File No. S360-36
Form C28-6810-1

44PS

IBM Systems Reference Library

IBM System/360 Model 44

Programming System

Concepts and Facilities

This publication describes the facilities provided by the IBM System/360 Model 44 Programming System.

The Model 44 Programming System consists of a FORTRAN compiler, an assembler, a supervisor, and system support programs. It provides FORTRAN and assembler languaqe processing and program execution in a monitored environment, with automatic job-to-job transition, interruption handling, and input/output supervision. The system has facilities for the creation and maintenance of libraries and the manipulation of their contents. It also provides extensive job control and program segmentation capabilities for flexibility and versatility in the preparation of programs for execution.

PREFACE

The first section of this publication is a survey of the basic concepts of the entire Model 44 Programming System. Subsequent sections discuss the operation of the supervisor and the system support programs. In addition, there is an appendix describing the publications that support the programming system and a glossary of terms.

In order to understand this publication, the reader should be familiar with basic data processing techniques and terminology, and with the functional characteristics of the Model 44, as described in the following publications

IBM System/360 System Summary, Form A22-6810

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

IBM System/360 Model 44: Functional Characteristics, Form A22-6875

Related literature on specific input/output devices, educational material, etc., is described in the IBM System/360 Bibliography, Form A22-6822.

Second Edition

This is a major revision of, and makes obsolete, Form C28-6810-0. A Model 44 Programming System publications plan has been added; the Appendix headed "Labels and Label Processing," which appeared in the first edition, has been deleted and can now be found in the publication IBM System/360 Model 44 Programming System: Guide to System Use, Form C28-6812. Other changes to the text are indicated by a vertical line to the left of the change; revised illustrations are denoted by the symbol ● to the left of the Figure caption.

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

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form for readers' comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, 1271 Avenue of the Americas, New York, N.Y., 10020.

CONTENTS

ILLUSTRATIONS

TABLES

The minimum machine configuration required for use of the Model 44 Programming System is as follows:

- IBM 2044 Processing Unit with its Console Printer-Keyboard, Single Disk Storage Drive, and at least 65,536 bytes of main storage.
- One multiplexor channel (#5248 or #4598).
- One IBM 2315 Disk Cartridge (used for system residence).
- One of the following input devices:

  IBM 1442 Model N1 Card Read-Punch
  IBM 2501 Model B1 or B2 Card Reader
  IBM 2520 Model B1 Card Read-Punch
  IBM 2540 Model 1 Card Read-Punch
  IBM 2401 or 2402 Model 1, 2, 3, 4, 5, or 6 Magnetic Tape Unit
  IBM 2403 Model 1, 2, 3, 4, 5, or 6 Magnetic Tape Unit and Control
  IBM 2404 Model 1, 2, or 3 Magnetic Tape Unit and Control
  IBM 2311 Disk Storage Drive

- One of the following devices for list output:

  IBM 1403 Model 2, 3, 7, or N1 Printer
  IBM 1443 Model N1 Printer
  Any of the magnetic tape or disk units listed above

- One of the following devices for punch output:

  IBM 1442 Model N2 Punch
  IBM 2520 Model B2 or B3 Punch
  Any of the card read-punches listed above
  Any of the magnetic tape or disk units listed above

Notes on configurations:

1. In addition to the foregoing requirements, the system supports the attachment of the following units:

   A second Single Disk Storage Drive (with 2315 Cartridge) which, alternatively, may be used for system residence.
   IBM 2311 Disk Storage Drives (with IBM 1316 Disk Pack)
   Additional magnetic tape units (any of the models listed above)
   Up to two additional multiplexor channels.

2. A system-residence 2315 Disk Cartridge can be created using the minimum machine configuration, provided the input device is a card reader. Assembly is not required in this procedure. However, if it is desired to assemble the IBM-supplied components of the system, a magnetic tape unit and a second single disk storage drive, if available, can be used in addition to the basic system to minimize card handling.
3. If more than 65,536 bytes of main storage are available, the system will take advantage of their availability.
4. The FORTRAN compiler requires that the 2044 be equipped with the floating-point arithmetic feature (#4427). The assembler also requires this feature if it is desired to assemble floating-point constants.
5. The user may modify the supervisor to include input/output routines for additional devices.
6. The Read-Backward feature of the 2400 Series Magnetic Tape Units is not supported.

BASIC CONCEPTS

The IBM System/360 Model 44 Programming System, like the computing system itself, is designed to meet the specific needs of the scientific user. The principal objectives of the Model 44 system are to increase the throughput (i.e., the rate at which work is handled) performance of the computing system and, at the same time, to relieve the programmer of much of the work involved in program preparation, so that he can concentrate on the problem-solving aspects of programming.

SYSTEM COMPONENTS

The system resides on an IBM 2315 Disk Cartridge mounted on the Model 44 Single Disk Storage Drive. It includes a supervisor, a set of support programs that perform system-related and utility functions, and two language processors: a full FORTRAN IV compiler and an assembler. It also includes stand-alone (i.e., not operating under system control) programs that are not resident on the 2315 Disk Cartridge as follows: (1) a program for constructing a system residence volume, (2) a loader for loading system-produced programs that are to be executed independently of system control, (3) two stand-alone disk initialization programs, (4) a program for making a back-up copy of the system residence volume, and (5) a program to print or punch all or part of an IBM-distributed tape containing the programming system.

SUPERVISOR

The supervisor controls the entire system and provides a common interface to all processing programs, including the FORTRAN compiler, the assembler, the system support programs, and user-written programs. Specifically, the supervisor:

- Manages the use of system resources. A resource is any facility of the system required by a job and includes input/output devices, data sets (the term applied to the major unit of data handled by the system), and processing programs.
- Loads the appropriate execution phases from the phase library (i.e., the library of programs in absolute form, ready for execution).
- Handles all standard tape label checking, input/output requirements, and input/output error recovery procedures. The programmer requests input/output operations without reference to a particular device type, so that he does not have to alter his source program when data sets are moved from one device to another.
- Services interruptions and transfers control to the appropriate system or user routine for interruption processing.
- Schedules channel use to effect overlap of processing with input/output operations.
- Provides for communication with the console operator.

The supervisor functions are described in greater detail later in this publication.

SYSTEM SUPPORT PROGRAMS

The system support programs provide a wide range of capabilities for use by both the system and the programmer.

- Job Control Processor. Job control statements provide the parameters that describe the program(s) to be executed and the resources required to do so. This program processes the job control statements and sets up the tables and communication areas necessary to execute a particular program. The job control processor allocates input/output resources and provides data set maintenance functions.
- Linkage Editor. The output produced by the compiler or assembler, called a module, is in relocatable form. This program edits the modules into absolute form, automatically combining these modules with modules from the module library (one of the system libraries) where necessary. Since the formats of the compiler-produced modules and the assembler-produced modules are the same, the linkage editor can combine modules from both sources. This facility permits preparation of a program from parts written in either language.
- Utility Programs. These programs provide data set transmission and external storage initialization and maintenance functions.

The system support programs are described in greater detail later in this publication.

LANGUAGE PROCESSORS

The FORTRAN compiler translates programs written in the FORTRAN language to System/360 Model 44 relocatable object program modules.

The FORTRAN language parallels that of mathematics and is a familiar tool in mathematics, science, and engineering. Variables and constants may be either fixed- or floating-point values in single or double precision. In addition to the capabilities of the language itself, there is a library of subroutines for performing common mathematical procedures such as finding square roots, trigonometric functions, logarithmic values, etc. The compiler recognizes the subroutine requirements and automatically supplies the linkages to the subroutine.

Detailed language specifications for the Model 44 FORTRAN language are given in IBM System/360 FORTRAN IV Language, Form C28-6515.

The assembler translates System/360 assembler language programs to relocatable object program modules. The assembler language consists of all System/360 Model 44 instruction mnemonics, literals, and a set of assembler instructions that direct the assembly process. The assembler language includes facilities for updating symbolic source language data sets by inserting, replacing, and deleting card images and reserializing the data set. Detailed language specifications for the assembler language are given in IBM System/360 Model 44 Programming System: Assembler Language, Form C28-6811.

The assembler language programmer may use instructions in his program that request functions from the supervisor. For example, an instruction to read, included in an assembler language program, is actually an instruction to the input/output procedures portion of the supervisor.

In the case of a FORTRAN program, the compiler generates the instructions that interact directly with the supervisor. For example, a READ statement in a FORTRAN source program causes the compiler to generate instructions for the supervisor.

SYSTEM CONSTRUCTION AND EDITING

Each system can be tailored to the specific system function requirements and input/output configuration of the installation. The installation may modify the IBM-supplied configuration, deleting functions not required by the installation and adding installation-created functions and programs. Depending on the nature of the modifications desired, there are two methods of creating a system residence volume: system construction and system editing.

System Construction

A stand-alone program is provided that constructs a system (on a 2315 Disk Cartridge) from absolute and relocatable decks containing the executable phases and the relocatable modules the installation elects to include in its system. Note that no reassembly of system components is necessary in this process.

All of the announced support will be included in the decks as distributed by IBM. As part of the initial program loading (IPL) procedure, the operator may specify alterations in the predefined device-channel configuration and unit functions.

System Editing

The installation can edit the system, by an assembly process, to change or extend the machine configuration, change the unit functions, alter the default conditions for system options, incorporate installation functions, etc.

ABSOLUTE LOADER

The absolute loader is a self-loading program that operates independently of system control. It can be used to load and transfer control to program card decks produced by the system's language processors or linkage editor.

The loader performs no functions other than loading the program text at the addresses specified in the cards containing the text and transferring control to the entry point specified in the input deck's END card. Therefore, if the input is language processor cards that have not been linkage edited, the deck should not contain external references or address constants that require relocation.

Programs loaded in this way are unable to use any of the programming system's features, such as input/output handling. There also is no protection against inadvertent destruction of the system residence volume if it is mounted on-line.

STAND-ALONE DISK INITIALIZATION

Two self-loading programs, operating independently of system control, are available for initializing 2315 Disk Cartridges and 1316 Disk Packs. Their function is to allocate the space for the volume label and the volume table of contents, and to write the label information necessary for the system to use these volumes.

The stand-alone disk initialization programs perform the additional function of checking for defective tracks. If a track is found to be defective, it is flagged, an alternate track is assigned, and the system thereafter uses the alternate track.

These same functions are also available under system control, via the utilities processor.

SAVE/RESTORE

The self-loading, stand-alone save/restore program copies the contents of a disk onto tape and, when needed, restores the material to the disk. The save/restore program can be used to make a tape copy of the system residence volume. If this disk volume is ever destroyed inadvertently, the save/restore program can then be used to copy the contents of the volume from the tape back onto the disk.

Detailed information about this program is available in the Operator's Guide, cited in the Appendix.

PRINT/PUNCH

The print/punch program enables users to copy the distributed version of the programming system onto a printer or punched cards. The output includes the absolute text and relocatable modules from the distributed tape reel. This program applies only to users who receive the system on magnetic tape.

Detailed information about this program is in the Operator's Guide, cited in the Appendix.

SYSTEM HIGHLIGHTS

The programming system for the Model 44 is designed to give the user the combined benefits of efficiency, flexibility, and ease of use.

For most applications, the programmer can state his processing requirements very simply, since most of the processing options have a default condition that represents the usual case; that is, in the absence of an option specification, the system performs in the manner most commonly required. At the same time, the options do exist that permit a wide range of flexibility.

Furthermore, the processing options are selected in statements that are supplied independently of the source language program itself, so that they can be varied from one run to the next without altering the source program. Thus, the programmer, within reasonable limits, can build and test his program without regard to the conditions under which it is finally to be executed and can, in fact, vary these conditions from execution to execution.

JOB PROCESSING

The system provides for sequential job processing with automatic job-to-job and step-to-step transition. A job, which consists of one or more job steps, is defined as a unit of processing from the standpoint of installation accounting and operating system control. A job step is a unit of processing from the standpoint of the programmer. It involves a description, via job control statements, of the resources required by and the execution of an explicitly defined program.

The job steps in a job need not be related. However, the system considers them to be interdependent to the extent that, if a job step fails in execution, the entire job is terminated. Furthermore, certain specifications may be made to extend over all or selected sequences of the job steps within the job.

The possible combinations of job steps that make up a single job permit great flexibility in the preparation of programs. The simplest application would involve executing a program already in the phase library. Another simple application, done in three job steps, is to compile (or assemble) a program into a relocatable module, edit the module into absolute form, called a program phase, and load and execute the absolute phase. The intermediate step, editing, is done by the linkage editor. The linkage editor can also edit several relocatable modules together, automatically adding required modules from the module library, resolving symbolic cross-references among them, and producing a single absolute phase. A more complex application might consist of several compilation and/or assembly steps, an editing step that combines their output modules (and perhaps library modules and modules that were assembled or compiled in a previous job) into one or more absolute phas-

es, and an execution of one or more of these phases.

The programmer may retain his program (or parts of his program) in its relocatable and/or absolute form in direct access storage or on magnetic tape or cards. The full range of job processing options for a sequence of related job steps is described below. Within one job there may be several such sequences that are logically independent of one another.

- Compile (or assemble) Only -- This option is useful when debugging a program or when creating a program module that is to be combined with other modules in later jobs.

- Compile (one or more) and Edit, or Edit Only -- This option is useful in debugging, to ensure that all cross-references between modules are correct and/or to retain a program in absolute format. Note that a job can consist of a single editing step, editing relocatable modules from previous jobs.

- Compile (one or more), Edit, and Execute; or Edit and Execute; or Execute Only -- Note that a job can consist of a single execution step, executing a program that was edited into absolute form in a previous job.

DATA MANAGEMENT

Setting up the mechanics of data and program storage and retrieval occupies a large percentage of programming time and effort. The system provides the programmer with several aids that reduce this time and effort both by simplifying the procedures and by eliminating the redundant effort and detailed record-keeping frequently associated with repeated use of data or programs.

Specifically, the system has facilities for systematically creating, manipulating, and keeping track of data sets. A data set is defined as a named collection of data. The definition of a data set is solely in terms of its name; the definition is completely independent of its data content or the way in which that data will be processed. Moreover, a data set is defined independently of any processing program.

Within a processing program, the programmer expresses his input/output requirements symbolically. That is, just as he manipulates symbols that represent variables with values to be supplied later, he specifies input/output operations on a symbolic data set. The relationship between a symbolic reference and the data set whose data content is to be processed is established at the time the processing program is executed. Thus, the actual data set that is processed may be varied from execution to execution without altering the processing program.

In order for a data set to be used by the system it must be identified to the system by both its name and its location. There are two ways of identifying a data set to the system. One is called cataloging. The catalog is itself a data set that the system uses in a special way. Each entry in the catalog contains a data set name and an indication of its location. Catalog entries are made at the programmer's request.

Once a data set has been cataloged, the programmer can refer to it by name without regard to its location. Since the catalog is a part of the system, a cataloged data set can be referred to and used indefinitely, by any number of jobs, until its entry is explicitly removed from the catalog.

The physical location of a data set is entered in the catalog in terms of a volume identification, i.e., the installation's designation of the volume on which it resides. A volume is defined as all of that portion of a storage medium that is accessible at one time by a unique channel and unit address. Thus, a volume might be a 2315 Disk Cartridge, a 1316 Disk Pack, a reel of magnetic tape, or a card deck in a card reader hopper.

If a data set is not cataloged, the programmer may still refer to it by name, but he must also supply the volume identification in a job control statement within his job. The data set is then identified to the system for the duration of the job.

The space on a volume occupied by a particular data set is called the extent of that data set. For a deck of cards or a reel of magnetic tape, it is not necessary for the programmer to define an extent for his data set, since only one data set at a time may occupy or reserve one of these volumes. For direct access storage volumes, however, the system must know the extent of each data set on the volume so that it can allocate space for new data sets, locate data sets already on the volume, etc.

Each direct access volume carries a volume table of contents (VTOC)[1] that iden-

---

[1]The VTOC actually contains the collected data set labels for all data sets on that volume.

tifies each data set on the volume by name and extent. In creating a data set that is to reside on a direct access volume, the programmer must specify the anticipated size of the data set. When the system allocates space for the data set, it automatically makes an entry in the VTOC. Later references to that data set cause the system first to locate the volume (either from a catalog entry or from a job control statement giving the volume identification) and then to search the VTOC for the actual address of the data set.

Note that the space for a data set is allocated and named independently of any reading or writing of the data within it. Thus, the data set may be empty (i.e., the space is reserved for data), partially full (with the potential for later additions), or completely full. In addition, a data set may be completely rewritten, i.e., the data content may be completely replaced.

Data Organization

The facilities for creating data sets exist principally in the job control processor. The user may organize his data set in one of two ways. The first of these, called sequential, is the familiar structure in which records are placed in sequence. Given one record, the next record to be processed is uniquely determined. The system processes all data sets (or members of a directoried data set; see below) sequentially. However, the programmer may alter the sequence of processing (e.g., for direct access applications) using input/output functions available in the system.

In the second organization, called directoried, each data set is organized in two parts, a directory and members. The directory contains the name of each of the members, a pointer to the location of each member in the data set, and an indication of the length of each member. The members may be program modules or any other data. Directoried data sets must reside in direct access storage. The system uses the directory to locate individual members when they are required.

Note that a symbolic data set reference can be related, via job control statements, to a specific member of a directoried data set, even though a member is not a data set, but only a part of one. A member has the characteristics of a sequential data set and the processing program is unaware that it is not a complete data set.

When a member is created, it may be given more than one name and the multiple names are also listed in the directory. This enables the user to obtain the member by using any one of several names. For example, consider the case of a member consisting of one subroutine that performs both sine and cosine evaluation. If it is listed in the directory under both SIN and COS, the programmer can call for the member by a name that is meaningful in the context of his program.

The system provides facilities, via the job control processor and utility programs, for adding and deleting directoried data set members. The addition of a member to a directoried data set is always made starting at the end of the last member added. A deletion consists of simply removing the member entry from the directory. Thus, it is possible for a directoried data set to appear to be too full for a new addition even though sufficient space is actually available because of previous deletions. In these circumstances, the data set can be condensed. The condense function shifts the members in the data set to fill up vacant areas; the order of the members is not changed. Figure 1 illustrates the condensing process.

Whichever organization is used for a particular data set, the following conventions apply:

1. All data sets on direct access devices must have fixed-length blocks.

2. More than one data set may occupy a direct access volume, but each data set must reside on contiguous tracks and cylinders.

3. Tape reels may not contain more than one data set.

LIBRARIES

The system incorporates two directoried data sets -- the module library and the phase library.

The module library contains relocatable program modules, produced by the compiler or assembler, and is a source of input for the linkage editor. The FORTRAN input/output conversion and mathematical subroutines are in this library.

The phase library contains program phases that have been edited into absolute form by the linkage editor; this library is the source from which programs are loaded for execution. The IBM-supplied processors reside permanently in this library. User-written programs may reside in this library either permanently or temporarily (i.e., just long enough to be executed once).

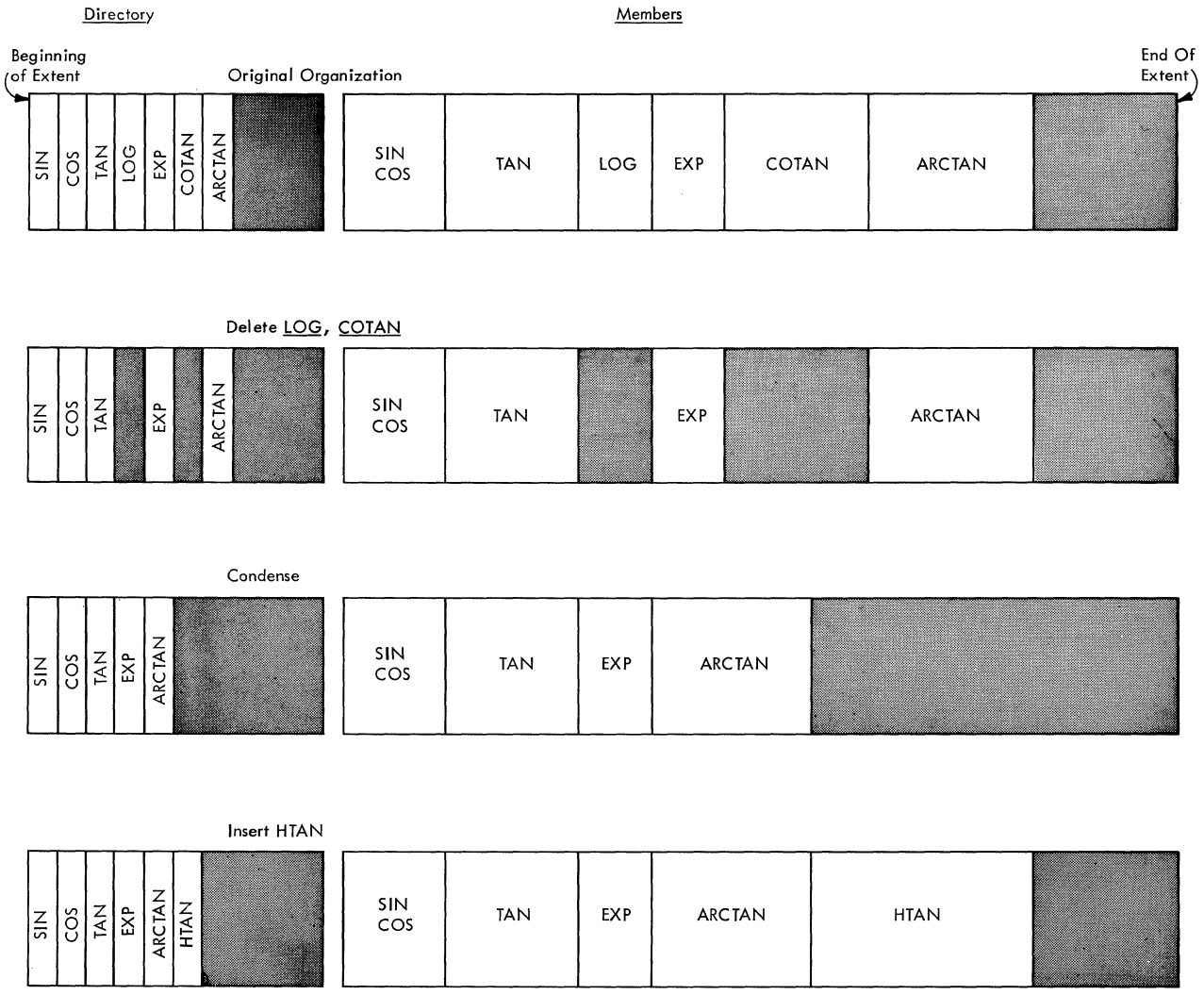


Figure 1. The Condensing Process

The presence of the phase library is essential to the operation of the system. Therefore, it resides on the 2315 Disk Cartridge designated as the system residence volume. The presence of the module library is optional and it need not reside on the system residence volume.

As with any other directoried data sets, the programmer may make additions to and deletions from the libraries, condensing as necessary. Note, however, that all additions to the phase library must be made via the linkage editor.

EXTERNAL STORAGE ASSIGNMENT

All references within a program to external storage are made in terms of symbolic data sets, which are in turn associated with symbolic unit names. The programmer uses job control statements to specify the assigning of a symbolic unit to an actual data set or member of a directoried data set.

The standard input/output device assignments and disk storage space allocation required by the system components and most problem programs are defined as data sets and associated with symbolic unit names during construction (or editing) of the system. The term for the predefined data set-symbolic unit relationships is system units. The programmer can use these data sets during his job simply by referring to their symbolic unit names; he need not be concerned with their type or whereabouts.

If the programmer requires access to data sets whose symbolic unit assignments are not predefined, or if he needs to preserve data sets that normally are associated with system units, he may request the assignment of a symbolic unit to the

desired data set (or member of a directoried data set). The volume and the location on the volume of the data set is either already known to the system (the data set already exists), or the system is to allocate an extent and name it (the data set is to be created).

If the symbolic unit name is the same as one of the predefined system unit names, the effect is to redefine the data set assignment temporarily. For example, the normal source for system input might be the card reader; the programmer can temporarily reassign this function to a data set on a specific reel of magnetic tape and the system will then take his input from that data set.

A symbolic unit can be associated with only one data set at a time, but different symbolic units can be assigned to the same data set at the same time.

Requests for assignments are made in job control statements that are part of the job but independent of any job step within the job. In any case, within a job step the programmer can refer to all data sets (or members of a directoried data set) symbolically, so that he is not concerned with device addresses and disk space allocation when writing his program.

Table 1 consists of a list of the predefined symbolic units by symbolic name, data set name, function, and potential device type.

SUMMARY OF DATA MANAGEMENT RELATIONSHIPS

Figure 2, using the 2315 Disk Cartridge as an example, illustrates the relationships involved in data management and the times at which they are established.

DIRECT ACCESS STORAGE MANAGEMENT

The system controls the disposition of all direct access storage space. In order to keep track of the space that is in use, the system creates a special label on each direct access volume as part of volume initialization. This label, called a format 5 label,[1] indicates the extents of any space that is not occupied by a data set. Each time a data set is added to or deleted from a volume, the system automatically updates the format 5 label.

All direct access storage space is allocated in units of a full track, starting at a track boundary. If the programmer requests an amount that results in a partial track, the system automatically extends the amount to the next track boundary.

The locations occupied by one data set must be contiguous in the volume. If there is sufficient space on a 2315 Disk Cartridge for a new data set but the space is not properly located (i.e., it is not contiguous), the volume may be squeezed in a manner similar to the condensing function available for directoried data sets. This function shifts the data sets on the volume to fill up lower numbered tracks that are not in use, deleting expired data sets. Once a disk volume has been squeezed, all the available space starts at the end of the last data set and continues to the end of the volume. The system updates the format 5 label and all affected volume labels as part of the squeezing process.

Note that squeezing a volume is a function of the utility processor, whereas condensing a directoried data set is a function of the job control processor.

Since the system identifies a data set by its extent as well as its name, any attempt to write beyond the extent is considered an error. Thus, if it is necessary to enlarge a data set beyond the boundaries initially set for it, the programmer must request allocation of an entirely new area, big enough to contain his entire enlarged data set.

INPUT/OUTPUT FACILITIES

The system provides two levels of input/output facilities. At the higher level, called read/write, the programmer invokes read/write functions to call system routines that will execute the input/output operation. The lower level, called execute channel program, uses a combination of routines supplied by the programmer and the system. Assembler-language programmers invoke the input/output functions, at both levels, via an assembler-language calling sequence. The FORTRAN programmer is not concerned with either level, since his input/output requests are handled by the FORTRAN compiler.

---

[1] This is a label designation of Operating System/360, which supports a volume label, five standard direct access storage data set labels and the standard magnetic tape data set labels. Of these labels, the Model 44 Programming System supports the creation and use of the volume, formats 1, 4, and 5, and the tape labels.

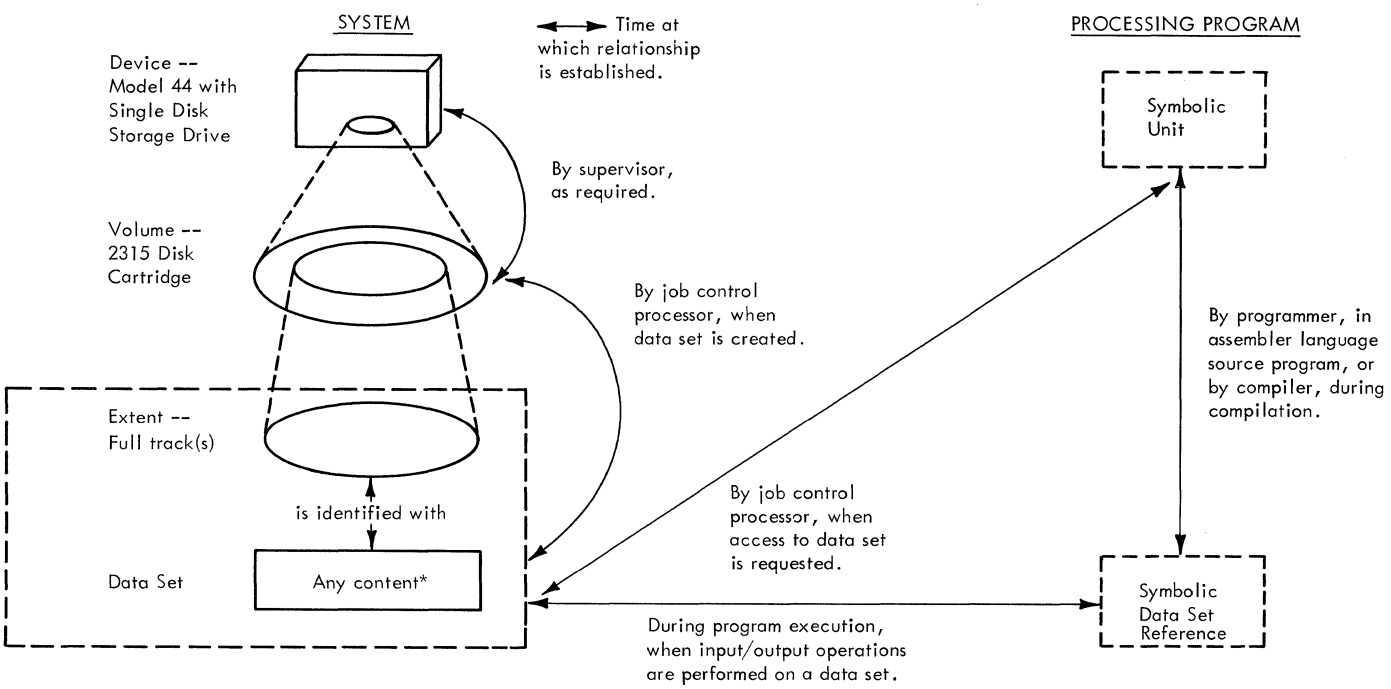
•Table 1. Symbolic Unit Assignments

Unit Name	Data Set Name[1]	Function	Permissible Devices[2]
SYSAB1	SDSABS[3]	Phase library; used by program fetch and load routines.	Disk
SYSAB2	SDSABS[3]	Phase library; used by linkage editor.	
SYSREL	SDSREL	Module library; used by linkage editor.	Disk
SYSLOG	SDSLOG	Operator-system communication.	Console Printer-Keyboard
SYSRDR	SDSRDR	Job control processor input.	Card Reader Magnetic Tape
SYSIPT	SDSIPT	Processing program input; may be same as SYSRDR.	Card Reader Magnetic Tape
SYSLST	SDSLST	System and diagnostic print output and all job control statements.	Printer Magnetic Tape
SYSOPT	SDSOPT	Processor and user print output; may be same as SYSLST.	Printer Magnetic Tape
SYSPCH	SDSPCH	Punch output.	Punch Magnetic Tape
SYSUAS	SDSUAS[3]	Job control unit assignment spill.	Disk
SYSPSD	SDSPSD	Pseudo-directory to data set on SYS000: output from FORTRAN compiler, assembler; input to linkage editor.	Disk
SYSDMY		General system use.	
	SDSCAT[3]	Catalog data set.	Disk
	SDSIPL[3]	Initializing routine for IPL procedure.	Disk
SYS000	SDS000	Output from FORTRAN compiler, assembler; input to linkage editor.	Disk Magnetic Tape
SYS001	SDS001	General system work data sets; may be used by any processing program.	Disk Magnetic Tape
SYS002 SYS003		Not used by system except that certain assignments are necessary for the utility processor and the assembler when using the update feature.	Any device
SYS004 . . . SYS240		Not used by system. The availability of all or some of these units above SYS005 depends upon the installation.	

[1] The names SDSABS and SDSUAS are used by the system and, therefore, cannot be changed; the other system data set names are installation-assigned. The names listed here are used throughout the Model 44 publications to refer to these data sets.

[2] Except for data sets on the system residence disk, these data sets may be on devices other than those listed in this column. In most cases, however, some loss in efficiency results.

[3] Must be on system residence volume (a 2315 Disk Cartridge).



*If a data set is directoried, the symbolic unit assignment can be to a specific member rather than to the entire data set, and the member is treated as a data set for the duration of the assignment.

Figure 2. Data Management Relationships

Use of the read/write level gives the assembler language programmer device independence. He can specify his input/output requirements without indicating a particular device type or device address. This enables him to concentrate on the main elements of his program and also eliminates the need to revise programs when data sets are moved to different devices.

Device independence is made possible by the system's use of control blocks for each active symbolic unit and data set. When a program calls for an input/output operation, the system examines the current control blocks for the unit and data set to determine which input/output routines must be used.

The system's execute channel program (EXCP) level is for programmers working with nonsupported devices and for those who wish to replace or supplement some of the system's routines with their own.

Data Format

All blocks within a direct access data set must be of the same length, although block lengths may vary among different data sets. Tape data sets may contain variable length blocks. The programmer is responsible for all blocking and unblocking of logical records within a block.

Channel Overlap

Input/output operations normally are overlapped with the main program's processing. The system makes maximum use of this overlap capability whenever possible. A programmer can, however, suspend processing until an input/output operation has been completed and he has made sure the transmission was successful. In any case, the system always examines the last operation that used the control block before starting another operation for that same control block. This examination is made each time the program calls for an input/output transmission. All unusual conditions are indicated to the program.

Control Blocks

All communication between the problem program and data sets at the read/write level is made through a request control block. This block is provided by the user and maintained by the system throughout the processing of one input/output request. When the program calls for a read/write operation, the system examines the request control block provided by the user for information about the request and data set involved. It uses this information to determine the status of the data set, which device is involved, and which device-dependent routines are needed to satisfy the request. (This information is

maintained by the system in control blocks provided by the system.)

The request control block contains an input/output block that is the basic structure for communication between the read/write level and the channel scheduler. The channel scheduler is a set of routines that keep track of the requests for use of devices on each channel and actually initiate the input/output operations.

Request control blocks are allocated by the user in main storage and partially filled out by the user; the system maintains the control information in the block during program execution.

Detailed information on constructing and using request control blocks is available in the publication IBM System/360 Model 44 Programming System: Guide to System Use, Form C28-6812.

Execute Channel Program (EXCP)

The execute channel program (EXCP) level of the system enables a programmer to work with devices not supported by the system, provided they transmit acceptable channel end, device end, and control unit end signals. The EXCP level also enables a programmer to replace or supplement some system routines with his own.

The EXCP level schedules input/output requests, starts command execution, directs interruption handling, and restarts channel activity, when necessary. An EXCP level programmer must be familiar with the system's control blocks and must provide a channel program and an interruption analysis routine for each request he issues.

Interruption analysis routines are device-dependent routines that maintain the status of an operation, examine the results of an interruption for errors or unusual conditions, and initiate any necessary error recovery procedures.

The EXCP level programmer must construct an input/output block for the system to use in order to communicate with the channel scheduler. He also must be familiar with the unit control block for each device he uses. This block reflects the current status of the device.

The system's EXCP scheduler examines input/output requests to ensure that they are properly constructed. An invalid request causes abnormal termination of the operation. The scheduler also checks to see whether there is room for the request in the channel queue and whether the channel and device are available. If the queue is full, the request is held until there is room for it, and control then returns to the calling program. The request is executed as soon as the device and channel are free and previous requests for the same facilities have been satisfied.

DUMP FACILITIES

The system provides two types of dump facilities:

1. The programmer can specify in his job control statements that, in the event of an abnormal termination (e.g., one that results from a program check interruption), the system is to produce a hexadecimal dump before proceeding to the next job.

   The console operator can also request this dump when he terminates a job with a cancel command (see "Operator-System Communication").

2. The subroutines DUMP and PDUMP, which reside in the module library, can be linkage-edited into the program and called by source program statements. These subroutines provide for dumping programmer-specified areas of main storage with the following format options: hexadecimal, integer, real, logical, complex, and literal.

   The DUMP routine causes the job to terminate after the dump is taken; the PDUMP routine returns control to the calling program.

SOURCE LANGUAGE INPUT

Input to the FORTRAN compiler may be in either Extended Binary-Coded-Decimal Interchange Code (EBCDIC) or Binary-Coded-Decimal Interchange Code (BCDIC).

Input to the rest of the system (i.e., the assembler language statements, job control statements, linkage editor control statements, utility control statements) must be in EBCDIC.

COMPATIBILITY

The Model 44 Programming System represents a selected subset of the features available in the IBM System/360 programming support systems designed for the Models 30, 40, 50, 65, and 75 -- specifically, System/360 Operating System (OS/360), System/360 Disk Operating System (DOS/360), System/360 Tape Operating System (TOS/360), and System/360 Basic Operating System

(BOS/360).[1] Thus, there are certain areas of compatibility and interchangeability among these systems, as described below.

Source Languages

Source programs written in the Model 44 FORTRAN language can, without modification, be compiled by the OS/360 FORTRAN IV compiler for execution under control of OS/360. This also applies to DOS/360, TOS/360, and BOS/360, provided that the source program observes the language level supported by those systems.

Source programs written in the Model 44 assembler language can be assembled, with minor modification, by the OS/360, DOS/360, or TOS/360 assembler. Modification includes any source statements involving subroutine linkages or supervisor functions and statements using machine instructions peculiar to the Model 44 or assembler instructions peculiar to the Model 44 assembler. The assembler language publication cited in the Preface includes a detailed list of the limits on the features of the Model 44 assembler language.

Data Sets and Volumes

Removable volumes (2400 Series Magnetic Tape Reels, 1316 Disk Packs) are interchangeable between the Model 44 using the Model 44 system and other System/360 models using OS/360, DOS/360, TOS/360, or BOS/360, subject to the following conditions:

All data sets to be read by the Model 44 system must be organized sequentially, and contained in single extents on single volumes. They may not contain any checkpoint records. If they are on a direct access device, they must be of fixed block length.

Data sets produced at the read/write level of the Model 44 system can be read and updated at the highest level of OS/360, DOS/360, TOS/360, or BOS/360 if the user maintains their standard fixed-length (type F) or variable-length (type V) logical record formats within the blocks.

A volume created by OS/360 can be read, updated, and written by the Model 44 system.

A volume created by DOS/360, TOS/360, or BOS/360 can be read and updated by the Model 44 system, but no additional data sets may be written on it except where the format 5 label is up to date. These systems create a format 5 label (i.e., reserve space for it) at the time of volume initialization, but do not maintain it. A Model 44 utility function, map, is available for updating the format 5 label.

Label formats 2 and 3 do not apply to the Model 44 system. If they are present on a volume they will be ignored except by the map function.

The 2315 Disk Cartridge volumes produced under the Model 44 programming system are not compatible with those produced under the programming systems for the IBM 1800 and 1130 systems, and vice versa.

---

[1] Publications detailing these systems are listed in the IBM System/360 Bibliography, Form A22-6822.

The supervisor is the portion of the system that controls system operation; it also performs functions required in common by the various processing programs, including user-written programs.

The routines that collectively make up the supervisor fall into two categories, resident and transient, defined by their use of main storage. The resident routines are always present in main storage and represent functions that frequently are used or that simply must be present for the system to operate at all. The functions provided by the resident supervisor include:

Communication region
Interruption handling
Program fetch and program load
Channel scheduler
Input/output routines
Resident input/output error recovery
Timer services

The transient routines share a predefined area of main storage. They are loaded only when their functions are needed and overlay each other in the transient area. The major functions of the transient supervisor include:

Input/output error recovery
Operator communications
Open, close functions

COMMUNICATION REGION

The communication region is an area within the resident supervisor main storage wherein the job control processor stores parameters to direct the system in processing the next job step. It provides for communication between the supervisor, and a processing program, and among processing programs. Included in this area are option parameters, parameters describing the limits of main storage, the job name, etc.

The job control processor is described in the section "System Support Programs."

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.

There are five kinds of interruptions, as follows:

1. Supervisor call
2. External
3. Program check
4. Machine check
5. Input/output

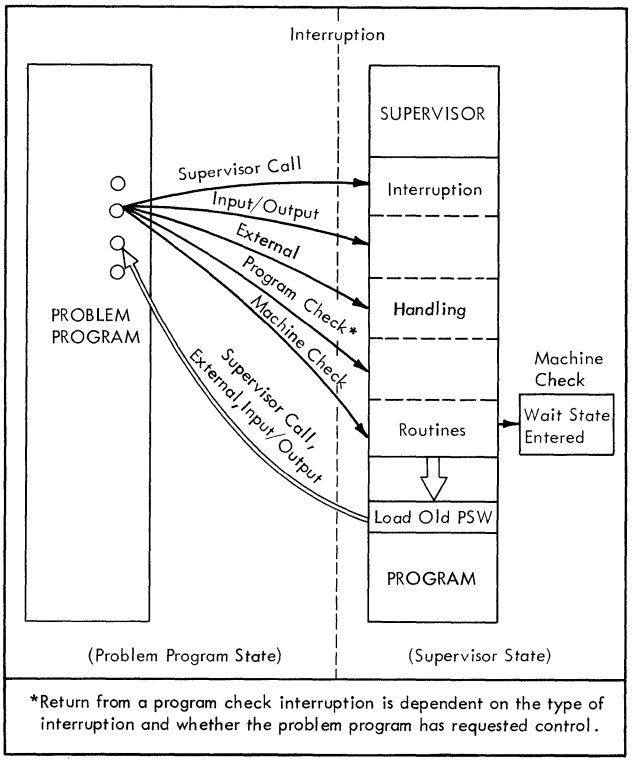


Figure 3. Flow of Control Between Supervisor and Problem Program During an Interruption

Figure 3 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, the status of the program is saved in the old Program Status Word

(PSW), and control is transferred to the supervisor. Depending on the type and reason for the interruption, control is given to an appropriate handling routine. Upon completion of the routine, the program may be restored to its original condition (via an old PSW). Control normally is given back to the problem program at the point where it was interrupted. If desired, the user may have control of certain types of program check and external interruptions.

SUPERVISOR CALL INTERRUPTION

The supervisor call interruption is caused when the SVC instruction is executed. The SVC instruction provides communication between the problem program and the supervisor. Each SVC has a certain interruption code that indicates to the supervisor which interruption handling routine is to be executed.

The interruption routine analyzes the code and transfers control to another routine within the supervisor, such as the program fetch routine, for the actual handling of the interruption.

EXTERNAL INTERRUPTION

An external interruption can be caused by the timer feature, or by the operator pressing the console interrupt key, or by an external signal. Interrupt-key and external-signal interruptions are not supported by the Model 44 system; control returns immediately to the point of interruption.

If a timer interruption occurs, control is given back to the interrupted program unless the user has provided an address of his own handling routine. When this is the case, control is transferred to the address specified.

The Timer Feature

The timer feature enables the supervisor to:

1. Maintain the time of day, to which the user can refer at any point within the execution of the problem program.

2. Time-stamp the beginning and end of a job. This information can be used for accounting and can be incorporated in a job log.

3. Set the timer for a specified interval of time and to transfer control to a prespecified user's routine after the time interval has elapsed, provided this occurs during execution of the job step that set the timer.

If the presence of the timer feature is not specified when the system is edited, all timer interruptions are ignored and control is returned immediately to the interrupted program.

PROGRAM CHECK INTERRUPTION

A program check interruption is caused by unusual conditions encountered in the program (e.g., overflow). A programmer can select one of the following options to be taken in the event of a program check interruption. The third option, however, is not available for all types of program check interruptions.

1. Cancel: The job being executed is terminated and a message to the operator describes the cause of the termination.

2. Dump and cancel: In addition to a message, all registers, all program status words, and the problem program area are printed in the system listing data set. The job is then terminated.

3. Transfer to user routine: If the address of a subroutine is supplied by the user, the program-check interruption routine will branch to that subroutine when an appropriate interruption occurs. The user routine can determine the cause of the interruption and handle it accordingly.

MACHINE CHECK INTERRUPTION

A machine check interruption results from a machine malfunction. When such an interruption occurs, the supervisor attempts to list, on the Console Printer-Keyboard, information regarding the status of the system when the interruption occurred. This information is used as an aid in diagnosing the source of the error.

The machine check old program status word and the failing instruction are listed. The information listed after this is in one of five formats, depending on the error type and condition of the system, as follows:

1. Parity check: The contents of the general registers, the numbers of registers that are out of parity, and the location and contents of any storage locations that are out of parity. Register 15 is an exception; if it is out of parity, the condition is indicated only if the error is permanent.

2. External machine check: The channel-unit addresses and the Channel Command Words of the last eight input/output devices started.

3. Channel check on initial selection: Same as external machine check, plus the contents of the general registers, the Channel Address Word, and the Channel Command Word of the selected device.

4. Parity and channel check: The combined information specified in items 1, 2, and 3, above.

5. Control check only: The contents of all general registers.

The system then can be restarted only through an initial program loading procedure.

INPUT/OUTPUT INTERRUPTIONS

An input/output interruption can be caused by:

1. End of input/output data transmission (channel end). The channel has completed sending to a device the information needed to carry out an input/output operation. The channel is now available for another operation.

2. Input/output attention. This results from pressing the request key of the Console Printer-Keyboard.

3. Device available (device end). A device that was busy or not ready is now available for use.

4. Control unit available (control unit end). A control unit that was busy is now available for use.

When one of these conditions is detected, control transfers to the channel scheduler.

CHANNEL SCHEDULER

The channel scheduler is a supervisor routine that keeps track of channel activity and initiates the actual input/output transmissions.

The channel scheduler examines the status of the requested facilities and schedules the request. If facilities are not available, the request will be queued. When they are free, the channel scheduler examines the type of request in the queue and the order in which the requests were received and schedules operations to make the most efficient use of all available channel resources.

The channel scheduler is device independent. It must know what device is requested and must keep track of whether that device is busy or available. It is concerned with the mode of operation of the device, whether burst, byte, or overrunable, but it is not concerned with the type of device. The channel scheduler examines the appropriate control block to determine whether a device-dependent initialization routine must be entered. The initialization routine sets up the command or chains of command needed to satisfy the request, if it has not been constructed previously. When its work is finished, it returns the address of the command list to the channel scheduler, which starts the transmission.

When there is a channel end interruption without device end, the interruption handler transfers control to the channel scheduler, which examines the queue for requests for other devices. If such a request is pending, the operation is started. When the interruption handler transfers control to the channel scheduler after a device end interruption, the channel scheduler passes control to a device-dependent interruption analysis routine. This routine examines data sent from the device and the channel to determine whether the transmission was successful. An error recovery procedure is entered when an error or unusual condition is detected. Then, unless execution of additional commands is required to satisfy the request, control returns to the interrupted program. If further command execution is needed, control returns to the channel scheduler. No other operation for the same device is initiated until this additional sequence is completed.

When an error cannot be recovered, the condition is noted for the problem program. Further action is up to the programmer.

Note that execute channel program (EXCP) level programmers must supply their own interruption handling routines. The system handles these functions for the read/write programmer.

INPUT/OUTPUT FUNCTIONS

This section contains a list of the input/output functions available to an assembler-language programmer; they are invoked by assembler-language calling sequences. A brief description of the function's purpose accompanies each entry. Note that in the function descriptions the term "data set" is used to mean both an

entire data set and a member of a directoried data set.

The first group, resident input/output functions, includes the functions most commonly used in a program, such as read and write. The supervisor routines required to execute these functions always are present in main storage.

The second group, transient functions, includes the open and close operations. The routines for these functions are less frequently used in a program and are called into main storage only when actually needed. If a program uses more than one data set, time can be saved by grouping open functions together and close functions together so that these transient routines do not have to be recalled frequently.

RESIDENT INPUT/OUTPUT FUNCTIONS

The Read Function: The read function is used to transmit data from a data set to an area of main storage. Its parameters include the name of a request control block set up earlier by the program. Before starting any new operation, the system examines this block to determine whether there was an error or unusual condition (EOF, EOV) in the last previous operation if any, using the same block. If there is any unusual condition, an indication is returned to the program and the read is not started.

The Write Function: The write function is used to transmit data from an area of main storage to a data set. The system makes the same examinations and takes the same action in cases of error as it does for the read function.

The Check Function: The check function is used to determine that an input/output operation has been completed successfully before processing resumes. This function resets the request control blocks and allows issuing of a new input/output request even if the checked operation was abnormally terminated.

The Note Function: The note function is used to determine the current position, in terms of a block number, of a symbolic unit relative to its beginning. The note function generally is used in conjunction with the point function (see below). These two functions enable the programmer to process data sets in a nonsequential manner.

The Point Function: The point function is used to reposition a symbolic unit to a specified data block. The programmer specifies the desired block by supplying its block number relative to the beginning of the data set (or member). Thus, an argument of zero results in the symbolic unit being positioned at the beginning of the first data block.

The Write-End-of-File (WEF) Function: The WEF function is used to write an end-of-file mark.

The Rewind Function: The rewind function is used to rewind a magnetic tape volume to its load point or, on a direct access volume, to position a symbolic unit to the first block of a data set.

The Unload Function: The unload function is used to rewind and unload a magnetic tape volume or, on a direct access volume, to position a symbolic unit to its first block.

The above read/write-level functions make use of the following resident functions, which can be called directly by the EXCP-level programmer:

The Execute Channel Program (EXCP) Function: The EXCP function is used by programmers at the execute channel program level to initiate an input/output operation. All control blocks and interruption analysis routines must be constructed before this function is invoked.

The Wait Function: The wait function is used to suspend all processing until an input/output operation is completed.

TRANSIENT INPUT/OUTPUT FUNCTIONS

The Open Function: The open function causes the system to validate tape labels, if any, and to reposition symbolic units, if necessary. All data sets should be opened, regardless of whether the programmer is operating at the read/write level or the EXCP level.

The Close Function: The close function, when applied to a data set on magnetic tape, instructs the system to write any necessary end-of-file marks and trailer labels. This function also is used to indicate the disposition of a data set or volume (e.g., whether a magnetic tape reel should be unloaded, rewound, or left as it is).

INPUT/OUTPUT ERROR RECOVERY

Examinations for input/output errors and unusual conditions are made at three points for programs at the read/write level.

The first check is made when a program calls for an input/output operation. The

program's request is examined, and the operation is cancelled if errors are found.

Another check is made immediately after the system initiates the physical operation. The system makes certain the command has been accepted by the channel and device and that the operation has started properly. Action is terminated if this examination reveals program errors or a non-operational device. In some cases, such as when the device has not finished execution of a previous command, control returns to the interrupted or calling program until the device is free and the system can reissue the command.

The final check is made immediately after completion of the physical operation. At this time, the system examines data provided by the channel and the device. If the possibility of error or an unusual condition is indicated, the system normally issues another command that causes the device to provide additional, more detailed information.

Interruption Analysis Routines

The examinations for unusual conditions at the completion of physical operations are made by the system's interruption analysis routines. When no errors are detected, these routines mainly act as record keepers, noting such data as the number of blocks read or written, and preparing the system for the next input/output operation for the data set.

Interruption analysis and error recovery procedures differ according to the type of device. In some cases (e.g., a reader check), when an error condition can be removed by operator intervention, a message is written on the Console Printer-Keyboard. Normally, the system repeats the input/output operation, a process that eliminates the most common errors. No other operation is initiated for the device while the error recovery procedure is in progress. If the error continues to appear and is classified as permanent, the system updates the count of permanent errors in the unit control block for the device. Unusual conditions are indicated to the program.

Programmers at the EXCP level must handle their own interuption analysis and error recovery routines. The same channel and device data that the system uses is made available to these programmers in the input/output control block and the unit control block. Detailed information on this data and the conditions to be checked can be found in the publication IBM System/360 Principles of Operation, Form A22-6821, and appropriate device manuals.

PROGRAM FETCH AND PROGRAM LOAD

Any program that is to be executed under system control must reside in absolute form in the phase library. This includes the supervisor transient routines, the language processors, the system support programs, and all user-written programs. Within the library, programs are stored as phases. Each phase represents a segment of code that is to be loaded into main storage at one time. The phase may be an entire program, or a part of a multiphase program.

The program fetch function loads a phase into main storage from the phase library and transfers control to the phase entry point, an absolute address specified at the time the phase was processed by the linkage editor. A fetch operation may be invoked automatically by the supervisor (e.g., when it is necessary to load a transient routine) or by the job control processor when it processes an EXEC (execute) statement, or explicitly by processing programs that are multiphase.

The program load function also loads phases into main storage from the phase library, but it does not transfer control to the entry point. Instead, it returns control to the invoking program. The phase load address may be one of the parameters supplied when the load function is invoked, thus permitting the programmer to relocate a phase at load time. The load operation can be invoked explicitly by processing programs. This function allows the programmer to load phases of nonexecutable code (e.g., tables).

The organization of programs into multiphase structures is discussed in "Linkage Editor."

OPERATOR-SYSTEM COMMUNICATION

The supervisor provides for two-way communication between the operator and the system, via the Console Printer-Keyboard. Three types of communication are permitted:

1. Messages to the operator.
2. Operator response to requests for operator action.
3. Operator-initiated commands to the system.

MESSAGES TO THE OPERATOR

Messages to the operator can be initiated by the system or by the programmer's job (via the job control PAUSE and comments statements).

The messages may indicate that an operator action (e.g., mounting a volume) or decision is required, or they may simply contain information (e.g., for a job log). In the first case, the system will suspend processing until the operator responds with an indication that the action or decision is complete. In the second case, processing continues without interruption.

OPERATOR COMMANDS

The operator may enter a command in any of the following instances:

1. He has pressed the request key.
2. The system has requested operator response and processing has been suspended.
3. The job has requested operator response with a PAUSE statement and processing has been suspended.

There are three types of operator commands: intervention, input/output, and information. Except for the intervention commands, the system will accept only commands that are entered between jobs.

Intervention Commands

The intervention commands, which are accepted at any time:

1. Immediately cancel the job currently being executed and proceed to the next job. The operator may request a dump in the cancellation command and the system will produce a hexadecimal dump before proceeding to the next job.
2. Cause the job control processor to pause at the end of the job currently being executed. This allows the issuing of other operator commands at a time when they can be accepted.

Input/Output Commands

The input/output commands provide the following functions:

1. Assign a symbolic unit to a physical input/output device.
2. Indicate that a device is temporarily unavailable to the system.
3. Indicate that a device has been restored to availability status.
4. Indicate the addition of a device to the machine configuration.
5. Indicate the deletion of a device from the machine configuration.

The fourth and fifth input/output commands permit temporary alteration of the machine configuration (until the next IPL). They may be issued only as part of an IPL procedure (see below).

Information Commands

The information commands provide the following functions:

1. List the input/output assignments currently in effect.
2. Initialize the date and time of day.
3. Indicate the end of operator commands so that the system can proceed to the next job step.

INITIAL PROGRAM LOADING (IPL) PROCEDURE

Operation of the Model 44 system is initiated through an initial program loading (IPL) procedure. An IPL procedure is required whenever it is necessary to load or reload the system. It consists of three steps.

During the first step, the operator mounts the system residence volume on the appropriate disk storage drive, sets the address of this unit on the console load address switches, and presses the Load key.

During the second step, the operator may issue the input/output commands to modify the list of available input/output devices. He then issues a command to declare the date and time of day. This is the only point in the IPL procedure at which these commands may be issued. These commands are effective only for the duration of one IPL procedure; they must be reissued if they are to be in effect during the next iteration of the procedure.

The standard system units are defined and the corresponding volumes are mounted during the third step. The system then begins job processing.

There are three processors, resident in the phase library, that are classified as system support programs. Two of these, the job control processor and the linkage editor, are integral to the operation of the system. The third processor is a collection of programs that provide utility functions under control of the system.

JOB CONTROL PROCESSOR

The job control processor is loaded (originally during the initial program loading procedure and, thereafter, by the program fetch routine) between successive job steps. It then proceeds to read and interpret job control language statements from the system reader unit.

The principal function of the job control language is to describe the job step to be done -- the program to be executed and the information required to do so. From the information in the job control statements, the processor sets up, within the supervisor, the input/output tables, program parameters, communication words, etc., that define the requirements for this step.

When all of the job control statements for this step have been processed, the processor returns control to the supervisor. The supervisor then fetches, from the phase library, the processing program associated with the step and loads it over the job control processor. Actual control of the step during execution is a function of the supervisor. At the end of the job step, the supervisor again loads the job control processor.

The job control language is also used to call special functions of the job control processor that are not associated with a particular job step (e.g., condensing a directoried data set).

JOB CONTROL LANGUAGE

The statements in the job control language fall into four categories, as follows:

1. Job Definition -- These statements define the program execution requirements (i.e., which programs and in what sequence) for the job.

2. Symbolic Unit Assignment -- These statements describe the data set and device resources the job will need during execution.

3. Data Set Maintenance -- These statements specify some action to be taken on a data set.

4. Miscellaneous Functions -- These statements specify comments, operator action pause, or repositioning a volume.

The job control statements are grouped by category and summarized briefly in Table 2. The double slash (//), the slash followed by an ampersand (/&) or an asterisk (/*), or the asterisk alone (*) identify these statements as job control statements.

Job Definition Statements

The JOB Statement: The JOB statement is required for each job and must be the first card of the job deck. It supplies accounting information to the installation accounting routine and indicates the start of the job control information for this job. It is also used to specify that a dump is to be taken in the event that the job fails in execution.

The EXEC Statement: The EXEC (execute) statement is required for each job step and must be the last job control statement before execution of a processing program. It indicates the end of control card information for a job step and that execution of the processing program for this step is to begin. The processing program named in an EXEC statement must reside in absolute form in the phase library.

If the program name field of the EXEC statement is blank, the preceding job step must have been an execution of the linkage editor. The system will then execute the first program phase produced from that linkage editor job step.

Table 2. Job Control Summary

STATEMENT	FUNCTION
JOB DEFINITION	
// JOB	Defines the start of a job.
// EXEC	Defines the start of a job step execution and the program to be executed.
/&	Delimits the end of a job.
// STOP	Delimits the end of all jobs.
/*	Delimits the end of a data set in the input stream.
SYMBOLIC UNIT ASSIGNMENT	
// ALLOC	Allocates space for a new data set.
// LABEL	Defines characteristics of a data set.
// ACCESS	Permits access to an existing data set.
// RESET	Restores unit assignments to status at start of job.
// LISTIO	Lists data set and device assignments on system log.
DATA SET MAINTENANCE	
// DELETE	Deletes a data set from a volume or a member from a directoried data set.
// CONDENSE	Condenses a directoried data set.
// RENAME	Renames a data set or a member of a directoried data set.
// CATLG	Enters a data set name into the catalog.
// UNCATLG	Removes a data set name from the catalog.
MISCELLANEOUS	
// PAUSE	Allows pause for operator action.
* (comments)	Allows logging of comments to system log.
// REWIND	Rewinds a tape volume; repositions a data set on a direct access volume to beginning.
// UNLOAD	Rewinds and unloads a tape.

The EXEC statement is used to request that the operator set the variable-precision switch. The EXEC statement also supplies any option parameters that are to be passed to the processing program. These include compiler and assembler options or options to be used by user-written programs. One of the options the user may specify in the EXEC statement for a compilation or assembly step is that the relocatable module produced by this step is to be linkage-edited later in this same job. The system will then direct the relocatable module to a data set where it can be immediately retrieved by the linkage editor.

The EXEC statement may also include accounting information particular to this job step.

The End-of-Job Statement: The end-of-job (/&) statement is required for each job and must be the last card of the job deck. It initiates the post-job housekeeping that restores any system variables that were altered during the job. This includes resetting the system units, performing any cataloging operations that were requested during the job, and reinitializing system records.

The STOP Statement: The STOP (end-of-jobs) statement is used to denote the end of a job input stream. It closes all data sets associated with system units.

The End-of-Data Statement: The end-of-data (/*) statement is required immediately following any input data on the system input unit when this unit is the same volume as the system reader unit.

This statement is not actually read by the job control processor; instead, it must be recognized as an end-of-file mark by whatever program is currently reading the system input data set.

Symbolic Unit Assignment Statements

The ALLOC Statement: The ALLOC (allocate) statement is used to create a data set; that is, to allocate space for it and name it. Within this statement the programmer can, if he desires:

1. Give the data set a name of up to eight characters.
2. Assign a symbolic unit name to the data set.
3. Specify residence on a particular volume or type of volume.
4. Specify that the volume be "fresh," i.e., no other data set is currently resident on it.
5. Indicate that the data set is to be directoried, and the desired size of the directory; the space will be allocated accordingly. The ACCESS statement (see below) is used to create

individual members within the data set.

6. Specify that the space allocated be rounded upward to the next higher cylinder boundary (rounding to the next higher track boundary is automatic).

7. Request that the data set be cataloged.

From the information in the ALLOC statement, the system locates the desired volume, allocates the space and, if the volume is a direct access one, updates the VTOC to reflect the creation of this data set.

The ALLOC statement is required only when the programmer wishes to create a new data set. Once it has been created, subsequent jobs can gain access to it using the data set name if it has been cataloged or the data set name and the volume identification if it has not been cataloged.

If the data set has been assigned to a symbolic unit name, this assignment remains in force until occurrence of one of the following:

1. The assignment is redefined within the job (i.e., by an ALLOC or ACCESS statement).

2. An applicable RESET is encountered (i.e., the unit named was one of the predefined system units).

3. The end of job is reached.

The ACCESS Statement: The ACCESS statement is used to gain access to a data set that has been created by an ALLOC statement either previously in this job or in a prior job. This statement is also used to add a new member, assigning multiple names if desired, or refer to an existing member of a directoried data set.

If the data set named in the ACCESS statement has been cataloged, then the name alone is sufficient to locate it. Otherwise, the volume identification must be supplied in addition so that the system can locate the data set.

Normally, when an existing data set is requested in an ACCESS statement and subsequently opened, the data set is positioned at its beginning. If, however, the programmer wishes to add to the data set, he can indicate this in his ACCESS statement and the data set will be positioned at the end of the last entry made in the data set.

Another ACCESS statement option allows the programmer to make references to a non-existent data set so his program can operate without actually performing input/output operations on a data set. When this option is used, input/output requests for this data set are handled as follows:

1. A read request results in an end of file.

2. A write request is recognized but no data is transmitted.

This option is useful not only in testing and debugging, but also for bypassing references to data sets used in regular procedures. For example, a new job that updates an existing master may use a dummy master until the first detail data set is processed and the first master is produced.

The ACCESS statement is required whenever a job step within the job:

1. Uses a data set that was allocated in a previous job.

2. Uses a data set that was allocated in this job, but the ALLOC statement did not give a symbolic unit assignment.

3. Requires reassignment of a symbolic unit.

4. Creates or refers to a member of a directoried data set.

5. Requires a dummy symbolic unit assignment for program testing.

The LABEL Statement: The LABEL statement is used to further define a data set's characteristics, such as block length, organization, expiration date, use of control characters, etc. From the information supplied in this statement, the system writes labels, fills out control blocks, or both, depending on the circumstances of its use.

When used, the LABEL statement follows immediately after the ALLOC or ACCESS statement identifying the data set with which it is associated. All data sets on disk must be labeled.

The RESET Statement: The RESET statement is used to restore either a selected one of the system unit assignments or all of them to their status at the beginning of the job. There is an implied RESET at the end of every job.

The LISTIO Statement: The LISTIO (list input/output) statement causes the printing on the system log of one or more symbolic unit names and their current data set and device assignments.

Data Set Maintenance Statements

The DELETE Statement: The DELETE statement can be applied to an entire data set or to a single member of a directoried data set. In the first case, the statement causes deletion of the VTOC entry for the named data set and the updating of the volume's format 5 label. If the data set is cataloged, its entry is also deleted from the catalog. In the second case, the data set member name is simply removed from the directory. If this member has multiple names in the directory, it is still accessible via the other name(s).

The CONDENSE Statement: The CONDENSE statement causes a directoried data set to be condensed. The condense function is described in the section "Data Organization."

The CATLG Statement: The CATLG (catalog) statement causes a data set entry to be added to the catalog. Since a cataloging option is also available in the ALLOC statement, this statement is necessary only when a data set has been created previously and it is now desired to catalog it.

The UNCATLG Statement: The UNCATLG (uncatalog) statement causes a data set entry to be deleted from the catalog. The data set itself and the volume on which it resides are unchanged.

The RENAME Statement: The RENAME statement can be used to rename either an entire data set or a member of a directoried data set. In the first case, the change is made in the VTOC; the catalog is also checked and, if an entry for the data set occurs in the catalog, the name is changed there as well. In the case of a member, the name is changed in the directory. No change is made to any other names for the same member that might occur in the directory.

Miscellaneous Statements

The PAUSE Statement: The PAUSE statement permits the programmer to specify a pause in processing so that the console operator can take some action. The statement, which can include the programmer's instructions to the operator, is printed on the Console Printer-Keyboard and the machine then waits for the operator's signal to proceed.

The Comments Statement: The comments (*) statement gives the programmer a convenient means of writing messages to the operator. The contents of the comments statement will be printed on the Console Printer-Keyboard. This statement, when followed by a PAUSE statement, can also be used to specify operator action in the event that the programmer's directions do not fit within the limits of the PAUSE statement.

The REWIND Statement: The REWIND statement is used either to rewind a tape volume or, on a direct access volume, to position a data set to its first block (block number 0).

The UNLOAD Statement: The UNLOAD statement is used to rewind and unload a specified reel of magnetic tape.

SAMPLE DECK SETUP

Figure 4 illustrates a sample deck setup using the ALLOC and ACCESS statements for data set creation and symbolic unit assignments. Note that the statements shown in Figure 4 are not complete; only those parameters that are meaningful to the example have been included. The parenthetical remarks in the comments column indicate assumptions that have been made for the purposes of this example.

LINKAGE EDITOR

Output from the compiler or the assembler is always in the form of relocatable object program modules. Each module consists of an external symbol dictionary, the text of the module (i.e., the instructions, in a relocatable format), and a relocation dictionary. The text consists of one or more control sections, as specified in the assembler or FORTRAN source language program. A control section is defined as a unit of text that can be independently relocated.

These modules must be processed by the linkage editor before they can be executed. The principal function of the linkage editor is to convert the relocatable text into absolute form, ready to be loaded and executed. The linkage editor includes facilities for linking together several control sections from one or more relocatable modules into one absolute executable program. The process of "linking" involves determining the absolute load addresses for each of the control sections and, from information in the external symbol dictionaries associated with the several modules, replacing symbolic cross-references between control sections with the absolute address of the referenced value.

The facilities provided by the linkage editor permit the programmer to construct programs that are multiphase; this topic is discussed in "Program Structures."

Statement			Comments
//JOB1	JOB		Beginning of job JOB1.
//SYSOPT	ALLOC	output data set,special unit	Give special assignment to system output unit.
//	EXEC	assembler (link)	Assemble following source deck.
assembler language source deck			(SYSRDR and SYSIPT are the same.)
/*			End of source deck.
//	RESET	output unit	Reset SYSOPT to standard assignment.
//	EXEC	linkage editor	Edit program just assembled.
linkage editor control statements			Defines phase structure for program.
/*			End of linkage editor control statements.
//	ALLOC	data set 1, disk volume 10	Data set 1 to be created on disk volume 10.
//	LABEL	label information	Label information for data set 1.
//SYS001	ACCESS	data set 1(member 1a), new	Set up to write member 1a of data set 1; unit assigned is SYS001.
//SYS002	ACCESS	data set 2, tape volume 20	Set up to use data set 2 (allocated to tape volume 20 in a previous job); unit assigned is SYS002.
//	CATLG	data set 2	Catalog data set 2.
//	EXEC		Execute the edited program.
/&			End of job JOB1.
//JOB2	JOB		Beginning of job JOB2.
//SYS001	ACCESS	data set 1 (member 1a),disk volume 10	Set up to use member 1a (written in JOB1); unit assigned is SYS001.
//SYS002	ACCESS	data set 2	Set up to use data set 2 (cataloged in JOB1); unit assigned is SYS002.
//SYS003	ALLOC	data set 3,tape volume 21	Data set 3 to be created on tape volume 21.
//	LABEL	label information	Label information for data set 3.
//	ALLOC	data set 4,disk volume 11	Data set 4 to be created on disk volume 11 (will be written by program bb).
//	LABEL	label information	Label information for data set 4.
//	EXEC	program aa	Execute program aa from phase library.
//SYS001	ACCESS	data set 5	Reassign SYS001 and set up to use data set 5 (cataloged in a previous job).
//SYS003	ACCESS	data set 4	Assign SYS003 to data set 4.
//	PAUSE	SAVE TAPE VOL 21	Operator action pause.
//	EXEC	program bb	Execute program bb from phase library.
//	UNCATLG	data set 53	Remove entry for data set 53 from catalog.
/&			End of job.
//	STOP		End of jobs.

•Figure 4. Sample Deck Setup

LINKAGE EDITOR PROCESSING

The programmer may specify at the execution of one or more compilation or assembly steps that the linkage editor will be called later in the job to edit the relocatable modules produced by the compilation or assembly step(s). The modules will then be stored in a data set from which the linkage editor can immediately retrieve them. In addition, the programmer may specify, through linkage editor control statements, that other modules are to be included as well. Or he can set up a job without compilation or assembly steps and specify, again through linkage editor control statements, that certain relocatable modules, produced in previous jobs, are to be edited together.

Input to the linkage editor consists of linkage editor control statements and relo-

catable object program modules. Generally speaking, the primary sources of linkage editor input are:

1. The system unit that serves as intermediate storage for the relocatable modules as they are produced by the compiler or assembler. That is, this is the source of modules that were produced in job steps within this same job.

2. The system input unit that contains the linkage editor control statements and any input modules produced in previous jobs. Module input from this source is first written on the system unit described in item 1 above. Thereafter, it is treated exactly as the other modules already on that unit.

3. The module library, which usually contains frequently used system and installation subroutines, such as the FORTRAN mathematical subroutines. The programming system incorporates module library routines into a program automatically, when required, or a programmer can specifically name routines that are to be included.

The output from the linkage editor is directed to the phase library. (A phase may be punched out as an absolute deck using the utility processor punch function.[1]) The programmer can indicate in his linkage editor EXEC card that residence in the phase library is to be either temporary (until after execution of the next EXEC statement, i.e., until the end of the next job step) or permanent (the program is available for execution at any time until explicitly deleted from the library).

PROGRAM STRUCTURES

As previously discussed, the input to the linkage editor is in units of modules. Output from the linkage editor is in units of phases. A phase is that portion of an absolute program that is to be loaded by a single program fetch or load operation. The phase may be an entire program or a part of a program.

The simplest case is a single-phase program. However, the linkage editor permits the programmer to set up his program with an overlay structure wherein each phase is a part of the program that may be combined with or loaded over other phases during execution of the program. That is, after a phase is loaded and executed, the next phase may be loaded into the same area of main storage, overlaying the previous phase. Each phase has a programmer-specified origin and the phases are executed in a programmer-specified sequence. Thus, the programmer has control over which parts of the program are to be overlaid and when.

The FORTRAN programmer who wishes to set up a multiphase structure for his program can use the FORTRAN CALL statement to call a special subroutine, LINK, which resides in the module library. This subroutine, which will then automatically be incorporated into his program by the linkage editor, contains the instructions necessary to invoke the fetch or load operation for loading subsequent phases.

Figure 5 illustrates the use of main storage by an overlay program. The loading sequence shown in Figure 5 is ROOTPH, A1, B1, B2 (overlaying B1), and A2 (overlaying A1 and B2). Although this illustration shows a root phase (ROOTPH) as resident in main storage throughout execution, there is no requirement for a root phase. Programs may be structured into phases all of which originate at SYSORG.

LINKAGE EDITOR CONTROL STATEMENTS

Following the EXEC statement specifying the linkage editor, the programmer uses linkage editor control statements to specify which control sections are to be included, the phase structure, and the origin of each phase. The sequence of execution of the phases is determined by the fetch or load requests within his program. There are four of these statements, as follows:

MODULE — Indicates that the sequence of cards or card images immediately following this statement on the system input unit constitutes a relocatable module intended for inclusion in the linkage editor input.

Any use of MODULE statements and their associated modules must precede any other linkage editor control statements in this job step.

PHASE — Defines a phase by providing the linkage editor with a phase name (the member name to be entered in the directory of the phase library) and the origin of the phase.

---

[1]All entries in the phase library must be made by the linkage editor. Therefore, to enter an absolute deck into the phase library for execution under system control, the deck must be reprocessed by the linkage editor as if it were a relocatable module.

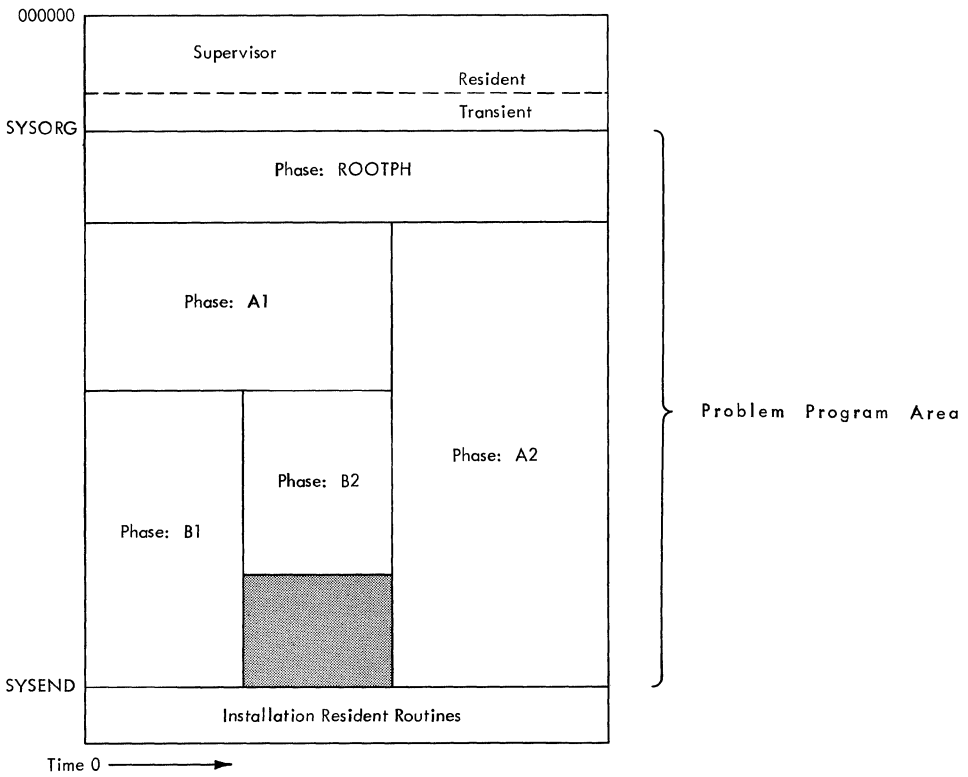


Figure 5. Example of Use of Main Storage by an Overlay Program

An option in this statement can be used to suppress, for this phase, the automatic linking to modules in the module library.

INCLUDE — Identifies a particular module, or control section(s) within a module, for inclusion in a phase.

ENTRY — Defines the phase entry point and indicates the end of linkage editor input on the system input unit. If ENTRY is omitted, the entry point is assumed to be the first entry name encountered in a module END card in the phase or, if none exists, the first location of the phase; also, the /* end-of-data statement must be used to denote the end of linkage editor input.

UTILITY PROGRAMS

The utility programs fall into two categories: (1) volume initialization and maintenance and (2) data set transmission. These programs are called by an EXEC statement specifying the utility processor, followed by a statement specifying the desired utility function.

VOLUME UTILITIES

There are three volume utility functions:

Initialize: Initializes a direct access volume or a tape volume, and creates the standard labels.

If the volume being initialized is direct access, this function performs an analysis of the recording surface, checking for defective tracks. If a defective track is found, it is flagged, an alternate track is assigned, and the system thereafter uses the alternate track in place of the defective one.

If a volume should develop a defective track after initialization, a message is printed to the operator giving the number of the defective track; the volume can then be partially initialized to accomplish assignment of an alternate track. The contents of the defective track are lost.

Map: Examines the VTOC of a direct access volume and, if necessary, updates the format 5 label. No data set is deleted or moved, but a map of the volume is produced with the expired data sets flagged. This utility should be executed before using any

disk volume created under another System/360 Programming System.

Squeeze: Condenses a 2315 disk cartridge. This condensing operation is similar to that described for directoried data sets (see "Data Organization"). However, the squeeze function does not condense members of any directoried data sets that might be on that volume. The VTOC is updated but is not moved; the format 5 label is also updated. Expired data sets are eliminated.

DATA SET TRANSMISSION UTILITIES

There are five data set transmission utility functions:

Copy: Reads a data set from a symbolic unit and writes it into a data set on another symbolic unit. Either the source or the destination may be a member or members of a directoried data set. Data sets may be reblocked by using the copy function.

Print: Reads a data set or one or more members from a symbolic unit and writes it on the system printer unit.

Punch: Reads a data set or one or more members from a symbolic unit and writes it on the system punch unit.

Print-punch: Reads a data set or one or more members from a symbolic unit and writes it on both the system printer unit and the system punch unit.

Punch absolute: Reads a phase from the phase library and writes it on the system punch unit.

Model 44 Programming System publications provide installation personnel with information requisite to the construction, use, modification, and maintenance of the system. This appendix is intended as a guide to the selection of the appropriate publications for each application. It contains:

1. A diagram, Figure 6, showing the available publications grouped according to the requirements of a particular audience and suggesting a sequence of use within each group.
2. A brief description of each publication.
3. A topic index, to direct the user to the proper publication for information about a particular subject (e.g., constructing a multiphase program).

In the short descriptions and in the diagram, abbreviated titles are used. Complete titles of the System Reference Library (SRL) publications are:

IBM System/360 Model 44 Programming System: Concepts and Facilities, Form C28-6810

IBM System/360, FORTRAN IV Language, Form C28-6515

IBM System/360 Model 44 Programming System: Guide to System Use for FORTRAN Programmers, Form C28-6813

IBM System/360: FORTRAN IV Library Subroutines, Form C28-6596

IBM System/360 Model 44 Programming System: Assembler Language, Form C28-6811

IBM System/360 Model 44 Programming System: Guide to System Use, Form C28-6812

IBM System/360 Model 44 Programming System: Systems Programmer's Guide, Form C28-6814

IBM System/360 Model 44 Programming System: Operator's Guide, Form C28-6815

IBM System/360 Model 44 Programming System: Formats for Machine Check Interruption Diagnostics (Reference Card), Form X28-6812

Program Logic Manuals (PLM) describe the internal design of programs and programming systems and are restricted in distribution to those responsible for program maintenance and to systems programmers who are responsible for making program modifications.

The titles of the PLMs supplied in support of the Model 44 Programming System are as follows:

IBM System/360 Model 44 Programming System: Supervisor and Job Control, Form Y28-6812

IBM System/360 Model 44 Programming System: Linkage Editor, Form Y28-6813

IBM System/360 Model 44 Programming System: Utilities and Stand-alone Programs, Form Y28-6814

IBM System/360 Model 44 Programming System: Assembler, Form Y28-6811

IBM System/360 Model 44 Programming System: FORTRAN IV Compiler, Form Y28-6815

Related machine manuals, which are not described here, are:

IBM System/360: System Summary, Form A22-6810

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

IBM System/360: Model 44, Functional Characteristics, Form A22-6875

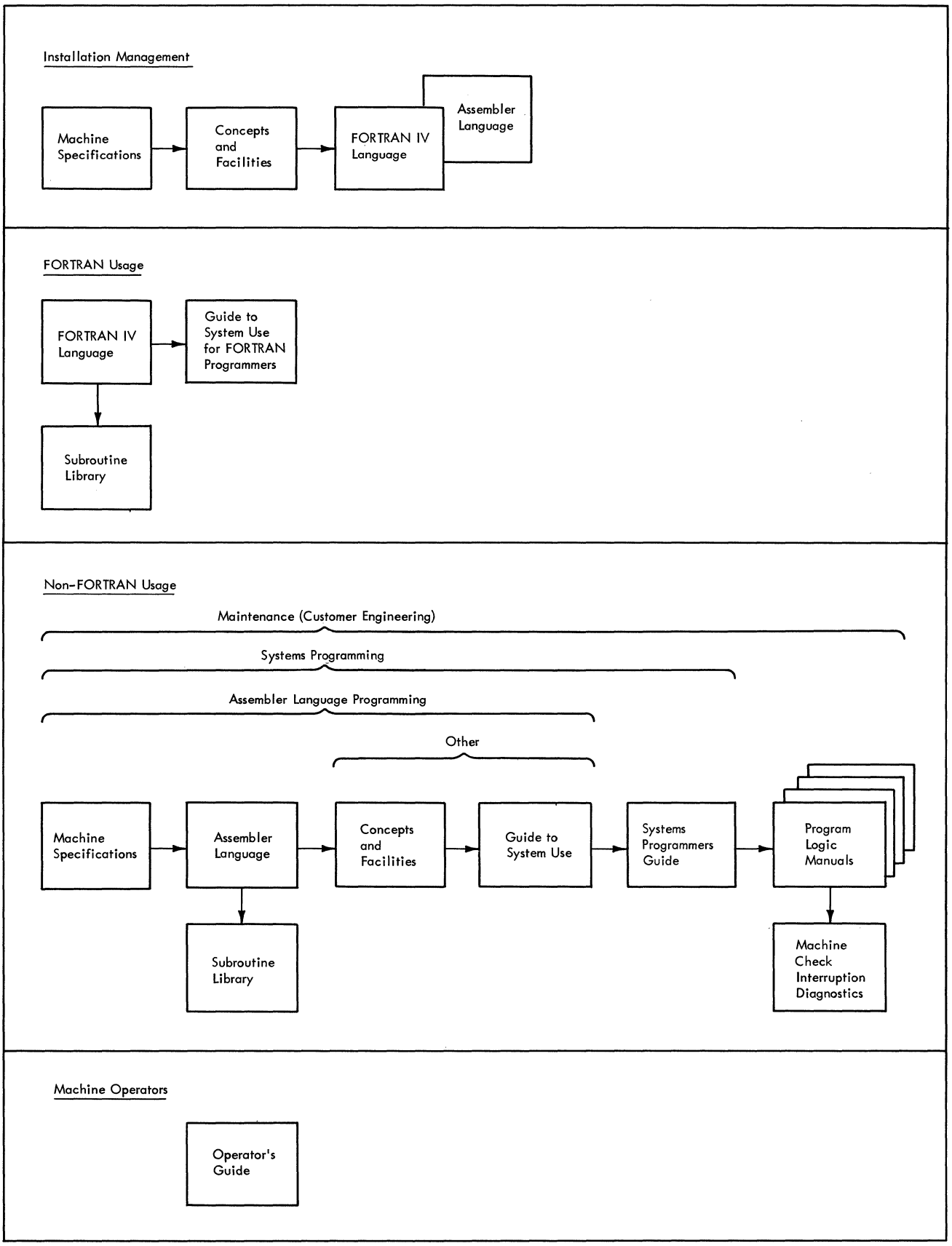


Figure 6. Model 44 Programming System Publications Plan

Concepts and Facilities describes the system in terms of basic concepts and operation of the supervisor and system support programs. It provides an understanding of the system's capabilities and a basis for planning their use. It is assumed that the user has a familiarity with basic data processing techniques and terminology, and with the functional characteristics of the Model 44.

FORTRAN IV Language is a reference manual that provides the applications programmer with information required for the preparation of source programs using the full capabilities of the FORTRAN IV language. A knowledge of the fundamentals of FORTRAN language programming is necessary.

Guide to System Use for FORTRAN Programmers instructs the applications programmer in the use of the system to compile, link edit, and execute FORTRAN programs. Discussions of program optimization and restrictions of the Model 44 FORTRAN IV compiler are included. This publication is intended to assist the new programmer in becoming familiar with the facilities available to him (e.g., job control, linkage editing, linking to assembler language subroutines, and constructing multiphase programs), and to serve the experienced programmer as a reference manual. It is restricted to FORTRAN applications; refer to the Guide to System Use for non-FORTRAN usage.

FORTRAN IV Library Subroutines describes the function and use of the mathematical and service subroutines in the module library. The calling sequences, argument formats, and main storage requirements are discussed as are these additional topics pertaining to the mathematical subroutines: algorithms, accuracy, error propagation, and timing. A knowledge of FORTRAN or assembler language is prerequisite to the use of this publication.

Assembler Language is a reference manual that contains the specifications for using the assembler language and its features. It describes the syntax of the language, and explains the symbolic machine instruction codes and the assembler program functions provided for the programmer's use. A knowledge of IBM System/360 machine operations, primarily storage addressing, data formats, and machine instruction formats and functions, is prerequisite.

Guide to System Use contains all necessary information about system requirements and capabilities. It gives complete specifications for using the job control language, the linkage editor, the utility processor, the absolute loader, and stand-alone disk initialization programs. It also discusses SVC functions for input/output operations at the read/write level, interruption handling, program phase loading, and interprogram and intraprogram communications. All system diagnostics except those for FORTRAN are found in this publication. The applications and/or systems programmer should read Concepts and Facilities before using this publication. Additional prerequisites would depend upon the particular application.

Systems Programmer's Guide explains how to construct, modify, and maintain the system. It is intended not only for systems programmers and those responsible for the installation's system, but also for those who are programming input/output functions at the EXCP level. Among the topics discussed are system construction, system editing, incorporation of installation routines, installation accounting, IPL procedures, EXCP-level programming and support of additional device types, and incorporation of IBM modifications. This publication presupposes a knowledge of Model 44 assembler language programming, and the facilities and use of the programming system.

Operator's Guide supplies machine operators with detailed information about operating the system. This publication, which presumes that the user has a knowledge of Model 44 machine operations, covers IPL procedures, general operations, abnormal end procedures, operator-to-system communications, and stand-alone operations.

Formats for Machine Check Interruption Diagnostics supplies customer engineers with the basic formats in which the system will list the diagnostic information when a machine check interruption occurs.

Program Logic Manuals are reference manuals that aid the user in analyzing the internal functions of a system component for purposes of correcting a program malfunction or making a program modification. They describe the flow of the program with both flowcharts and narrative, and include descriptions of internal tables, program interfaces, intermediate data formats, and other analysis aids. They are intended to be used with a symbolic listing of the component under discussion. Refer to the list of PLM titles for the subject matter of a particular Model 44 programming system PLM.

TOPIC INDEX

This index is intended to assist readers of the Model 44 publications in locating topic discussions relevant to their particular needs. The Program Logic Manuals are not included; their titles are descriptive of their content.

The key to abbreviations for titles throughout the index is as follows:

Systems Programmers Guide	SPG
Guide to System Use	GSU
Guide to System Use for FORTRAN Programmers	FORTRAN GSU
Assembler Language	Assembler
Operator's Guide	OG
FORTRAN IV Subroutines	FORTRAN Library
FORTRAN IV Language	FORTRAN
Formats for Machine Check Interruption Diagnostics	MCID

Note that some items are indicated as being "covered in both GSU and FORTRAN GSU." This means that, in order to reduce cross-references between publications, full information appears in both. The user should select a particular publication applicable to the needs of his installation.

for the format of machine check interruption diagnostic information, see MCID.

Directoried data sets
covered in both GSU and FORTRAN GSU.

Disk initialization
under system control, see GSU.

for stand-alone disk initialization, see GSU.

Dump facilities
use of DUMP and PDUMP subroutines and abnormal end dumps, covered in both GSU and FORTRAN GSU.

for console operator requested dumps, see OG.

Error recovery
at EXCP level, see SPG.

at read/write level, see GSU.

(not applicable to FORTRAN users.)

Error messages
for all system messages to programmers (including assembler) with the exception of FORTRAN compiler diagnostic messages, see GSU.

for all system messages directly related to FORTRAN applications, including the compiler diagnostic messages, see FORTRAN GSU.

for the format of machine check interruption diagnostic information, see MCID.

for system-to-operator messages, see OG.

External-storage assignment
covered in both GSU and FORTRAN GSU.

EXCP level programming
for input/output functions and support of additional device types, see SPG.

FORTRAN
language specifications for, see FORTRAN.

for compiling, linkage editing (including multiphase programs), and execution, see FORTRAN GSU.

for full descriptions of mathematical and service subroutines, see FORTRAN Library.

Initial Program Load (IPL)
for operating procedures, see OG.

for special considerations of sytem construction, see SPG.

Input/Output
for assembler language programs at the read/write level, see GSU; at the execute channel program level, see SPG.

for FORTRAN programs, see FORTRAN GSU.

Interruption handling
for problem program handling of timer interruptions and certain types of program check interruptions, see GSU.

(not applicable to FORTRAN users.)

Job control
for FORTRAN applications, see FORTRAN GSU.

for all other applications, see GSU.

Labels and label processing
for FORTRAN applications, see FORTRAN GSU.

for full formats and descriptions of all standard labels, see GSU.

Libraries
for use of phase and module libraries, see both GSU and FORTRAN GSU.

for system construction allocation, see SPG.

Linkage editor
covered in both GSU and FORTRAN GSU.

for use of LINK subroutine to construct multiphase FORTRAN programs, see FORTRAN GSU.

Machine Check Interruption
see MCID.

Machine configuration
covered in this publication.

Maps
see GSU and FORTRAN GSU.

Mathematical subroutines
covered in FORTRAN Library

Messages
for all system messages to programmers (including assembler) with the exception of FORTRAN compiler diagnostic messages, see GSU.

for all system messages directly related to FORTRAN applications, including the compiler diagnostic messages, see FORTRAN GSU.

absolute form: A form of program text wherein the instructions have a predetermined load address and all symbolic address references have been replaced with machine address values.

absolute loader: A stand-alone program that loads decks in absolute form for execution independent of system control.

allocate: To reserve external storage space for a data set.

block (records):
1. To group records for the purpose of conserving storage space or increasing the efficiency of access or processing.
2. A physical record so constituted.

catalog:
1. The data set containing the names and volume identifications of selected data sets; used by the system to locate data sets specified by name only.
2. To include in the catalog the name and volume identification of a data set.

cataloged data set: A data set that is represented in the catalog.

communication region: A control block within the resident supervisor that provides for communication between the supervisor and a processing program, and among processing programs.

condense: For directoried data sets, to shift the members and, separately, their directory entries, maintaining their original sequence, to fill space vacated through the deletion of other members. After condensing, members occupy contiguous locations. Similar to the squeeze function for direct access volumes.

control block: A storage area through which a particular type of information required for control of the system is communicated among its parts.

control section: The smallest separately relocatable unit of a program; that portion of text specified by the programmer to be an entity, all elements of which are to be loaded into contiguous main storage locations.

data management: A general term that collectively describes those functions of the system that provide creation of and access to data sets, enforce data storage conventions, and regulate the use of input/output devices.

data organization: A term that refers to the data management conventions for the arrangement of a data set, i.e., sequential and directoried.

data set: The major unit of data storage and retrieval in the system, consisting of a collection of data in a prescribed arrangement and described by control information to which the system has access.

data set label: A collection of information that describes the attributes of a data set and that is normally stored with the data set.

device independence: The ability to request input/output operations without regard to the characteristics of the input/output devices.

directoried data set: A data set in direct access storage that is organized so that the first part contains an index (directory) to the members following.

directory: The initial portion of a directoried data set that indexes the subsequent members by name; it provides the means of gaining access to the members.

dump (main storage):
1. To copy the contents of all or part of main storage onto an output device, so that it can be examined.
2. The data resulting from action described in 1.
3. A routine that will accomplish the action described in 1.

entry point: Any location within a module to which control can be passed by another module.

extent: The physical locations on a volume occupied by or reserved for a particular data set.

fetch (program):
1. To obtain a requested phase, load it into main storage at the locations assigned by the linkage editor, and transfer control to the phase entry point.
2. A routine that accomplishes the action described in 1.

initial program loading (IPL): As applied to the system, the initialization procedure that loads the supervisor and the job control processor and begins normal operations.

installation: A general term for a particular computing system, in the context of the overall function it serves and the individuals who manage it, operate it, apply it to problems, service it, and use the results it produces.

job: An externally specified unit of work for the computing system from the standpoint of installation accounting and system control. A job consists of one or more job steps.

job control processor: The processing program that reads and interprets job control statements and sets up the system to execute a specific program using specific resources.

job control statement: Any one of the control statements in the input stream that identifies a job or defines its requirements.

job step: A unit of work for the computing system from the standpoint of the user, presented to the system by job control statements as a request for execution of a specific program and a description of the resources required by it.

library: A collection of objects associated with a particular use and having a directory to locate individual objects. In this context, see module library, phase library.

linkage: The means by which communication is effected between two routines or control sections.

linkage editor: A program that produces one or more program phases by transforming relocatable modules into a format that is acceptable to the program fetch routine, combining separately produced modules, replacing, deleting, and adding control sections as requested, and resolving symbolic cross-references among them.

load:
1. Generally, to read a phase into main storage.
2. Program load -- to read a phase into main storage, and return control to the invoking program.
3. The routine that accomplishes the above two steps.

main storage: All addressable storage from which instructions can be executed or from which data can be loaded directly into registers.

member: An entity within a directoried data set, indexed in the data set's directory and having data content.

module: The unit of output from a single execution of the assembler or compiler, in relocatable form and consisting of one or more control sections with control information to permit relocation and symbolic cross-references to other modules.

module library: A directoried data set containing selected modules and serving as an automatic source of input to the linkage editor.

multiphase program: A program in absolute form that requires more than one fetch or load operation to complete execution.

multiple names: In a directoried data set, more than one name entry in the directory referring to the same member.

name: A set of one or more characters that identifies a statement, data set, module, phase, etc., and that is usually associated with the location of that which it identifies.

operator command: A statement to the supervisor, issued via the Console Printer-Keyboard, that causes the supervisor to provide requested information, alter normal operations, terminate a job, etc.

phase: The unit of output of the linkage editor, in absolute form, that is loaded by a single program fetch or program load operation; may represent an entire program or part of a program.

phase library: The directoried data set that contains program phases, processed and entered by the linkage editor; the source from which program phases are loaded for execution.

problem program: Any of the class of routines that perform processing of the type for which a computing system is intended, and including routines that solve problems, perform computations, monitor and control industrial processes, etc.

processing program: A general term for any program other than the supervisor.

record: A general term for any unit of data that is distinct from all others when considered in a particular context.

relocatable form: A form of program text wherein the instructions have variable load addresses and symbolic cross-references, plus control information to permit later conversion to absolute form.

relocation: The modification of address constants required to compensate for a change of origin of a module or control section.

resource: Any facility of the computing system or operating system required by a job and including input/output devices, data sets, and processing programs.

sequential data set: A data set organized so that, given one record, the next record to be processed is uniquely determined.

squeeze: To eliminate expired data sets on a direct access volume and shift remaining data sets to lower numbered tracks to fill space vacated by expired and deleted data sets. After squeezing, data sets occupy contiguous locations. Similar to the condense function for members of directoried data sets.

stand-alone program: Any program that operates independently of system control; generally it is either self-loading or loaded by another stand-alone program.

supervisor: As applied to the Model 44 system, the routines executed in response to a requirement for altering or interrupting the flow of operations through the central processing unit, or for performance of input/output operations, and, therefore, the medium through which the use of system resources is coordinated and the flow of operations through the central processing unit is maintained.

symbolic data set: In coding a program, the designation used to refer to data; the actual data set whose data content is to be processed during a particular execution of the program is determined later. The later assignment may be an entire data set or a specific member of a directoried data set.

symbolic unit: In coding a program, the designation used to refer to external storage; the actual storage to be used during a particular execution of the program is determined later.

system data set: A data set that has a particular system use and/or content and a predefined relationship to a system unit.

system residence volume: The volume containing the phase library, the catalog, and the IPL routine job control work tables.

system support programs: Those processing programs that contribute directly to the use and control of the system and the production of results: the job control processor, the linkage editor, and the utility programs.

system unit: A symbolic unit that has a particular system use and a predefined relationship to a system data set.

text: The instructions or data content of a phase or of the control sections of a module, collectively.

throughput: A measure of system efficiency; the rate at which work can be handled by a computing system.

user: Anyone who requires the services of a computing system.

utility programs: A collection of programs (together, the utility processor) that perform volume initialization and maintenance and data set transmission functions.

volume: All of that portion of a single unit of storage media that is accessible at a unique channel and unit address.

volume identification: The installation's designation for a particular tape or direct access volume.

volume table of contents (VTOC): A table associated with a direct access volume that describes each data set on the volume.

Printed in U.S.A. C28-6810-1

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 COMMENTS

Title: IBM System/360 Model 44 Programming System Concepts and Facilities

Form: C28-6810-1

Your comments assist us in improving the usefulness of our publications; they are a major part of the input used for technical newsletters and revisions.

Please do not use this form for technical questions about the system; it only delays the response. Instead, direct your technical questions to your local IBM representative.

Corrections or clarifications needed:

Page	Comment

If you wish a reply, please include your name and address below:

fold fold

FIRST CLASS
PERMIT NO. 33504
NEW YORK, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM CORPORATION

1271 AVENUE OF THE AMERICAS
NEW YORK, N.Y. 10020

Attention: PUBLICATIONS

fold fold

Printed in U.S.A. C28-6810-1

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]