field in Main Storage

M=1 refers to SYS003; and M=2 refers to SYS004.

B,B, is reserved for cell number (relates to the IBM 2321 Data Cell Drive). These two bytes are always zero (0,0) for 2311 disk storage references.

C,C, is used for indicating the number of the cylinder (0-202) in which the record is located. The first byte is always zero. The second byte specifies one of the 203 cylinders in a disk pack.

H,H, is used to indicate the number of the read/write head (0-9) to be used for a particular track in the specified cylinder. The first byte is always zero. The second byte specifies one of the 10 disk surfaces in a disk pack.

R, is the record number. This byte can have a binary number of 0 to 255 to identify the physical location of the record on the specified track (CCHH). Space for this byte must always be provided in the reference field, but the number is required only when records are to be referenced by their physical location on the track.

IOCS uses the track reference field to construct the data-address field of the channel command word (CCW) for seek commands. When records are referenced by their physical location (record ID), IOCS uses the field to construct the CCW data address field for search commands. The user must specify the location of the field by a symbolic name in the DTFDA entry SEEKADR.

Record Reference

The position of a record on a track can be referred to by either of two methods.

- Record identifier(ID): the physical location of a record on a track. The record identifier is the CCHHR part of the count area of any 2311 disk record.
- Record Key: the control information of a record. Records with key areas can be referred to by the control information.

Record Reference by ID

The record identifier (ID) is part of the count area of each 2311 disk record. It consists of five bytes (CCHHR shown in Figures 45 and 46). The first four bytes identify the location of the track (cylinder number and head number) and the fifth byte (record) uniquely identifies the particular record on the track. When records are to be referenced by ID, they must be numbered in succession, without missing numbers, on each track. The first data record (after Record Zero) must be record number 1, the second record number 2, etc.

When a READ or WRITE instruction that searches by ID is executed, IOCS refers to the track reference field to determine which record is requested by the program. The number in this field (CCHHR) is compared with the corresponding field in the count areas of the disk records.

When the READ, ID instruction is executed, IOCS searches the specified track for the particular record. If the record is found, the key area (if present and defined in DTFDA entry KEYLEN) and the data area of the record are transferred into the main storage I/O area. If a corresponding ID is not found, a no-record-found indicator will be placed in the user's error/status field. The location of this field must be specified, by the user, in the DTFDA entry ERRBYTE.

When the WRITE, ID instruction is executed, IOCS searches the specified track for a record with an equal ID. If an equal ID is found, the information in the main-storage output area is transferred to the key area (if specified) and the data area of the disk record. This new record replaces the key and data previously recorded on the disk track. Because of the manner in which the WRITE, ID instruction functions, the disk storage area used must have been previously formatted with appropriate count and data areas or count, key, and data areas. The Clear Disk utility program can be used to format count and data areas or count, key, and data areas.

The count field of the original record is used to control the writing of the new record. A record shorter than the original record will be automatically padded with zeros. A record longer than the original record will be written only to the extent of the area indicated in the count field on the track. Any bytes in excess of the number of bytes recorded in the count field will be lost. In either case, short or long records, IOCS will turn on the wrong-length-record bit in the error/status field.

Records Reference: Record Zero

This reference should be used each time the problem program reuses a certain portion of a pack. It may be used as a utility function to initialize a limited number of tracks or cylinders. Only one track at a time, however, may be initialized. This may be done by issuing a WRITE RZERO instruction with the address of each track to be initialized.

When the WRITE RZERO instruction is executed, IOCS writes a new Record Zero (R0) with the maximum capacity of the track (3625 characters) and erases the full track after R0. The user must supply the track information (cylinder and track number) in the track-reference field. Any record number is valid, but will be ignored.

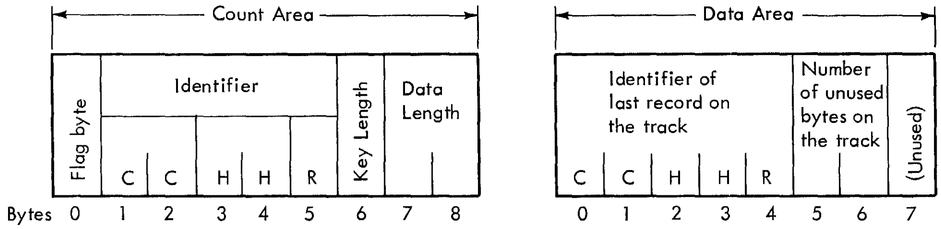


Figure 46. Record Zero (R0) with Capacity Record

Record Reference: After

This operation permits adding a record to a file without regard to the record key or record ID.

When WRITE AFTER is executed, IOCS examines and maintains the capacity record (Figure 46) in Record Zero to determine the location and amount of space available on the specified track.

If the space remaining on the track is sufficient, the information in the main-storage output area is transferred to the disk track immediately following the last record. The count area, the key area (if specified), and the data area are written on the track. The remainder of the track is erased. The capacity record is updated by IOCS.

If the remaining space on the track was not large enough for the record, IOCS will not write the record. A no-room-found indicator will be set in the user's error/status field.

Capacity Record: When random files are created, the address conversion routines should allow more file space than would normally be required to contain the number of records. Because of the nature of random file organization, the extra space is scattered throughout the file. The DAM technique provides an efficient means for maintaining a capacity inventory for this extra space. A capacity record is maintained by IOCS if the AFTER method is used to add records to a file.

The capacity record can be initialized with the Initialize Disk or the Clear Disk utility programs.

When records are to be added to a disk file by the WRITE AFTER method, IOCS uses the capacity record to ensure that each record fits on the track specified for the record. If the record fits, IOCS writes the record and updates the capacity record.

If the record is too large for the remaining area on the track, IOCS sets a no-room-found indicator in the user's error/status field. When the problem program detects a no-room-found indicator under these conditions, a problem program routine should be entered to place the record in an overflow area. A discussion of overflow areas for random files can be found under Random Addressing Techniques.

IOCS uses a portion of the data area of the first record on each track (R0) to maintain the track capacity inventory. Record Zero contains a count area and a data area as shown in Figure 46. The count area has a:

1-byte flag (is not normally transferred to or from main storage)

5-byte record identifier (CCHHR)

1-byte key length (K_L)

2-byte data length (D_L)

The capacity-record information is kept in the first bytes of the data area and consists of:

5 bytes - the identifier (CCHHR) of the last record written on the track

2 bytes - the number of unused bytes remaining on the track

1 byte - not used by DAM.

Record Reference by Key

The referencing of records by their key is significant to random processing of records. It permits the user to refer to records by the logical control information associated with the records. Whenever this method of reference is used, the problem program must supply the key of the desired record to IOCS before a READ or WRITE instruction is issued. When a READ or WRITE instruction is executed, IOCS will search the track, identified by the track reference field for the desired key. The search will be confined to one track unless multiple-track search is specified by the user (see Multiple-Track Search).

If the desired key is not found on the track, IOCS will place a no-record-found indicator in the user's error/status field.

When a READ instruction is executed and the desired key is found, IOCS will read the data area of the disk record into the main-storage input areas.

When a WRITE instruction is executed and the desired key is found, IOCS will transfer the data in the main-storage output area to the data area of the disk record. This replaces the information previously recorded in the data area.

The count field of the orginal record is used to control the writing of the new record. A record shorter than the original record will be automatically padded with zeros. A record longer than the original record will be written only to the extent of the area indicated in the count field on the track. Any bytes in excess of the number of bytes recorded in the count field will be lost. In either case, short or long records, IOCS will turn on the wrong-length-record bit in the error/status field.

MULTIPLE-TRACK SEARCH

The READ and WRITE macro routines of the DAM technique for processing disk files normally searches one track for the desired logical record. In many applications, a search through a complete logical file or a portion of the file would be appropriate to efficient system design. Therefore, the DAM technique provides a multiple-track-search capability. The user can specify a search of multiple tracks by including the DTFDA entry SRCHM (search multiple tracks) when the file to be processed is defined. When multiple-track search is specified, IOCS begins the search for a specified record key on the track specified in the track reference field. The search continues until one of two conditions occur. The search terminates when:

1. An equal compare occurs between the key argument (record key) in main storage and the record key on the disk tracks,

   or

2. The end of the specified cylinder is reached.

The search of multiple tracks continues through the cylinder even though part of the cylinder may be assigned to a different logical file. Therefore, it is important that the user organize his files or his program in such a manner that one of the foregoing two conditions properly terminates the search. One of the techniques used in this kind of file processing is to place the key argument into the last record location of the file on disk, before beginning the search. At the termination of the search the user must determine whether the record found is the record desired or the last record on the file.

By proper file organization and programming, the user can extend the multiple-track search to more than one cylinder. This is a problem-program responsibility, but IOCS provides the user with an end-of-cylinder indicator when the search reaches the end of a cylinder. This indicator is placed into the user's error/status field by IOCS.

ID LOCATION

The DAM technique requires that the user provide IOCS with both a track reference (MBBCCHH) and a record reference (CCHHR or KEY) for every record that is read or written. Because of this requirement DAM provides optional routines to aid the user when processing in certain conditions. For example, when record reference is by key and multiple tracks are searched, IOCS will make the ID (CCHHR) of the specified record available to the problem program. The user can use this ID to determine whether the record found was in the last location of the file.

To request that IOCS supply the ID, the user must place the symbolic name of a five-byte field in the DTFDA entry IDLOC. The symbolic name should be the same name the user writes in the DS statement to reserve the field in main storage.

IOCS supplies the ID of the record specified in the READ/WRITE instruction, or the next record on the track. The type of READ/WRITE function determines which ID is returned. The search multiple-track feature (SRCHM) is considered part of the definition of a READ/WRITE function.

Read/Write Function	ID Returned	
	With SRCHM	Without SRCHM
READ by KEY	Same record	Next record
READ by ID	(invalid)	Next record
WRITE by KEY	Same record	Next record
WRITE by ID	(invalid)	Next record
WRITE by AFTERID	(invalid)	none
WRITE by AFTER	(invalid)	none

Figure 47. The ID Returned by IOCS

Figure 47 indicates the various functions and the corresponding ID returned to the problem program.

ERROR/STATUS FIELD

When records in a disk-storage environment are being processed, certain exceptional conditions must be handled within the program. Because the method used for handling these exceptional conditions depends upon the application and the operating environment, the DAM IOCS routines provide the user with exception indicators.

The user must specify a symbolic name for the address of a two-byte field where IOCS is to place error/status information. The symbolic name is written by the user in the DTFDA entry ERRBYTE. When appropriate, IOCS will set one or more of the bits in this error/status field to indicate the following conditions. Before each I/O operation is started IOCS resets any bits that were set by a previous I/O operation.

Error/Status Field

FIRST BYTE

Bit	Error/Status Condition	Remarks
0	*	None
1	Wrong length Record (WLR)	For fixed unblocked records, wrong block size specified while using READID, WRITEID, READKEY, and WRITEKY.
2	*	None
3	*	None
4	No Room Found	Occurs only on WRITE AFTER and AFTERID when CAPREC=YES. No room to write record on track specified.
5	*	None
6	*	None
7	*	None

SECOND BYTE

Bit	Error/Status Condition	Remarks
0	Data Check in Count Area	Data error detected. Occurs when reading or searching a count field.
1	Track Overrun	Occurs only when using AFTERID. If CAPREC=YES is not specified, user must ensure record will fit or IOCS writes portion that fits and indicates record only partially written.
2	End of Cylinder	Occurs when record is not found on specified cylinder while using SRCHM.
3	Data Check Reading Key or Data	Data error detected when reading key or data or while searching keys.
4	No Record Found	Occurs when searching ID or key without SRCHM and ID or key is not found in specified track.
5	End of File	Count field with data length of zeros detected.
6	*	None
7	*	None

Note: * denotes bits reserved for future use.

These indicators are available to the problem program upon completion of the record transfer (after WAITF time). The user can tailor his exception-handling routine to fit his particular requirements.

LABELS

Any disk pack to be processed by logical IOCS must have System/360 Standard Disk Labels. No provision is made for processing disk files without labels. However, the DAM techniques provide for an exit from the standard label IOCS processing routines to the symbolic name of the user's routine for processing user labels. If the user requires one or more labels in addition to the standard labels, he must include his own routine to build and check the additional labels. The symbolic name of his routine must be specified in the DTFDA entry LABADDR. After IOCS has processed the standard labels, the program will branch to the user's routine. At the end of his routine, the user must return to IOCS by use of LBRET macro.

INDEXED SEQUENTIAL FILE MANAGEMENT SYSTEM

The Indexed Sequential File Management System (ISFMS) has been designed to provide a comprehensive file organization method for using disk-storage devices on the System/360. ISFMS accomplishes the disk-storage file organization objective of providing efficient handling of data records, while maintaining the over-all processing objectives of the system. The ISFMS has these advantages:

- An IOCS file management system specifically designed for direct access storage devices.
- Sequentially organized files that can be processed in random order or in sequential order.
- Both READ/WRITE and GET/PUT macro instruction routines available to the problem program.
- Routines for processing blocked or unblocked records.
- Record processing directly in the I/O area or in a work area.
- Presorted logical records that are loaded onto disk while a series of indexes are established for subsequent processing.
- An efficient chaining method for handling additions requiring an overflow area.

The ISFMS has these restrictions:

- Only one I/O area is permitted when a file is loaded.
- All physical data records must contain key areas, and all key areas must be the same length.
- Data records must be fixed-length only.
- Only Standard Disk Labels are permitted.
- For multi-pack files, all packs must be on-line for <u>any</u> function (loading, adding, retrieving randomly or retrieving sequentially) performed for the file.
- The prime data area for a logical file must be contained within <u>one</u> extent on a disk pack. It must start on the first track (track 0) of a cylinder, and it must end on the last track (track 9) of the same or a different cylinder. Prime data extents cannot start or end in the middle of a cylinder. For a multi-pack file, the prime data area must continue from the <u>last</u> track of one pack to the <u>first</u> track (track 0) of <u>cylinder 1</u> on the next pack so that the area is considered continuous by ISFMS. The first cylinder (cylinder 0) is reserved for labels.

PROCESSING DISK RECORDS BY THE ISFMS METHOD

ISFMS is designed to permit processing disk records in both random order and/or sequential order by control information. For random processing, the programmer supplies the control information (record key) of the desired record to ISFMS, and then issues READ or WRITE macro instructions to transfer the specified record. For sequential processing, the programmer specifies the <u>first</u> record to be processed, and then issues GET and/or PUT macro instructions to retrieve successive records (in sequential order by record key). For either types of processing, all packs in a multi-pack file must be on-line.

The DTFIS (<u>D</u>efine <u>T</u>he <u>F</u>ile for <u>I</u>ndexed <u>S</u>equential System) declarative macro is teamed-up with the READ/WRITE and GET/PUT imperative macros to permit the following:

- A logical file of records can be loaded onto a disk.

- Individual records can be read from, added to, or updated in the ISFMS created file.

- Overflow areas for handling additional records may be built by one of three methods, cylinder overflow, independent overflow or a combination of cylinder overflow and independent overflow.

- Several user exits permit the programmer to handle exceptions in a manner suitable for his particular requirements.

- A file can be reorganized by using the macro instructions for Sequential Retrieval and macro instruction for Load.

- A file can be updated when processing is done in random order or sequential order or when processing is done in both random and sequential order.

RECORD TYPES

Logical records in an ISFMS organized file must be fixed-length records either blocked or unblocked and every physical record in the file must contain a key area. If the records are blocked, the record key (control information) of the highest (last) logical record in the block is stored in the key area of the block.

One physical record format is available with ISFMS.

```
┌────────────┐ ┌──────────┐ ┌───────────┐
| Count Area | | Key Area | | Data Are  |
└────────────┘ └──────────┘ └───────────┘
```

The count area includes such information as a program-generated track and record address, the number of bytes in the key area, and the number of bytes in the data area.

The key area contains the record key or identifier of the information record.

The data area contains the information record.

The reader is referred to the section on Disk Record Formats in this manual for further details.

Two types of records can be used with ISFMS.

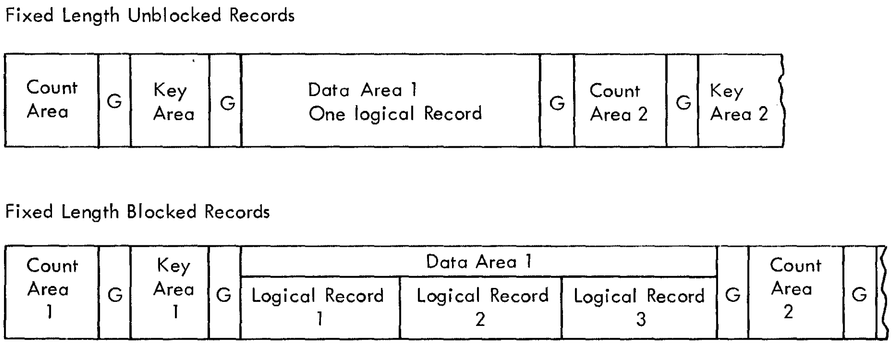
These record types are shown in Figure 48 and are described in the section Types of Records.

MAIN-STORAGE AREAS

I/O Areas

One I/O area must be specified for each ISFMS file to be processed in a problem program. This I/O area must be defined to contain sufficient space for the data area. If unblocked records are to be retrieved or records are to be loaded or added, space for a key field is required. Space for the count area must also be provided, when the file is being loaded or additions to the file are being made. Space for a sequence-link field is required, when additions are to be made to the file or when records are retrieved from a file. The sequence-link field is used for overflow records as described later (see Adding Records to a File).

The following main storage I/O areas required with ISFMS processing techniques are shown schematically in Figure 49.


Fixed Length Unblocked Records

Fixed Length Blocked Records

Figure 48. ISFMS Record Types

LOAD. To create or extend a disk file of blocked or unblocked records. This area must be defined with sufficient capacity for an 8-byte count field, a control-information field (key area), and the data record(s).

ADD, unblocked records. The output area for adding unblocked records to an ISFMS organized file must be defined with sufficient capacity for an 8-byte count field, a control-information field (key area), and a data-record area. The data-record area must have space for a 10-byte sequence-link field that is used in conjunction with overflow records (see Adding Records to a File). The sequence-link field is required when a record is written on an overflow track. ISFMS determines the correct sequence link and stores this information at the beginning of the data section of the I/O area. When the sequence-link field is not used, the ten unused bytes fall at the end of the data section and are ignored.

ADD, blocked records. The output area for adding blocked records to an ISFMS organized file must contain sufficient space for an 8-byte count field, a control-information field (key area) and a data section large enough to contain the block of logical records. The minimum size for the data section is one logical record plus 10-bytes to be used for a sequence-link field when required.

RETRIEVE, unblocked records. The input area for reading unblocked records must contain sufficient capacity for a key area and a data area. The data area must include enough space for the logical record plus 10-bytes for the sequence-link field of overflow records. If a record does not have a sequence-link field, the extra 10-bytes in the I/O area fall at the end of the data section and are ignored by the program.

RETRIEVE, blocked records. The input area for reading blocked records must contain space for a data area. The data area must be large enough to contain a full block of records. The minimum size of the data area is one logical record plus 10-bytes for the sequence link used with overflow records.

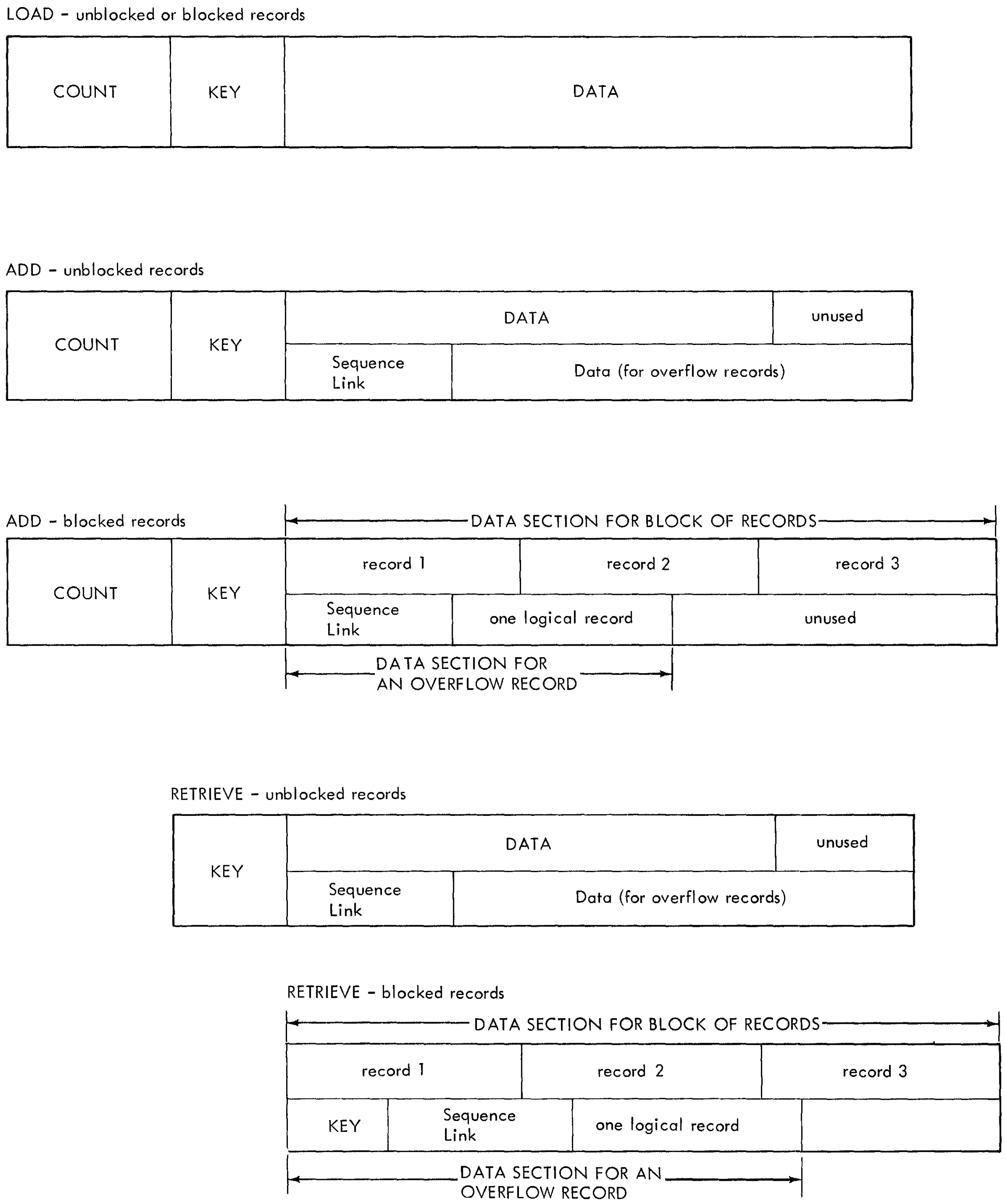

LOAD - unblocked or blocked records
COUNT
KEY
DATA
ADD - unblocked records
COUNT
KEY
DATA
unused
Sequence Link
Data (for overflow records)
ADD - blocked records
DATA SECTION FOR BLOCK OF RECORDS
COUNT
KEY
record 1
record 2
record 3
Sequence Link
one logical record
unused
DATA SECTION FOR AN OVERFLOW RECORD
RETRIEVE - unblocked records
KEY
DATA
unused
Sequence Link
Data (for overflow records)
RETRIEVE - blocked records
DATA SECTION FOR BLOCK OF RECORDS
record 1
record 2
record 3
KEY
Sequence Link
one logical record
DATA SECTION FOR AN OVERFLOW RECORD

Figure 49. Schematic of ISFMS Input/Output Main Storage Requirements

I/O Register

When blocked or unblocked records are to be retrieved and processed directly in the I/O area, a register must be specified. This register is used for indexing, to point to the beginning of each logical record when it is needed for processing.

Work Areas

A work area must be specified by a DTFIS entry when records are loaded or added. The work area must provide space for one logical record (including a key field if records are unblocked) when a file is loaded or extended. If records are being added to a file, the work area must contain space for one logical record (data area) and a control-information field (key area).

When blocked records and a work area are specified in the DTFIS entries for a file, an additional work area will be generated at assembly time. This additional work area is a duplicate of the user-specified work area and is used exclusively by the ADD macro. The ADD macro uses this area when a record is added to a file and records are shifted. For example, if a record is to be inserted into a block of records, each record in the block higher than the inserted record must be shifted to the next higher location. The second work area is required for this shift operation.

When a work area is specified on input, ISFMS moves each record from the I/O area to the work area. The problem program can then process the record in the work area. When a work area is specified on output, ISFMS moves the record from the work area to the I/O area in preparation for transferring the record to disk storage. If a work area is specified, an I/O register is not required.

LOADING OR EXTENDING A DISK FILE WITH ISFMS

Data records to be loaded onto a disk file must be sorted into sequence by record key, before being presented to the ISFMS load routines.

The data records are written by ISFMS onto disk tracks in an area of the file (called the prime area) specified by the user. The position of each logical record is a function of the record key used in the pre-sort operation; that is, each record is written one after the other into the prime area of the logical file. The user must specify one extent for the prime data area on one pack. If a file is to be loaded onto more than one pack, the prime data area must continue from the last track of one pack to the first track of another pack. Extents must be adjacent. In addition, all packs to be used for a multi-pack file must be on-line throughout the load operation.

Indexes

As ISFMS loads the records, it creates a set of two or three indexes to be used to control the processing and location of the data records. Two indexes, the track index and the cylinder index, are always developed for each file. The third, a master index, is built only when specified by the user. A master index should be specified only for large files. As a guide line, if a cylinder index occupies less than five tracks, it is usually faster to search only the cylinder index (followed by a search on the track index) than to search a master index, also.

Indexes are developed as a series of entries, each including the address of a disk track and the highest (last) record key on that track or cylinder. Each entry is a separate disk record composed of a key on that track or cylinder. Each entry is a separate disk record composed of a count area, a key area, and a data area. The key area contains the highest key on the track or cylinder, and its length (number of bytes) is the same as the key-area length specified by the user for the data records. The data area of each index record is 10-bytes in size and contains the physical address of the logical record or of another index.

TRACK INDEX: The lowest level index for a logical file is the track index. This index has two important functions.

- Point to the correct track in the cylinder that contains the specified key.
- Provide direct linkage to the record-overflow areas.

Each track index is built on the cylinder that it is indexing. The index of the first cylinder will be located on the first track specified by the user's job-control XTENT card for the prime area of the file. The XTENT specification must start with the first track of a cylinder. If the prime area is contained in more than one cylinder, the track index will be located on the first track of each additional cylinder. The index can occupy a partial track, a full track, or more than one track. If the track index does not fill a track, data records will be stored on the remaining portion of the track.

When first created, the track index is formatted (by ISFMS routines) with two entries for each track used on the cylinder. These two entries can be called the normal entry and the overflow entry.

Each entry is a disk record containing a key area and a data area.

The normal entry is the first of the two entries. After a track is loaded with records for a file, this entry has in its data area, the address of the track referenced by the entry. The key of the last record on the track is maintained in the key area of the normal entry. The key area is changed each time a record is added to the track, so that it will always reflect the key of the last record on the track. (See Adding Records to a File.)

The overflow entry is used only in the track index. It is required for handling overflow record chaining when additional records are inserted into the file. Before a record is added to a track, the overflow entry for that track is similar to the normal entry in that they both contain the key of the last record on the track and the address of the track. Note, at this point the last record on the track is the last record placed on the track when the file was originally loaded. When overflow records occur, the data area of the overflow entry is changed to reflect the address of the lowest record in the overflow chain. An overflow chain is developed for each track. The key area of the overflow entry is not changed, but will always contain the key of the highest record, because records added to a track will always have keys lower than the highest key originally loaded onto the track. The technique used to add records is explained in the section Adding Records to a File.

The last entry on a track index is always a dummy entry. The dummy entry indicates the end of the track index and indicates that any following records are logical file data records.

The key area of the dummy record is the same length as the user's key length and contains bytes of all one-bits. The data field is the same length as the normal entries but is a null field.

When the cylinder overflow option is specified by the user, the Record Zero in the track index will be a Cylinder Overflow Control Record (COCR). This entry will be set up in the data area of Record Zero (R0). The address of the last overflow record on the cylinder and the number of tracks remaining in the cylinder overflow area are maintained by ISFMS in this record. (see Adding Records to a File).

CYLINDER INDEX: The cylinder index is present for all ISFMS-organized files. It is an intermediate-level index used to point to the proper track index. A cylinder index is built by ISFMS to contain one index entry for each cylinder in the prime area of the file. This entry contains the highest record key associated (in the cylinder or a corresponding overflow area) with the cylinder, and the address of the track index for that cylinder.

The cylinder index can be located wherever the user chooses except on one of the cylinders that contains data records for the file. It can be placed on a separate disk pack, if the pack will be on-line whenever the logical file is processed. The cylinder index may also be located on one or more successive cylinders. When more than one cylinder is required, the last entry on each cylinder points to the first track of the next cylinder. However, the cylinder index can not be continued from one disk pack to another. A job-control XTENT card must be used to specify the proper location for this index.

The last entry on the cylinder index is a dummy entry. The key of the dummy entry is the same length as the user's key length and contains bytes of all one-bits. The data field is of the same length as the normal entries, but is a null field.

MASTER INDEX: The Master Index is the highest-level index for a logical file built by the ISFMS technique. This index is optional; and if required, must be specified by the user in the DTFIS entry MSTIND. The master index must be placed on a set of tracks that immediately precede the cylinder-index area. The last track assigned to the master index area must be contiguous to the first track of the cylinder index area. A job-control XTENT card must be used to specify the proper location.

The entries to this index point to each track of the cylinder index. Each entry contains the highest record key on the cylinder-index track and the address of that track.

The last entry on the Master Index is a dummy entry. The key of the dummy entry is the same length as the user's key length and contains bytes of all one-bits. The data field is of the same length as the normal entries, but is a null field.

Care should be used in selecting the Master Index option. Because of the speed and flexibility of the 2311 disk storage drives used on the System/360 only very large files will benefit from the use of this index. If the cylinder index occupies four tracks or fewer, it is usually more efficient to go directly to the cylinder index.

The disk storage space requirements for an Indexed Sequential organized file can be computed by using the formula given in Appendix M.

ISFMS Macro Instructions for Loading a Disk File

Three imperative macro instructions are available for loading a disk file with the ISFMS technique. These three macros are always required in the problem program used to originate or extend an ISFMS logical file on a disk pack.

The SETFL macro instruction initializes the load function. The macro sets up the disk file by formatting the track index on each cylinder specified in the job control XTENT card for the prime area of the data file. The prime area is where the data records are to be written. The SETFL macro also formats the master index area and cylinder index area if specified by job control XTENT cards. Entries to these indexes are in the form of a key area of the same length as the user's key length, and a data area of 10 bytes.

When used to extend a file, the SETFL macro instruction bypasses the formatting function, and simulates a restart condition to allow the load function to be performed. If the upper limit of the file is being redefined, the SETFL macro will format the new area added to the extent. The SETFL macro also handles the setup of the cylinder overflow control record (COCR) on each track index when this option is specified.

The WRITE macro instruction transfers each record to be loaded to the disk file and completes the construction of the appropriate indexes.

Before issuing the WRITE macro instruction, the problem program must place the record to be loaded into a user-specified work area. The user specifies the symbolic name of this work area with the DTFIS entry WORKL. Also, the problem program must place the record key in a key field in the work area (DTFIS WORKL).

When a WRITE macro is issued for unblocked records, the ISFMS routines will retrieve the record key and the data record from the work area and place them in the I/O area (see Figure 49). A count area will be constructed in the I/O area. Then the count, key, and data will be transferred to the proper location in the prime area of the file on the disk pack.

If blocked records are being loaded, ISFMS builds the count area and the block of records in the I/O area. Then the ISFMS moves the key of the highest record in the block into the key area. ISFMS then transfers count, key and data to disk storage.

ISFMS checks each record as it is presented for out-of-sequence condition and duplicate records. If an out-of-sequence or duplicate record is detected, ISFMS will branch to the corresponding user's routine. The user specifies the symbolic name for these routines in the DTFIS entries SQCHEX and DUPREX, respectively.

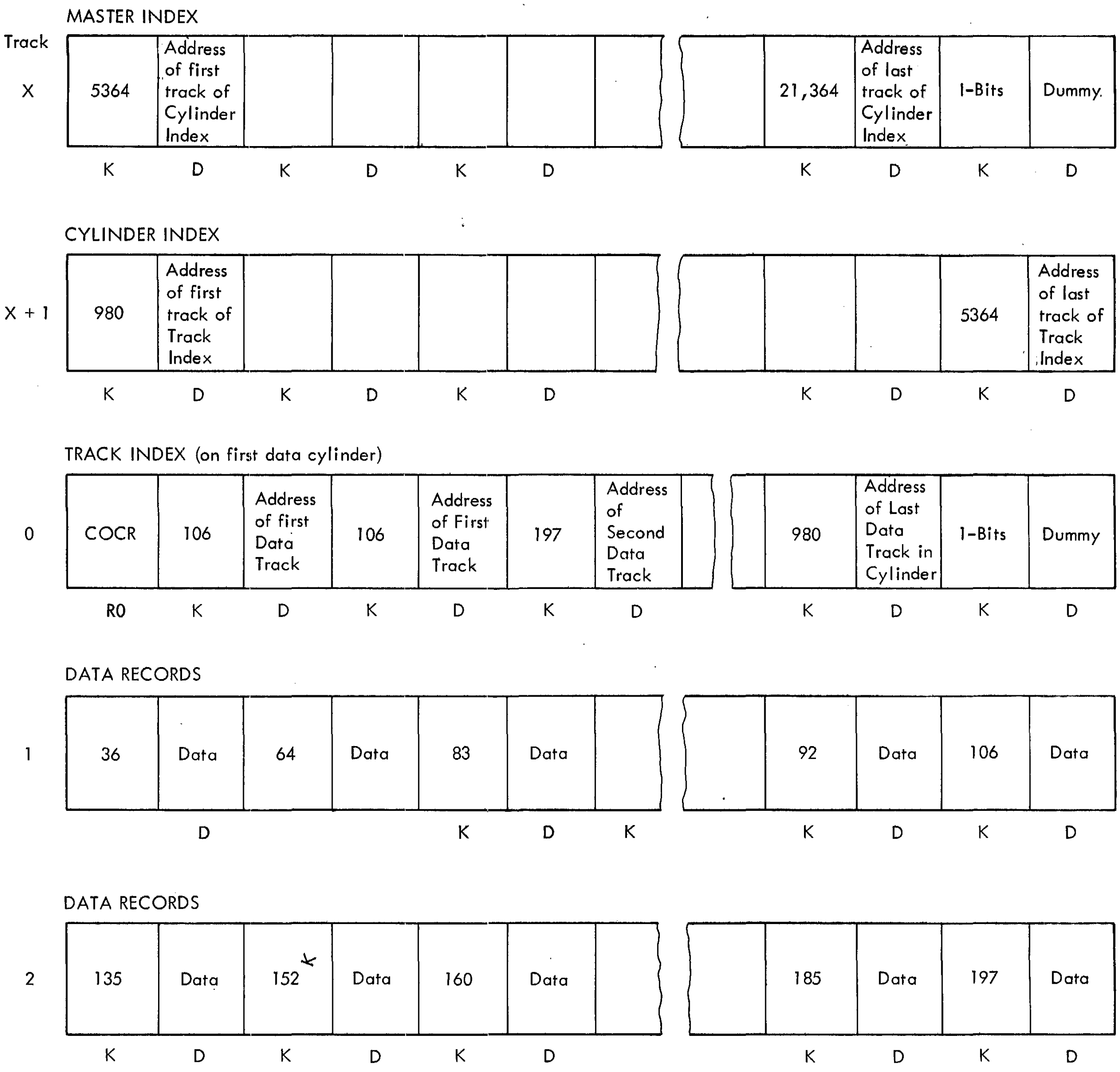
The ENDFL macro instruction performs a close-like operation for the file that has been loaded. When this macro is issued in the problem program, the last block of data records is written into disk storage. Then, an end-of-file record is written into the next record location in the logical file on the disk pack. An end-of-file record is a record with a data length of zero. The ENDFL macro completes the indexes by writing any index records required and a dummy entry at the end of each index.

Extending a Disk File

The same macro instructions used to load a file orginally can be used to extend the file. If it becomes necessary to increase the size of a file to contain records with keys higher than the last key on the original file, the records can be loaded at the end of the file. The upper limit of the prime area of the file can be adjusted by the specification in a job-control XTENT card, and the new records can be added by loading them into the file. No overflow area is required. The file is merely extended further on the disk pack.

Loading Records

As records are loaded onto disk, ISFMS writes track-index records each time a track is filled, a cylinder-index record each time a cylinder is filled, and a master-index record (if specified) each time a track of the cylinder index is filled. When a track index is completed ISFMS writes a dummy record following the last index record. This is used in subsequent operations to indicate the end of the index, and the beginning of data records. A schematic example of the three indexes after loading a file is shown in Figure 50. The highest record key on the first prime data track is 106. This is reflected on the track index in the first 2 entries. The highest record key recorded on the track index is 980, the highest key in the cylinder. This key is put in the first entry of the cylinder index. The highest record key recorded on the first cylinder index track is 5364. This record key is put in the first entry of the master index. The highest key in the file, 21364, is the highest key on the master index. If the record with key number 160 was required on a random processing step, the first comparison would be made in the master index. Because 5364 is higher than 160, the address at 5364 would be used to point to the cylinder index. The 160 would be compared to the first key. Again, 160 is lower than 980, so the 980 address points to track index 1. A key search is made on the track index. Again 160, is higher than the first index entry, but lower than the second index entry. The second index entry points to the second data track. A key search is again made and on an equal compare the record for key 160 is found.


MASTER INDEX
Track
X
5364
Address of first track of Cylinder Index
21,364
Address of last track of Cylinder Index
1-Bits
Dummy
K D K D K D K D K D
CYLINDER INDEX
X + 1
980
Address of first track of Track Index
5364
Address of last track of Track Index
K D K D K D K D K D
TRACK INDEX (on first data cylinder)
0
COCR
106
Address of first Data Track
106
Address of First Data Track
197
Address of Second Data Track
980
Address of Last Data Track in Cylinder
1-Bits
Dummy
R0 K D K D K D K D K D
DATA RECORDS
1
36
Data
64
Data
83
Data
92
Data
106
Data
D K D K K D K D
DATA RECORDS
2
135
Data
152
k
Data
160
Data
185
Data
197
Data
K D K D K D K D K D

K, indicates key area
D, indicates data area

Figure 50. Examples of Three Indexes After a File is Loaded (ISFMS)

ADDING RECORDS TO A FILE

Additional records for a data file can usually be expected in most disk-file applications. Usually these additions are too few in number to warrant reorganization of the file or they occur when reorganization is inconvenient. ISFMS macro routines are available for adding records to a file without disrupting the file organization and without causing undue system scheduling conflicts.

Because additions can occur anywhere in a file, it is more likely that they will occur between items stored within a track of data than at the end of a track. ISFMS, therefore, will insert additional records in their proper sequence on a track by moving higher numbered (record key) records toward the end of the track. Records shifted off the end of the track are moved to an overflow track on the same cylinder or to a separate overflow area on some other cylinder in the same pack or another pack on-line during file processing. Entries are then made in the track index to reflect the new records and to point to the overflow records. An insertion to a file affects three tracks: the prime data track that receives the addition; the track index of that cylinder; and the track in the overflow area that receives the record displaced from the prime track.

Further insertions into a prime track will lead to additional displacement of records from the prime area into the overflow area. These additions to the same track in the prime area cause an overflow chain to develop. This chain is defined by the sequence-link field appended to each record placed in an overflow area. Any insertion made between the last record of a track in the prime area and a record in the overflow area will be referred directly to the overflow area by the overflow entry on the track index. Each overflow record for any given track will be in sequence by the sequence links, but positioning (physical location) in the overflow area is according to the relative time of assignment to that area.

Except when adding records to the end of the file, no insertions will be made that have a key higher than the highest key already loaded into a cylinder. Therefore, the cylinder index (and the master index, if used) will not be changed by the insertion of records within the file. The reason for this is that a search for any key higher than the highest key associated with a cylinder (on the cylinder or a corresponding overflow area) will be referred to the next cylinder in the file. The cylinder index will make this referral directly to the track index on the next cylinder. This is true for all cylinders except the cylinder containing the end-of-file records.

If there is an addition that has a record key higher than all the records that are presently in the file, a special routine is used to add this record. This ISFMS routine will determine whether the last track containing data records is full. If it is not full, the addition will replace the end-of-file record, on the track. The end-of-file record will then be written in the next record location on the track or the next available prime data track, if no locations remain on the original track.

When the end-of-file record becomes the first record on a track, all records higher than the last record on the preceding track will be written in the appropriate overflow area. The sequence link for records that overflow after the end-of-file record is written, will be chained to the last track containing data records.

After each new record is inserted in its proper location, ISFMS adjusts all indexes that are affected by the addition.

When the file is loaded, the last track of the last cylinder in the prime data area must be left empty. This track prevents the end-of-file record from being written over a record for another file.

All packs in a multi-pack file must be on-line when records are being added to a file.

ISFMS Macro Instructions for Adding Records to a File

To add records to a file, the user can specify either the add or add/retrieve functions of ISFMS. The file may contain either blocked or unblocked records.

Only one macro instruction, WRITE is required to add records to a file.

Before the WRITE macro is issued, the program must store the record to be added into a work area specified in the DTFIS entry WORKL. Also, the record key must be stored in the main-storage key field specified in the work area, WORKL. Before records are transferred, ISFMS checks for duplicate record keys. If a duplication is found, ISFMS branches to the user's routine specified in the DTFIS entry DUPREX.

Blocked Record File. Records can be added to a blocked record file one at a time. Each record to be added must contain a key field in the same location as the records already in the file. The high-order position of this key field relative to the left-most position of the logical record must be specified to ISFMS by the user. The DTFIS entry KEYLOC is used for this specification.

When the WRITE macro is issued in the problem program, ISFMS first locates the correct track by referring to the necessary indexes. Then a search on the key areas of the disk records on the track is made to locate the desired block of records. The block of records including the key area is read into the I/O area (Figure 49). ISFMS then examines the key fields within each logical record to find the exact position to insert the record. The record in this position is moved by ISFMS to a special work area established by the ADD routine. The record to be inserted (in the user's work area) is then moved into the proper position in the block. Each succeeding record in the block is shifted to the right until the last record is moved into the special work area. ISFMS then updates the key area of the disk record and writes the block back onto the disk pack. The remaining blocks on the track are similarly processed until the last logical record on track is moved into the special work area. This record is then set up as an overflow record with the proper sequence links and moved to the overflow area. The indexes are updated and ISFMS returns to the problem program for the next record to be added.

If the proper track for a record is an overflow track, ISFMS writes the record, preceded by a sequence-link field in the data area of the record. The appropriate linkages are adjusted so as to maintain sequential order.

Unblocked Record File. When records are added to an unblocked file, ISFMS searches the indexes to find the correct track for the record. The record key placed in the key field is used by ISFMS to locate the correct track and the position on the track where the record should be inserted. If the correct track is not an overflow track, the record in the position where the new record must be inserted is read into the I/O area. The new record in the user's work area is written directly to the track from the work area. The record in the I/O area is moved by ISFMS to the user's work area. The next record on the track is read into the I/O area. Then the record in the work area is written on the track. Succeeding records are shifted until the last record on the track is set up as an overflow record. This last record is then written into the appropriate overflow area. This is the cylinder overflow area, if CYLOFL has been specified for this file, and the area has not been filled. If the cylinder overflow area does not have space available, or if only an independent area has been specified by a job-control XTENT card, the end record is transferred to the independent overflow area. ISFMS will branch to the user's routine specified by the DTFIS entry ADAREX, when all specified overflow areas become full and another overflow record has to be stored.

REORGANIZING THE DATA FILE

Reorganizing the data file may be necessary in many situations. This requires unloading the data file, merging the overflow records, and reloading the file into the original area. The Sequential Retrieval macro routines and the Load macro routines can be used for this operation.

If a file has grown since the previous loading, additional space will be required for the data file. The overflow records can be merged during the data file unloading operation. During this operation an output device, magnetic tape, cards, or printer can be used to produce a complete record of the data file. This record may be useful in a normal reporting procedure or an auditing function.

The original file can be unloaded onto another disk pack, a magnetic tape, or cards. If another disk pack is used, (two or more disk drives available) the reload step may not be necessary, because Sequential Retrieval and Load macros can be used to reorganize the file in one pass. The original file can be reorganized (in one pass) within the same disk pack, if there is sufficient space on the pack. None of the area occupied by the original file can be used by the reorganized file. If tape or cards are used, the data can be reloaded back on to the original disk pack. The tape or cards can then be filed to fulfill record retention protection requirements. Reorganizing a file can be a regular part of normal operations, but a careful analysis of the need for reorganization should be made. The time required to reload and reindex the file may be wasted if reorganizing is done too frequently. This time must be weighed against the processing time that can be wasted if overflow records cause too many additional accesses to the overflow areas. Also, reorganizing the file periodically frees space used by obsolete records.

The user can easily tag (or mark) records for deletion during normal processing runs. These deletion records can be omitted from the file by the user when he reorganizes the file. The space they occupied is then available for new records.

RANDOM RETRIEVAL

The indexes built by the ISFMS technique provide the ability to randomly retrieve and/or update records in a disk file. The user specifies the type of processing and other related items in the DTFIS entries for the file to be processed.

Because random reference to the file is by record key, the problem program merely needs to supply ISFMS with the desired record key prior to each READ command.

Two macro instructions are provided for retrieving and updating records randomly: READ, and WRITE.

Before issuing the READ instruction to retrieve a record, the user moves the record key of the desired record into a key field, the name of which he has specified in a DTFIS entry. The key specified in the KEY field for the READ instruction determines where the record is written, if the update function is specified.

The READ macro routines search the indexes to locate the track on which the record is stored. Then a key search is made upon the track to locate the record corresponding to the record key. If the records are unblocked, ISFMS transfers the appropriate record to the I/O area. When a work area is specified, the record is moved to the work area by ISFMS.

If the records on the file are blocked, ISFMS transfers the proper block of records into the I/O area.

If a work area has been specified by the user for blocked records, ISFMS moves the individual record to the work area for processing. If a work area has not been specified, the user must specify a register that ISFMS will use as a pointer to the proper record in the block.

The WRITE macro routines are used for random updating. These routines perform the required operation to rewrite the record retrieved by the preceding READ macro routines. The key specified in the KEY field for the READ instruction determines where the record is written.

SEQUENTIAL RETRIEVAL

Records in a file organized by the ISFMS technique can be retrieved sequentially in record key order. The user makes the proper entries in the DTFIS statements for the file. ISFMS uses the track index entries and the sequence-link fields to retrieve overflow records in their proper sequential order.

Sequential retrieval of records in key order can start with:

- The beginning of the logical file. (BOF)

- The location of any record in the file, identified by a record key. (KEY)

- The location of any record in the prime data area identified by the ID of the record. (The ID is in the form MBBCCHHR.)

The user specifies, in the SETL macro, the type of reference he will use to identify the first record to be processed.

If the file contains blocked records and the starting reference is by key, ISFMS must know the position of the key field in each logical record. This can be specified in the DTFIS entry KEYLOC by the user. With blocked records, the user can specify that retrieval begin with any record in the block by indicating to ISFMS the key of the first record to be retrieved.

If the user wishes to begin sequential retrieval at the beginning of the file, he specifies BOF as the second parameter of the SETL macro that initiates sequential retrieval. The first record retrieved will be the first data record of the file.

When sequential retrieval is to begin with a record associated with a particular key, the word KEY must be used as the second parameter of the SETL macro instruction. The record key of the beginning record must be placed in the field defined by the DTFIS entry KEYARG=name, prior to issuing the SETL macro.

Sequential retrieval can begin at any record location in the prime data area identified by its ID (MBBCCHHR). To establish the beginning point for retrieval, the IDs of the appropriate records can be determined as the file is loaded. During loading of the file, the ID of each physical record is available when ISFMS returns to the user program after each WRITE instruction. The ID is located in an 8-byte field referenced by filenameH (the name of the file suffixed by H). The user program can print or punch these IDs for later use. For example, assume that payroll records are being loaded in departmental sequence to establish a file named PAYRD. The ID of the first record in each departmental group can be saved (punched or printed) by referring to PAYRDH at the appropriate points while loading the file. In subsequent processing operations, retrieval can begin at the first record of any department by specifying the ID of that record as the beginning point.

To begin sequential retrieval at a record location identified by the ID of the record, the user specifies the symbolic name of the field in which he will place the ID of the beginning record in the second parameter of the SETL macro. The ID supplied by the user in this field must be in the form MBBCCHHR. Where, M is a number designating the extent where the records may be found. An extent must be specified for each disk pack that contains the prime area of a file. There can be only one prime area extent per file on a disk pack;

therefore the prime area must be contiguous. The user specifies the file extent with job-control XTENT cards.

The BB portion of the record identifier is reserved for cell number (relates to the IBM 2321 Data Cell Drive). These two bytes are always zero (0,0) for 2311 disk storage references.

CCHH is the track address in the prime data area where the record is located.

R is the physical location of the record on the specified track.

All numbers must be in binary notation.

Four macro instructions are available to the user for retrieving and updating records sequentially.

The SETL (set limit) macro instruction initializes the sequential retrieval routines. This macro sets up the starting record address from information supplied by the user.

The GET macro routines provide instructions to retrieve the record located at the starting address when the first GET is issued in the problem program. Thereafter each GET issued will retrieve the next record in sequence. The GET routines will transfer the record from disk to the user specified I/O area. If a work area is specified, the record is moved to the work area.

If blocked or unblocked records and no work area are specified, ISFMS makes each record available by supplying the address of the record in a register specified by the DTFIS entry IOREG. The key for unblocked records will be at the beginning of the I/O area.

When blocked records and updating are specified in the DTFIS entries, each GET that transfers a block of records to main storage will also write the preceding block back into the disk file, if a PUT instruction has been issued for at least one of the records in the block. When updating is not required for any record in the block (no PUT instruction has been issued), the block is not rewritten.

The PUT macro routines are provided to sequentially update a disk file. These routines provide the necessary instructions for transferring unblocked records directly from I/O areas to disk storage. When blocked records are processed, the routines develop instructions for moving a logical record from a work area (if specified) to the I/O area. Then, these instructions signal the GET routines that this block has been updated. At least one logical record must be updated (a PUT issued by the problem program) before the GET routines will transfer the block (completely processed) from the I/O area to the disk file. When a block has been completely processed, the GET that is issued to read the next block of records into main storage will determine whether the preceding block is to be returned to the file. If the block is to be returned (updated), the GET routines will write the block back onto the file in its proper location.

The ESETL macro instruction has two functions: one, write the last block of records back onto the disk file if a PUT has been issued for this block; two, put an end to the sequential mode of processing initiated by the SETL macro instruction.

PROCESSING WITH STR DEVICES

STR stands for Synchronous Transmitter Receiver. This term is used to classify a large group of devices that can be remotely attached to a System/360, Model 30, 40, 50, 65, or 75, through an IBM 2701 Data Adapter Unit with an IBM Synchronous Data Adapter - Type I. A System/360, so equipped, is itself an STR device. The device at the other end of the line can be:

- Another System/360, Model 30, 40, 50, 65, or 75 with a 2701 Data Adapter Unit attached.
- A System/360, Model 20, with a Communications Adapter.
- An IBM 1401, 1440, or 1460 System attached through an IBM 1009 Data Transmission Unit (or any system using an IBM 1414 I/O Synchronizer attached through a 1009).
- An IBM 1013 Card Transmission Terminal.
- An IBM 1974-2 Data Transmission Terminal.
- An IBM 1978 Print, Read, Punch Terminal.
- An IBM 7701 or 7702 Magnetic Tape Transmission Terminal
- An IBM 7711 Data Communication Unit.

All of these STR devices appear essentially the same to a program in the System/360. From the "point of view" of the program, the physical I/O unit actually being operated is the Synchronous Data Adapter, Type I, on the 2701. Each

Synchronous Data Adapter, Type I, (there can be two on a 2701, each with one or two communications lines attached) has a unique physical device address.

It is not meaningful to speak of a single "logical file" being processed from this device, because any or all of these STR devices can be transmitting and/or receiving over the line or lines. However, the device can be defined for logical I/O support using a macro language similar to that for conventional data files.

A DTFSN macro instruction in a problem program defines each specific synchronous data adapter for the STR routines. The only information supplied in the DTFSN macro instruction is the symbolic name (SYSnnn) of the adapter and the symbolic addresses of two fields used to supply the location and the length of the data to be transmitted or received.

A unique imperative macro instruction, SOPEN, is used with the STR routines to initialize the adapter for what really constitutes <u>logical file</u> processing for these routines. SOPEN turns on the adapter and establishes synchronization with another STR device. If the communications line is a dial network, the Automatic Call Feature must be present on the Synchronous Data Adapter, Type I. The SOPEN macro instruction monitors the line(s) for ringing, and establishes the connection and synchronism. If the other equipment necessary to complete the dialing operation is present (see Figure 51), the SOPEN macro instruction also performs the initial dialing operation. Another macro instruction, DIALO, supplies the telephone number and the parameters for SOPEN to complete the dialing operation, make the connection, and establish synchronism. Other parameters in the SOPEN macro instruction determine:

1. whether the first, second, or both line interfaces are to be used. Note: The Dual Communications Interface feature must be present to select the second, or both line interfaces;

2. the type of data checking to be performed;

3. the data transmission rate to be used expressed in characters per second;

4. the data transmission mode to be used: full duplex, four wire half duplex, or two wire half duplex.

Once the connection is established between the two devices, the user's program can issue the following imperative macros:

CNTRL PREPARE	to prepare the adapter for read operation.
CNTRL INQ	to signal the other device; to prepare to read.
CNTRL TEL	to signal the other device; to change mode.
CNTRL EOF	to signal end of file to the other device.
READ	to receive a single record from the other device.
WRITE	to transmit a single record; to the other device.
WAIT or WAITM	to wait for completion of one (WAIT) or more (WAITM) previously initiated operations.

After the transmission or reception of data to or from a given device is completed, a unique close macro instruction, SCLOS, is issued to break the connection. SOPEN can then be reissued with the same or different parameters to begin transmission or reception of another file to the same or another device.

The STR routines also provide a macro instruction, CDCNV, to convert the standard STR transmission code (fixed count four out-of-eight 4/8) to or from the EBCDIC used internally in the System/360.

Both the declarative and imperative macros for STR are discussed in detail in the Assembler with Input/Output Macros publication, listed in the Preface of this manual.

Communications Speed/Facilities	Common Carrier Switched Telephone Networks	Common Carrier Leased Private Line Services	Common Carrier Broadband Communication Services	Common Carrier Communication Facilities
1200 bps (150 char/sec) See Note.	Western Electric Data Set 202A, 202C, or equivalent.	Western Electric Data Set 202B, 202D, or Western Union Data Set 2121A or equivalent.	Not Applicable	Modem equipment (data set) having a proper interface must be used.
2000 bps (250 cps) See Note.	Western Electric Data Set 201A4, 201A or equivalent.	Western Electric Data Set 201A, 201A3, 201A4 or equivalent.	Not Applicable	Modem equipment (data set) having a proper interfact must be used.
2400 bps (300 cps) See Note	Not Applicable	Western Electric Data Set 201B or equivalent, Western Union 2241A, or equivalent.	Not Applicable	Modem equipment (data set) having a proper interface must be used.
19,200 bps (2,400 cps)	Not Applicable	Not Applicable	TELPAK A with Channel Terminal (includes Western Electric 303A10 data set) or equivalent.	Modem equipment (data set) having a proper interface must be used.
40,800 bps (5,100 cps)	Not Applicable	Not Applicable	TELPAK A with Channel Terminal A2 (includes Western Electric Data Set 301B) or equivalent.	Modem equipment (data set) having a proper interface must be used.
50,000 bps (6,250 cps)	Not Applicable	Not Applicable	TELPAK A with a suitable Channel Terminal (includes Western Electric 303A20 data set) or equivalent.	Modem equipment (data set) having a proper interface must be used.
230,000 bps (28,000 cps)	Not Applicable	Not Applicable	TELPAK C with a suitable Channel Terminal (includes Western Electric 303A30 data set), or equivalent.	Modem equipment (data set) having a proper interface must be used.

NOTE: The internal clock feature is required if the communication facility is half duplex or the data set does not provide clock pulses. In this chart, this feature is always required for Western Union Data Sets, for Switched Telephone Network operation, and for Western Electric 202C, 202D, and 201A4 data sets. The Internal Clock Feature cannot be used with the Western Electric 201A3 data set, and is not available with speeds other than 1200 bps, 2000 bps, 2400 bps, 4000 bps, and 4800 bps.

Figure 51. Communication Facilities and Transmission

BINARY SYNCHRONOUS COMMUNICATION

BSC (Binary Synchronous Communication) is used to refer to the communications environment consisting of an IBM 2701 Data Adapter Unit with an IBM Synchronous Data Adapter - Type II (SDA II) and connected by leased or dial line to a remote IBM System/360, Model 30, 40, 50, 65, or 75, equipped with an IBM 2701 or 2703 Data Adapter Unit, also with an SDA II.

From the problem program point of view, the actual physical I/O unit involved is the 2701, SDA II. The 2701, SDA II has a unique physical device address.

Macro support then provides for point-to-point Binary Synchronous Communication between the CPU and a remote CPU.

A DTFBS macro instruction in the problem program defines the Synchronous Data Adapter (SDA II). Information supplied in the macro instruction includes:

- The symbolic unit name of the SDA II
- The symbolic addresses to be associated with two fields used to supply the location and length of the data to be transmitted or received and with the BSC flag bytes
- A count of the number of times

operations resulting in I/O errors are to be retried before returning to the problem program or issuing an operator's message

The maximum count that can be specified is 15. The recommended retry count is either 3 or 7.

The unique imperative macro instruction, BOPEN, is used to establish the mode, and on a dial line, to turn on the Data Adapter (SDA II) for logical file processing with BSC macro routines. Parameters on the BOPEN macro instruction include the symbolic name of the DTFBS for the dial line, the adapter interface (A or B) to be used, and whether the line is a dial line or a leased line.

On a dial line, the BOPEN macro instruction must be followed by an IDIAL macro instruction. The IDIAL macro instruction performs initial line control functions necessary or dial lines; it handles dialing a number, monitoring a line for "ringing" and ID-verification procedures. IDIAL also reads or writes one text record from or to the remote CPU. The dial digits (telephone number) and ID-characters are provided in a parameter list pointed to by the general register specified among the IDIAL operands. Other operands on the IDIAL macro instruction determine:

1. Whether the CPU is calling, answering, or establishing the connection manually. If the CPU is calling, the Automatic Call Feature must be installed.
2. The kind of ID-verification procedure, if any, to be performed.
3. Whether the text record is to be read, written, or written as transparent text.

The ID-verification procedures, if included, provide a means for two CPUs connected by a dial line to identify themselves by exchanging sequences of graphic characters. The problem program may send ID-characters, receive ID-characters, or both send and receive ID-characters. ID-checking is performed by the CPU that receives the ID-sequence. Once the indicated ID-sequences and responses are validated, the text record is read or written.

On a leased line, the BOPEN macro instruction should be followed by a CNTRL (control) operation (either Prepare or ENQ).

CNTRL	Prepare	Monitors the line for activity in preparation for a READ.
CNTRL	ENQ	Bids for the line by signalling the remote CPU to prepare to READ - normally precedes a WRITE.

Once the connection has been established between the two CPUs, the following imperative macros may be used by the problem program.

CNTRL Prepare	See above.
CNTRL EOT	Signals end of transmission to the remote CPU.
CNTRL WABT	Signals Wait Before Transmitting to the remote CPU and waits for a response.
CNTRL Disconnect	Signals to the remote CPU that the connection is being broken (the line is being disabled) at this CPU. Used only on a dial line.
CNTRL ENQ	See above.
READ Continue	Receives one record from the remote CPU.
READ Continue with leading graphics	Sends graphic characters to the remote CPU, then receives one record from the remote CPU.
READ Repeat	Requests retransmission of the last record.
READ Repeat with leading graphics	Sends graphic characters to the remote CPU before requesting retransmission of the last record.
READ Inquiry	Receives the ENQ control character
WRITE Continue	Transmits one record to the remote CPU.
WRITE Transparent Text and WRITE Transparent block	Transmits one record (or block) of transparent text to the remote CPU with the correct End-character sequence.

WRITE Conversational   Sends one record to the remote CPU and receives one record (or control character) from the CPU.

WRITE Transparent Conversational   Sends one record of transparent text with the correct End-character sequence to the remote CPU and receives one record (or control character) from the remote CPU.

WAIT or WAITM   Waits for the completion of a previously initiated I/O operation.

When data transmission or reception is completed, the unique macro instruction, BCLOS, is used to break the connection. BOPEN (and IDIAL) can then be reissued with the same or different parameters to initiate processing of another logical file.

When end-of-job is reached, the ERRPT macro must be used to display the error statistics (data check, lost data, intervention required, time out, and unit check) and the transmission count. The ERRPT macro should be followed by the EOJ macro.

The declarative and imperative macro instructions for BSC Support are more fully described in the Assembler with Input/Output Macros publication, listed in the Preface of this manual.

IBM 2311 DISK STORAGE DRIVE

This section is organized as shown in Figure 52. The IBM 2311 Disk-Storage Drive features removable, interchangeable disk packs, offering virtually unlimited data-storage capacity. A disk pack can be easily removed and replaced with another pack in less than a minute. These units

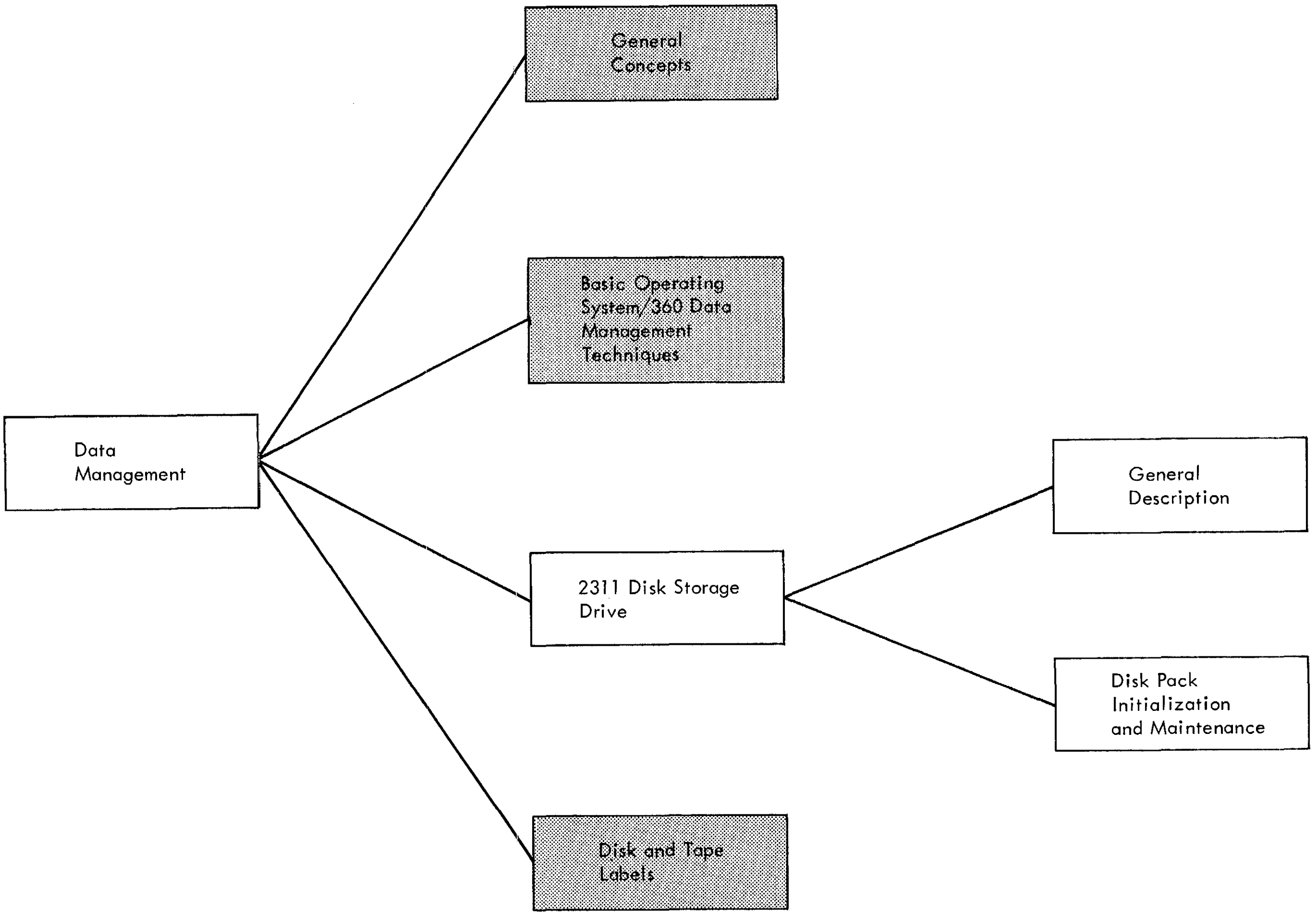

General Concepts
Basic Operating System/360 Data Management Techniques
Data Management
General Description
2311 Disk Storage Drive
Disk Pack Initialization and Maintenance
Disk and Tape Labels

Figure 52. Data Management: 2311 Disk Storage Drive

have flexibility comparable to a tape system, plus the advantages of direct-access processing.

Disk Pack

The interchangeable 1316 disk packs contain six, 14-inch diameter magnetic recording disks in an assembly weighing about 10 pounds. The packs provide ten disk-storage surfaces each (the top surface of the upper disk and the bottom surface of the lower disk are not used for recording).

Cylinder Concept

The corresponding recording tracks on each disk surface are physically located one above the other, and may be pictured as forming 203 concentric cylinders of 10 data tracks each. Figure 53 is a schematic representation of the cylinder concept.

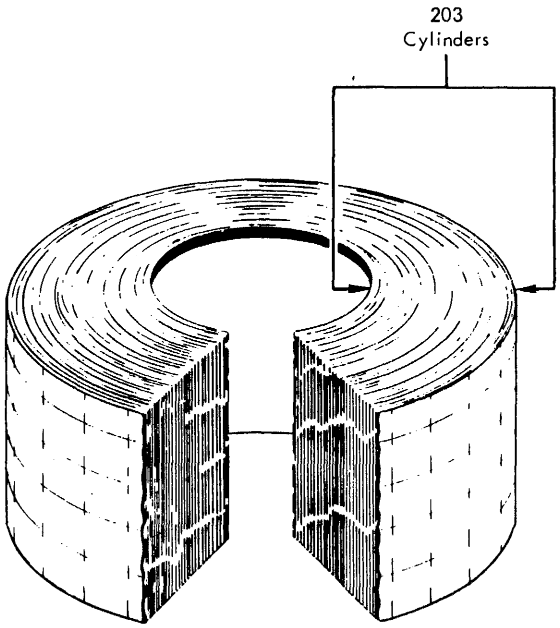


Figure 53. Cylinder Concept

Access Mechanism

Ten read/write heads are mounted on a vertical assembly. The heads are aligned vertically and are all moved together horizontally to any of 203 positions. Therefore, each time the read/write heads are moved into position, one entire cylinder of ten data tracks is accessible for reading and writing. Only electronic switching of the heads is necessary to select a particular track within the cylinder. Figure 55 illustrates the access mechanism and the disk pack.

Storage Capacity

The 7.25 million byte capacity of each disk pack is based on 200 tracks per disk surface. With the high-density recording of the 2311, minute contamination particles can affect data reading and writing. Therefore, 203 tracks per disk surface are provided to ensure that the stated capacity is maintained for the life of the disk pack.

Because each record has certain nondata characters, such as disk addresses, the net data-storage capacity of tracks may vary. Figure 55 indicates that, on the basis of one 3625-byte record per track, each disk pack can store 7,250,000 bytes. Figure 56 shows the number of bytes per record according to the number of equal-length records per track. The figure is used only for illustration; in actual practice records on the same track can vary in length in both the key areas and the data areas. Formulas for determining record capacity per track are available in the Component Descriptions publication, listed in the Preface of this manual.

Disk Track Format

The 2311 Disk-Storage Drive uses a track format consisting of an index marker, a home address, and one or more data records. An address marker precedes each data record (except the first one, as explained later) to indicate the beginning of a new record. Each track area is separated by a gap. Figure 57 is a schematic representation of a disk track. The various areas of the disk track format are described as follows:

Index Marker. The index marker indicates the physical beginning of each track. There is one Index Marker per track.

Home Address. There is one home address per track. The address is 7 bytes in length and is recorded in binary code. The home address defines the location of the track in terms of the physical parameters of the files. Home addresses are written on a track by an initializing program, which will be explained later.

Figure 58 is a schematic representation of the home-address area.

The Flag Byte: The flag byte is recorded on the track during a write-home address operation. The flag information byte indicates the condition of the track and is automatically propagated to all records as they are recorded on the track.

The Address: The four bytes containing the cylinder number and head number give the physical location of the track.

The Check Bytes: Two check bytes contain the 16 check bits used to verify the validity of reading and writing. These check bytes are a function of the record-verification circuits of the system. They are automatically appended to each separate area written on a disk track. The two bytes are not included in the count field's key length and data length when the length of the corresponding areas are defined.

Record Zero (R0)

The first record on every track (record number zero: R0) is used primarily to facilitate the use of an alternate track, when the original track is found to be defective. Referred to as the Track Descriptor Record or Record Zero (R0) this record is unique, in that it is not preceded by an address marker and does not contain a key area (key length is always zero). Figure 59 is a schematic representation of the Record Zero.

The Count area is similar to other records as described below. The Data Area can be used to maintain updated information about the data records on the track. A discussion of the capacity-record portion of this data area is contained in the section Direct Access Method. For information about the Cylinder Overflow Control Record (COCR) see section Track Index.

Disk Record Format

There are three basic parts to each 2311 Disk Record; the Count Area, the Key Area (optional), and Data area. Figures 60 and 61 are schematic representations of a disk record with a key area and without a key area.

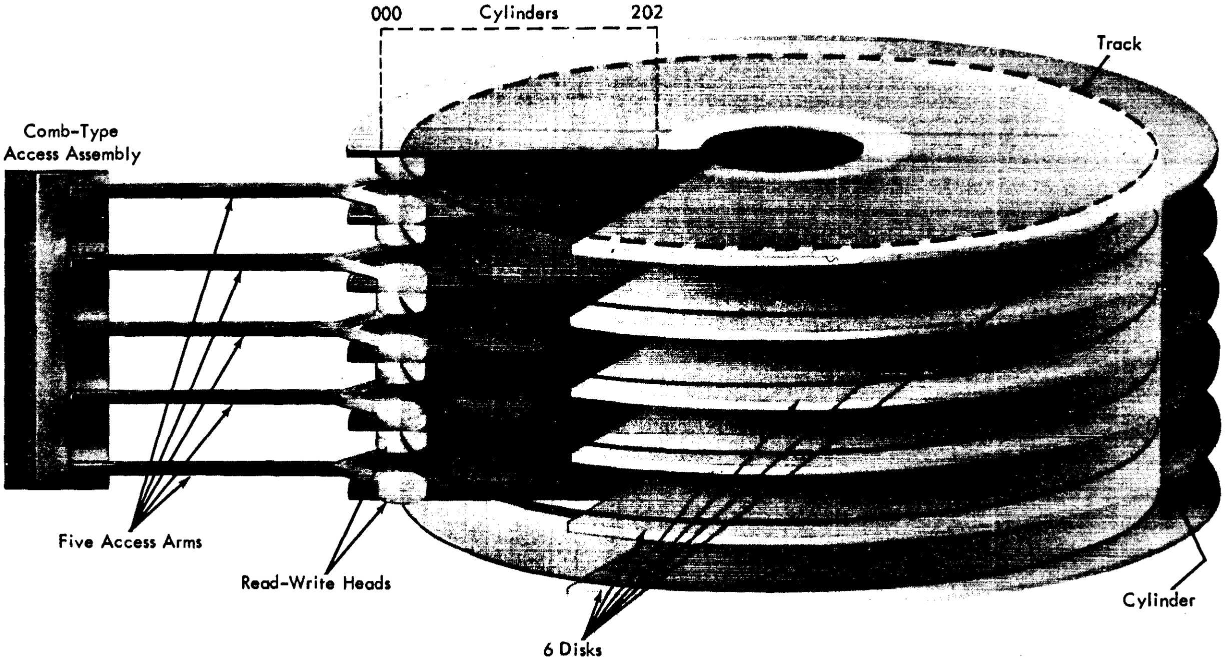


Figure 54. IBM 2311 Access Assembly and Disk Pack

	Per Track	Per Cylinder	Per Disk Storage Drive	Per Storage Control Unit
Disk Storage Drives				8
Cylinders			200	1,600
Tracks		10	2,000	16,000
Bytes (Alphameric Characters)	3,625	36,250	7,250,000	58,000,000
Packed Decimal Digits (Numeric Only)	7,250	72,500	14,500,000	116,000,000

NOTE: All figures are based on one record per track.

Figure 55. 2311 Disk Storage Drive Capacity

	Number of Equal-Length Records per Track																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Maximum Number of Bytes per Record without Key Field	3625	1738	1130	829	650	530	446	382	333	293	261	234	212	192	175	161	148	136	126	117
Maximum Number of Bytes per Record with Key Field	3605	1719	1110	810	630	511	426	363	314	274	242	215	192	173	156	142	129	117	107	97

Figure 56. Bytes per Record vs Records per Track

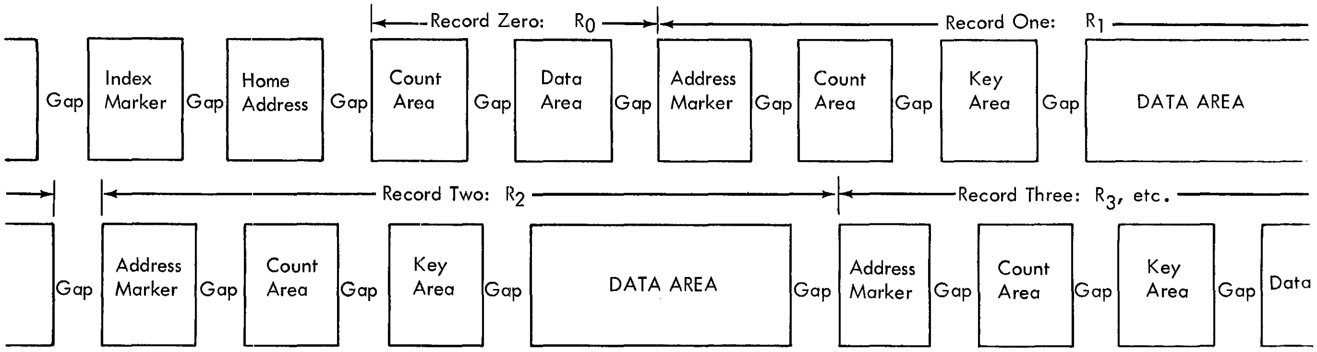


Figure 57. Schematic Representation of 2311 Disk Track (with Key Area)

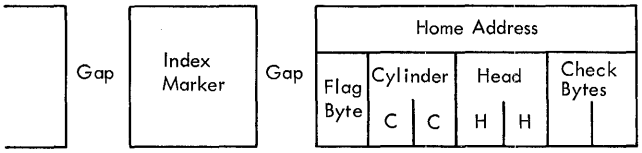


Figure 58. Schematic Representation of the Home Address

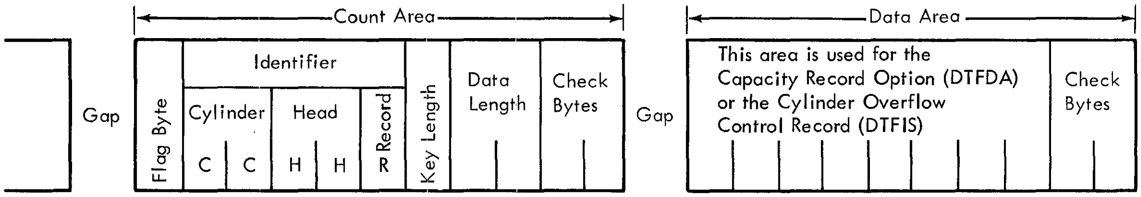


Figure 59. Schematic Representation of Record Zero

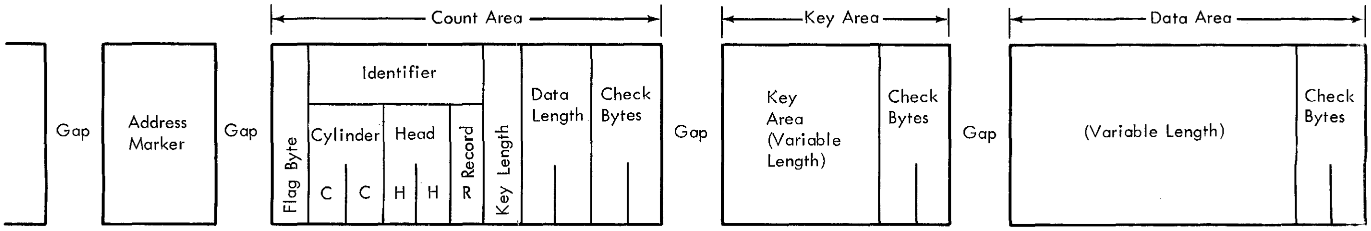


Figure 60. Schematic Representation of a 2311 Disk Record with a Key Area

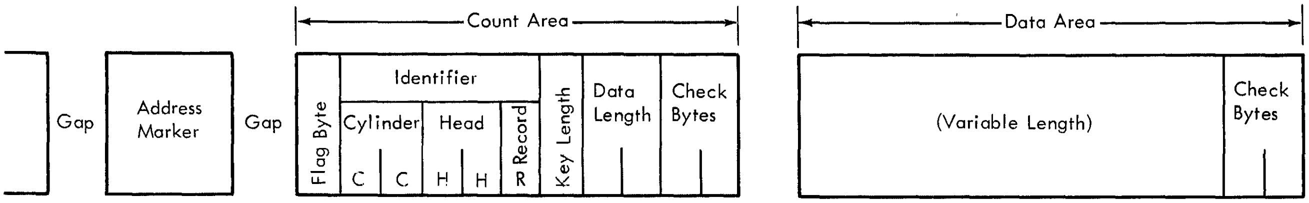


Figure 61. Schematic Representation of a 2311 Disk Record without a Key Area

The Count Area: The count area consists of the flag byte recorded from the home address flag byte, the identifier field, the key length, the data length, and check bytes. This area is recorded in binary notation.

The Identifier Field: The identifier field (Record ID) of five bytes contains the cylinder number, the head number and the record number to define the physical location of the record. The record number is the sequential position of the record on the track. Record number zero (R0) is the first record on the track and each succeeding record is numbered in ascending order.

The Key Length: The key length is one byte and denotes the number of bytes in the key portion of the record, but does not include the two check bytes. If the record does not contain a key area, key length is recorded as zero.

The Data Length: The data length is two bytes long and specifies the number of bytes in the data portion of the record, but does not include the two check bytes.

The Key Area: The key area consists of the key field and two check bytes. The key is an external number, such as part number, employee number, that identifies the information stored in the data portion of the record. This field will usually be the major control field of the logical data record to which it is appended. The key length can vary from zero to a maximum of 255 bytes.

The Data Area: The data area contains the information stored in the file, plus two check bytes. The data area can be one logical record or many logical records blocked together. Records on the same track can vary in length in the key area and in the data areas. The length of the data area is defined by the data length field.

DISK PACK INITIALIZATION AND MAINTENANCE

INITIALIZE DISK PROGRAM

An Initialize Disk utility program is provided as part of the IBM System/360 Basic Programming Support package. Initialize Disk is provided to prepare disk packs to be used on 2311 disk drives. As disk packs are received by an installation, this program is used to write Standard Home Addresses and Track Description Records (Record Zero) and to make a disk surface analysis to identify defective recording surfaces (if any). The program will write the volume label and establish the Volume Table of Contents (VTOC) area to be used for cataloging file labels. For further details on the VTOC see the section Labels.

The Initialize Disk program is also used to re-initialize disk packs when job requirements change. To guard against accidental reinitialization of packs containing valid data fields, the VTOC is checked for labels reflecting unexpired data files.

Home Address and Record Zero Generation

When the Initialize Disk Program writes the Home address and record zero fields on each track, a verification of the accuracy of recording is made. If a home address and record zero (R0) cannot be successfully written on a track, a message will be printed indicating the error. The pack will be deleted from the job if a home address and/or record zero cannot be written on one or more tracks. The remainder of the pack will be analized before the pack is deleted. The program will continue processing the next pack, if present.

Surface Analysis

Each track will be written and checked to ensure that data can be successfully written on the track. If a track is detected upon which data can not be validly written, an alternate track is established and a defining message is printed.

Alternate Track Capability

With the high-density recording of the 2311, minute contamination particles can affect data reading and writing. Therefore, three extra tracks per disk surface are provided to ensure that the stated disk pack capacity is maintained for the life of the disk pack. These extra tracks (in cylinders 200, 201, 202) are used as alternate data tracks when a defective track is discovered.

PHYSICAL IOCS AND ALTERNATE TRACKS

Physical IOCS disk error routines maintain a constant check on the condition of disk surfaces, if the record verification option is chosen when processing disk files. The VERIFY entry in a DTF statement will cause the record verification (a read-check CCW) operation to be performed on output files. The user can write a CCW (channel command word) to read each record following a write operation, when using physical IOCS. The execution of any read command will cause the entire record read to be checked for errors. The SKIP flag must be present in the read command CCW. The record is not transferred to main storage. Therefore, no additional main storage is required for the record verification operation.

Discovery of a Defective Track

During normal processing runs, defective disk tracks are originally discovered when physical IOCS has entered a disk error routine following a verified disk write operation. The track is considered to be defective after ten retries (that is, repeated write and read-check operations) in this disk error routine. When a record cannot be successfully written on a track, a message is given to the operator indicating that a defective track has been discovered by IOCS. The message will contain the address (cylinder and head number) of the defective track. Also, IOCS will post an indicator bit in the corresponding command control block (CCB). Recovery procedures must be provided by the user.

Previously Established Defective Tracks

Anytime a write disk command is executed during normal processing runs, the track condition bits in the identifier field (ID) of record zero (R0) are examined.

If the Flag byte (track condition bits) indicates that a track is defective, Channel End, Device End, and Unit Check (track condition check) signals are generated and an I/O interrupt will occur, indicating that the track is defective. The physical IOCS error routine will recognize this condition and take the following action:

Read the Record Zero (R0) of the defective track into main storage.

Extract the address of the alternate track from Record Zero.

Seek to the alternate track, and perform the desired I/O operation.

DISK AND TAPE LABELS

DISK LABELS

The contents of the section are illustrated in Figure 62.

The IBM System/360 Basic Operating 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.

Certain standard disk labels are required on all disk packs, although it is possible to process files with the physical I/O macros (EXCP) without processing any labels. When using any of the logical IOCS routines (DTFSR, DTFDA, DTFIS), labels must be processed for each data file. A special IOCS routine (DTFPH) is also provided to allow label processing when using physical I/O macros. In addition to the required standard labels, logical IOCS provides facilities for optional user labels.

The standard labels include one volume label for each pack and one or more file

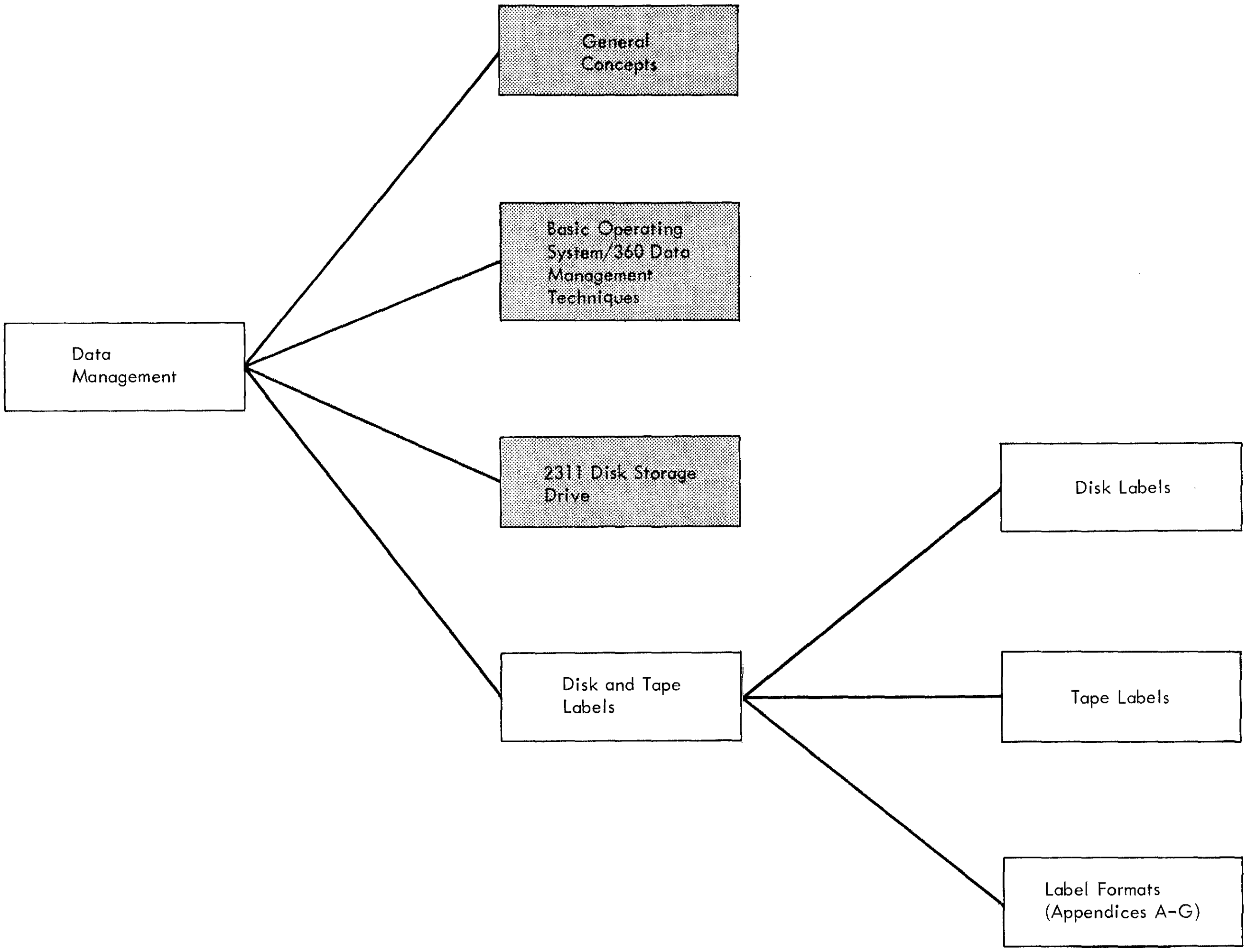


Figure 62. Data Management Disk and Tape Labels

labels for each logical file on the pack. There may be additional volume labels and, in some cases, user header labels and user trailer labels.

STANDARD VOLUME LABEL

The standard volume label identifies the entire volume (or pack). Every disk pack used in the Basic Operating System environment must have a standard volume label. It is always the third record on cylinder 0, track 0 (the first two records on this track are always IPL records). The volume-label record has a four-byte key field and an eighty-byte data field. Both the key field and the first four bytes of the data field contain the label identifier VOL1. The format of the data field is shown in Appendix A.

The volume label contains a volume serial number. This number is assigned to the pack when it is prepared for use in the system, and the number is never changed. It is repeated in the labels for all files on the pack.

The only other field in this label that is used by the Basic Operating System programs is the address of the area containing the file labels.

Additional Volume Labels

The standard volume label can be followed by one to seven additional volume labels (starting with the fourth record on cylinder 0, track 0). These labels must contain the label identifier VOL2, VOL3, etc in the four-byte key fields and in the first four bytes of the data fields. The other 76 bytes can contain whatever information the user requires. These labels are not read or processed by IOCS. If required, they must be read by using physical I/O macros.

Creation of Volume Labels

All volume labels (the standard label and any additional labels) are written by the Initialize Disk Utility program at the time a disk pack is prepared for use. The information in the standard volume label is checked, but never altered, during file processing.

STANDARD FILE LABELS

A standard file label or set of file labels identifies a particular logical file, gives its location(s) in the disk pack, and contains information to prevent premature destruction of current files. The Indexed Sequential File Management System also supplies and maintains information in the file labels to further define an indexed sequential file. Other fields within the file labels are set aside for use by the full IBM System/360 Operating System data management features. The number and format of labels required for any one file depends on the file organization structure and the number of separate areas of the pack (extents) used by the file (see Standard File Label Formats).

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 of 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) and is defined as such with its own file label. The label of the VTOC is the first record in the VTOC. This label identifies the file as the VTOC file and gives the file limits of the VTOC.

Preformatting the VTOC

The VTOC is preformatted by the Initialize Disk program. The user specifies its location and length when initially preparing a disk pack for use. It can be placed anywhere within the pack with the following restrictions:

1. It must be within cylinders 0-199 (cylinders 200-202 are used as the alternate-track area).
2. If in a pack used for system residence, it must be outside of the residence area.
3. It must be one or more full tracks, with the single exception noted in number 5 below.
4. It must be contained within one cylinder. It cannot overflow onto another cylinder.
5. If in a pack that is not used for system residence, it can begin on cylinder 0, track 0, immediately following the last volume label.
6. If a multi-pack file is to be processed by ISFMS and the prime data extends over two or more disk packs, the VTOC must start on cylinder 0 on all packs, except the first and last pack. On the first pack, the VTOC may be on any cylinder that precedes the prime data area. On the last pack, the VTOC may be either on cylinder 0 or on a cylinder that follows the prime data area. (See the section Indexed Sequential File Management System).
7. If only one disk pack is used, this pack can contain the system residence and the data file to be processed by ISFMS. In this case, the VTOC may be anywhere on the pack, except within the system residence or data file.

The Initialize Disk utility program preformats the entire VTOC by writing the foundation disk records that are to be filled in later. Each record location is written with a 44-byte key field and a 96-byte data field. Both these fields in each record are filled with binary zeros. The Initialize Disk program then writes the label for the VTOC itself in the first record location. This label is described as Format 4 under Standard File Label Formats.

The second record in the VTOC is also reserved at this time by inserting a hexadecimal 05 in each of the first four bytes of the key field and a EBCDIC value 5 in the first byte of the data field. This label is used by the Direct Access Device Space Management (DADSM) facility of the Operating System. It is not used or maintained by BOS programs. The label is described as Format 5 in the next section.

Standard File Label Formats

All standard disk file labels are written in the preformatted 140-byte records in the VTOC (44-byte key and 96-byte data). The field types contained within the labels written for data files follow three standard formats. In addition to these three formats, there are the two special labels: the one used for the VTOC itself, and the DADSM label (see Preformatting the VTOC). The format of a label is identified by the value in the first byte of the data field. This is an EBCDIC value from 1 to 5 indicating label format 1 to 5.

Format 1: This format is used for all logical files. It is always the first of the series of labels when a file requires more than one label on a disk pack (as discussed in Formats 2 and 3).

The Format 1 label identifies the logical file (by a file name assigned by the user and included in the 44-byte key area), and it contains file and data-record specifications. It also provides the addresses for three separate disk areas (extents) for the file. If the file is scattered over more than three separate areas on one pack, a Format 3 label is also required. In this case, the Format 1 label points to the second label set up for the file on this pack.

If a logical file is recorded on more than one disk pack, the Format 1 label is always the first label for the file in the VTOC on each pack. The Format 1 label is illustrated in Appendix B.

Format 2: This format is required for any file that is organized by the Indexed Sequential File Management System. The 44-byte key area is not used by BOS programs. The Operating System uses this area to describe the second-level and third-level master indexes. The 96-byte data area contains additional specifications unique to this kind of file organization (such as the track reserved for indexes).

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 additional packs (as the Format 1 label is). The Format 2 label is illustrated in Appendix C.

Format 3: If a logical file uses more than three extents on any pack, this format is used to specify the addresses of the additional extents. It is used only for extent information, and the entire 140-bytes provide for as many as 13 extents.

The Format 3 label is pointed to by the Format 1 label for the logical file or by a preceding Format 3 label. It is included as required on the first pack, or on additional packs if the logical file is recorded on two or more packs. The Format 3 label is illustrated in Appendix D.

Format 4: The Format 4 label is used to define the VTOC itself. This is always the first label in the VTOC. This label is also used to provide the location and number of available tracks in the alternate track area. The Format 4 label is illustrated in Appendix E.

Format 5: This label is used by the Operating System for Direct Access Device Space Management. The Basic Operating System programs do not use or maintain this label, although it is reserved by the Initialize Disk utility program. The Format 5 label is illustrated in Appendix F.

User Header and Trailer Labels on Disk

The user can include additional labels to further define his file, if he desires, provided the file is processed consecutively (DTFSR), by the Direct Access Method (DTFDA), or with the physical I/O macros (DTFPH). The DTFSR routine allows as many as eight user header labels and as many as eight user trailer labels. The DTFDA and DTFPH routines allow as many as eight user header labels, but no user-trailer labels. The DTFIS routine (for indexed sequential files) makes no provision for any user labels.

User header and trailer labels are not stored in the VTOC. Instead, they are written on the first track of the first extent allocated by the user for the logical file. The user label track is defined by IOCS as a separate extent in the Format 1 label for the file. If a file is written on two or more packs, the user label track is reserved in the first extent of each of the packs.

The user header labels are read after processing the standard file labels on the pack. As each user header label is read, the DTF routine branches to a routine supplied by the user to process the label.

User trailer labels are read when DTFSR reaches the end of the last extent on each pack. They are furnished to the user's routine in the same way as the header labels. Upon entry to the user's label routine, general registers 14 and 15 contain data that should not be destroyed. If the user's routine uses the GET, PUT macro statements or otherwise requires the use of these registers, the contents of these registers must be saved. The registers (14 and 15) must be restored prior to issuing the return macro LBRET.

All user labels must be eighty bytes long and they must contain standard information in the first four bytes. The remaining 76 bytes may contain whatever information the user wants. IOCS writes these labels with a key area of four bytes and a data area of eighty bytes.

User header labels are identified by UHL1,UHL2,...UHL8. The user must supply this identification in the first four bytes of the label. IOCS repeats this identification in the key area.

The identification for trailer labels that must be supplied by the user is UTL1,UTL2,...UTL8. IOCS writes UTL0,UTL1,...UTL7 in the corresponding key area of the user trailer label record. When user header labels are used without user trailer labels, IOCS writes a trailer label end-of-file record following the header label end-of-file record. The trailer label end-of-file record in the form UTL0 in the key area, with a data length of zero, indicates that there are no trailer labels.

Each user label set (header or trailer) is terminated by an end-of-file record (a record with data length 0). For example, if a file has five header labels and four trailer labels, the user label track contains:

R0	Standard Information	D_L = 80
R1	UHL1--user's 1st header label	
R2	UHL2--user's 2nd header label	
R3	UHL3--user's 3rd header label	
R4	UHL4--user's 4th header label	
R5	UHL5--user's 5th header label	
R6	UHL6--end-of-file record	D_L = 00
R7	UTL1--user's 1st trailer label	D_L = 80
R8	UTL2--user's 2nd trailer label	
R9	UTL3--user's 3rd trailer label	
R10	UTL4--user's 4th trailer label	
R11	UTL5--end-of-file record	D_L = 00

When the files are processed by the direct access method, or by physical IOCS defined with the DTFPH macro, only user header labels can by used. In this case the user label track contains:

R0	Standard information	
R1	UHL1--user's 1st header label	D_L = 80
R2	UHL2--user's 2nd header label	
	• • •	
R(n)	UHL(n)--user's nth header label, where n ≤ 8	
R(n+1)	UHL(n+1)--end-of-file record	D_L = 00
R(n+2)	UTL0--end-of-file record	

DISK LABEL PROCESSING

All disk label processing is performed by the transient label-processing routines of the Supervisor. These routines use the information supplied in the job control cards (VOL, DLAB, and XTENT) that was stored in the label information area in the resident pack. VOL (volume) and DLAB (disk label) cards must be supplied for each logical file, and an XTENT card must be supplied for each extent in which the file is located.

The DTFSR routine processes the labels of a consecutive file (input or output) one pack at a time. When the end of the last extent on a pack is reached, an automatic open is issued for the next pack. The DTFDA (Direct Access Method) and DTFIS (Indexed Sequential) routines require that all packs be on-line for the initial OPEN. The DTFPH routine will process labels one pack at a time and when the end of the last extent is reached, an automatic open will be issued for the next pack. Also, the DTFPH routine will process labels with all of the packs on-line for the initial OPEN.

The actual label processing consists of the following checks:

Disk Input Files

- The volume serial numbers in the volume labels are compared to the file serial numbers in the XTENT cards.
- Fields 1-3 in the Format 1 label are compared to the corresponding fields in the DLAB card. Fields 4-6 are then checked against their EBCDIC equivalents in the DLAB continuation card.
- Each of the extent definitions in the Format 1 and Format 3 labels is checked against the limit fields supplied in the XTENT cards. When using DTFDA or DTFPH, the user must provide an extent routine if he desires to process XTENT card information.
- If user header labels are indicated (when using DTFSR, DTFPH, or DTFDA), they are read as each pack is opened. After reading each label, the OPEN routine branches to the user's label routine to perform any processing necessary.
- If user trailer labels are indicated (when using DTFSR), they are read after reaching the end of the last extent on each pack. As with the user header labels, the trailer labels are processed by the user's routine.

Disk Output Files

- The volume serial numbers in the volume labels are compared to the volume serial numbers in the XTENT cards.
- The extent definitions in all labels in the VTOC are checked to determine whether any extend into those defined in the XTENT cards. If any do overlap, the expiration date is checked against the "today's date" in the communication region. If the expiration date has passed, the old labels are zeroed. If not, the operator is notified of the condition.
- The new Format 1 label is written with information supplied in the DLAB card and the DLAB-continuation card. If an indexed sequential file is being processed, the DTFIS routine supplies information for the Format 2 label.
- The information in the XTENT cards is placed in the Format 1 labels and, if necessary, additional Format 3 labels.
- If user header labels are indicated (when using DTFSR, DTFPH, or DTFDA), the user's label routine is entered to furnish the labels as each pack is opened. This can be done for as many as eight user header labels per pack. As each label is presented, IOCS writes it out on the first track of the first extent of the pack.
- If user trailer labels are indicated (when using DTFSR), the user's label routine is entered to furnish the labels when the end of the last extent on each pack is reached. This can be done for as many as eight user trailer labels. As each label is presented, IOCS writes it out on the first track of the first extent of the pack.

TAPE LABELS

A tape file processed by the logical IOCS routines must conform to certain standards. These standards concern labels, placement of tape marks, and the grouping (or blocking) of tape records. Considerations of blocked records were discussed previously in Part 4 under Types of Records. This section (Tape Labels) discusses the topics of tape labeling and the placement of tape marks (on both labeled and unlabeled files).

Tape files can be processed with or without labels. If labels are present, they are classified as either standard or nonstandard. The standard label set includes the following types of labels:

1. Standard Volume Label, fixed in length and format, processed by IOCS.
2. Additional Volume Labels, fixed in length and identifier, but not fixed format; bypassed by IOCS.

3. Standard File Label, fixed in length and format, processed by IOCS.

4. Additional File Labels, fixed in length, identifier, and format; bypassed by IOCS.

5. User Labels, fixed in length and identifier, but not fixed format; read and written by IOCS, processed by user routine.

The additional volume labels, additional file labels and user labels are classified as part of the standard label set even though they are not fixed-format. They are, however, standard in length (eighty bytes) and have standard label identifier fields. Nonstandard labels, on the other hand, are unrestricted in size, format, or identification. All these label types, and the rules governing their positioning are described in the following sections.

STANDARD TAPE LABEL SET

When standard tape labels are specified in the DTFSR entries, the mininum set of labels (Figure 63) allowed consists of:

1. One standard volume label per reel.

2. Two standard file labels for each logical file on the reel (one header label preceding the file and one trailer label following the file).

The user has the option of adding additional header and trailer labels (see example of Figure 64):

1. Up to seven additional volume labels.

2. Up to eight user header labels and up to eight user trailer labels.

TAPE VOLUME LABELS

Standard Volume Labels

The standard volume label identifies the entire volume (or reel). If standard labels are specified for a file, every reel used for the file must have the standard volume label. It is always the first record on the reel. It is eighty bytes long and follows a fixed format. The first four bytes contain the label identifier VOL1. The standard tape volume label is identical to the standard disk volume label except that the disk label contains the address of the file label area on the disk pack. The format of the standard volume label is shown in Appendix A.

The standard volume label contains a volume serial number. This number is assigned to the reel when it is prepared for use in the system. This number is never changed. It is repeated in the file labels for all files on the reel.

Additional Volume Labels

The standard volume label can be followed by up to seven additional volume labels. These labels are eighty bytes long and must contain the label identifier VOL2, VOL3, etc. in the first four bytes. The other 76 bytes can contain whatever information the user requires. These labels are not read or processed by IOCS. If required these labels must be read by using physical I/O macros.

Creation of Volume Labels

All volume labels (the standard label and any additional labels) are written by an IBM-supplied utility program at the time a reel is prepared for use. The information in the standard volume label is checked, but never altered, during file processing.

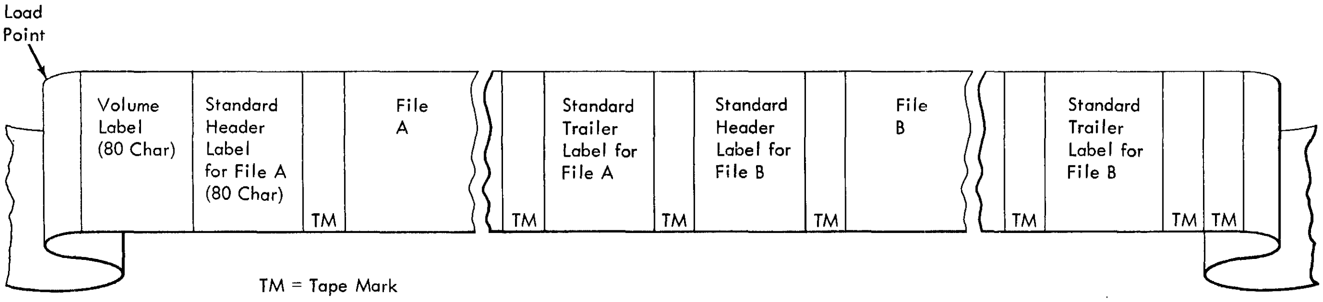

Load Point
Volume Label (80 Char)
Standard Header Label for File A (80 Char)
TM
File A
TM
Standard Trailer Label for File A
TM
Standard Header Label for File B
TM
File B
TM
Standard Trailer Label for File B
TM
TM
TM = Tape Mark

Figure 63. Tape File with Standard Labels

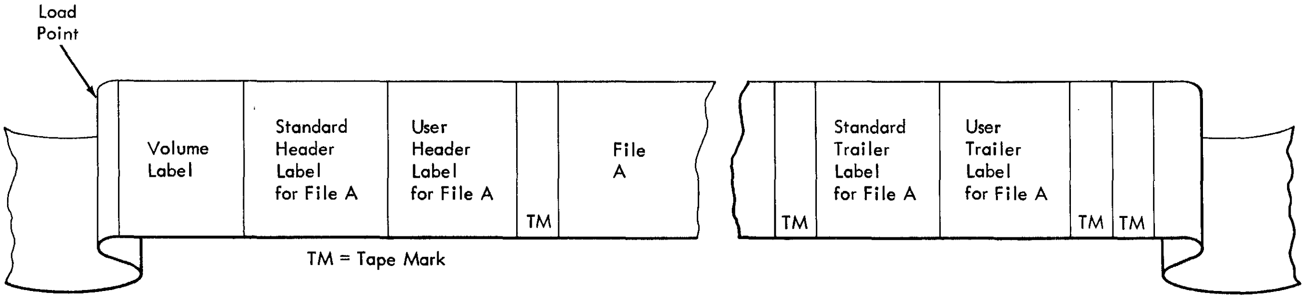

Load Point
Volume Label
Standard Header Label for File A
User Header Label for File A
TM
File A
TM
Standard Trailer Label for File A
User Trailer Label for File A
TM
TM
TM = Tape Mark

Figure 64. Tape File with Standard and User Labels

TAPE FILE LABELS

Standard File Label

Standard file labels are written before and after every logical file on a reel, if specified in the DTF. These labels are referred to as file header labels or file trailer labels, depending on their position and use. They are always eighty bytes long and always have the same format and content, with the following exceptions:

1. The label identifier field (bytes 1-3) contains:

    a. HDR to indicate a header label (precedes the data file).

    b. EOV to indicate an end of volume (end of reel) trailer label (written at the end of a reel, indicating that the file is continued on another reel).

    c. EOF to indicate an end of file trailer label (written at the end of the logical file).

2. The block count field is used only in the EOF and EOV trailer labels. This field is blank in the HDR label.

The standard tape file label is illustrated in Appendix G.

Additional File Labels

Each standard file label (one header and one trailer) can be followed by up to seven additional file labels. These labels are eighty bytes long and must contain the label identifier HDR, EOV or EOF in the first three bytes. The fourth byte should contain a character 2,3,...or 8, indicating the second, third ...or eighth file label. These labels are not read or processed by IOCS. If required, these labels must be read by using physical I/O macros.

User Header and Trailer Labels on Tape

The user can include additional header and trailer labels to further define his file, if he desires. As many as eight additional header labels can be written after the standard file header label, and as many as eight additional trailer labels can be written after the standard file trailer label (EOF and EOV). Each additional label in the set is eighty characters long. The first four characters of each additional label must contain standard identifying information. The remaining 76 characters can contain any information and arrangement desired by the user. The user header labels are identified by UHL1, UHL2...UHL8 in bytes 1-4. The user trailer labels are identified by UTL1, UTL2...UTL8 in bytes 1-4.

Header and trailer labels form the set of labels designated as file labels. The header label appears at the beginning of a file and the trailer label appears at the end of a file. Both contain the same fields. The fields which are checked differ, depending on whether the label is a header or a trailer.

When the file is to be read backward, the trailer labels must be complete so that, for checking purposes, the label can be treated as a header label. Thus only those fields which are pertinent to the type of check (header or trailer) are examined.

On 7-track tape, standard labels are written in the same density as the data on that tape. (All information on a tape reel must be written in a single density.) These standard labels are written with even parity in the translation mode.

TAPE MARKS WITH STANDARD TAPE LABELS

Figures 63 and 64 illustrate the use of tape marks with files that use the standard label sets. The sequence of items on the tape is:

1. No tape mark preceding header label set.

2. Header label set:
   Standard volume label (required)
   Additional volume labels (none to seven, optional)
   Standard file header label (required)
   Additional file labels (none to seven, optional)
   User header labels (none to eight, optional).

3. Tape mark between header label set and first data record.

4. Physical records for file.

5. Tape mark between last data record and trailer label set.

6. Trailer label set:
   Standard file trailer label (required at end of file and end of volume)
   User trailer labels (none to eight, optional).

7. Tape mark after trailer label set.

8. If multi-file reel, (EOF label) next standard file header label follows here. If single file reel (EOF label) or if last file of a multi-file reel, another tape mark follows here. If multi-reel file (EOV label) a tape mark follows here.

STANDARD TAPE LABEL PROCESSING

Standard tape label processing is performed by the transient label-processing routines of the Supervisor. These routines use the information supplied in the job control cards (VOL and TPLAB) that was stored in the label information area in the resident pack. Only one VOL (volume) card and one TPLAB (tape label) card need be supplied for each logical file, regardless of the number of reels on which the file is recorded.

The actual label processing consists of the following checks:

Tape Input File

- The volume serial number in the standard volume label on the first or only reel is compared to the file serial number in the TPLAB card. All other volume labels on all reels of the file are bypassed.

- The fields in the standard file header label on the first reel are compared to the corresponding fields in the TPLAB card. In the file header label, fields 1-10 are required; fields 11-14 are optional. For successive reels of a multi-reel file, the volume sequence number from the TPLAB card is increased by one for each reel.

If more than one file is written on a reel of tape (multi-file reel), the file sequence number determines the file to be processed. All files are bypassed until a file sequence number in a standard label matches the file sequence number in the TPLAB card, or until the end of the tape is reached. If the tape is positioned beyond the desired file when the search is started, a message is given to the operator.

- If user labels are indicated, they are read into main storage for processing by the user's label routines. The user labels are read one at a time, until all have been processed.
- When a standard file trailer label is read, the block count is compared to a count accumulated by IOCS.
- If user trailer labels are indicated, they are read into main storage for processing by the user's label routine. The user trailer labels are read one at a time until all have been processed.

Tape Output File

- The volume serial number in the standard volume label on the first or only reel is compared to the file serial number in the TPLAB card. All other volume labels on all reels of the file are bypassed.
- The expiration date in the standard file header label is checked against the "today's date" in the communication region. If the expiration date has passed, the reel is backspaced to write the new standard file label. If not, the operator is notified of the condition. This check is performed on each reel of a multi-reel output file. If no file label is present (tape mark after VOL label) the tape is considered expired.
- The new standard file label is written with the information supplied in the TPLAB card. For multi-reel files, the volume sequence number is increased by 1 for each successive reel.

  More than one file of records may be written on a tape reel (multi-file reel). The standard file header label for each file after the first is written with information taken partially from the corresponding TPLAB card, and partially from the preceding standard trailer label. The information from the trailer label is: the file serial number, the volume sequence number, and the file sequence number. The file sequence number is increased by 1 for the new file. If the tape has not been rewound (or otherwise re-positioned) after a file is closed, it is located at the correct position for writing the standard file header label of the next file. The header label of the new file is written immediately after the tape mark that follows the trailer label(s) of the previous file. If the tape has been moved, however, the user must properly position it before the header label is written. He can move the tape from the load point to the proper position by using a Job Control FILES card and skipping three tape marks for each file on the tape.

- If user header labels are indicated, the user's label routine is entered to furnish the labels as each reel is opened. This can be done for as many as eight user header labels per reel.
- If end of reel is sensed before completing the file, an EOV trailer label is written with all fields presented in the TPLAB card plus a block count.
- When end of file is reached, an EOF trailer label is written identical to the EOV label mentioned above.
- If user trailer labels are indicated, the user's label routine is entered to furnish the labels after each trailer (EOV or EOF) label is written. This can be done for as many as eight user trailer labels.

NONSTANDARD TAPE LABELS

Any tape labels that do not conform to the standard label specifications are considered nonstandard and must be read, checked, or written by the user. On input files, the nonstandard labels may or may not be followed by a tape mark. Therefore, four conditions are possible:

1. Nonstandard label(s), followed by a tape mark to be checked.
2. Nonstandard label(s), <u>not</u> followed by a tape mark, to be checked.
3. Nonstandard label(s), followed by a tape mark, which are <u>not</u> to be checked.
4. Nonstandard label(s), <u>not</u> followed by a tape mark, which are <u>not</u> to be checked.

For conditions 1 and 2, the DTFSR entries must specify nonstandard labels and the address of a user-written routine to do the reading or writing.

For condition 3, nonstandard labels must be specified, but the address of a user routine is omitted. IOCS skips all labels, passes the tape mark, and positions the tape at the first data record to be read.

For condition 4, nonstandard labels and a user address are specified, IOCS can not distinguish labels from data records because there is no tape mark to indicate the end of the labels. Therefore, to position the tape at the first data record, the user must read all labels.

With nonstandard labels when an end-of-file or an end-of-volume condition exists, the user indicates to IOCS which condition it is. On end of file, IOCS branches to the user's end-of-file address. On end of volume, IOCS initiates the end-of-volume procedures to close the completed volume and open the next volume for processing.

On output files, nonstandard labels are written by the user's routine by using physical IOCS. The OPEN routine writes a tape mark between the user's nonstandard header labels and his first data record, unless the TPMARK=NO entry is present in the DTFSR statements. The CLOSE routine writes a tape mark after the user's <u>last</u> data record before he writes his nonstandard trailer labels, and after the trailer labels.

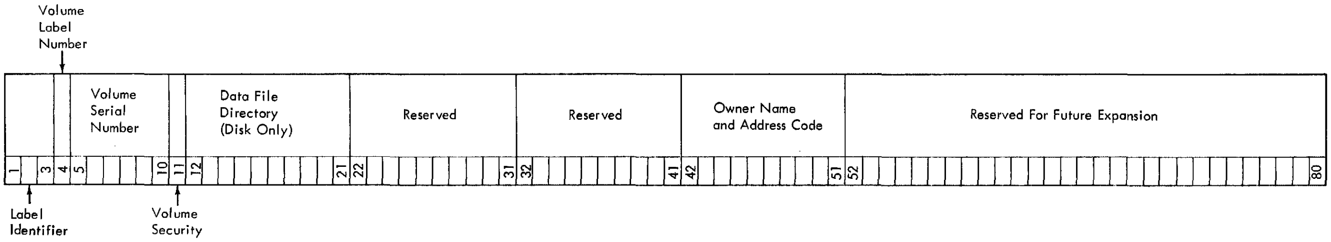
UNLABELED TAPE FILES

The first record of unlabeled <u>input</u> tape files (nine or seven track) may or may not be a tape mark. If a tape mark is not present as the first record, IOCS will process the first record as a data record.

When an unlabeled <u>output</u> file is specified, the OPEN routine assumes the mounted output tape is also unlabeled. No label checking is performed and any labels on the output tape are erased. The OPEN routine will write a tape mark as the first record of the output tape, unless the TPMARK=NO entry is present in the DTFSR.

<u>Note:</u> Unlabeled tapes (nine or seven track) can be read backwards if they: were written by a System/360, have a tape mark as the first record, and have not been written in the data conversion mode.

APPENDIX A. STANDARD VOLUME LABEL, TAPE OR DASD



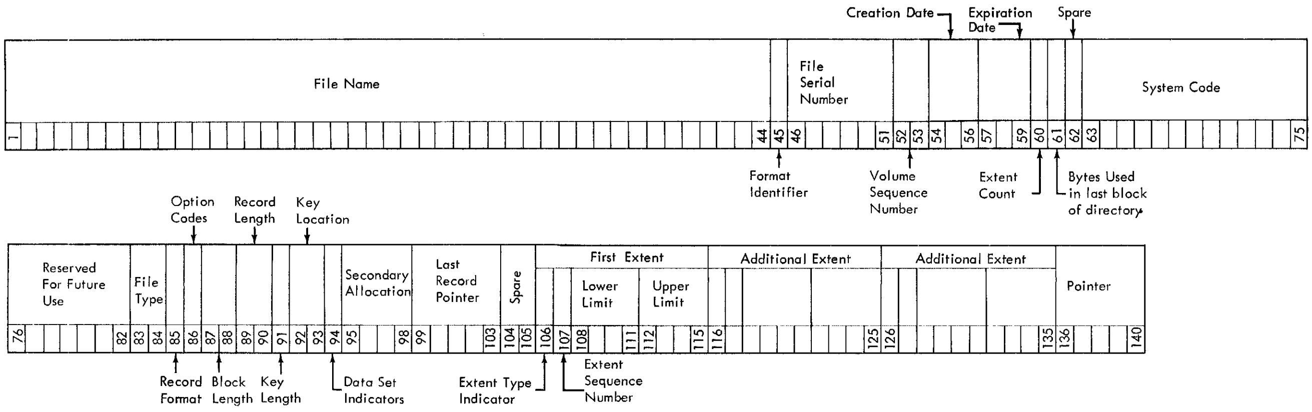
Volume Label Format (80 bytes) for Tape or DASD

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
1	1-3	LABEL IDENTIFIER 3 bytes	must contain VOL to indicate this is a Volume Label.
2	4	VOLUME LABEL NUMBER 1 byte	indicates the relative position (1-8) of a volume label within a group of volume labels.
3	5-10	VOLUME SERIAL NUMBER 6 bytes	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 alphameric.
4	11	VOLUME SECURITY 1 byte	indicates security status of the volume: 0 = no further identification for each file of the volume is required. 1 = further identification for each file of the volume is required before processing.
5	12-21	DATA FILE DIRECTORY 10 bytes	for DASD only. The first 5 bytes contain the starting address (CCHHR) of the VTOC. The last 5 bytes are blank. For tape files, this field is not used and should be recorded as blanks.
6	22-31	RESERVED 10 bytes	reserved for manufacturers.
7	32-41	RESERVED 10 bytes	reserved for American Standards Associated (A.S.A.).

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
8	42-51	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	52-80	RESERVED 29 bytes	reserved for future use.

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 Basic Operating System programs as well as the Operating System programs.

APPENDIX B. STANDARD DASD FILE LABELS, FORMAT 1



Format 1: This format is common to all data files on DASD.

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
1	1-44	FILE NAME 44 bytes, alphameric EBCDIC	this field serves as the key portion of the file label. Each file must have a unique file name. Duplication of file names will cause retrieval errors. The file name can consist of three sections:

1. File ID is an alphameric name assigned by the user and identifies the file. Can be 1-35 bytes if generation and version numbers are used, or 1-44 bytes if they are not used.

2. Generation Number. If used, this field is separated from File ID by a period. It has the format Gnnnn, where G identifies the field as the generation number and nnnn (in decimal) identifies the generation of the file.

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
			3. Version Number of Generation. If used, this section immediately follows the generation number and has the format Vnn, where V identifies the file as the version of generation number and nn (in decimal) identifies the version of generation of the file.

Note: Basic Operating System compares the entire file name field against the file name given in the DLAB card. The generation and version numbers are treated differently by the Operating System.

The following fields (2-33) comprise the DATA portion of the file label:

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
2	45	FORMAT IDENTIFIER 1 byte, EBCDIC numeric	1 = Format 1
3	46-51	FILE SERIAL NUMBER 6 bytes, alphameric EBCDIC	uniquely identifies a file/volume relationship. It is identical to the Volume Serial Number of the first or only volume of a multi-volume file.
4	52-53	VOLUME SEQUENCE NUMBER 2 bytes, binary	indicates the order of a volume relative to the first volume on which the data file resides.
5	54-56	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	57-59	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	60	EXTENT COUNT 1 byte, binary	contains a count of the number of extents for this file on this volume. If user labels are used, the count does not include the user label extent. This field is maintained by the Basic Operating System programs.
7B*	61	BYTES USED IN LAST BLOCK OF DIRECTORY 1 byte, binary	used by the Operating System only for partitioned (library structure) data sets. Not used by the Basic Operating System.
7C	62	SPARE 1 byte	reserved for future use.
8	63-75	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.

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
9*	76-82	RESERVED 7 bytes	this field is reserved for future use.
10*	83-84	FILE TYPE 2 bytes	the contents of this field uniquely identify the type of data file: Hex 4000 = Consecutive organization Hex 2000 = Direct-access organization Hex 8000 = Indexed-sequential organization Hex 0200 = Library organization Hex 0000 = Organization not defined in the file label.
11	85	RECORD FORMAT 1 byte	the contents of this field indicate the type of records contained in the file:

Bit Position	Content	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 only)
3	0	Unblocked records
	1	Blocked records
4	0	No truncated records
	1	Truncated records ir file
5 and 6	01	Control character ASA code
	10	Control character machine code
	00	Control character not stated

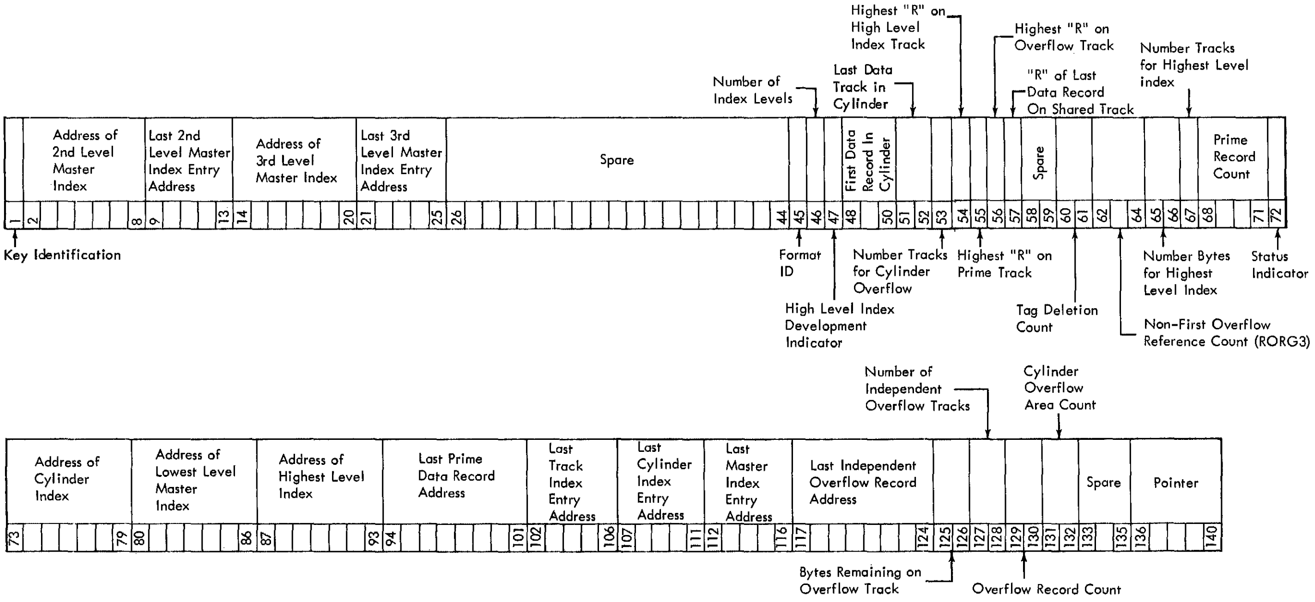
FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
			7 0 Records have no keys 1 Records are written with keys.
12*	86	OPTION CODES 1 byte	Bits within this field are used to indicate various options used in building the file. Bit 0-2 unused 3 If on, indicates independent overflow option. 4 If on, indicates cylinder overflow option. 5-7 unused
13**	87-88	BLOCK LENGTH 2 bytes, binary	indicates the block length for fixed length records or maximum block size for variable length blocks.
14**	89-90	RECORD LENGTH 2 bytes, binary	indicates the record length for fixed length records or the maximum record length for variable length records.
15**	91	KEY LENGTH 1 byte, binary	indicates the length of the key portion of the data records in the file.
16**	92-93	KEY LOCATION 2 bytes, binary	indicates the high order position of the data record.
17	94	DATA SET INDICATORS 1 byte	Bits within this field are used to indicate the following: Bit 0 If on, indicates that this is the last volume on which this file normally resides. This bit is used by the Basic Operating System DTFSR routine only. None of the other bits in this byte are used by Basic Operating System. 1 If on, indicates that the data set described by this file must remain in the same absolute location on the direct access device. 2 If on, indicates that Block Length must always be a multiple of 8 bytes.

** BOS supports fields 13-16 for ISFMS only.

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
			3 If on, indicates that this data file is security protected; a password must be provided in order to access it. 4-7 Spare. Reserved for future use.
18*	95-98	SECONDARY ALLOCATION 4 bytes, binary	indicates the amount of storage to be requested for this data file at end of extent. This field is used by Operating System only. It is not used by Basic Operating System routines. The first byte of this field is an indication of the type of allocation request. Hex code "C2" (EBCDIC "B") indicates blocks (physical records), hex code "E3" (EBCDIC "T") indicates tracks, and hex code "C3" (EBCDIC "C") indicates cylinders. The next three bytes of this field is a binary number indicating how many bytes, tracks or cylinders are requested.
19*	99-103	LAST RECORD POINTER 5 bytes, discontinuous binary	points to the last record written in a sequential or partition-organization data set. The format is TTRLL, where TT is the relative address of the track containing the last record, R is the ID of the last record, and LL is the number of bytes remaining on the track following the last record. If the entire field contains binary zeros, the last record pointer does not apply.
20	104-105	SPARE 2 bytes	reserved for future use.
21	106	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. 40 user label track area 8n shared cylinder indicator, where n = 1,2, or 4.

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
22	107	EXTENT SEQUENCE NUMBER 1 byte, binary	indicates the extent sequence in a multi-extent file.
23	108-111	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	112-115	UPPER LIMIT 4 bytes	the cylinder and the track address specifying the ending point (upper limit) of this extent component. This field has the format CCHH.
25-28	116-125	ADDITIONAL EXTENT 10 bytes	these fields have the same format as the fields 21-24 above.
29-32	126-135	ADDITIONAL EXTENT 10 bytes	these fields have the same format as fields 21-24 above.
33	136-140	POINTER TO NEXT FILE LABEL WITHIN THIS LABEL SET 5 bytes, discontinuous binary	the address (format CCHHR) of a continuation label if needed to further describe the file. If field 10 indicates Indexed Sequential organization, this field will point to a Format 2 file label within the label set. Otherwise, it points to a Format 3 file label, and then only if the file contains more than three extent segments. This field contains all binary zeros if no additional file label is pointed to.

* These fields are not supported by the Basic Operating System.


Address of 2nd Level Master Index
Last 2nd Level Master Index Entry Address
Address of 3rd Level Master Index
Last 3rd Level Master Index Entry Address
Spare
Number of Index Levels
Last Data Track in Cylinder
Highest "R" on High Level Index Track
First Data Record In Cylinder
Highest "R" on Overflow Track
"R" of Last Data Record On Shared Track
Spare
Number Tracks for Highest Level index
Prime Record Count
Key Identification
Format ID
High Level Index Development Indicator
Number Tracks for Cylinder Overflow
Highest "R" on Prime Track
Tag Deletion Count
Number Bytes for Highest Level Index
Status Indicator
Non-First Overflow Reference Count (RORG3)
Address of Cylinder Index
Address of Lowest Level Master Index
Address of Highest Level Index
Last Prime Data Record Address
Last Track Index Entry Address
Last Cylinder Index Entry Address
Last Master Index Entry Address
Last Independent Overflow Record Address
Number of Independent Overflow Tracks
Cylinder Overflow Area Count
Spare
Pointer
Bytes Remaining on Overflow Track
Overflow Record Count

Format 2: This format is applicable only to Indexed Sequential data files. It is always pointed to by a Format 1 label.

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
K1	1	KEY IDENTIFICATION 1 byte	this byte contains the Hex Code 02 in order to avoid conflict with a file name.
K2*	2-8	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*	9-13	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*	14-20	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.

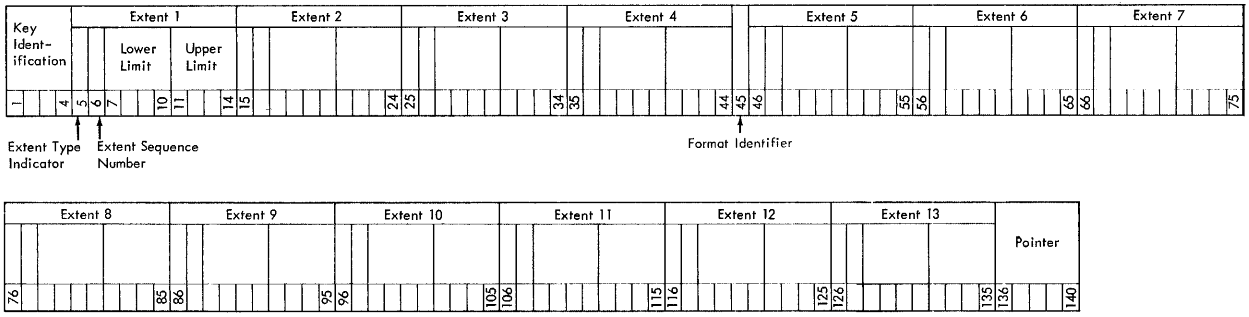
FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
K5*	21-25	LAST 3RD LEVEL MASTER INDEX ENTRY ADDRESS 5 bytes, discontinuous binary	this field contains the address of the last entry in the third level of the master index, in the form CCHHR.
K6*	26-44	SPARE 19 bytes	reserved for future use.
D1	45	FORMAT IDENTIFIER 1 byte, EBCDIC numeric	2 = Format 2
D2	46	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*	47	HIGH LEVEL INDEX DEVELOPMENT INDICATOR 1 byte, binary	this field contains the number of tracks determining development of Master Index.
D4	48-50	FIRST DATA RECORD IN CYLINDER 3 bytes	this field contains the address of the first data record on each cylinder in the form HHR.
D5	51-52	LAST DATA TRACK IN CYLINDER 2 bytes	this field contains the address of the last data track on each cylinder, in the form HH.
D6	53	NUMBER OF TRACKS FOR CYLINDER OVERFLOW 1 byte, binary	this field contains the number of tracks in cylinder overflow area.
D7	54	HIGHEST "R" ON HIGH LEVEL INDEX TRACK 1 byte	this field contains the highest possible R on a track containing high-level index entries.
D8	55	HIGHEST "R" ON PRIME TRACK 1 byte	this field contains the highest possible R on prime data tracks for form F records.
D9	56	HIGHEST "R" ON OVERFLOW TRACK 1 byte	this field contains the highest possible R on overflow data tracks for form F records.
D10	57	"R" OF LAST DATA RECORD ON SHARED TRACK 1 byte	this field contains the R of the last data record on a shared track.
D11*	58-59	SPARE 2 bytes	Reserved for future use.
D12	60-61	TAG DELETION COUNT 2 bytes, binary	this field contains the number of records that have been tagged for deletion.
D13	62-64	NON-FIRST OVERFLOW REFERENCE COUNT (RORG3) 3 bytes, binary	this field contains a count of the number of random references to a non-first overflow record.

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
D14	65-66	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*	67	NUMBER OF TRACKS FOR HIGHEST-LEVEL INDEX 1 byte, binary	this field contains a count of the number of tracks occupied by the highest-level index.
D16	68-71	PRIME RECORD COUNT 4 bytes, binary	this field contains a count of the number of records in the prime data area.
D17	72	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	73-79	ADDRESS OF CYLINDER INDEX 7 bytes	this field contains the address of the first track of the cylinder index, in the form MBBCCHH.
D19	80-86	ADDRESS OF LOWEST-LEVEL MASTER INDEX 7 bytes	this field contains the address of the first track of the lowest-level index of the high level indexes, in the form MBBCCHH.
D20	87-93	ADDRESS OF HIGHEST-LEVEL INDEX 7 bytes	this field contains the address of the first track of the highest-level master index, in the form MBBCCHH.
D21	94-101	LAST PRIME DATA RECORD ADDRESS 8 bytes	this field contains the address of the last data record in the prime data area, in the form MBBCCHHR.
D22	102-106	LAST TRACK INDEX ENTRY ADDRESS 5 bytes	this field contains the address of the last normal entry in the track index on the last cylinder in the form CCHHR.
D23	107-111	LAST CYLINDER INDEX ENTRY ADDRESS 5 bytes	this field contains the address of the last index entry in the cylinder index in the form CCHHR.
D24	112-116	LAST MASTER INDEX ENTRY ADDRESS 5 bytes	this field contains the address of the last index entry in the master index in the form CCHHR.
D25	117-124	LAST INDEPENDENT OVERFLOW RECORD ADDRESS 8 bytes	this field contains the address of the last record written in the current independent overflow area, in the form MBBCCHHR.
D26*	125-126	BYTES REMAINING ON OVERFLOW TRACK 2 bytes, binary	this field contains the number of bytes remaining on current independent overflow track.

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
D27	127-128	NUMBER OF INDEPENDENT OVERFLOW TRACKS (RORG2) 2 bytes, binary	this field contains the number of tracks remaining in independent overflow area.
D28	129-130	OVERFLOW RECORD COUNT 2 bytes, binary	this field contains a count of the number of records in the overflow area.
D29	131-132	CYLINDER OVERFLOW AREA COUNT (RORG1) 2 bytes, binary	this field contains the number of full cylinder overflow areas.
D30	133-135	SPARE 3 bytes	this field is reserved for future use.
D31*	13-140	POINTER TO FORMAT 3 FILE LABEL 5 bytes	this field contains the address (in the form CCHHR) of a Format 3 file label if more than 3 extent segments exist for the data file within this volume. Otherwise it contains binary zeros.

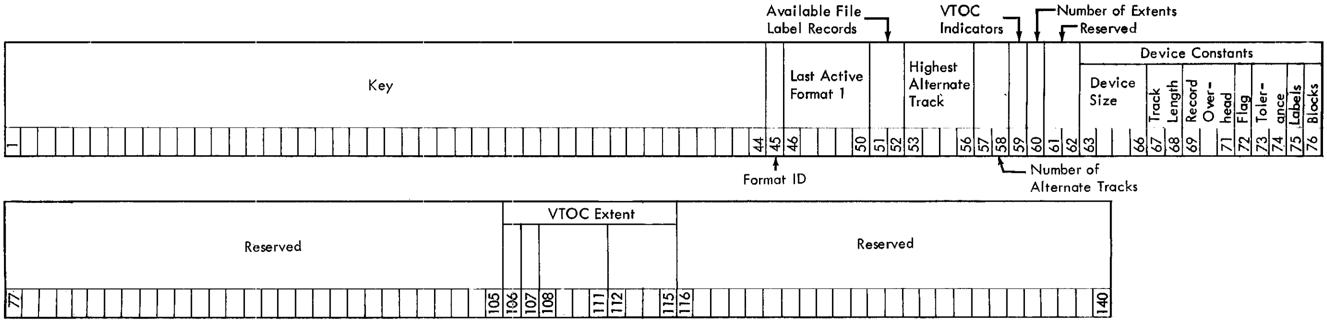
* These fields are not supported by the Basic Operating System.

APPENDIX D. STANDARD DASD FILE LABELS, FORMAT 3



Format 3: This format 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 pointed to by a Format 1, Format 2, or another Format 3 file label.

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
1	1-4	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	5-44	EXTENTS (in KEY) 40 bytes	four groups of fields identical in format to fields 21-24 in the Format 1 label.
18	45	FORMAT IDENTIFIER 1 byte, EBCDIC numeric	3 = Format 3
19-54	46-135	ADDITIONAL EXTENTS 90 bytes	Nine groups of fields identical in format to fields 21-24 in the Format 1 label.
55	136-140	POINTER TO NEXT FILE LABEL 5 bytes	this field contains the address (in the form CCHHR) of another Format 3 label if additional extents must be described. Otherwise, it is all binary zeros.



Format 4: This format 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	1-44	KEY FIELD 44 bytes, binary	each byte of this field contains the Hex Code 04 in order to provide a unique key.
2	45	FORMAT ID 1 byte, EBCDIC numeric	4 = Format 4
3*	46-50	LAST ACTIVE FORMAT 1 5 bytes	contains the address (in the form CCHHR) of the last active Format 1 file label. It is used to stop a search on a file name.
4	51-52	AVAILABLE FILE LABEL RECORDS 2 bytes, binary	contains a count of the number of unused records in the VTOC.
5	53-56	HIGHEST ALTERNATE TRACK 4 bytes	contains the highest address (in the form CCHH) of a block of tracks set aside as alternates for bad tracks.
6	57-58	NUMBER OF ALTERNATE TRACKS 2 bytes, binary	contains the number of alternate tracks available.
7	59	VTOC INDICATORS 1 byte	Bit 0 if on, indicates no DADSM (format 5) label, or DADSM label does not reflect true status of volume. Bits 1-7 not used.
8A	60	NUMBER OF EXTENTS 1 byte	contains the hexadecimal constant 01, to indicate one extent in the VTOC.

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
8B	61-62	RESERVED 2 bytes	reserved for future use.
9	63-76	DEVICE CONSTANTS 14 bytes	this field contains constants describing the device on which the volume was mounted when the VTOC was created. The following describes each of the subfields.

Device Size (4 bytes)--The number of cylinders (CC) and tracks per cylinder (HH).

Track Length (2 bytes)--The number of available bytes on a track exclusive of home address and record zero (record zero is assumed to be a non-keyed record with an eight byte data field).

Record Overhead (3 bytes)--The number of bytes required for gaps, check bits, and count field for each record. This value varies according to the record characteristics and thus is broken down into three subfields.

I--Overhead required for keyed record other than the last record on the track.
L--Overhead required for a keyed record that is the last record on the track.
K--Overhead bytes to be subtracted from I or L if the record does not have a key field.

Flag (1 bytes)--Further defines unique characteristics of the device.

Bits	Meaning
0-5	reserved
6	CC and HH must be used as 1-byte values, as in the case of the 2321.
7	A tolerance factor must be applied to all but the last record on the track.

Tolerance (2 bytes)--A value that is to be used to determine the effective length of the record on the track. The effective length of a record is calculated in the following manner:

1. Add the key length to the data length of the record.
2. Test bit 7 in the flag byte:
   a. if 0 go to 3
   b. multiply value from 1 by the tolerance factor
   c. shift result 9 bits to the right
3. Add overhead bytes to the result.

NOTE: Step 2 is not required if the calculation is for the last record on the track.

Labels/Track (1 byte)--A count of the number of labels that can be written on each track in the VTOC. (Number of full records of 44-byte key and 96-byte data lengths that can be contained on one track of this device.)

Directory Blocks/Track (1 byte)--A count of the number of directory blocks that can be written on each track for an Operating System partitioned data set (number of full records of 8-byte key and 256-byte data lengths that can be contained on one track of this device).

The following illustrates the device constants field for the various direct access devices:

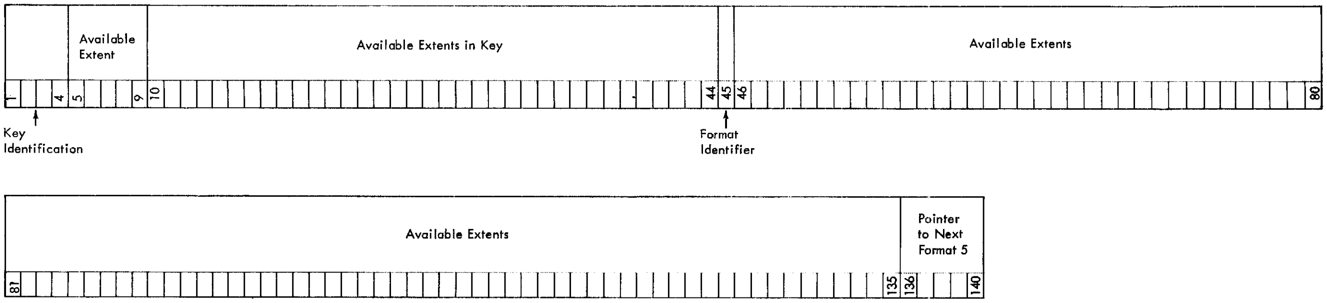
Device	CC	HH	Track Length	I	L	K	Flag	Tolerance	Labels/ Track	Directory Blocks/ Track
2311	203	10	3656	82	55	20	1	537	16	10
2321	20 10	5 20	2027	101	47	16	3	537	8	5
2301	0	200	20616	186	186	53	0	512	63	45
2302	250	46	5070	82	55	20	1	537	22	14
7320	0	400	2129	111	43	14	1	537	8	5

Note: CCHH for the 2321 above are separate 1 byte quantities.

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
10	77-105	RESERVED 29 bytes	reserved for future use.
11-14	106-115	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).
15	116-140	RESERVED 25 bytes	reserved for future use.

* This field not supported by the Basic Operating System.

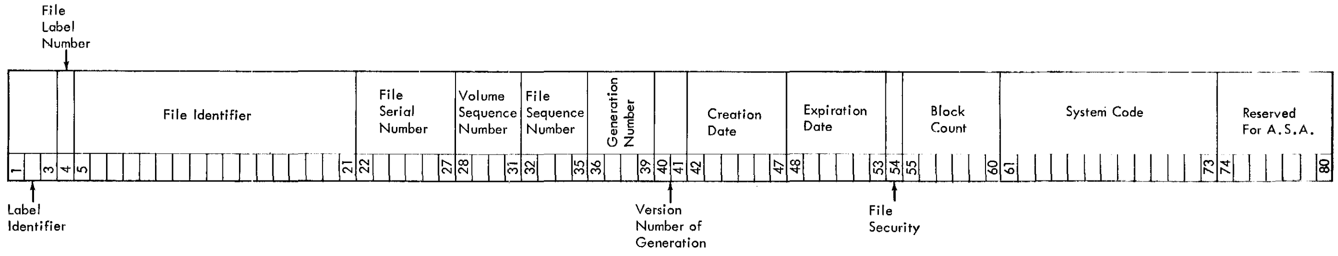
APPENDIX F. STANDARD DASD FILE LABLES, FORMAT 5



Format 5: This format is used for Direct Access Device Space Management (DADSM) only.*

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
1	1-4	KEY IDENTIFICATION 4 bytes	each of these four bytes is a hex 05.
2	5-9	AVAILABLE EXTENT 5 bytes	this field indicates an extent of space available for allocation to a data file. The first two bytes are relative track address. The next two are the number of full cylinders included in the extent. The last byte is the number of tracks in addition to the cylinders in the extent.
3-9	10-44	AVAILABLE EXTENTS IN KEY 35 bytes	these fields are identical to field 2. They are in relative track address sequence.
10	45	FORMAT IDENTIFIER 1 byte EBCDIC numeric	5 = Format 5
11-28	46-135	AVAILABLE EXTENTS 90 bytes	these fields are the same as Field 2. There are 26 available extent fields in the Format 5 label.
29	136-140	POINTER TO NEXT FORMAT 5 5 bytes	contains the address (in the form CCHHR) of the next Format 5 file label if one exists.

* Not supported by the Basic Operating System.



The standard tape file label format and contents are as follows:

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
1	1-3	LABEL IDENTIFIER 3 bytes, EBCDIC	identifies the type of label HDR = Header--beginning of a data file EOF = End of File--end of a set of data EOV = End of Volume--end of the physical reel .
2	4	FILE LABEL NUMBER 1 byte, EBCDIC	always a 1
3	5-21	FILE IDENTIFIER 17 bytes, EBCDIC	uniquely identifies the entire file; may contain only printable characters.
4	22-27	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	28-31	VOLUME SEQUENCE NUMBER 4 bytes	indicates the order of a volume in a given file or multi-file set. This first must be numbered 0001 and subsequent numbers must be in proper numeric sequence.
6	32-35	FILE SEQUENCE NUMBER 4 bytes	assigns numeric sequence to a file within a multi-file set. The first must be numbered 0001.

FIELD	BYTES	NAME AND LENGTH	DESCRIPTION
7	36-39	GENERATION NUMBER 4 bytes	uniquely identifies the various editions of the file. May be from 0001 to 9999 in proper numeric sequence.
8	40-41	VERSION NUMBER OF GENERATION 2 bytes	indicates the version of a generation of a file.
9	42-47	CREATION DATE 6 bytes	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, 1965 would be entered as 65031).
10	48-53	EXPIRATION DATE 6 bytes	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 Field 9. On a multifile reel, processed sequentially, all files are considered to expire on the same day.
11	54	FILE SECURITY 1 byte	indicates security status of the file. 0 = no security protection 1 = security protection. Additional identification of the file is required before it can be processed.
12	55-60	BLOCK COUNT 6 bytes	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	61-73	SYSTEM CODE 13 bytes	uniquely identifies the programming system.
14	74-80	RESERVED 7 bytes	reserved for American standards Association (A.S.A.). At present, should be recorded as blanks.

The core sizes and timings given are estimated and should be used for planning only.

SUPERVISOR CORE SIZES

The size of the Supervisor depends upon the parameters specified in the following Supervisor-assembly macros:

SUPVR (Supervisor)
SYMUN (Symbolic Units)
IOCFG (I/O Configuration)
SEND (Supervisor End).

These macro instructions are described fully in the Assembler with Input/Output Macros publication, listed in the Preface of this manual. The following chart gives the core requirements for the routines generated by the various operands of the supervisor macros. The IBM-supplied processing programs (Assembler, Sort, etc.) require the availability of a problem program area of 4096 bytes. Thus, in an 8K (8192 bytes) system, the Supervisor assembled to handle these programs must not exceed 4096 bytes, including any user patch area.

The computed value and the actual value of the Supervisor size may vary up to 30 bytes becuase of boundary alignment conditions. This variance is handled in the boundary buffer. A comma (,) between operands indicates (AND), meaning both operands must be present, and a slash (/) indicates (OR), meaning any one or all may be present.

BOS SUPERVISOR: Core Size Chart

MACRO	MAJOR OPERAND	SUB-OPERANDS	SUB BYTES	MAJOR BTYES
----	Common Area			1954
SUPVR	DISK=YES			1568
		BACKWRD=YES		12
		CONFG≥64K(0110nnnn->1010nnnn)	8	
		SAVEREG=YES	4	
SUPVR	CONFG≥64K(0110nnnn->1010nnnn)			56
SUPVR	CONFG=Not Model 30 (nnnn1nnn)			244
SUPVR	CONFG=1052 (nnnnnnn1)			172
		CONFG≥64K(0110nnnn->1010nnnn)	8	
SUPVR	CR=Yes			164
		CONFG≥64K(0110nnnn->1010nnnn)	4	
SUPVR	TR=Yes			128
		CONFG≥64K(0110nnnn->1010nnnn)	4	
		CONFG=1052 (nnnnnnn1)	8	
		CR=No	4	
		SAVEREG=YES	16	

MACRO	MAJOR OPERAND	SUB-OPERANDS	SUB BYTES	MAJOR BYTES
SUPVR	TR=NO,CR=NO			4
SUPVR	CHKPT=YES			110
SUPVR	SAVEREG=YES			80
SYMUN	Program Pub Units			4n
SYMUN	Pub Units>57			8
IOCFG	BSC=YES			4122
IOCFG	MPX>1			84+4(n-1)
IOCFG	MPX>1/SEL>0			24
IOCFG	MPX>1, SEL>0			36
IOCFG	MPX<2, SEL=0			2
IOCFG	MPX<2/(MPX>1,SEL>0)			6
IOCFG	MPX<2/SEL=0			4
IOCFG	SEL>0			6+4n
IOCFG	DVE>0			194+6n
		CONFG≥64K(0110nnnn->1010nnnn)	8	
		MPX>1, SEL>0	2	
IOCFG	R1=YES/P3=YES			10
IOCFG	R2=YES/P1=YES/P3=YES			16
IOCFG	R3=YES/R4=YES/R5=YES/P2=YES			6
IOCFG	L1=YES			18
		R1=NO	6	
IOCFG	L2=YES			20
IOCFG	T=YES			420
		TAU=YES	94	
		BACKWRD=YES	12	
		TRK7=YES	8	
IOCFG	R0=YES			36
IOCFG	RR=YES			134
IOCFG	ST=YES			820
		CONFG<64K(0000nnnn->0100nnnn)	4	
		ANSWR=n	316+ 4(n-1)	
SEND	REP (specified)			186

CORE SIZES AND TIMINGS FOR ASSEMBLED PROGRAMS

The core requirements for object programs resulting from an assembly depend on:

1. The types of file organization and processing used for the data files;

2. The types of I/O devices used; and

3. The imperative macro instructions included in the program.

The following charts give the core sizes of routines in the Direct Access Method, in the Indexed Sequential File Management System, and for Consecutive Processing. For Consecutive Processing, a separate chart is included for each type of I/O device. Estimated timings are given for records processed in consecutive order.

DIRECT ACCESS METHOD (DAM) CORE SIZES

The size of a DAM routine depends upon the parameters specified in the DTFDA macro instruction. The following list gives the size of the routines generated for the Direct Access Method.

	RECFORM=	
	FIXUNB	UNDEF
Basic Logic	122	122
CONTROL=YES	62	62
VERIFY=YES	24-48	24-48
AFTER=YES	266	278
WRITEID=YES	58	62

WRITEID=YES
IDLOC=Name
	114	118

WRITEKY=YES	58	62

WRITEKY=YES
IDLOC=Name
	114	118

WRITEKY=YES
SRCHM=YES
	66	70

READID=YES	58	110

READID=YES
IDLOC=Name
	114	166

READKEY=YES	58	90

READKEY=YES
IDLOC=Name
	114	146

READKEY=YES
SRCHM=YES
	66	98

INDEXED SEQUENTIAL FILE MANAGEMENT SYSTEM (ISFMS) CORE SIZES

Four basic logical IOCS routines are available from the DTFIS macro. The routine generated depends on the I/O routine (IOROUT) parameter specified. The four routines are:

IOROUT=LOAD	Routine to build or extend a file.
IOROUT=ADD	Routine to add new records to a file.
IOROUT=RETRVE	Routine to retrieve records for processing and, optionally, to write back the updated records.
IOROUT=ADDRTR	Routine to add new records and to retrieve records for processing and updating.

The following chart gives the core requirements for the various routines generated by ISFMS.

	IOROUT= LOAD	IOROUT= ADD	IOROUT=ADDRTR Plus TYPEFLE=			IOROUT=RETRVE Plus TYPEFLE=		
			RANDOM	SEQNTL	RANSEQ	RANDOM	SEQNTL	RANSEQ
RECFORM=FIXUNB								
I/O Reg., Basic	----	----	2140	2370	2530	586	576	936
I/O Reg.,Basic,Update	----	----	2190	2420	2630	660	648	1118
Workarea,Basic	980	1750	2170	2396	2546	590	590	910
Workarea,Basic,Update	980	1750	2220	2460	2660	676	682	1108
RECFORM=FIXBLK								
I/O Reg.,Basic	----	----	2350	2640	2820	598	656	1066
I/O Reg.,Basic,Update	----	----	2400	2698	2928	672	738	1258
Workarea,Basic	1070	1900	2370	2694	2840	600	660	1040
Workarea,Basic,Update	----	----	2426	2766	2968	696	762	1258
MSTIND=YES	124	36	30	----	30	---	---	---
VERIFY=YES	36	64	72	72	80	---	---	---
CYLOFL=n	----	96	96	96	96	---	---	---

Notes:

1. Figures for LOAD, ADD, and ADDRTR are per DTF.
2. Key length (KEYLEN=n) and record size (RECSIZE=n) must be added to the ADD and ADDRTR figures. Key length must be added to the LOAD figures.
3. Figures for RETRVE are per program, for all files that have the same specifications. For example, the specifications of RANDOM, FIXUNB, I/O Reg., Basic require 586 bytes of main storage, regardless of the number of files having that set of specifications. If different RETRVE files have different sets of specifications, the main-storage requirements must be combined. The maximum main-storage requirement for all specifications is 1370 bytes per program.
4. A 144-byte table per DTF must be added to the RETRVE figures.

CONSECUTIVE PROCESSING CORE SIZES

The logical IOCS routines generated from the DTFSR macro instructions vary greatly in size depending upon the type of device and the record format. The following charts reflect the size of the DTF routines generated depending on the DEVICE parameter specified. Parameters not listed either do not cause additional code to be generated or they make minor changes that are accounted for in the size given for the basic routine.

DEVICE=PRINTER	Basic	CONTROL=YES	CTLCHR=YES	PRINTOV=YES
FIXUNB, 1 I/O	52	68	16	74
1 I/O,WORKA	84	68	8	66
2 I/O	84	68	16	66
2 I/O,WORKA	100	68	8	66
VARUNB, 1 I/O	84	68	8	74
1 I/O,WORKA	100	60	0	74
2 I/O	116	76	0	58
2 I/O,WORKA	116	68	8	66
UNDEF, 1 I/O	60	60	16	66
1 I/O,WORKA	84	68	8	66
2 I/O	92	68	16	66
2 I/O,WORKA	108	60	8	58

DEVICE=DISK11	TYPEFLE=		CONTROL=YES	ERROPT=			UPDATE=YES	VERIFY=YES	WLRERR=Name	If RECSIZE greater than 256	TRUNCS=YES	
	INPUT	OUTPUT		IGNORE	SKIP	Name					Input	Output
FIXUNB, 1 I/O	284	300	44	-20	16	64	48	24	28	$6 \times \frac{\text{RECSIZE}}{256}$	--	--
1 I/O,WORKA	316	320	44	-20	16	64	52	24	28		--	--
2 I/O	320	336	44	-20	16	64	100	24	28		--	--
2 I/O,WORKA	332	348	44	-20	16	64	124	24	28		--	--
FIXBLK, 1 I/O	404	348	44	-12	20	68	76	24	32		16	84
1 I/O,WORKA	464	408	44	-12	36	84	56	24	32		60	120
2 I/O	436	432	44	-12	36	84	104	24	32		16	76
2 I/O,WORKA	448	440	44	-12	48	96	132	24	32		32	112
VARUNB, 1 I/O	348	348	44	44	80	96	52	24	48	56	--	--
1 I/O,WORKA	392	416	44	52	72	104	60	24	48	56	--	--
2 I/O	380	424	44	52	80	104	100	24	48	56	--	--
2 I/O,WORKA	412	444	44	52	72	104	124	24	48	56	--	--
VARBLK, 1 I/O	456	444	44	52	72	116	56	24	68	56	--	--
1 I/O,WORKA	572	548	44	52	72	108	40	24	52	56	--	--
2 I/O	496	504	44	52	72	108	96	24	30	56	--	--
2 I/O,WORKA	536	584	44	52	72	108	136	24	48	56	--	--
UNDEF, 1 I/O	344	340	44	56	80	112	52	24	48	56	--	--
1 I/O,WORKA	392	404	44	56	80	112	60	24	48	56	--	--
2 I/O	376	416	44	56	80	116	104	24	48	56	--	--
2 I/O,WORKA	416	436	44	52	80	116	124	24	48	56	--	--

DEVICE=READ42 (1442)	TYPEFLE=		CMBND	CONTROL=YES		CTLCHR=YES
	INPUT	OUTPUT		Input	Output	
FIXUNB, 1 I/O	76	84	116	52	60	16
1 I/O,WORKA	108	108	164	28	68	8
2 I/O	116	116	156	44	68	16
2 I/O,WORKA	124	132	---	28	60	8
VARUNB, 1 I/O	---	108	---	--	68	8
1 I/O,WORKA	---	124	---	--	68	8
2 I/O	---	140	---	--	68	8
2 I/O,WORKA	---	148	---	--	60	8
UNDEF, 1 I/O	---	84	---	--	68	16
1 I/O,WORKA	---	116	---	--	68	8
2 I/O	---	116	---	--	68	16
2 I/O,WORKA	---	132	---	--	68	8

DEVICE=TAPE	TYPEFLE=		CHECKPT=n	CKPTREC=YES	CONTROL =YES		ERROPT=			READ=BACK	WLRERR=Name	FILABL=STD
	INPUT	OUTPUT			Input	Output	IGNORE	SKIP	Name			
FIXUNB, 1 I/O	160	168	56	56	88	44	-16	8	64	0	24	88
1 I/O,WORKA	200	224	72	48	88	44	-24	8	72	8	32	88
2 I/O	200	232	80	56	88	44	-16	16	72	16	32	88
2 I/O,WORKA	216	248	72	48	88	44	-24	8	72	8	32	88
FIXBLK, 1 I/O	272	316	56	48	88	48	-40	0	64	0	24	88
1 I/O,WORKA	352	418	56	56	88	50	-40	16	80	0	32	88
2 I/O	312	444	72	56	88	0	-40	8	64	8	24	88
2 I/O,WORKA	324	416	80	56	88	44	-32	16	72	8	32	88
VARUNB, 1 I/O	184	192	56	48	88	44	-40	0	64	40	24	88
1 I/O,WORKA	216	246	72	48	88	44	-40	8	72	32	32	88
2 I/O	224	264	80	56	88	44	-32	16	80	40	32	88
2 I/O,WORKA	232	272	72	48	88	44	-40	8	72	40	32	88
VARBLK, 1 I/O	280	296	56	48	88	44	-60	0	64	0	24	88
1 I/O,WORKA	376	400	48	56	88	44	-32	16	80	0	32	88
2 I/O	296	344	72	56	88	44	-8	-16	0	0	56	88
2 I/O,WORKA	320	432	72	56	88	44	0	0	8	0	56	88
UNDEF, 1 I/O	152	168	56	48	88	44	0	8	40	16	--	88
1 I/O,WORKA	184	232	80	56	88	44	0	16	56	16	--	88
2 I/O	192	232	72	48	88	44	-48	16	48	8	--	88
2 I/O,WORKA	200	256	80	56	88	44	0	16	56	16	--	88

DEVICE=READ40 (2540)	TYPEFLE=			CONTROL=YES		CRDERR=RETRY	CTLCHR=YES
	INPUT	OUTPUT	CMBND	Input	Output		
FIXUNB, 1 I/O	76	52	116	52	40	64+BL	16
1 I/O,WORKA	108	84	164	28	40	64+BL	8
2 I/O	116	84	156	44	40	72+BL	16
2 I/O,WORKA	124	100	----	28	40	56	8
VARUNB, 1 I/O	---	84	----	--	40	80+BL	8
1 I/O,WORKA	---	100	----	--	32	80+BL	0
2 I/O	---	116	----	--	32	88+BL	0
2 I/O,WORKA	---	116	----	--	40	80	8
UNDEF, 1 I/O	---	60	----	--	32	72+BL	16
1 I/O,WORKA	---	84	----	--	40	80+BL	8
2 I/O	---	92	----	--	40	88+BL	16
2 I/O,WORKA	---	108	----	--	32	80	8
Note: BL denotes Block Length							

DEVICE=CONSOLE (1052)	TYPEFLE=	
	INPUT	OUTPUT
FIXUNB, 1 I/O	64	56
1 I/O,WORKA	80	72
UNDEF, 1 I/O	72	60
1 I/O,WORKA	96	76

DEVICE=READ20 (2520)		TYPEFLE=			CONTROL=YES			CTLCHR=YES	CRDERR=RETRY
		INPUT	OUTPUT	CMBND	Input	Output	Cmbnd		
FIXUNB	1 I/O	84	90	130	50	40	30	10	20
	1 I/O,WORKA	116	120	175	50	40	—	10	24
	2 I/O	124	120	—	—	40	—	10	24
	2 I/O,WORKA	140	135	—	—	40	—	10	24
UNDEF	1 I/O	—	90	—	—	40	—	10	20
	1 I/O,WORKA	—	120	—	—	40	—	10	24
	2 I/O	—	125	—	—	40	—	10	24
	2 I/O,WORKA	—	140	—	—	40	—	10	24
VARUNB	1 I/O	—	110	—	—	40	—	10	20
	1 I/O,WORKA	—	135	—	—	40	—	10	24
	2 I/O	—	150	—	—	40	—	10	24
	2 I/O,WORKA	—	150	—	—	40	—	10	24

DEVICE=PTAPERD (2671)	Basic	TRANS=Name	If RECSIZE is greater than 256
FIXUNB, 1 I/O	180	18	--
1 I/O,WORKA	200	18	10 to 54
2 I/O	200	18	--
2 I/O,WORKA	220	18	10 to 54
UNDEF 1 I/O	190	18	--
1 I/O,WORKA	210	18	54
2 I/O	210	18	--
2 I/O,WORKA	225	18	54

DEVICE=READ85 (1285)	TYPEFLE=INPUT	HEADER=YES	CONTROL=YES
FIXUNB 1 I/O	364	46	120
1 I/O,WORKA	422	46	120
2 I/O	410	46	120
2 I/O,WORKA	440	46	120
UNDEF 1 I/O	360	46	120
1 I/O,WORKA	410	46	120
2 I/O	420	46	120
2 I/O,WORKA	440	46	120

CORE SIZES FOR IMPERATIVE MACRO INSTRUCTIONS

The range of core sizes for imperative macro instructions, such as GET, PUT, RELSE, etc, is 10 to 24 bytes. The core size for OPEN and CLOSE is 24 bytes plus 4 times the number of files to be opened or closed.

TIMINGS (ESTIMATED)

The estimated timings given here are for consecutive files processed by the DTFSR routine of logical IOCS. The figures refer to a System/360 Model 30 with a 1.5 microsecond main storage cycle time. Except for the OPEN macro, they represent only the actual process time required and do not include input/output times. (Input/output times are a function of the I/O device itself.) Distinction is made only between disk and all other types of files (tape, card, printer, etc).

The times for both GET and PUT are the same:

	Disk	Other
1. Each record of unblocked file or first record of block in blocked file.	2.9ms	2.0ms
2. Additional for successive records after first record in blocked file:		
a. Fixed length	0.4ms	0.4ms
b. Variable length	0.4ms	0.4ms
3. Additional if separate work area (plus the record move time)		
a. Fixed length	0.2ms	0.2ms
b. Variable length	0.3ms	0.3ms

When two I/O areas are used, the overlapped I/O transfer time can be subtracted.

The times for the OPEN macro are:

1. No labels (any device except disk)	650ms
2. Tape Input, labeled	850ms
3. Tape Output, labeled	850ms
4. Disk input	
(basic, each extent)	2500ms
(additional, each label searched)	2ms
5. Disk output	
(basic, each extent)	2500ms
(additional, each label in VTOC)	25ms
(additional, each label deleted)	50ms

REPORT PROGRAM GENERATOR CORE SIZES

This section provides the core sizes for the various phases of the IBM System/360 Basic Operating System RPG object program. It consists of main storage requirements for input/output devices and areas, operation code expansions, and field formats. These main storage sizes are presented in a format corresponding to the types of entries specified by the user in the RPG Specification sheets. In most cases, there is a definite main storage requirement for each line of coding on these sheets.

The total main storage estimate calculated from the information in this section will be within 10% of the actual main storage requirement; that is, the total estimate will be the actual number of bytes ± 10%.

Note: This estimate does not include the amount of main storage required by the user's supervisor.

FILE DESCRIPTION

This section is composed of three subsections, each designated by a particular file type, with consideration given to certain other options the user has in defining his files. The user specifies the file types in column 15 of the File Description Specification sheet.

STR CORE SIZES

The following figure gives core sizes for macros used with STR (Synchronous Transmitter/Receiver) devices.

	Number of Bytes
CDCNV Macro	*see below
CNTRL Macro	16
DIALO Macro (This macro includes the end-of-numbers character)	7+number
DTFSN Macro	520
Add: 100 bytes/DTFSN, after first	
READ Macro	10
SCLOS Macro	89
Add: 15 bytes/SCLOS, after first	
SOPEN Macro:	
1st DIAL = OUT or DIAL not specified	224
Add: 72 bytes/SOPEN, after first	
1st DIAL = IN	202
Add: 124 bytes/SOPEN, after first	
WAITM	52
Add: 2 bytes/DTF	
WRITE	10

*CDCNV

TYPE	1st CNV	2 - nth CNV	Also:
A	874 bytes	22 bytes	1st A, B, D, E, F
B	436 bytes	22 bytes	Following A, B, D, E, F
C	756 bytes	22 bytes	Subtract 148
D	896 bytes	22 bytes	1st A, B, D, E, F following C
E	432 bytes	22 bytes	Subtract 56
F	432 bytes	22 bytes	1st C Following A, B, D, E, F
			Subtract 56

BSC CORE SIZES

The following figure gives core sizes for macros used with BSC support.

	Number of Bytes
BCLOS macro	126
BOPEN macro	138
CNTRL macro	12
DTFBS macro	1174
ERRPT macro	6
IDIAL macro	542
READ macro	12
WRITE macro	12

Case 1: If the user specifies an Input or Update file (entry I or U in column 15), the following entries determine the number of main storage bytes required by this type of file:

Device Name (Col. 40-46)	File Format (Col. 19)	Block Length (BL) : Record Length (RL)	Type of File Organization (Col. 32)	The Number of Bytes Required
READ42	F	BL=RL	Blank	198
READ01	F	BL=RL	Blank	182
READ20	F	BL=RL	Blank	198
READ40	F	BL=RL	Blank	182
TAPE	F	BL=RL	Blank	284
TAPE	F	BL≠RL	Blank	452
TAPE	V	BL=RL	Blank	340
TAPE	V	BL≠RL	Blank	520
DISK11	F	BL=RL	Blank	478
DISK11	F	BL≠RL	Blank	622
DISK11	V	BL=RL	Blank	598
DISK11	V	BL≠RL	Blank	810
DISK11	F	BL=RL	D or T	386
DISK11	F	BL≠RL	T	386
DISK11	V	BL=RL	D or T	402

For Random ISAM:

Device Name (Col. 40-46)	File Format (Col. 19)	Block Length (BL) : Record Length (RL)	Type of File Organization (Col. 32)	Mode of Processing (Col. 28)	Number of Bytes	
					First File	Each Additional File
DISK11	F	BL=RL	I	R	942	322
DISK11	F	BL≠RL	I	R	1004	384

For sequential ISAM:

Device Name (Col. 40-46)	File Format (Col. 19)	Block Length (BL) : Record Length (RL)	Type of File Organization (Col. 32)	Mode of Processing (Col. 28)	Number of Bytes	
					First File	Each Additional File
DISK11	F	BL=RL	I	Blank	1464	464
DISK11	F	BL≠RL	I	L	1530	530
Disk11	F	BL=RL	I	Blank	1460	460
DISK11	F	BL≠RL	I	L	1524	524

Case 2: If the user specifies an Output file (entry O in column 15), the following entries determine the number of main storage bytes required by this type of file:

Device Name (Col. 40-46)	File Format (Col. 19)	Block Length (BL) : Record Length (RL)	Type of File Organization (Col. 32)	The Number of Bytes Required
READ42	F	BL=RL	Blank	252
READ42	V	BL=RL	Blank	324
READ20	F	BL=RL	Blank	164
READ20	V	BL=RL	Blank	225
READ40	F	BL=RL	Blank	156
READ40	V	BL=RL	Blank	368
TAPE	F	BL=RL	Blank	252
TAPE	F	BL≠RL	Blank	284
TAPE	V	BL=RL	Blank	420
TAPE	V	BL≠RL	Blank	460
DISK11	F	BL=RL	Blank	364
DISK11	F	BL≠RL	Blank	464
DISK11	V	BL=RL	Blank	436
DISK11	V	BL≠RL	Blank	628
PRINTER	F	BL=RL	Blank	262

Case 3: If the user specifies a Combined file (entry C in column 15), the following entries determine the number of main storage bytes required by this type of file:

Device Name (Col. 40-46)	File Format (Col. 19)	Block Length (BL) : Record Length (RL)	Type of File Organization (Col. 32)	The Number of Bytes Required
READ42	F	BL=RL	Blank	296
*READ40	F	BL=RL	Blank	440

*To exercise this option, the Punch-Feed-Read feature must be present.

To estimate the number of bytes required for input/output areas, the following points should be noted:

1. For a file containing unblocked records, add the Block Length (columns 20-23) to the number of main storage bytes required.

2. For a file containing fixed-length blocked records, add the Block Length and the Record Length (columns 24-27) to the number of main storage bytes required.

3. For a file containing variable-length blocked records, add BL and the Record Length to the number of main storage bytes required. BL is calculated using the following formula:

   BL = Block Length + 4(BF + 4) + Record Length

   where BF = Block Length/Record Length (round high)

4. For an IBM 1442/2540 combined file, the size of the input/output area is doubled.

FILE EXTENSION

This section gives the main storage sizes required for various types of files and tables, which are summarized as follows:

1. For each Record Address File present, 196 bytes are required. If TAG Sort is used, an additional 28 bytes are required.

2. For each Chaining file present, 96 bytes are required.

3. For each table present, the following entries determine the number of bytes required. The chart assumes the To Filename entry (columns 19-26) to be blank.

   If columns 19-26 contain other than a blank entry, add 98 to the number of bytes as listed in the chart. If there is more than one To Filename specification, use the following formula to determine the additional bytes required:

   [24 + 102*(Total number of To Filename entries)]

   When more than one table is specified, the following formula determines the number of bytes required in addition to the number specified by the chart for each table:

   [24 + 4*(Total number of Table Name entries)]

Table Name (Col. 27-32)	Sequence (Col. 45)	Table Name (Col. 46-51)	Sequence (Col. 57)	Number of Bytes
TABLENAME	A or D	Blank	Blank	150*
TABLENAME	Blank	Blank	Blank	132*
TABLENAME	A or D	TABLENAME	Blank	202**
TABLENAME	A or D	TABLENAME	A or D	220**
TABLENAME	Blank	TABLENAME	A or D	202**
TABLENAME	Blank	TABLENAME	Blank	182**

*If the table area is larger than 44 bytes, this amount should be subtracted from the amount shown in the chart.

**If the total area of the two tables is larger than 84 bytes, this amount should be subtracted from the amount shown in the chart.

The Table Area is found by multiplying the contents of columns 36-39 by the contents of columns 40-42. If the table is numeric, the packed length is to be used.

INPUT

This section gives the number of main storage bytes required when exercising various options related to the input record types and input field types. Information pertaining to the record type of an input file is entered in columns 7-42 of the Input Specification sheet, and information pertaining to the field types is contained in columns 43-74.

The following must be considered for the input record type:

1. If Sequence (columns 15-16) contains an entry, 4 additional bytes are required. If the file is in numeric sequence, an additional 74 bytes are required. If N is specified in column 17, 4 additional bytes are required.

2. The entries in C/Z/D (columns 26,33, or 40) require the following:

   C ... 6 bytes

   Z ... 12 bytes, plus 10 if Character contains &, -, or is blank.

   D ... 12 bytes

An N entry in columns 25,32, or 39 does not affect the preceding storage requirements.

3. If Stacker Select (column 42) contains an entry, an additional 10 bytes are required.

The following must be considered for the input field type:

1. If the field type specified in columns 43-58 is an unpacked numeric field, 14 bytes are required. For a packed numeric field (P in column 43) or an alphabetic field, 6 bytes are required. If Field-Record Relation (columns 63-64) contains an indicator, the preceding storage requirements are increased by 8 bytes for each field containing field-record relation; an additional 8 bytes are required for each record type containing field-record relation.

2.a. If Control Level (columns 59-60) contains an indicator (L1-L9) and Field-Record Relation is blank, 8 bytes are required; if Field-Record Relation contains an indicator, 16 bytes are required.

b. If a Chaining Fields indicator (C1-C3) is specified in columns 61-62, 16 bytes are required.

c. With or without Matching Fields specifications in columns 61-62, use the following formulas to determine the number of bytes required:

NF = Number of primary and secondary files

ML = Sum of matching field lengths

MF = Sum of the number of matching fields in all record types of the file

RF = Number of record types

For NF = 1:

$$84+(2*ML)+MF*\left(6+\left|\begin{array}{ll} 8 & \text{if Field-Record Relation} \\ \underline{10} & \text{if Numeric Field} \\ \underline{10} & \text{if ALTSEQ} \\ \underline{10} & \text{if Sequence Checking} \end{array}\right|\right)+(18*RF)$$

For NF > 1:

$$228+ML+(NF+1)*(ML+8)+(42*NF)+MF*\left(6+\left|\begin{array}{ll} \underline{8} & \text{if Field-Record Relation} \\ \underline{10} & \text{if Numeric Field} \\ \underline{10} & \text{if ALTSEQ} \\ \underline{10} & \text{if Sequence Checking} \end{array}\right|\right)+(18*RF)$$

3. If one entry is specified in Field Indicators. (columns 65-70) and the field is numeric, 18 bytes are required for the first indicator; if the field is alphabetic, 26 bytes are required for the first indicator. 12 bytes are required for each additional indicator specified.

CALCULATION

The following series of diagrams gives the amount of main storage required by the RPG operation codes upon expansion. The optional entries on the specification sheet determine the main storage requirement for each operation. (F1 = Factor 1, F2 = Factor 2, RF = Result Field).

1.

OPERATION CODE	Decimal Length of F1 : F2	Resulting Decimal Length of F1/F2 Operation : Decimal Length of RF	Half-Adjust	Result Field	Number of * Bytes
ADD/SUB	Equal	Equal		Same as F1	6
ADD/SUB	Equal	Equal		Not same as F1	18
ADD/SUB	Equal	Unequal			36
ADD/SUB	Unequal	Equal			36-42
ADD/SUB	Unequal	Unequal			48-54
ADD/SUB	Equal	Unequal	H		54
ADD/SUB	Unequal	Unequal	H		66-72
Z-ADD		Equal			6
Z-ADD		Unequal			24
Z-ADD		Unequal	H		42
Z-SUB		Equal		Same as F2	18
Z-SUB		Equal		Not same as F2	12
Z-SUB		Unequal			36
Z-SUB		Unequal	H		54
MULT		Equal			22
MULT		Unequal			34-44
MULT		Unequal	H		46-50
DIV					22
DIV	Add Zeros to Factor 1				44-50
DIV	Add Zeros to Factor 2				44-50
DIV			H		34-62
MVR		Equal			18**
MVR		Unequal			36**

*If the Result Field entry is a table name, an additional 10 bytes are required.

**No table update is possible with the MVR operation code.

2.

OPERATION CODE	Factor 2	Result Field	Number of Bytes*
MOVE	Alphameric	Alphameric	10
MOVE	Alphameric	Numeric	10-16
MOVE	Numeric	Alphameric	10
MOVE	Numeric	Numeric	10-16
MOVEL	Alphameric	Alphameric	10
MOVEL	Alphameric	Numeric	22
MOVEL	Numeric	Alphameric	16
MOVEL	Numeric	Numeric	28-34
MLLZO	Alphameric	Alphameric	10
MLLZO	Alphameric	Numeric	24
MLLZO	Numeric	Alphameric	16
MLLZO	Numeric	Numeric	10
MLHZO	Alphameric	Alphameric	10
MLHZO	Numeric	Alphameric	16
MHLZO	Alphameric	Alphameric	10
MHLZO	Alphameric	Numeric	30
MHHZO	Alphameric	Alphameric	10

*If the Result Field entry is a table name, an additional 10 bytes are required.

3.

OPERATION CODE	Sequence of Factor 2	Result Field	Resulting Indicators	Number of Bytes
LOKUP	Any	Blank	Equal	60
	Any	Table Name	Equal	70
	Ascending	Blank	High	60
	Ascending	Table Name	High	70
	Ascending	Blank	Low	82
	Ascending	Table Name	Low	92
	Ascending	Blank	High/Equal	76
	Ascending	Table Name	High/Equal	86
	Ascending	Blank	Low/Equal	102
	Ascending	Table Name	Low/Equal	112
	Descending	Blank	High	82
	Descending	Table Name	High	92
	Descending	Blank	Low	60
	Descending	Table Name	Low	70
	Descending	Blank	High/Equal	102
	Descending	Table Name	High/Equal	112
	Descending	Blank	Low/Equal	76
	Descending	Table Name	Low/Equal	86

4. COMP

   a. If Factor 1 and Factor 2 are numeric fields and have equal decimal lengths, 10 bytes are required for the compare, plus 12 bytes for each resulting indicator specified.

   b. If Factor 1 and Factor 2 are numeric fields and have unequal decimal lengths, 34 bytes are required for the compare, plus 12 bytes for each resulting indicator specified.

   c. If Factor 1 and Factor 2 are alphameric fields and have equal field lengths, 10 bytes are required for the compare, plus 12 bytes for each resulting indicator specified.

   d. If Factor 1 and Factor 2 are alphameric fields and have unequal field lengths, 32 bytes are required for the compare, plus 12 bytes for each resulting indicator specified.

5. TESTZ

   a. Plus - 34 bytes

   b. Minus - 34 bytes

   c. Zero - 50 bytes

   d. Plus and Minus - 58 bytes

   e. Minus and Zero - 62 bytes

   f. Plus and Zero - 62 bytes

   g. Plus, Minus, and Zero - 74 bytes

6. The following operation codes require additional bytes as specified:

   a. GOTO - 6 bytes

b. EXIT - 6 bytes

c. TAG - 26 bytes

d. SETON and SETOF - 4 bytes for each indicator specified

e. EXTCV - 12 bytes

16 bytes if followed by KEYCV

7.a. For each indicator specified in columns 7 - 17, 8 additional bytes are required.

b. When using the operation codes ADD, SUB, Z-ADD, Z-SUB, MULT, and DIV, 18 bytes are required for the first indicator specified in columns 54 - 59. 12 bytes are required for each additional indicator.

Note: For each field name defined as a ULABL which is used in Factor 1, Factor 2, or Result Field, an additional 4 bytes are required.

OUTPUT-FORMAT

This section gives the main storage sizes required by the various output field types and output record types.

1. Output Record Types (Columns 15-31)

   a. H, D, or T in column 15 -- 6 bytes.

   b. OR in columns 14-15 -- 6 bytes.

   c. AND in columns 13-15 -- no additional requirements other than those specified in (f).

   d. Stacker Select is specified -- 6 bytes.

   e. Space and/or Skip -- 8 bytes.

   f. For each indicator specified for a record (columns 23-31) -- 6 bytes.

   g. No Stacker Select and no Space/Skip specified -- 2 bytes.

2. Output Field Types (Columns 23-74)

   a. For each indicator specified for a field (columns 23-31) -- 8 bytes.

   b. Blank After (column 39) specified without an associated indicator:
      Numeric Field -- 6 bytes.
      Alphameric Field (length equal to 1) -- 4 bytes.
      Alphameric Field (length greater than 1) -- 10 bytes.

      Blank After specified with an associated indicator:
      Numeric Field -- 14 bytes.
      Alphameric Field (length equal to 1) -- 12 bytes.
      Alphameric Field (length greater than 1) -- 18 bytes.

c.

Field Name (Columns 32-37)	Zero Supress (Column 38)	Packed Field (Column 44)	Constant or Edit Word (Columns 45-70)	Number of Bytes
Alphameric	Elank	Blank	Blank	6
Blank	Blank	Blank	Constant	6
Numeric	Blank	Blank	Blank	6
Numeric	Blank	P	Blank	6
Numeric	Z	Blank	Blank	18

d. Page counter -- 6 bytes

e. The following diagram contains the main storage requirements for the edit operations. These requirements are based on the assumptions that Field Name contains a numeric field entry, and Zero Suppress and Packed Field are blank. (In the diagram, LE = the length of the edit word and LF = the length of the field.)

Constant or Edit Word	Number of Bytes	
(Columns 45 - 70)	LE = LF	LE > LF
Simple Edit	18	24
Fixed $	22	28
Floating $	28	34
CR Symbol	28	34
Minus (-) Symbol	26	32
Fixed $ and CR Symbol	32	38
Fixed $ and Minus (-) Symbol	30	36
Floating $ and CR Symbol	38	44
Floating $ and Minus (-) Symbol	36	42

Note: If any of the preceding output fields are defined as ULABL, an additional 4 bytes are required.

MISCELLANEOUS

1. Sterling Routines

   a. Linkage to Sterling subroutines for Input fields - 14 bytes/field

   b. Linkage to Sterling subroutines for Output fields - 12 bytes/field (No edit) 20 bytes/field (Edit)

   c. Sterling input subroutine

      1. With input IBM code for Shillings - 342 bytes
      2. With input BSI code for Shillings - 376 bytes

   d. Sterling output subroutine

      1. Printed output only - 1128 bytes

      2. Printed output and output IBM code for Shillings - 1228 bytes

      3. Printed output and output BSI code for Shillings - 1268 bytes

2. Calculation and Output Literals

   a. Numeric:

      $\left(\frac{\text{Length of literal } +2}{2}\right)$ for each unique literal

   b. Alphameric:

      1). Non-editword -- the length of each unique literal

      2). Editword

         a). the length + 1 of each unique edit word which contains an uneven number of blanks

         b). the length + 2 of each unique edit word which contains an even number of blanks

3. Input and Calculation Result Fields

   a. Numeric:

      $\left(\frac{\text{Length of field } + 2}{2}\right)$ for each unique field

   b. Alphameric: the length of each unique field

4. Control Levels

a. 22 bytes, plus 20 bytes for each control level specified.

b. Hold Area - 2* (Number of Control levels specified)

5. Indicators
   1 byte per unique indicator

6. Overhead - consisting of work area, address constants, linkage to the OPEN and CLOSE routine, EOJ routine, linkage between specifications, and common subroutines - 1000 bytes.

For the overall amount of main storage used, the sizes of the Supervisor and the user's routines should be added to the preceding total.

The format of the ESD card follows:

Card Columns

1 Multiple punch (12-2-9). Identifies this as a loader card.

2 - 4 ESD -- External Symbol Dictionary card.

11 - 12 Number of bytes of information contained in this card.

15 - 16 External symbol identification number (ESID) of the first SD, PC, or ER on this card (EBCDIC). relates the SD, PC, or ER to a particular control section.

17 - 72 Variable information.

8 positions - Name

1 position - Type code to indicate SD, PC, LD, or ER

3 positions - Assembled origin

1 position - Blank

3 positions - Control section length, if an SD-type or a PC-type. If an LD-type, this field contains the external symbol identification number (ESID) of the SD or PC containing the label.

73 - 80 May be used by the programmer for identification.

The format of the TXT card follows:

Card Columns

1 Multiple punch (12-2-9). Identifies this as a loader card.

2 - 4 TXT -- Text card.

6 - 8 Assembled origin (address of first byte to be loaded from this card).

11 - 12 Number of bytes of text to be loaded.

15 - 16 External symbol identification number (ESID) of the control section (SD) containing the text (EBCDIC).

17 - 72 Up to 56 bytes of text -- data or instructions to be loaded.

73 - 80 May be used for program identification.

The format of the RLD card follows:

Card Columns

1 Multiple punch (12-2-9). Identifies this as a loader card.

2 - 4 RLD -- Relocation Dictionary card.

11 - 12 Number of bytes of information contained in the card.

17 - 72 Variable information (multiple items).

a. Two positions - pointer to the relocation factor of the contents of the load constant.
b. Two positions - pointer to the relocation factor of the control sections in which the load constant occurs.
c. One position - flag indicating type of constant.
d. Three positions - assembled address of load constant.

73 - 80 May be used for program identification.

The format of the END card follows:

Card Columns

1 Multiple punch (12-2-9). Identifies this as a loader card.

2 - 4 END

6 - 8 Assembled origin of the label supplied to the Assembler in the END card (optional).

15 - 16 ESID number of the control section to which this END card refers.

17 - 22 Symbolic label supplied to the Assembler if this label was not defined within the assembly.

73 - 80 Not used.

The format of the XFR card follows:

Card
Columns

1 Multiple punch (12-2-9). Identifies this as a loader card.

2 - 4 XFR -- Transfer card.

6 - 8 Assembled origin of entry point (after the program is loaded it will receive control at this point).

15 - 16 External symbol identification number (ESID) of the control section in which the transfer occurs (EBCDIC).

17 - 22 Symbolic label of the entry point.

73 - 80 May be used for program identification.

The format of the REP (User replace card) follows:

Card
Columns

1 Multiple punch (12-2-9). Identifies this as a loader card.

2 - 4 REP -- Replace text card.

5 - 6 Not used.

7 - 12 Assembled address of the first byte to be replaced (hexadecimal).

13 Not used.

14 - 16 External symbol identification number (ESID) of the control section (SD) containing the text (Hexadecimal). Right adjusted with leading zeros optional. See the assembly output listing for this number.

17 - 72 From 1 to 11 four-digit-hexadecimal fields separated by commas, each replacing one previously loaded halfword. A blank indicates the end of information in this card.

73 - 80 May be used for program identification.

The format of the SYM card follows:

Card
Columns

1 Multiple punch (12-2-9). Identifies this as a loader card.

2 - 4 SYM - Symbol card.

11 - 12 Number of bytes of information contained in this card.

14 - 16 External symbol identification number (ESID) of the control section (SD) containing the text (Hexadecimal).

17 - 72 Variable information.
12 columns - 8 positions - symbol name
1 position - type identification (machine or assembler instruction other than EQU DC, or DS).
3 positions = Value attribute (the displacement within the CSECT).

or

17 columns - 8 positions - symbol name
1 position - type identification (EQU, DC, or DS).
3 positions - Value attribute the displacement within the CSECT)
1 position - constant type
1 position - length (one byte less than the constant
3 positions - multiplicity

73 - 76 Program identification taken from the name field of the first TITLE statement preceding the START card.

77 - 80 Sequence number starting with 0001.

APPENDIX J. REGISTER USAGE

Certain general registers have been set aside for special uses within the Basic Operating System. They are:

0-1	Parameter registers.
2-11	Problem program registers.
12-13	Supervisor interrupt registers.
14	Return register.
15	Entry point register.

PARAMETER REGISTERS (0-1): These are used with macro instructions, and may contain a value or an address. An address may point to a list of values called a parameter list. These registers may be used by the problem program with the following restrictions. When the problem program issues a Supervisor or IOCS macro, these registers will be used. If the problem program needs the contents of the register, it must save and restore the register's contents.

PROBLEM PROGRAM REGISTERS (2-11): These registers are available without restriction. If any are used by the Supervisor or logical IOCS routines, they are saved and then restored to their original value. Registers 10 and 11 are saved upon entry into the 1052, or interval timer routines. They are available to the problem program at this time and are restored upon completion of these routines.

SUPERVISOR INTERRUPT REGISTERS (12-13): These are used by the Supervisor interruption routines. Since interruptions are unpredictable, these registers are not available to any other routine, unless SUPVR SAVEREG=YES was specified when the supervisor was assembled. This routine is optional, but it must be included if the programmer plans to use registers 12 and 13 for programs that are executed in a disk-resident system. (For a description of the supervisor macro, see the Assembler with Input/Output Macros, publication listed in the Preface of this manual.)

Note: If autotest is used, the programmer must *not* use registers 12 and 13 because these registers are used by the Autotest Master Control routine.

RETURN REGISTER (14): This is a general register used by the Supervisor and IOCS routine to return to the problem program after completion of a macro. It is also used by the problem program to return to IOCS after executing an IOCS option (LABADDR return). It normally contains the 24-bit absolute address of the instruction in the calling routine to which control is to be transferred at the completion of the execution of the subroutine.

ENTRY POINT REGISTER (15): This is used to branch to the problem program from IOCS routines.

Logical IOCS requires two registers for linkage. These are 14 and 15. Register 15 is used to branch to the called routine and register 14 is used to return to the calling routine. When a macro call is issued (GET), IOCS will use the registers without saving their contents.

Relocatable Library Module Name	Number of Blocks in RL	Core Image Library Phase Name	Number of Blocks in CL	Description of Program	Comments
AORGZ	43	AORGZ AORGZ2	5	Relocatable library limits	LORGZ must be available on system residence, if AORGZ is to be used.
CORGZ	74	CORGZ* CORGZ1* CORGZ2* CORGZ3*	9	Copy system	
CSERV	25	CSERV	3	Core image library service	
DSERV	29	DSERV	3	Directory service	
LORGZ	27	LORGZ	3	Condense libraries	
MMAINT	87	MMAINT MMAIN1 MMAIN2	12	Macro Library maintenance	
MSERV	40	MSERV	5	Macro library service	
PSERV	28	PSERV	5	PUB table service	
RMAINT	33	RMAINT	4	Relocatable library maintenance	
RSERV	37	RSERV	5	Relocatable library service	
SYSCMA (key)	21	SYSCMA*	3	Core image library maintenance	Requires SYSLDR in core image library.
SYSDMP (key)	14	SYSDMP*	1	Storage Dump	Must be cataloged in the core image library of the system residence.
SYSEOJ (key)	45	SYSEOJ* SYSBPD*	6	Job Control	Must be cataloged in the core image library of the system residence.

Relocatable Library Module Name	Number of Blocks in RL	Core Image Library Phase Name	Number of Blocks in CL	Description of Program	Comments
SYSLDR (key)	115	SYSLDR* SYSTXT* SYSESD* SYSRLD* SYSXFR* SYSEND* SYSREP* SYSCTL* SYSINC* SYSPH1* SYSPH2* SYSENT* SYSMAP* SYSERR* SYSCMB*	19	Linkage Editor	Must be available for: load and execute, compile and execute, assemble and execute, and SYSCMA program
SYSOA1 (key)	51	SYSOA1* SYSOA2* SYSOB1* SYSOC1* SYSOC2*	5		Must be available for: AORGZ, CORGZ, Assembler, or logical IOCS OPEN
SYSOJ1	68	SYSOJ1 SYSOJ2 SYSOJ3 SYSOJ4 SYSOJ5	8		Must be available if logical IOCS for direct access files is used.
SYSOLA	64	SYSOLA SYSOLC SYSOLD SYSOLZ	8		Must be available if logical IOCS for indexed sequential files is used.
SYSOQA (key)	97	SYSOQA* SYSOQC* SYSOQD* SYSOQE* SYSOQH* SYSOQI* SYSOQO* SYSOQX*	11		Must be available for AORGZ, CORGZ, Assembler, or if logical IOCS for consecutive files is used.

Relocatable Library Module Name	Number of Blocks in RL	Core Image Library Phase Name	Number of Blocks in CL	Description of Program	Comments
SYSOT0	108	SYSOT0 SYSOT1 SYSOT2 SYSOT3 SYSOT4 SYSOT5 SYSOT6 SYSOT7 SYSOT8 SYSOT9 SYSOTA SYSOTB	12		Must be available if logical IOCS is used to handle tape files.
SYSRSD	22	SYSRSD SYSCPD	2		Must be available for check-pointing and re-starting programs on a checkpoint area on the system pack.
SYSRST	19	SYSRST SYSCPT	2		Must be available for check-pointing and re-starting programs on tape.

Relocatable Library Module Name	Number of Blocks in RL	Core Image Library Phase Name	Number of Blocks in CL	Description of Program	Comments
SYSSTA (key)	15	SYSSTA*	2		Must be available for: AORGZ, LORGZ, MMAINT, MSERV, RMAINT, RSERV, SYSCMA, and SYSLDR.
		SYSSUP*		Supervisor	Must be cataloged in the core image library of the systems residence.
ASSEMB	403	ASSEMB* ZZZ30A* ZZZ31A* ZZZ32A* ZZZ33A* ZZZ35A* ZZZ39A* ZZZ401* ZZZ403* ZZZ405* ZZZ406* ZZZ408* ZZZ410* ZZZ501* ZZZ815* ZZZ910* ZZZ920* ZZZ925*	50	Assembler	Must be available in system residence if Assembler is to be used.
ZZZ56A	490	ZZZ551* ZZZ56A* ZZZ57A* ZZZ57B* ZZZ57C* ZZZ60A* ZZZ62A* ZZZ62B* ZZZ63A* ZZZ65A* ZZZ66A* ZZZ67A* ZZZ68A* ZZZ69A* ZZZ70A* ZZZ71A* ZZZ72A* ZZZ73A* ZZZ75A* ZZZ76A* ZZZ77A* ZZZ772* ZZZ78A* ZZZ80A*	77		Must be available in System Residence, if Assembler is to be used.

Relocatable Library Module Name	Number of Blocks in RL	Core Image Library Phase Name	Number of Blocks in CL	Description of Program	Comments
DATBID	544	DATBAP DATBCT DATBCV DATBFL DATBID DATBLE DATBLF DATBMD DATBMG DATBPE DATBPL DATBPO DATBPT DATBST DATBTR	100	Autotest	

Note: All items listed in the first column (Relocatable Library Module Name) are contained in the relocatable library of the system pack supplied by IBM.

The relocatable library module name would be used in the following ways:

1. As the operand of a JOB card when used with EXEC LOADER,R
2. As the operand of a CATAL card (CATAL modulename,R) when cataloging from the relocatable library to the core image library.
3. As the operand of an INCLUDE card when cataloging to the core image library via the INCLUDE function.

The items identified by (key) are system programs that must be edited to run with the new Supervisor when generating a new system residence.

The number of blocks shown in the second and fourth columns are the estimated sizes of the respective items. The section Disk Storage Space Required for Libraries and Directories contains additional information on this subject.

All items in the third column (Core Image Library Phase Name) which are marked with an asterisk (*) are contained in the core image library of the system pack supplied by IBM.

APPENDIX L. NAMES OF IBM-SUPPLIED MACROS

The numbers represent the approximate number of blocks required to contain the macros in the macro library.

DTFSR	80	Consecutive Processing Macros (plus CHNG, CLOSE, CNTRL, DTFBG, DTFEN, GET, LBRET, OPEN, PUT)
DTFST	33	
DTFSU	65	
DTFSV	30	
DTFSW	56	
DTFSX	32	
DTFSY	28	
DTFSZ	66	
DTFTA	27	
DTFTC	13	
DTFTE	17	
DTFTG	15	
DTFTI	17	
DTFTK	12	
DTFTL	4	
DTFTM	16	
DTFTO	14	
DTFTQ	15	
DTFTS	8	
DTFTU	12	
DTFTW	9	
DTFTY	12	
DRFUA	9	
DTFUD	13	
DTFUE	10	
DTFUG	12	
DTFUI	12	
DTFUL	14	
DTFUM	9	
DTFUO	10	
DTFZA	42	
DTFZC	18	
DTFZE	38	
FEOV	1	
PRTOV	1	
RELSE	1	
TRUNC	1	
DTFDA	39	Direct Access (DAM) Macros (plus CLOSE, CNTRL, DTFBG, DTFEN, LBRET, OPEN, READ, WRITE)
DTFDC	25	
WAITF	1	
DTFIA	55	Indexed Sequential (ISFMS) Macros (plus CLOSE, DTFBG, DTFEN, GET, LBRET, OPEN, PUT, READ, WRITE)
DTFIC	25	
DTFIG	18	
DTFIH	10	
DTFIL	69	
DTFIM	37	
DTFIQ	42	
DTFIR	10	
DTFIS	38	
ENDFL	2	
ESETL	2	
SETFL	2	
SETL	2	
CHKPT	9	System Control Macros
COMRG	1	
DUMP	1	
EOJ	1	
EXIT	1	
FETCH	1	
IOCFG	93	
JBCTL	62	
MSG	1	
MVCOM	1	
RSTRT	8	
SEND	9	
STXIT	5	
SUPVR	29	
SYMUN	5	
CCB	3	Physical I/O Macros (plus CHNG, CLOSE, LBRET, OPEN, WAIT, WAITM)
DTFPH	9	
EXCP	1	
CDCNV	55	STR Processing Macros (plus READ, WRITE, CNTRL, WAIT, WAITM)
DIALO	4	
DTFRF	1	
DTFSN	15	
SCLOS	3	
SOPEN	10	

Common Macros

CHNG	3	Consecutive Processing and Physical IOCS
CLOSE	19	Consecutive Processing, Direct Access, and Indexed Sequential
CNTRL	9	Consecutive Processing, Direct Access, and STR Processing
DTFBG	1	Consecutive Processing, Direct Access, and Indexed Sequential
DTFEN	79	Consecutive Processing, Direct Access, and Indexed Sequential
GET	4	Consecutive Processing and Indexed Sequential
LBRET	1	Consecutive Processing Direct Access, and Indexed Sequential
OPEN	44	Consecutive Processing, Direct Access, and Indexed Sequential
PUT	4	Consecutive Processing and Indexed Sequential
READ	4	Direct Access, Indexed Sequential, and STR Processing
WAIT	1	Physical IOCS and STR Processing
WAITM	5	Physical IOCS and STR Processing
WRITE	5	Direct Access, Indexed Sequential, and STR Processing

Three formulas are used to compute disk storage requirements for an Indexed Sequential file. The known quantities for the computations given are:

D_L = Data Length
K_L = Key Length
B_L = Block Length (Data Length x Number of Records)
X = Number of prime data tracks per cylinder
L = Number of bytes (10) for overflow link information.

I. To calculate the number of prime data records per cylinder (Npr)

Let: A = Number of prime data records on a shared track
B = Number of records on a non-shared track.
(Note: These values must be whole numbers.)

Then: a. Determine the size of the track index (T_1),

$$T_1=[2X+1][91.49+1.049(K_L)]$$

b. Determine the number of bytes remaining on a track for prime records (T_2),

$$T_2=3625-T_1$$

c. Determine the size of the last prime record on a track (T_3),

$$T_3=20+K_L+B_L$$

d. Determine the number of prime data records on a shared track (A),

$$T_4=T_2-T_3$$

if the result (T_4) is negative, set A=0,
if the result (T_4) is zero, set A=1,
if the result (T_4) is positive, set

$$A=1+\frac{T_4}{81+1.049(K_L+B_L)}$$

e. Determine the number of records on a non-shared track (B),

$$B=1+\frac{3605-(K_L+B_L)}{81+1.049(K_L+B_L)}$$

Compute the number of prime records per cylinder (Npr) by substituting for A, B, and X in

$$Npr=A+B(X-1)$$

II. To determine the number of overflow records per track (Nor)

Compute:

$$Nor=1+\frac{3605-(K_L+D_L+L)}{81+1.049(K_L+D_L+L)}$$

III. To determine the number of cylinder or master index records per track (Nir)

Compute:

$$Nir=1+\frac{3595-K_L}{91.49+1.049(K_L)}$$

(Note: Allow for a dummy record.)

GLOSSARY

The explanations of the following terms relate only to their use in this publication.

Basic Operating System: A disk resident system; device dependent and non-modular that provides basic operating system capabilities for 8K and larger System/360 disk configurations.

Block: 1. To group records physically for the purpose of conserving storage space or increasing the efficiency of access or processing.
2. A physical record on tape or disk.

Buffer: 1. A storage device in which data is assembled temporarily during data transfers. It is used to compensate for a difference in the rate of flow of information or the time occurrence of events when transferring information from one device to another. For example, the IBM 2821 Control Unit (a control and buffer storage unit for card readers, card punches, and printers in a System/360).
2. A portion of main storage used for an input or output area.

Burst Mode: A means of transferring data to or from a particular I/O device on either the multiplexor or selector channel. All channel controls are monopolized for the duration of data transfer.

Byte Mode: (see multiplex mode)

Catalog: To enter a phase, a module, or macro in one of the system libraries as a permanent entry, and to include control information about the entry in the corresponding library directory.

Channel Program: One or more channel command words (CCWs) that control(s) a specific sequence of channel operations. Execution of the specific sequence is initiated by a single start I/O instruction.

Channel Scheduler: The part of the Supervisor that controls the movement of data between main storage and input/output devices.

Checkpoint: A point in a program at which sufficient information can be stored to permit restarting the computation from that point.

Checkpoint Record: Disk or tape records that contain the status of the job and the system at the time the records are written by the checkpoint routine. These records provide the necessary information for restarting a job without having to return to the beginning of the job.

Checkpoint/Restart: A means of restarting execution of a program at some point other than the beginning. When a checkpoint macro is issued in a problem program, checkpoint records are created. These records contain the status of the program and the machine. When it is necessary to restart a program at a point other than the beginning, the restart procedure uses the checkpoint records to reinitialize the system.

Checkpoint Routine: A routine that stores information for a checkpoint.

Command Control Block: An eight-byte field (four halfwords) required for each I/O device controlled by physical IOCS. This field is used for communication between physical IOCS and the problem program.

Communication Region: An area of the Supervisor set aside for interprogram and intraprogram communication. It contains information useful to both the Supervisor and the problem program.

Control Programs: A group of programs that provides functions such as the handling of input/output operations, error detection and recovery, program loading, and communication between the program and the operator. The IPL Loader, the Supervisor, and the Job Control programs are classified as control programs in the Basic Operating System.

Core Image Library: An area of the resident disk pack used to store programs that have been processed by the Linkage Editor. Each program is in a form identical to that which it must have to be executable in main storage. The programs in the core image library include system programs, the librarian programs, and other IBM-supplied programs such as Assembler, RPG, and sort programs. User programs are also stored in the core image library.

Core Storage: All addressable storage from which instructions can be executed or from which data can be loaded directly into registers.

Data File: A collection of related data records organized in a specific manner. For example, a payroll file (one record for each employee, showing his rate of pay, deductions, etc) or an inventory file (one record for each inventory item, showing the cost, selling price, number in stock, etc).

Directory: A group of records containing information used in locating and retrieving elements of the disk-resident system. There are six directories in the Basic Operating System: system directory, transient directory, core image directory, macro directory, relocatable directory, and phase directory.

Discontinuous Binary Number: A number containing two or more subordinate numbers, each subordinate number treated as a separate binary value.

Disk-Resident System: An operating system that uses disk storage for on-line storage of system routines.

Extent: A continuous area of direct access storage between defined upper and lower limits.

Fetch: 1. To bring a program phase into main storage from the core image library.
2. The routine that retrieves requested phases and loads them into main storage.
3. The name of a macro (FETCH) used to transfer control to the System Loader.

File: See Data File.

Fixed-Length Record: A record having the same length as all other records with which it is logically or physically associated.

Initial Program Loading (IPL): The initialization procedure that causes Basic Operating System to read programs into main storage.

Input/Output Control System (IOCS): A group of macro routines provided by IBM for handling the transfer of data between main storage and external storage devices. IOCS consists of two parts: physical IOCS and logical IOCS.

Interruption: A break in the normal sequence of instruction execution. It causes an automatic transfer to a preset storage location, where action is taken to satisfy the condition that caused the interruption.

I/O Area: An area (portion) of main storage into which data is read or from which data is written. In Operating System publications, the term buffer is often used in place of I/O area. I/O means Input/Output.

IPL Loader: A program that reads the Supervisor into main storage and then transfers control to the Supervisor.

Job Control: A control program that is called into storage to prepare each job to be run. Some of its functions are to assign I/O devices to certain symbolic names, set switches for program use, log (or print) job control cards, and call the requested program.

Job Control Statement: Any one of the control statements in the input stream that identifies a job or defines its requirements.

Job Statement (JOB): The control statement in the input stream that identifies the beginning of a series of job control statements for a single job.

Language Translators: A general term for any assembler, compiler, or other routine that accepts statements in one language and produces equivalent statements in another language. The Basic Operating System has two language translators: Assembler and Report Program Generator (RPG).

Librarian: The set of programs that maintains, services, and organizes the system libraries.

Library: An organized collection of programs, macro definitions, or object modules maintained on the system-resident disk pack. Three libraries are used by the Basic Operating System: core image library, macro library, and relocatable library.

Linkage Editor: A system service program that edits the output of language translators and produces executable program phases in the core image library. It relocates programs or program sections and links together separately assembled sections.

Load System Program: One of the system service programs. It is used to build a resident system from punched cards.

Logical File: A data file that has been described to the Basic Operating System through the use of a file-definition (DTF) macro instruction. Note that a data file is described to Operating System through a different defining method. Operating System publications refer to a data file described in this different manner as a data set.

Logical IOCS: A comprehensive set of macro routines provided to handle the creation, retrieval, and maintenance of data files.

Logical Record: A record identified from the standpoint of its content, function, and use rather than its physical attributes. It is meaningful with respect to the information it contains. (Contrasted with Physical Record.)

Macro Library: A collection of macro definitions cataloged onto the resident disk pack by the librarian.

Main Storage: See Core Storage.

Module (Programming): The input to or output from, a single execution of a language translator or a Linkage Editor; a separate program unit that can be combined with other units.

Multiplex Mode: A means of transferring records to or from low-speed I/O devices on the multiplexor channel, by interleaving bytes of data. The multiplexor channel sustains simultaneous I/O operations on several subchannels. Bytes of data are interleaved and then routed to or from the selected I/O devices or to and from the desired locations in main storage. Multiplex mode is sometimes referred to as byte mode.

Object Module: The output of a single execution of a language translator, that constitutes input to the Linkage Editor. An object module consists of one or more control sections in relocatable, though not executable, form and an associated control dictionary.

Operating System: 1. A collection of programs that enables a data processing system to supervise its own day-to-day operations, automatically calling in programs, routines, languages, and data as needed for continuous throughput of an uninterrupted series of jobs.
2. A modular and device independent operating system requiring direct access storage device residence; minimum main storage requirement is 32K.

Overlap: To do something at the same time that something else is being done; for example, to perform input/output operations while instructions are being executed by the central processing unit.

Overlay: 1. A section of a program (phase) loaded into main storage, replacing all or part of a previously loaded section.
2. The technique of repeatedly using the same blocks of internal storage during different stages of a problem. For example, when one routine is no longer needed in internal storage, another routine can replace all or part of that storage.

Phase: The smallest complete unit that can be referenced in the core image library. Each overlay of a program or (if the program contains no overlay) the program itself is a single complete phase.

Physical IOCS: Macros and Supervisor routines that schedule and supervise the execution of channel programs. Physical IOCS controls the actual transfer of records between the external storage medium and main storage.

Physical Record: A record identified from the standpoint of the manner or form in which it is stored and retrieved; that is, one that is meaningful with respect to access. (Contrasted with Logical Records.)

Problem Program: 1. The user's object program. It can be produced by either the Assembler or RPG language translators. It consists of instructions necessary to solve the user's problem.
2. A general term for any routine that is executed in the data processing system's problem state; that is, any routine that does not contain privileged operations. (Contrast with Supervisor.)

Processing Program: A general term for any program that is both loaded and supervised by system control programs. The term processing programs is in contrast to the term control programs.

Queue: A list of entries in order or in line, usually in the sequence of arrival. The entries identify things contending for service or attention.

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

Relocatable-Library Module: A module consisting of one or more complete object modules cataloged as a single entry in the relocatable library.

Restart: See Checkpoint/Restart.

Service Programs: Any of the class of standard routines that assists in the use of a data processing system and successful execution of problem programs. These programs include language translators, Autotest, Sort/Merge, and Utilities.

Source Module: A set of source statements in the symbolic language of a language translator, that constitutes the entire input to a single execution of the language translator.

Source Statement: Statements written by a programmer in symbolic terms related to a language translator such as Assembler or RPG.

Stacked Job Processing: A technique that permits multiple job definitions to be grouped (stacked) for presentation to the system. This allows the system to automatically process each job in sequence.

Supervisor: One of the control programs. It provides over-all control of operations. It consists of routines to control the functions of machine interrupts, external interrupts, operator communications, and physical IOCS requests and interrupts.

Symbolic I/O Assignment: A means by which problem programs can refer to an I/O device by a symbolic name. Before a program is executed, the Job Control program can be used to assign a specific I/O device to that symbolic name.

System Loader: One of the Supervisor routines. It is used to retrieve program phases from the core image library and load them into main storage.

System Residence: The external storage space allocated for storing the Basic Operating System. If refers to the space on an on-line disk pack that contains the necessary programs and disk areas required for executing a job on the data processing system.

System Service Programs: Programs that perform the functions of generating the initial operating system, generating specialized systems, creating and maintaining the library sections, and loading and editing programs into disk residence. These programs are: Linkage Editor, Librarian, and Load System.

Throughput: A measure of system efficiency; the rate of which work can be handled by a data processing system.

Transient Area: This is a main storage area (within the Supervisor area) used for temporary storage of transient routines.

Transient Routines: These routines are permanently stored on the system-residence disk pack and loaded (by the Supervisor) into the transient area when needed for execution.

Variable-Length Records: A record having a length independent of the length of other records with which it is logically or physically associated. (Contrasted with Fixed-Length Record.)

Volume: That portion of a single unit of storage media that is accessible to a single read/write mechanism. For example, a reel of magnetic tape on a 2400 magnetic tape drive or a disk pack on a 2311 disk storage drive.

INDEX

For Planning Purposes Only:

Whenever one reference has more significance than the others for an item, that page number is listed first.

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 Basic Operating System
Programmer's Guide C24-3372-6

- Your comments, accompanied by answers to the following questions, help us produce better publications for your use. If your answer to a question is "No" or requires qualification, please explain in the space provided below. All comments will be handled on a non-confidential basis. Copies of this and other IBM publications can be obtained through IBM Branch Offices.

	Yes	No
- Does this publication meet your needs?	□	□
- Did you find the material:		
Easy to read and understand?	□	□
Organized for convenient use?	□	□
Complete?	□	□
Well illustrated?	□	□
Written for your technical level?	□	□

- What is your occupation?____
- How do you use this publication?

As an introduction to the subject?	□	As an instructor in a class?	□
For advanced knowledge of the subject?	□	As a student in a class?	□
For information about operating procedures?	□	As a reference manual?	□

Other____

- Please give specific page and line references with your comments when appropriate.

COMMENTS:

- Thank you for your cooperation. No postage necessary if mailed in the U. S. A.

YOUR COMMENTS, PLEASE . . .

This publication is one of a series 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, help us produce better publications for your use. Each reply is 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 using 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. 170
ENDICOTT, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
P. O. Box 6
Endicott, N. Y. 13760

Attention: Programming Publications, Dept. 157

Fold Fold

Cut Along Line

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]

Additional Comments: