File No. 7040-36
Form C28-6318-5

IBM Systems Reference Library

IBM 7040/7044 Operating System (16/32K)

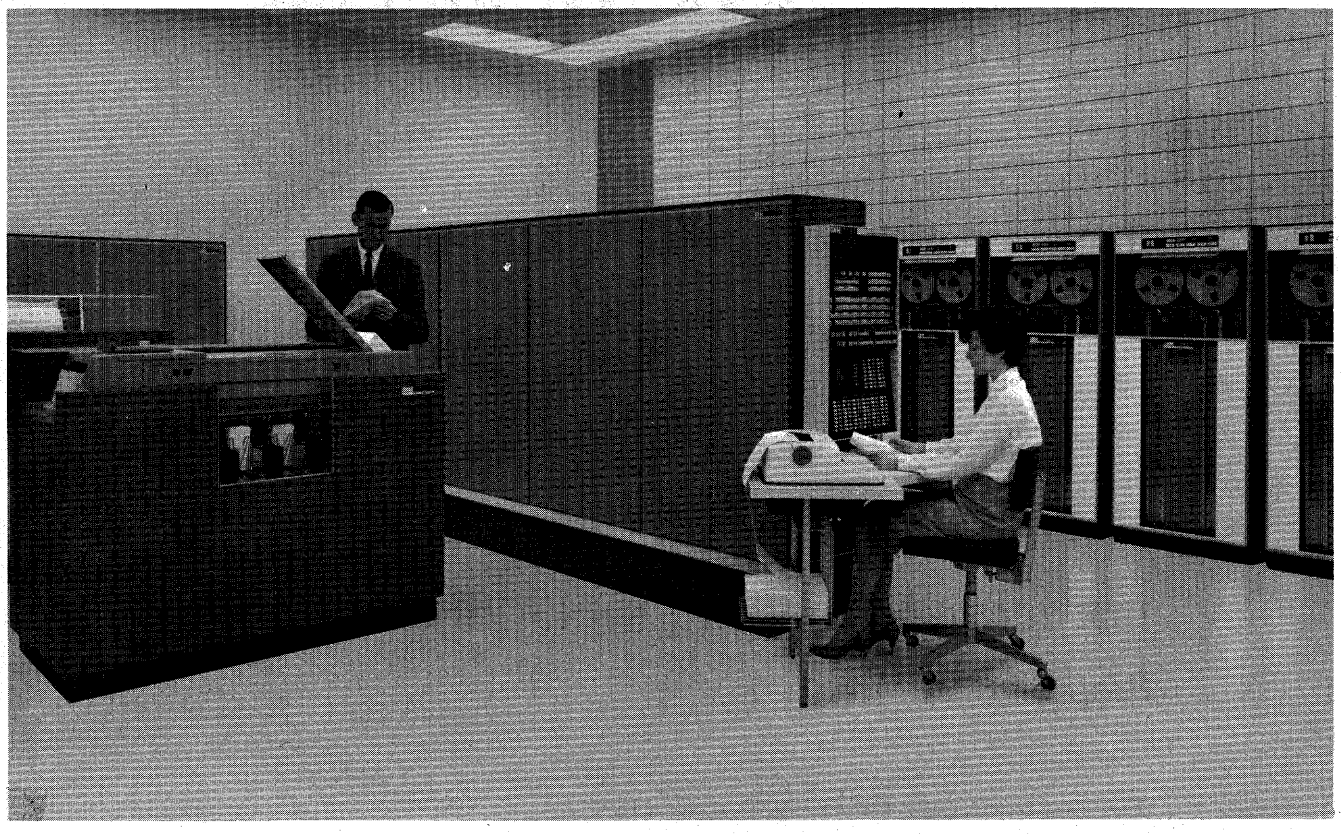
Programmer's Guide

The IBM 7040/7044 Operating System (16/32K) — 7040-PR-150 — is an integrated group of programs that permits continuous job processing on 7040/7044 Data Processing Systems with 16/32K capacity. This publication contains information the 7040/7044 programmer needs to run his jobs under this Operating System. It includes general descriptions of the following system components:

System Monitor	(#7040-SV-951)
Input/Output Control System	(#7040-IO-952)
Generalized Sorting System	(#7040-SM-953)
Monitored Utility Programs	(#7040-UT-975)
Processor	(#7040-PR-954)
Monitor	(#7040-SV-811)
Loader	(#7040-SV-812)
Library	(#7040-LM-813)
Macro Assembly Program	(#7040-SP-814)
FORTRAN IV Compiler	(#7040-FO-815)
COBOL Compiler	(#7040-CB-816)
Debugging Processor	(#7040-TA-817)
Update Program	(#7040-UT-955)

Descriptions of the Operating System control cards prepared by the programmer are provided.

Separate publications describe the MAP, FORTRAN IV, COBOL, and Debugging Languages, the Input/Output Control System, and the Generalized Sorting System. Other related publications contain instructions for the machine operator, information needed by the system programmer, and information on the contents of the Subroutine Library, which is a Processor component.



IBM 7040 Data Processing System

MAJOR REVISION (March 1965)

This publication, Form C28-6318-5, supersedes Form C28-6318-4 and associated Technical Newsletter N28-0518. This revision corresponds to Version 9 of the 7040/7044 Operating System (16/32K). Changes include the addition of the Debugging Processor as a component of the Processor and the use of 1302 Disk Storage as an input/output unit.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM Corporation, Programming Systems Publications, Dept. D39, 1271 Avenue of the Americas, New York, N. Y. 10020

7040/7044 Operating System Publications

The publications describing the 7040/7044 Operating System form an integrated set of publications that meet the needs of several types of readers. Figure 1 illustrates the sequence in which these publications should be read.

The Applications Programmer

For the applications programmer, the publication *IBM 7040/7044 Operating System (16/32K): Programmer's Guide,* Form C28-6318, is provided. It gives general descriptions of the 7040/7044 Operating System components and includes the instructions needed by applications programmers to run their programs under control of the operating system.

While the applications programmer can obtain operating information that he requires from this publication, the following 7040/7044 Operating System publications can provide him with more detailed information in his area of interest.

MAP PROGRAMMERS

IBM 7040/7044 Operating System (16/32K): Macro Assembly Program (MAP) Language, Form C28-6335

IBM 7040/7044 Operating System (16/32K): Input/Output Control System, Form C28-6309

FORTRAN PROGRAMMERS

IBM 7040/7044 Operating System (16/32K): FORTRAN IV Language, Form C28-6329

OTHER PROGRAMMERS

A programmer who needs to use the Generalized Sorting System can find the necessary instructions in the publication *IBM 7040/7044 Operating System (16/32K): Generalized Sorting System,* Form C28-6337. COBOL programmers should read the publication *IBM 7040/7044 Operating System (16/32K): COBOL Language,* Form C28-6336.

The Operator

For the operator, the publication *IBM 7040/7044 Operating System (16/32K): Operator's Guide,* Form C28-6338, is provided. Its contents include descriptions of the System Monitor control cards and detailed operating procedures for all the Operating System components.

The System Programmer

A system programmer is an experienced programmer assigned to place this system into operation, modify it according to the special requirements of his installation, maintain it, and ensure adequate control over its contents.

For the system programmer, the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339, is provided. It contains the editing instructions and detailed descriptive information of the Operating System components needed by programmers authorized to modify the operating system.

Programmer's Guide

This publication is the starting point for a study of the 7040/7044 Operating System. Included are descriptions of the features of the system, the functions performed by each system component, and programmer operating information, such as control card descriptions and examples of input card decks.

The first section introduces the reader to the operating system and describes its organization and operation. Later sections present the features of each of the system components and describe the control cards prepared by the programmer.

Special features of this guide include:

1. A table giving the format of all control cards used with the 7040/7044 Operating System, including a cross reference to a complete description of each.
2. Checklists of control cards required to process typical jobs.
3. A glossary of technical terms.

It has been assumed that the reader of this publication is familiar with the contents of the following publications:

IBM 7040/7044 Principles of Operation, Form A22-6649

IBM 7040/7044 System Summary, Form A28-6289

IBM 1301, Models 1 and 2, Disk Storage and IBM 1302, Models 1 and 2, Disk Storage with IBM 7040 and 7044 Data Processing Systems, Form A22-6768

IBM 7320 Drum Storage with 7040 and 7044 Systems, Form A22-6793

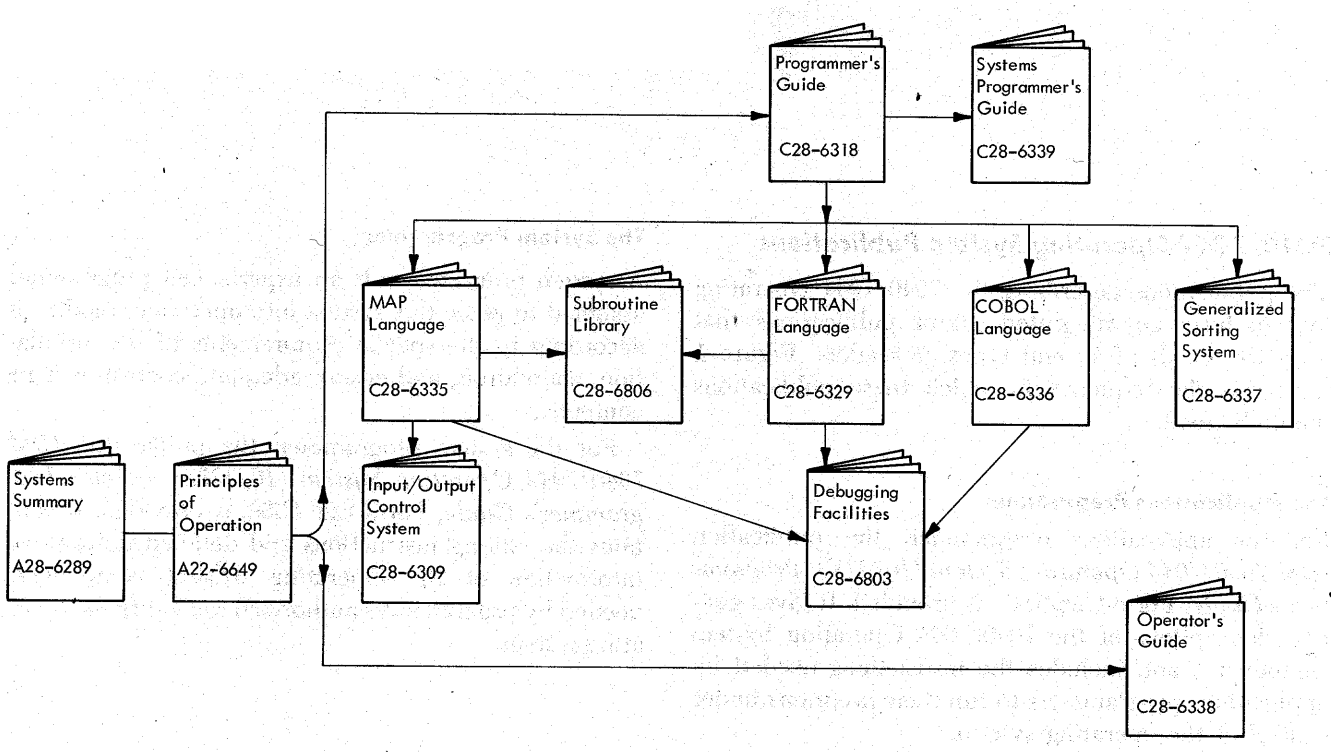


Figure 1. Reading Sequence for 7040/7044 Operating System Publications

Using the Programmer's Guide

A programmer who wishes to make full use of all the features of the Operating System would normally read the Programmer's Guide from cover to cover and then proceed to the appropriate manual that covers his specific area of interest. However, for the basic knowledge required to run a simple job, the MAP, FORTRAN, or COBOL programmer may read through to the section "The Nucleus," and then skip to the section "Input/Output Unit Assignment." When a programmer has familiarized himself with the control card formats, he may simply refer to the control card check list and the control card format index in the appendixes to this manual.

Machine Requirements

The following machine configuration is required for use of the 7040/7044 Operating System:

Processing System: An IBM 7040/7044 Data Processing System with the extended performance instruction set and with at least 16,384 locations of core storage. The single-precision floating-point instruction set is also required in order to use FORTRAN.

Input Unit: An IBM 1402 Card Read Punch with an IBM 1414-4 Input/Output Synchronizer that has the column binary feature, a magnetic tape unit, IBM 1301 Disk Storage, IBM 1302 Disk Storage, or IBM 7320 Drum Storage. An IBM 1622 Card Read Punch may be used if the input is entirely symbolic. If the 1622 Card Read

Punch is used, it must have the Expanded Character Set, Feature #3831, to obtain proper translation of IBM card code (H code) to BCD characters.

Punch Unit: An IBM 1402 Card Read Punch, a magnetic tape unit, IBM 1301 Disk Storage, IBM 1302 Disk Storage, or IBM 7320 Drum Storage. (This unit may be attached to the same device as the output unit for off-line processing.)

Output Unit: An IBM 1403 Printer, a magnetic tape unit, IBM 1301 Disk Storage, IBM 1302 Disk Storage, or IBM 7320 Drum Storage.

Library Unit: A magnetic tape unit, IBM 1301 Disk Storage, IBM 1302 Disk Storage, or IBM 7320 Drum Storage.

Utility Units: These units may be magnetic tape units, IBM 1301 Disk Storage, IBM 1302 Disk Storage, or IBM 7320 Drum Storage. Utility unit requirements are shown in Figure 2.

Checkpoint Unit: A magnetic tape unit, IBM 1301 Disk Storage, IBM 1302 Disk Storage, or IBM 7320 Drum Storage. If this unit is not provided, snapshots are not taken and core storage dumps are incomplete.

NOTE: The checkpoint unit may not be a magnetic tape unit attached through a 1401.

The console typewriter is used for operator messages.

Application	Required Utility Units
Compile or assemble only	3
COBOL Compilation	4
Load only	3
Compile or assemble, and Load	4
Compile, assemble, and Load (COBOL)	5
Load and go	3
Compile or assemble, Load, and go	4
Compile, assemble, Load, and go (COBOL)	5
Two-way merge	4*
Three-way merge	6*
Four-way merge	8*
Five-way merge	10*
Six-way merge	12*
Seven-way merge	14*
Eight-way merge	16*
System Editing (absolute modification cards only)	2
*Magnetic tape units and/or disk storage units (not attached to a 1401)	

Figure 2. Utility Unit Requirements

Contents

The 7040/7044 Operating System

The 7040/7044 Operating System is an integrated group of programs designed to permit continuous job processing on 7040/7044 Data Processing Systems with 16/32K capacity. As this operating system minimizes the need for operator intervention, the high speed of these computers can be applied to an uninterrupted series of jobs.

The Requirement

Machine time available at a computer installation is usually shared among many applications. Setting up the various job runs once involved numerous manual operations that took considerable time during which the computer was idle. The resulting loss of computing potential became proportionately larger as computer speeds increased.

To reduce manual procedures to a minimum and to eliminate the need for operator intervention between computer applications, monitored operating systems were developed. The monitor acts upon control information supplied by control cards, positions a library to the next program to be executed, loads it into core storage, and passes control to the program. Control is eventually returned to the operating system monitor, and the cycle is repeated.

This type of system permits the user to stack his job input decks on an input unit and process them through the computer in a continuous flow. In addition, it provides a convenient means for using data control programs, such as a sort or merge program.

The System

The 7040/7044 Operating System permits continuous machine processing of a stack of jobs. It includes a generalized sorting program and a language processor. This processor provides FORTRAN IV and COBOL compilers, an assembler, a relocatable program loader, and a library of subroutines. An operating system monitor includes a communications area and supervisory and service routines. User's programs and subroutines can be added to the system by means of the editing program, which also provides for system maintenance.

A flexible input/output control system permits the MAP programmer to specify on the $IBJOB card which sections of the input/output control system he needs to have in core storage at execution time. See the section "The Processor Monitor" for a description of the $IBJOB card. An auxiliary program, the 7040/7044-1401 Input/Output Control Program, is available to increase the input/output capacity of the system.

The use of all input/output devices is controlled through a flexible unit assignment scheme. Although some units are reserved for system use, others may be shared by the system and object programs. If the programmer wishes to retain any files, he must avoid assigning them to units that may be used by the system during his job. The programmer can designate units as intersystem units, reserving them for use from application to application within a job.

If an installation uses labeled files, the programmer can request that units be assigned as a result of a label searching procedure. This decreases the need for operator intervention during a job because files can be assigned to specific devices before processing is begun.

Symbolic input/output unit assignment frees the programmer from making absolute references to units and channels. The Input/Output Control System issues absolute instructions for input/output devices and keeps a record of the status of each device.

These features of the Operating System permit an installation to make efficient use of input/output devices for a series of applications. The programmer is also freed from having to determine which devices will be available when his program is executed.

Under normal conditions, it is not necessary for the computer to be idle; however, it may be idled for operator action. For example, facilities are provided for operator interruption for unusual, priority, or error conditions.

The most significant features that the 7040/7044 Operating System provides within one system are listed in Figure 3. The "Glossary" at the end of this publication gives definitions for many of the terms used in that figure.

COMPONENTS

The relationship of the following major components of the 7040/7044 Operating System is shown in Figure 4.

System Monitor (IBSYS): The System Monitor has three major functions. It provides, first, an area of core storage for communicating between subsystems; second, supervisory routines for processing control cards; and third, a variety of routines for use by system and object programs.

System Monitor

1. Reduces manual operations
2. Types operating instructions on the console typewriter
3. Permits scheduling of jobs by machine operators
4. Allows interruption of the processing of stacked jobs for a job given installation precedence
5. Provides rapid transition between applications
6. Controls job-to-job transition
7. Provides job accounting facilities
8. Standardizes control information on punched cards
9. Permits a flexible machine environment
10. Provides access to a language processor, a sorting program, an editing program, utility programs, and update facilities
11. Allows sorting, compiling, assembling, and executing within one job
12. Provides automatic recovery in case of job failure
13. Includes facilities for job skipping
14. Provides program checkout documentation, including a variety of storage prints
15. Pre-positions library devices
16. Includes checkpoint capability
17. Includes restart capability
18. Controls use of input/output devices
19. Provides symbolic input/output device reference
20. Provides input/output control routines
21. Provides standard error routines
22. Checks labels

Processor

1. Compiles and/or assembles source decks in any sequence
2. Compiles and assembles multilanguage programs
3. Facilitates use of FORTRAN and COBOL compilers and the Macro Assembly Program
4. Provides for source language debugging of relocatable segments
5. Includes a relocatable program loader
6. Provides access to and relocation of subroutines
7. Includes a library of system/user shared subroutines
8. Renames sections of coding for cross-referencing
9. Forms a multiphase program that exceeds a single storage load, if necessary
10. Provides a loader for absolute object programs
11. Provides for load-time debugging

Generalized Sorting Program

1. Sorts and/or merges records with the following characteristics:
   - Binary or BCD records
   - Fixed-length or variable-length records
   - Blocked or unblocked records
   - Records of up to 2,000 words
2. Sorts and/or merges on up to 32 control fields in the following manner:
   - Logically or algebraically
   - In ascending or descending order
   - According to commercial or scientific collating sequence
3. Merges previously sorted data with a current sort run
4. Inserts modification subroutines at specified exit points
5. Incorporates a complete merge program

System Editor

1. Provides routines to modify, add, delete, or duplicate the library of system programs and subroutines
2. Allows compact, efficient storage of all system programs
3. Allows patching of system programs
4. Permits source language editing of system programs

Monitored Utility Programs

1. Provide device-printing facilities
2. Provide utility routines for disk or drum storage

Update Facilities

1. Provide a means of creating and maintaining a file of symbolic or binary program decks on tape
2. Generate tape files from punched card decks
3. Provide for listing system output tapes

Figure 3. Features of the 7040/7044 Operating System

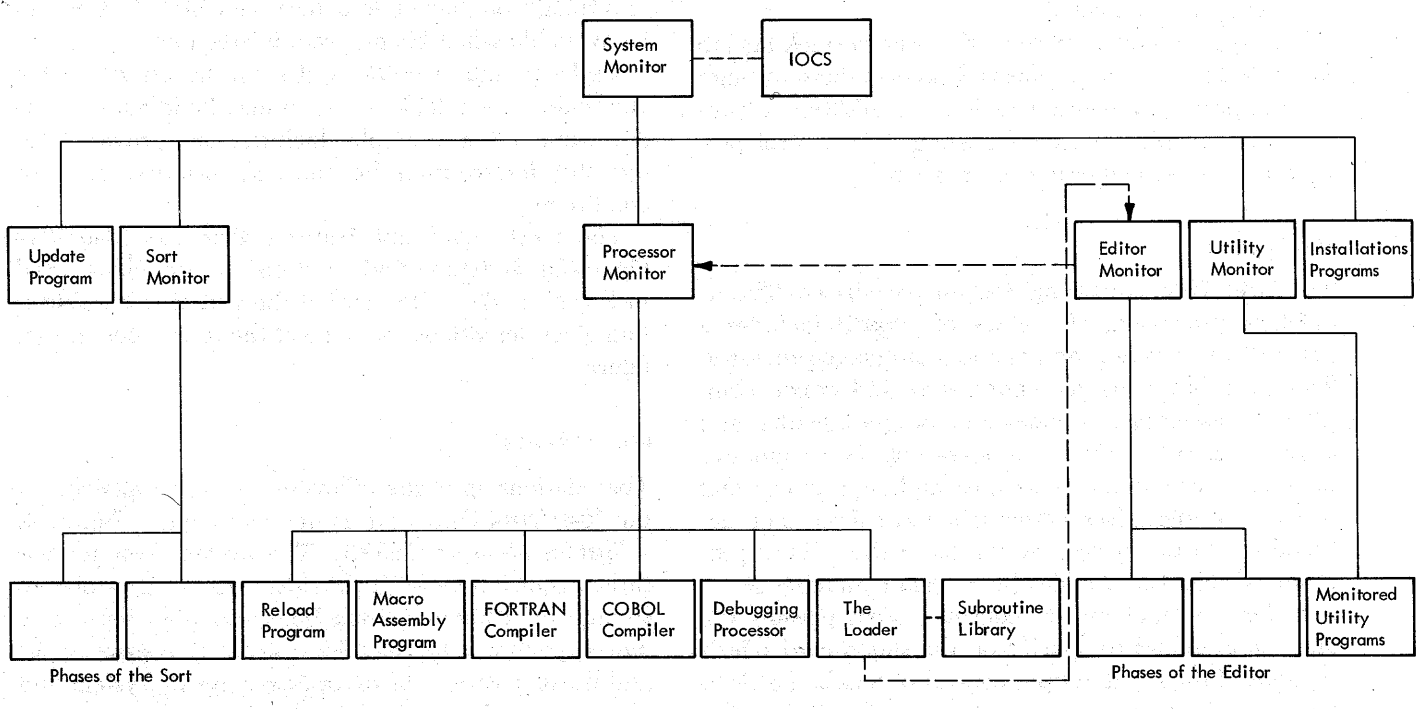


Figure 4. 7040/7044 Operating System Components

Processor (IBJOB): The Processor is a subsystem that facilitates use of the compilers and the assembler. It is composed of the Processor Monitor, the FORTRAN compiler, the COBOL compiler, the Macro Assembly Program, the Loader (IBLDR), the Subroutine Library, and the Debugging Processor.

Generalized Sorting System (IBSRT): The Generalized Sorting System is a subsystem that performs a wide variety of functions for efficient sorting and merging of data. It can process files of blocked and unblocked records of either fixed or variable length.

System Editor (IBEDT): The System Editor is a subsystem that provides facilities for maintenance of the operating system. These editing facilities are described in the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339.

Monitored Utility Programs: The Monitored Utility Programs perform device-printing functions and also various utility functions for disk or drum storage.

Update Facilities: The Update Facilities provide for maintenance of tape files containing symbolic or binary program decks. In addition to the primary update facilities, they have the capability of generating tape files from punched card decks and of listing system output tapes.

Installation Programs: Any programs, tables, data, etc., can be added to the operating system by an installation. The System Editor and facilities in the System Monitor, such as the $EXECUTE card and the System Loader described later in this text, permit the user to incorporate, call, load, and execute installation programs.

Definition of a Job

A job may consist of any sequence of applications of the operating system components. For example, a single job could include the compilation, assembly, and execution of an object program (a processor application), a sort and/or merge of the resulting data (a sort application), and subsequent compilations and executions (processor applications) for editing the output into a report format.

Using the Operating System

The System Monitor enables several subsystems to run within a coordinated environment supervised by a control program. Input to this system consists of a variety of jobs that may consist of one or more applications of one or more subsystems.

The 7040/7044 Operating System can be used for a variety of applications that fall into the following categories:

1. Language processing
   a. Compiling and/or assembling
   b. Loading only, for analysis of assigned storage
   c. Compiling (and/or assembly) and loading
   d. Loading and executing an object program
   e. Compiling (and/or assembly), loading, and executing a source program
2. Sorting or Merging
3. Executing installation programs

Control cards ensure an even flow of processing by specifying the subsystem required and by adapting the generalized subsystem to the specific needs of the user. This publication furnishes the information that the programmer needs to prepare the necessary control cards for his job.

Input

Control cards and other input to the system are stacked in a system input file. The input to the compilers, Sort, Editor, or object programs may be in the system input file or in any other file designated by the programmer. The system input file may be blocked or unblocked, but all $ control cards are always unblocked and in BCD form.

Input may consist of control cards, binary cards, BCD cards, data cards, symbolic cards, absolute cards, relocatable cards, or images of these cards on other storage media.

EXAMPLE OF AN INPUT DECK

A portion of a typical input deck containing several jobs is illustrated in Figure 5. The control cards in that figure are described in detail later in the text. For the purpose of this illustration, they can be briefly defined as follows:

$IBSYS	Returns control to System Monitor
$JOB	Delimits a job
$IBJOB	Designates a processor application
$IBSRT	Designates a sort or merge
$EXECUTE	Passes control to an installation program

PROCESSING THE INPUT DECK

The System Monitor is entered from the System Loader, which brings the supervisory routines into core storage. When the System Monitor encounters the $IBJOB card, it transfers control to the Processor Monitor, since this portion of the job is a processor application. The Processor Monitor controls the processor application according to control card specifications. The $IBSYS card indicates to the Processor Monitor that the next operation is not within the scope of the Processor, and control is returned to the System Monitor.

The System Monitor recognizes the $IBSRT control

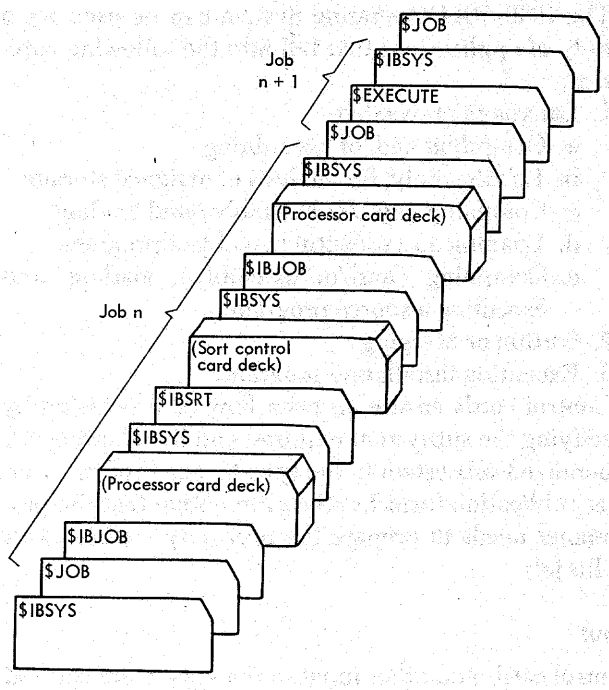


Figure 5. Portion of an Input Deck

card and passes control to the Sort Monitor, the supervisory portion of the Generalized Sorting System. The Sort Monitor controls the execution of the sort run according to the sort control card specifications. Control is returned to the System Monitor when sort is completed.

The $JOB card indicates the beginning of a new job. Notice that the System Monitor is supervising both subsystem-to-subsystem and job-to-job transition. The subsystem monitors supervise run-to-run transition for a series of subsystem runs.

When the $EXECUTE card is encountered, the installation program to be executed is loaded by the System Loader. The $IBSYS card causes the System Monitor to regain control after the installation program is executed.

Output

The listing output of all system programs is written on a system output file. Punched-card output appears in the system punch file. The object program may use either these files or any files designated by the programmer.

Job Termination

In the event of unexpected job termination, the subsystem in control completes all possible housekeeping and transfers control to the Dump routine of the System Monitor, which brings in the Dump program. Parameters assembled with the Dump program and identified by the error number given in the Dump calling sequence specify the error message that is typed and/or listed and the extent of internal and external storage that is dumped. See the section titled "The Dump Program" in this publication.

Programs that are being tested can be run under control of the System Monitor in the same manner as fully-tested programs. The Dump program provides a storage printout if an error occurs, and processing continues with the next job.

Job Skipping

The machine operator can skip jobs, one at a time, by using the operator interrupt procedures. Control cards within a skipped job that affect unit assignment are processed. For details of the job skipping procedure and the names of control cards that are processed, see the publication *IBM 7040/7044 Operating System (16/32K): Operator's Guide,* Form C28-6338.

Control Card Format

Several control card formats are used in the Operating System. Parameter cards, which are used to adapt a sort or edit run to the specific needs of the user, have special formats. A description of the Sort control cards is in the publication *IBM 7040/7044 Operating System (16/32K): Generalized Sorting System,* Form C28-6337. A description of the Edit control cards is in the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339.

The general format of the System Monitor and Processor Monitor control cards is:

COLUMNS	CONTENTS
1	$
2-8	Control card name, left-justified
16-72	Variable field information, no embedded blanks, fields separated by commas

Column 8 is not examined by the System Monitor. Columns 8 through 13 of all monitor control cards may contain a name, left-justified.

A comma delimits an option; a blank ends the list of options and begins the comments portion of a card. A comma must not appear in column 16 unless otherwise specified.

The order of the options in the variable field is not significant unless otherwise stated. Except where indicated, an option is assumed by a monitor when a field is omitted. Thus, the amount of information needed on the card in general is minimized.

In this manual, the following conventions are used for variable field information:

1. Lower-case letters indicate that a substitution must be made.

2. Upper-case letters must be punched exactly as shown if used.

3. Brackets [ ] contain an option that may be omitted or included at the user's choice.

4. Braces { } indicate that a choice of the contents is to be made. If no option is specified by the user, the standard option assumed by the monitor, if any, is underlined.

5. A number over the first character in a field indicates the first card column of the field.

The following sections explain the features of the System Monitor and how each is used by the programmer. The System Monitor consists of the Supervisor and the Nucleus. A more detailed description of the System Monitor is contained in the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339.

The System Monitor

The 7040/7044 Operating System Monitor coordinates the use of program and machine facilities and provides for continuous, efficient operation of the system. This continuous machine processing is the function of the Supervisor, a program that allows changes in the core storage environment, processes changes in the input/output device configuration, and permits reassignment of symbolic input/output units. The Supervisor is the portion of the System Monitor that provides for program control.

The Nucleus contains information that must be available to object programs, to subsystems, and to the Supervisor. Consequently, the Nucleus is in lower core storage at all times.

Figure 6 defines the relationship between the Combined Monitors and the System Monitor.

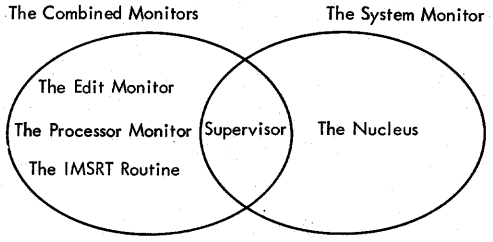


Figure 6. Relationship Between the System Monitor and the Combined Monitors

Several routines that can be used by object programs and subsystems are made available by the System Monitor. This category includes the Input/Output Control System and the Dump Program. The System Monitor also provides a Bootstrap routine and a Housekeeping routine to load and initialize the Nucleus at every initial start. They are overlaid by portions of the Nucleus, the Input/Output Control System, and the Supervisor before control card processing begins. The lower levels of the Input/Output Control System (IOEX and IOOP1) always remain in core storage with the Nucleus. Other portions that may be overlaid by object programs or subsystems are restored each time the combined monitors are loaded.

The Dump program is stored with other system programs in the System Library, which is the collection of system programs and subroutines that make up the Operating System. The Dump program is called into core storage by the Dump routine in the Nucleus when a dump is necessary.

Use of the System Monitor

System Monitor Control Cards

System Monitor control cards are provided for operator and programmer control of the 7040/7044 Operating System. They always appear unblocked on the system input unit. Four types of cards are used:

Subsystem Control Cards
- $IBJOB
- $IBSRT
- $IBEDT
- $EXECUTE

Operational Control Cards
- $IBSYS
- $JOB
- $UNLIST
- $LIST
- $ID
- $PAUSE
- $STOP
- $RESTART

Information Control Cards
- $DATE
- $TIME
- $*

Unit Assignment Control Cards
- $DETACH
- $ATTACH
- $SWITCH
- $RESTORE
- $CHANNEL
- $OPEN
- $CLOSE
- $UNITS

The Subsystem control cards designate the next subsystem or program to which control should be passed. The $IBSYS card is used to return control to the System Monitor; the $JOB card delimits a job.

A series of applications to be run under one subsystem may be submitted without a $IBSYS or $JOB card.

A description of the $IBJOB card is presented later in this publication. Descriptions of the $JOB card, $IBSYS card, $EXECUTE card, $OPEN card, $CLOSE card, $SWITCH card, and $CHANNEL card follow.

$JOB CARD

The format of the $JOB card is:

1	8	16
$JOB		any text

This optional card denotes the beginning of a job. It is typed on-line and the system accounting routine is called. When a $JOB card is read, all previous programmer unit reservations are canceled. (See the section "Intersystem Reservation Technique" for a description of unit reservation codes.) The $JOB control card can be preceded by any System Monitor control card except a Subsystem control card.

The variable field of the $JOB control card may contain a message for the machine operator or programmer. The first 30 characters, starting in card column 16, are saved in the Nucleus and appear in the listing as part of the page heading.

An example of a typical $JOB card follows:

1	8	16
$JOB		FORTRAN PROGRAM 63

This card is typed on the console typewriter to identify the job and the subsequent typed output. It cancels all programmer unit reservations and it transfers control to the system accounting routine.

$IBSYS CARD

The format of the $IBSYS card is:

1	8	16
$IBSYS		

This card is used to return control to the System Monitor. Each return to the System Monitor will release all units reserved as system work units and all units reserved for object programs, other than intersystem units. The Supervisor is loaded if it is not already in core storage. Except at initial start, the $IBSYS card must precede all other System Monitor control cards or any control cards that cannot be processed by the subsystem that is currently in control.

An example of a typical $IBSYS card follows:

1	8	16
$IBSYS		

This card forces the transfer of control to the System Monitor.

$EXECUTE CARD

The format of the $EXECUTE card is:

1	16
$EXECUTE	name

This card is used to pass control to a program that runs under direct control of the System Monitor.

The variable field of the $EXECUTE card must contain the name of the program to which control is to be passed. From one through six BCD characters may be punched, starting in card column 16, to identify a program or system on the System Library. The characters punched should be identical to the contents of the first word of the appropriate entry in the Table of Contents for the System Library.

When the $EXECUTE card is encountered, a control card processing routine in the Supervisor scans the System Library Table of Contents for the program name that is specified on the $EXECUTE card. (The publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide*, Form C28-6339, contains a description of the method used for editing a program into the System Library.)

If the program name is found in the Table of Contents, the load addresses of all the program's phases are found. The Supervisor transfers control to the System Loader, the absolute program loader in the Nucleus, which positions the proper library unit, loads the first phase, and releases control to it.

If the program name is not found in the Table of Contents, an error message is typed and the next control card is read.

An example of a typical $EXECUTE card follows:

1	16
$EXECUTE	PROGRM

This card causes the program with the identifying name PROGRM to be loaded into core storage from the System Library and executed under control of the System Monitor.

Modified forms of the $EXECUTE card are used in connection with the Update Facilities, the Monitored Utility Programs, and the IBM 7740 Communications Control Package. Descriptions of the cards used for the Update Facilities and the Monitored Utility Programs appear in the sections of this publication that describe these features of the Operating System. The card used for the 7740 Communications Control Package is described in the publication *IBM 7040/7044 Operating System (16/32K): Input/Output Control System*, Form C28-6309.

$OPEN CARD

The format of the $OPEN card is:

1	8	16
$OPEN		{ S.SPP1 / unit=Iyy / S.SUxx / Iyy } [, REWIND]

This card may be used for any of the following:

1. To change the combined system print/punch function to uncombined.

2. To assign an intersystem reservation code to a unit.

3. To perform a rewind operation on a unit.

The contents of the variable field are:

{S.SPP1 / unit=Iyy / S.SUxx / Iyy}

The Unit options: The S.SPP1 option causes system punch output to be produced on the system punch unit. The option is used when the system print output and the system punch output are being produced on the same unit and the programmer wishes each to be produced on a separate unit.

The *unit*=Iyy option causes the specified unit to be assigned the intersystem reservation code yy. The specification for *unit* may be:

S.SUxx	System utility unit xx
U	Any available tape, disk, or drum unit
T	Any available tape unit
D	Any available disk or drum unit

The S.SUxx option is used to specify the system utility unit xx. Unless REWIND is also specified, this option is meaningless and the card is ignored.

The Iyy option is used to specify the unit that has been assigned the intersystem reservation code yy. Unless REWIND is also specified, this option is meaningless and the card is ignored.

[, REWIND]

This option causes the specified unit to be rewound when the $OPEN card is processed. The REWIND option is meaningless when the unit option specifies S.SPP1.

Three examples of the $OPEN card follow:

1	8	16
$OPEN		S.SPP1

This card changes the combined print/punch function to uncombined.

1	8	16
$OPEN		T=I14, REWIND

This card causes the first available tape unit to be assigned the intersystem reservation code 14. A message is typed to indicate the physical unit to which the code has been assigned. The tape on the unit is rewound.

1	8	16
$OPEN		S.SU07, REWIND

When this card is encountered, the system utility unit 07 is rewound.

$CLOSE CARD

The format of the $CLOSE card is:

1	8	16
$CLOSE		{S.SPP1 / S.SUxx / IyyR} [, MARK] [{REWIND / REMOVE}]

This card may be used for either of the following:

1. To change the uncombined system print and punch functions to combined.

2. To cancel system or programmer reservations.

The contents of the variable field are:

{S.SPP1 / S.SUxx / IyyR}

The Unit options: The S.SPP1 option causes the system print output and the system punch output to be produced on one unit rather than on two separate units.

The S.SUxx option causes the reservation code (if any) assigned to the system utility unit xx to be canceled, that is, set to 00.

The IyyR option causes the intersystem reservation code yy to be canceled. The unit to which the code was assigned is then assigned the code 00.

[, MARK]

If MARK is specified, a file mark is written on the unit that has been specified by the unit option.

[{REWIND / REMOVE}]

The Rewind options: If REWIND is specified, the unit is rewound. If REMOVE is specified, the unit is rewound and unloaded.

If neither option is specified, the unit is not repositioned.

If the unit to be closed is a system utility unit, no test is made for invalid operations, such as writing a file mark on an input reel. No trailer label facilities exist for a unit that is closed.

An example of the $CLOSE card follows:

1	8	16
$CLOSE		S.SU10, MARK, REMOVE

This card closes system utility unit 10. A file mark is written on the unit. The unit is rewound and unloaded and its reservation code is set to 00.

$SWITCH CARD

The format of the $SWITCH card is:

1	8	16
$SWITCH		{S.Sxxx / Iyy[R]} {S.Sxxx / Iyy[R]}

This card interchanges two physical units assigned as symbolic units. Neither of the physical units is repositioned.

The contents of the variable field indicate which units are to be switched. The specification for a unit may be:

S.Sxxx

S.Sxxx represents any one of the symbolic units for which there is an entry in the Symbolic Units Table.

Iyy[R]

This represents the unit to which the intersystem reservation code yy has been assigned. If R is included in this specification, the intersystem code is to be released when the units are switched.

The rules for the switching of units are as follows:

1. Units having identical codes (for example, both 00, both 60) may be switched.

2. Two units, each having a unique intersystem reservation code (01-20), may be switched. In this case, each code is assigned to a new symbolic unit but remains assigned to the same physical unit.

3. A unit assigned an intersystem reservation code (01-20) may be switched with a unit assigned the code 00. In this case, each code is assigned to a new symbolic unit but remains assigned to the same physical unit.

4. A unit assigned an intersystem reservation code (01-20) may be switched with a system unit (59-63). The reservation codes will be switched unless the $SWITCH card indicates that the intersystem code is to be released (IyyR). This specification causes both units to be assigned the system unit reservation code after the units are switched.

5. A unit that is not reserved may be switched with a system unit (59-63). In this case, the system unit code is assigned to both units after they are switched.

Two examples of the $SWITCH card follow:

```
1          8          16
$SWITCH               S.SU08,S.SU12
```

This card causes the device assigned as utility unit 08 to be reassigned as utility unit 12 and the device assigned as utility unit 12 to be reassigned as utility unit 08.

```
1          8          16
$SWITCH               I12R,S.SIN1
```

This card causes the device that has been assigned the intersystem reservation code 12 to be reassigned as S.SIN1. The device assigned as S.SIN1 is reassigned as the symbolic unit that was previously assigned intersystem code 12. Both units are assigned the system unit reservation code for the system input unit (60).

$CHANNEL CARD

The format of the $CHANNEL card is:

```
1                    16
$CHANNEL             c(f1, f2, . . . , fn)c(f1, f2, . . . , fn). . .
```

This card designates the symbolic channels that are to be used during a job. The programmer specifies the *n* conditions required for each symbolic channel. This enables a relationship to be established between the symbolic channels and the physical channels A through E. This relationship remains in effect until changed or canceled by a new $CHANNEL card or until canceled by a $JOB card.

The contents of the variable field are:

c

The letter *c* is replaced by one of the characters V, W, X, Y, or Z, which identifies the symbolic channel that is being defined.

f

The letter *f* represents a channel definer, that is, a requirement or a restriction that applies in assigning a physical channel as the symbolic channel being defined. As many channel definers may be specified as are necessary to define the symbolic channel fully. More than one definer of the same type may be specified, provided that no contradiction exists. A channel definer may have any of the following forms:

$n_1[-n_2]k$

This form indicates the number of units of a specified type that are associated with the symbolic channel being defined. The device type *k* can be:

T—to indicate a magnetic tape unit
D—to indicate a disk or drum unit
U—to indicate any unit (tape, disk, or drum) that can be used for both input and output

The number n_1 indicates the minimum number of units of the specified type that must be available on the physical channel associated with the symbolic channel. The value of n_2 is the maximum number of units that will be referred to as being on the symbolic channel, though only n_1 of them need actually be on the associated physical channel.

$$nDm \left[\begin{Bmatrix} Iyy \\ /Iyy \end{Bmatrix} \right]$$

This form indicates that the physical channel associated with this symbolic channel must include a disk or drum module that has at least n symbolic units available on it. The programmer assigns the module a number, m, which may be any number from 1 through 5. This number is used in a variable unit reference of the form Dm to refer to this module. (Variable unit references are discussed in the section "Symbolic Channel Assignment Technique" in this publication.)

If Iyy is included in this channel definer, the module will be the one on which the unit assigned the intersystem reservation code yy is located. If /Iyy is included in the channel definer, the module will not be the one on which the unit assigned the intersystem reservation code yy is located.

$$\begin{Bmatrix} Iyy \\ /Iyy \end{Bmatrix}$$

The channel definer Iyy indicates that the physical channel associated with the symbolic channel being defined will be the channel that includes the unit to which the intersystem reservation code yy has been assigned.

The channel definer /Iyy indicates that the physical channel associated with the symbolic channel being defined will not be the channel that includes the unit to which the intersystem reservation code yy has been assigned.

$$\begin{Bmatrix} r \\ /r \end{Bmatrix}$$

The letter *r* is replaced by one of the characters used to identify a physical channel (A, B, C, D, or E). The definer *r* indicates that the symbolic channel being defined will be associated with the physical channel specified.

The definer */r* indicates that the symbolic channel being defined will not be associated with the physical channel specified.

$$\begin{Bmatrix} Ryy \\ /Ryy \end{Bmatrix}$$

The definer Ryy indicates that the symbolic channel being defined will be associated with the physical channel on which the system unit identified by the reservation code yy is located. The code must be a decimal number from 59 through 63.

The definer /Ryy indicates that the symbolic channel being defined will not be associated with the physical channel on which the system unit identified by the reservation code yy is located.

Up to five symbolic channels may be defined by a single $CHANNEL card. $ETC cards may be used to continue the definers that extend beyond the capacity of the $CHANNEL card. If the requirements of the $CHANNEL card cannot be met, the job is terminated.

Example 1:

```
1                    16
$CHANNEL             V(3T)W(3T)
```

The requirements of this $CHANNEL card can be met if the system configuration has at least two channels. Each channel (V and W) must have a minimum of three tape units available.

Two possible configurations are:

CONFIG.	PHYSICAL CHANNEL	UNITS AVAILABLE	SYMBOLIC CHANNEL
1	A	3T	V
	B	3T	W
	C	..	..
	D	..	..
	E	..	..
2	A	..	..
	B	5T	V
	C	1T, 1D	..
	D	4T	W
	E	..	..

Example 2:

```
1                    16
$CHANNEL             X(2D,3T)Y(/B,/I01,3U)
```

The requirements of this $CHANNEL card can be met if the system configuration has at least two channels.

One channel (X) must have at least two disk units and three tape units available. Another channel (Y), which cannot be channel B and which cannot have a unit to which intersystem reservation code 01 is assigned, must have at least three disk or tape units available.

Two possible configurations are:

CONFIG.	PHYSICAL CHANNEL	UNITS AVAILABLE	SYMBOLIC CHANNEL
1	A	..	..
	B	3T, 2D	X
	C	..	..
	D	1T, 2D	Y
	E	..	..
2	A	3T	Y
	B	3T, 3D	..
	C	3T, 2D	X
	D	1T(I01)*,2T	..
	E	..	..

*Not an available unit

Example 3:

```
1                    16
$CHANNEL             Z(/R60,4T)Y(2D1,3T)V(2D2I05)
```

The requirements of this $CHANNEL card can be met if the system configuration has at least three channels.

One channel (Z), which cannot have the system input unit (R60 = S.SINX) assigned to it, must have at least four tape units available. One channel (Y) must have at least three tape units available, and one disk or drum module with at least two symbolic units assigned and available. Another channel (V) must have a disk or drum module, which has been assigned the symbolic unit with intersystem reservation code 05, with at least two symbolic units assigned and available.

Two possible configurations are:

CONFIG.	PHYSICAL CHANNEL	UNITS AVAILABLE	SYMBOLIC CHANNEL
1	A	..	..
	B	1T(R60)*	Y
		3T	
		2D on Mod. 1	Z
	C	4T	V
	D	1D on Mod. 1 (I05)*	
		2D on Mod. 1	
	E	..	..
2	A	..	..
	B	3T	Y
		1D on Mod. 1	
		4D on Mod. 2	
	C	1T(R60)*	..
		4T	V
	D	1D on Mod. 1 (I05)*	
		2D on Mod. 1	
	E	5T	Z

*Not an available unit

Note that, in configuration 2, physical module 2 on channel B will be assigned as symbolic module 1 on symbolic channel Y.

Example 4:

```
1                    16
$CHANNEL             W(3-5T)V(4T)
```

The requirements of this $CHANNEL card can be met if the system configuration has at least two channels.

One channel (W) must have at least three tape units available. Two additional tape units will be referred to as being on symbolic channel W, but need not be on the physical channel assigned to W. Another channel (V) must have at least four tape units available.

Two possible configurations are:

CONFIG.	PHYSICAL CHANNEL	UNITS AVAILABLE	SYMBOLIC CHANNEL
1	A	..	..
	B	4T	V
	C	5T	W
	D	..	..
	E	..	..
2	A	..	..
	B	4T	V
	C	4T	W
	D	1T	..
	E	..	..

Example 5:

```
1                    16
$CHANNEL             W(0-5T)V(4T)
```

The requirements of this $CHANNEL card can be met if the system configuration has at least one channel.

One channel (W), which need not exist, must have at least zero tape units available. Five additional tape units will be referred to as being on symbolic channel W, but need not be on the physical channel assigned to W. Another channel (V) must have at least four tape units available.

Three possible configurations are:

CONFIG.	PHYSICAL CHANNEL	UNITS AVAILABLE	SYMBOLIC CHANNEL
1	A	..	..
	B	4T	V
	C	5T	W
	D	..	..
	E	..	..
2	A	..	..
	B	..	..
	C	9T	V
	D	..	..
	E	..	W*
3	A	..	..
	B	4T	V
	C	3T	W
	D	2T	..
	E	..	..

*Note that internally symbolic channel W is assigned to some channel that meets the requirements, in this case, channel E, which has "at least zero tape units available."

Use of Subsystem Control Cards

Figure 7 shows a composite system input deck containing several types of jobs. The Subsystem control cards (shaded cards in the figure) are stacked on the system input unit, where they delimit each application or run. Other control cards in the figure are described later in the text.

The remainder of this chapter describes the System Monitor in greater detail. The general reader may now skip to the section "Input/Output Unit Assignment."

The Nucleus

The Nucleus remains in core storage at all times. It contains the data and tables that must be passed from subsystem to subsystem and routines that may be useful to any object program. If the storage protection feature is available, the contents of the Nucleus may be protected against any inadvertent change.

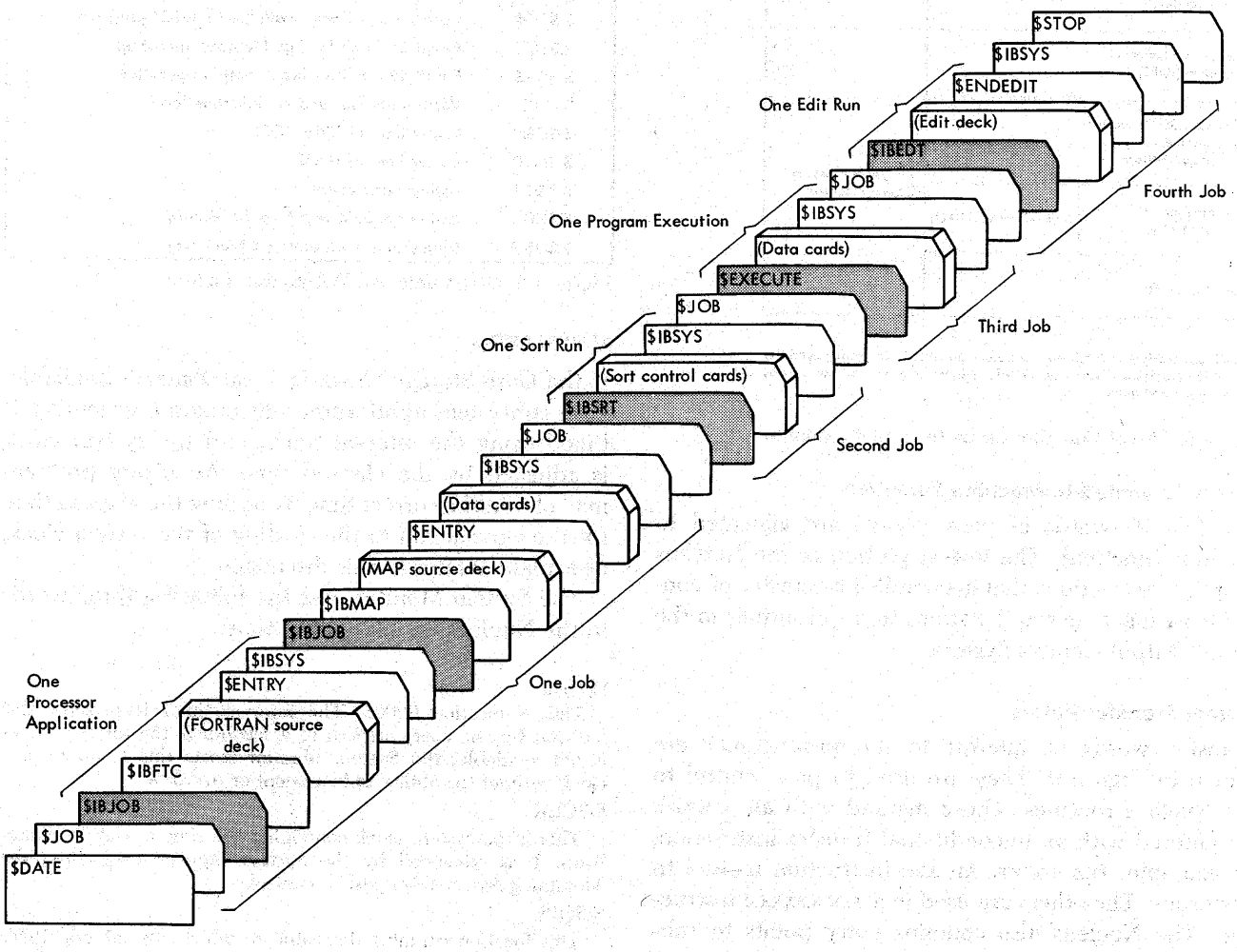
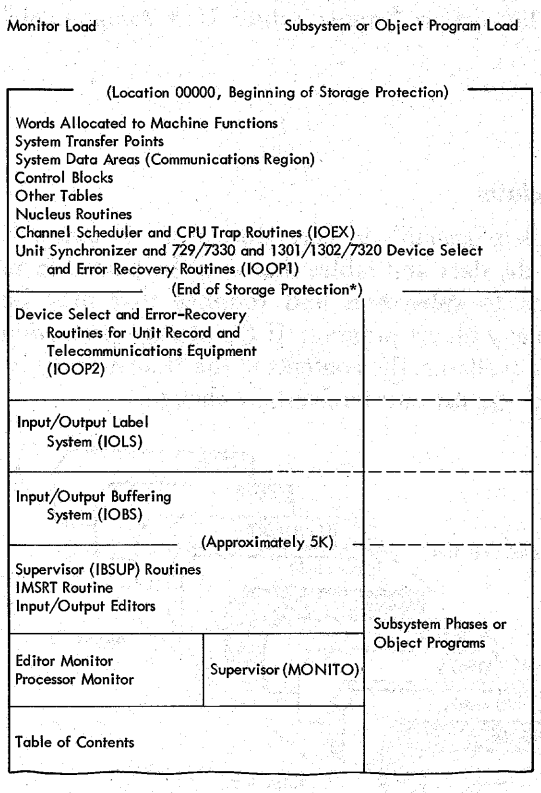


Figure 7. Composite System Input Deck

The Nucleus is divided into the following sections:

1. Words Allocated to Machine Functions
2. System Transfer Points
3. System Data Areas
4. Control Blocks
5. Other Tables
6. Nucleus Routines

Figure 8 shows the relationship of the contents of the Nucleus to the Supervisor and the subsystems.



*IOOP2 and portions of IOLS may be storage protected, depending upon the size of the Nucleus, IOEX, and IOOP1. Object programs will not overlay storage protected areas.

Figure 8. Use of Core Storage by the System Monitor

Words Allocated to Machine Functions

The first 93 words of core storage are allocated to machine functions. The lowest portion of the Nucleus contains instructions that accomplish a transfer of control from machine trap locations to trap routines in the Input/Output Control System.

System Transfer Points

Transfer words of interest to the programmer are shown in Figure 9. They are used to pass control to the Nucleus routines. Those marked with an asterisk are entered with an unconditional transfer instruction, for example, TRA S.SRET. An XEC instruction is used to test S.SRPT. The others are used in a TSX S.SXXX,4 instruction. The Nucleus also contains entry points to subroutines in the Input/Output Control System.

S.SLDR	Transfer to System Loader to load next phase in System Library
S.SRPT	Test for operator interrupt
S.SDMP	Transfer word to Dump routine
S.SRUP*	Transfer to recall the System Monitor
S.SRET*	System Return routine
S.SRST*	Transfer to restore portion of the Restart
S.SCCR	Transfer to Change Communication Region routine
S.SIDR	Transfer word to installation's accounting routine

Figure 9. Transfer Words to Nucleus Routines

System Data Areas

The locations that contain data of use to the programmer are shown in Figure 10. This data can be obtained in the accumulator by using a CLA instruction.

S.SDAT	Current date expressed as day of the month, i.e., mmddyy
S.SDAT+1	Current date expressed as day of the year, i.e., 0yyddd
S.SCUR	Name of system in control
S.SFAZ	Name of phase of system in control
S.SSWI	Switch for system in control
S.SCLK	Time of day (binary)
S.SCIS	Two's complement of the time allotted
S.SCMX	Standard time allotment for a run
S.SCOR	Core storage limits available to object program
S.SSCH	Result of a call for input/output operation
S.SSNS	Result of a call for input/output operation
S.SLVL	Version number and modification level
S.SCSN	Upper limit of IOLS, IOBS
S.SPND	Upper limit of IOOP
S.PGCT	Listing page count
S.SHDR	Listing heading text from $JOB card
S.SDBG	Debugging work unit and block size

Figure 10. Data Contained Within the Nucleus

TIMEKEEPING

If the Core Storage Clock-Interval Timer is available, each subsystem application and program execution is timed using the interval timer, and the system clock is adjusted by the elapsed time. An object program may obtain the correct time by adding the elapsed time for the current run to the reading of the system clock, as explained later in this discussion.

The System Monitor uses the following three words in the Nucleus for measuring time:

IBCLK

This is location 00005. The interval timer increments this location by one every sixtieth of a second. If the interval timer is not available, the System Monitor treats this location as a clock without the ability to increment or overflow.

S.SCLK

This is the system clock containing the time of day in binary form. It is advanced by the System Monitor each time the Monitor gains or relinquishes control.

S.SCIS

This location contains the value to which the interval timer was last set by the System Monitor.

The time of day is kept in the system clock as follows:

1. Location s.SCLK is set to 0 at the initial start.
2. Upon each exit from the System Monitor to a subsystem or to an object program to be executed directly under the System Monitor, the Timer routine of the Supervisor advances the system clock by the difference of the contents of location IBCLK minus the contents of location s.SCIS. The timer routine then places the two's complement of the number of minutes allotted to a run, converted to sixtieths of a second in binary form, into IBCLK and s.SCIS. This value will cause an interval timer overflow at the end of the time period specified by the $TIME card or the assembly parameter MXCLK.
3. Upon each entry to the System Monitor from a subsystem or an executed program, the Timer routine again advances the system clock by the difference between the contents of locations IBCLK and s.SCIS.

An object program may use a similar technique to determine the time of day. A sample routine to obtain the time of day and store it in location TIMEOD in the object program is given in Figure 11.

```
TIMNOW  PZE     **
        CLA     5       PICK UP TIMER VALUE
        SUB     S.SCIS  MINUS INITIAL SETTING
        ADD     S.SCLK  PLUS PREVIOUS TIME
        STO     TIMEOD  TAKE YOUR TIME
        TRA*    TIMNOW
```

Figure 11. Obtaining the Time of Day in Binary Sixtieths of a Second

POINTER TO TABLES

A group of constants set from the system assembly parameters provides information concerning the location and length of the tables within the Nucleus. These constants are considered pointers to the tabulated data. They are shown in Figure 12.

S.SUNI	Pointer to system utility units
S.SUBC	Pointer to unit control blocks by channel
S.SSBC	Pointer to system control blocks by channel
S.SDEX	Information used to load the Table of Contents

Figure 12. Pointers to Tabulated Data

The publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide*, Form C28-6339, contains a description of the contents of these pointers.

SYMBOLIC UNITS TABLE

The Symbolic Units Table lists the input/output units available to the system and the addresses of descriptive tables for these units. There is one word in the table for each symbolic unit. These entries are initialized by the System Monitor housekeeping routines from the ATTACH macro-instructions used during system assembly.

Figure 13 shows a schematic Symbolic Units Table. Figure 14 shows the minimum symbolic units necessary for system operation. These are S.SLB1, S.SIN1, S.SOU1, S.SPP1, and the utility units S.SU00, S.SU01, and S.SU02. A sample configuration is shown in Figure 15.

The Supervisor maintains this table and modifies it when processing $SWITCH, $ATTACH, $DETACH, or $RESTORE control cards.

The following definitions apply to discussions of input/output units or devices in this publication:

Input/output device
The physical equipment from which information can be read or to which information can be transmitted.

Logical unit
A subsystem unit that is equated to a symbolic unit.

Symbolic unit
One of the mnemonics in the Symbolic Units Table. Each symbolic unit is equated to an input/output device, or a portion of such a device.

Mnemonic	Contents		Function
S.SLB1	pfx	ucb, , scb	LIBRARY1
S.SLB2	pfx	ucb, , scb	LIBRARY2
S.SIN1	pfx	ucb, , scb	INPUT1
S.SIN2	pfx	ucb, , scb	INPUT2
S.SOU1	pfx	ucb, , scb	OUTPUT1
S.SOU2	pfx	ucb, , scb	OUTPUT2
S.SPP1	pfx	ucb, , scb	PUNCH1
S.SPP2	pfx	ucb, , scb	PUNCH2
S.SCK1	pfx	ucb, , scb	CHECKPOINT
S.SU00	pfx	ucb, , scb	UTILITY0
S.SU01	pfx	ucb, , scb	UTILITY1
S.SU02	pfx	ucb, , scb	UTILITY2
S.SU03	pfx	ucb, , scb	UTILITY3
S.SU04	pfx	ucb, , scb	UTILITY4
S.SU05	pfx	ucb, , scb	UTILITY5

The prefix bits of each entry are interpreted as follows:

Bit 0 = 0	Unit may or may not be unloaded after being rewound.
= 1	Unit must be unloaded after being rewound. This bit is set for S.SOU1, S.SOU2, S.SPP1, and S.SPP2 for data protection.
Bit 1 = 0	Unit is not in use by current program.
= 1	Unit is in use by current program. This bit should be set by any object program that does not use the IOBS level of IOCS, so that checkpoint and restart procedures can be performed.
Bit 2	Not used.

The prefix codes that can be used to set these bits are:

PZE	Bit 0 = 0; Bit 1 = 0
MZE	Bit 0 = 1; Bit 1 = 0
PTW	Bit 0 = 0; Bit 1 = 1
MTW	Bit 0 = 1; Bit 1 = 1

The address portion of each entry contains the location of the unit control block (ucb) for the device assigned to the symbolic unit; the decrement portion contains the location of the system control block (scb) for the part of the device assigned to the symbolic unit.

Figure 13. A Schematic Symbolic Units Table

System unit
One of the symbolic units from S.SLB1 through S.SCK1. See Figure 13.

Utility unit
One of the symbolic units S.SU00 through S.SU99.

Use of this symbolic unit reference scheme allows maximum flexibility in the use of input/output devices by both system and object programs. In addition, this allows the machine operator to plan the use of each input/output device in advance.

The Symbolic Units Table is referenced by any part of the Operating System that uses an input/output unit in processing and by all file control blocks. The calling sequence to S.IOOP references this table.

The symbolic units S.SIN2, S.SOU2, and S.SPP2 are provided for reel switching purposes. These are called secondary units. They should not be referenced by a program. Word FCUNI in a file control block referencing one of these should have the primary unit (S.SIN1, S.SOU1, and S.SPP1) in both the address and decrement portions or the decrement portion should be zero. They may have the same unit control block and system control block as the corresponding primary unit.

If a symbolic unit is not attached, the corresponding entry in the Symbolic Units Table is a word of zeros.

Control Blocks

UNIT CONTROL BLOCKS

A nine-word block of information is created at system assembly time for each input/output device attached to the system. These blocks are maintained by the System Monitor and the Input/Output Control System. These blocks contain information for each input/output device. For a description of unit control blocks, see the publication *IBM 7040/7044 Operating System (16/32K): Input/Output Control System,* Form C28-6309.

SYSTEM CONTROL BLOCKS

At an initial start, the housekeeping routines of the System Monitor create a system control block for each symbolic unit to which a device is attached. The system control blocks are generated in an abbreviated form at system assembly time from the ATTACH macro-instruction. The length of these blocks and the information they provide depend on the attached device. The system control blocks are used by the unit synchronizer of IOOP to save information pertaining to an operation requested on a symbolic unit. See the publication *IBM 7040/7044 Operating System (16/32K): Input/Output Control System,* Form C28-6309, for a description of a system control block.

Other Tables

In addition to the Symbolic Units Table, two tables, the Abbreviated Table of Contents and the Recognizable Control Card Table, are included in the Nucleus with the Nucleus routines that use them.

ABBREVIATED TABLE OF CONTENTS

The Abbreviated Table of Contents contains the portions of the Table of Contents that identify the names and positions in the System Library of each phase of the subsystem currently in control. This allows the Return routine and the System Loader to obtain the next subsystem phase to be executed without reloading the combined monitors.

RECOGNIZABLE CONTROL CARD TABLE

The Recognizable Control Card Table is a list of all control cards that can be recognized by the Return routine and used by the System Loader to call a specified subsystem component.

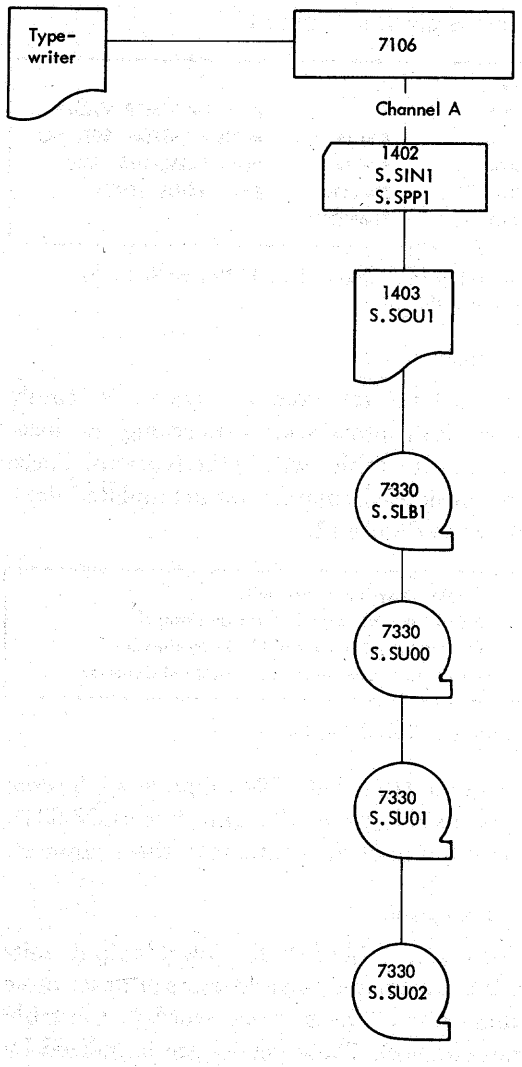


Figure 14. A Configuration for a Minimum System

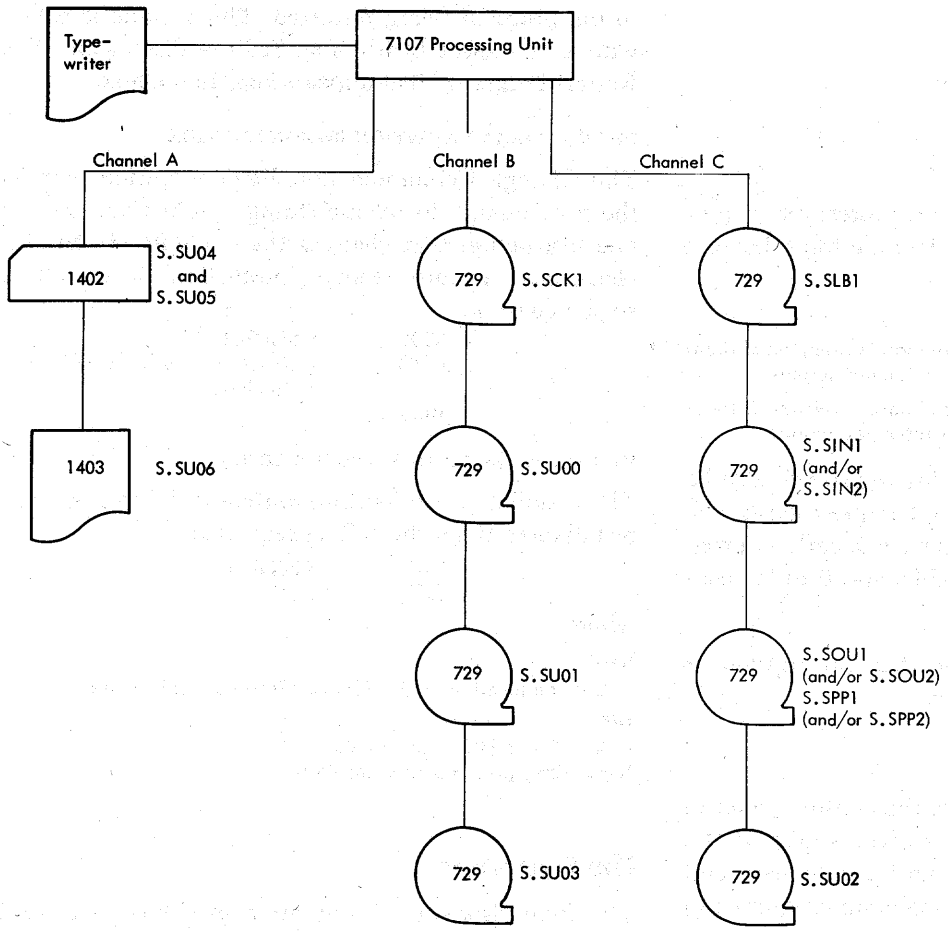


Figure 15. Sample Configuration for a System

Nucleus Routines

The Nucleus contains the minimum routines necessary for smooth transition between the subsystem monitors. The following routines are available to the programmer through the transfer points in the Nucleus.

1. System Loader (S.SLDR)
2. Interrupt Test (S.SRPT)
3. System Dump (S.SDMP)
4. System Monitor Recall (S.SRUP)
5. System Return (S.SRET)
6. System Restart (S.SRST)
7. Change Communication Region (S.SCCR)
8. Installation Accounting (S.SIDR)

SYSTEM LOADER

The System Loader is an absolute program loader that can load and verify a phase in System Library format by using information supplied to it in the Abbreviated Table of Contents. The System Loader optimizes loading from disk by skipping directly to the track containing the first record to be loaded. This loader performs the following functions:

1. It pre-positions the device specified in the Abbreviated Table of Contents, loads a phase from the device, and verifies the accuracy of the positioning.
2. It initiates post-positioning of the device to the next phase to be loaded.
3. It transfers control to the phase just loaded.

Functions 1, 2, and 3 are optional, as indicated in the calling sequence to the System Loader, which follows:

```
TSX     S.SLDR,4
pfx     ptr
```

where the prefix codes have the following meanings:

Sign Bit
1 – Do not post-position
0 – Post-position

Bit 1
1 – Do not load
0 – Load

Bit 2
1 – Do not transfer control
0 – Transfer control

and ptr has the following interpretation:

ptr=0

Use the pointer, as it is, to reference the Abbreviated Table of Contents in the established sequence.

ptr≠0

Adjust the reference to the Abbreviated Table of Contents to location ptr, which must be the first word of a three-word table of contents entry for the phase or program that the user wishes to have loaded.

INTERRUPT TEST

The Interrupt Test tests for operator interrupt. A program may at a convenient point include the following instructions:

```
XEC     S.SRPT
OPC     This instruction is executed if there
          is no interrupt request.
OPC     This instruction is executed if there
          is an interrupt request.
```

OPC is any operation code. Index register 4 may be altered by this test. If an interrupt request exists, the caller should complete all operations currently in progress, take a checkpoint if applicable, and transfer control to S.SRUP.

Each subsystem tests for operator interruption at logical points during its processing.

THE SYSTEM DUMP ROUTINE

The System Dump routine stores the console panel in the Nucleus and saves an area of core storage on the system checkpoint unit, if that unit is attached, and calls the System Loader to load the Dump program into that area.

SYSTEM MONITOR RECALL ROUTINE

The System Monitor Recall routine causes the Supervisor to be brought into core storage. It is called with a TRA S.SRUP instruction.

SYSTEM RETURN ROUTINE

When the program in control cannot identify a control card, it saves the card in S.SAVE and transfers control to the System Return routine. The System Return routine compares columns 1-6 of the card that has been saved in location S.SAVE with the cards in the Recognizable Control Card Table. If the control card name is recognized, i.e., is contained in the Recognizable Control Card Table, the System Loader loads the required subsystem component and passes control to it. If there is no card S.SAVE, or if the card is not recognized, the System Loader loads the Supervisor record containing the combined monitors. This routine is called with a TRA S.SRET instruction.

SYSTEM RESTART ROUTINE

The System Restart routine is the restore portion of the Restart Program. It reads in the checkpointed contents of core storage, restores the panel, and transfers control to the program being restarted. This routine is called with a TRA S.SRST instruction. See the discussion "The Restart Program," that appears later in the text.

CHANGE COMMUNICATION REGION ROUTINE

The Change Communication Region routine permits the programmer to release storage protection, execute one instruction that changes the contents of the Nucleus, and restore storage protection. The calling sequence is:

```
TSX     S.SCCR,4
OPC     Instruction that affects the
          Nucleus
Return
```

INSTALLATION ACCOUNTING ROUTINE

The Installation Accounting routine is defined by each installation. It has the calling sequence:

```
TSX     S.SIDR,4
PZE     L,,N
```

where:

L=0
L is the location of the $ID or $JOB card information.
L=0
No $ID or $JOB card exists.
N=Calling program identification.

The Supervisor

The Supervisor provides uninterrupted flow of control from subsystem to subsystem, and from job to job, during the processing of a stack of diversified jobs.

The Supervisor consists of control routines that are brought into core storage between jobs to process System Monitor control cards to determine the way in which an application is processed. All operations of the machine are controlled by the Supervisor in conjunction with the subsystem monitors. Control instructions are provided by the programmer or the machine operator by using control cards on the system input unit.

The Supervisor is called in from the system library unit when required for transferring control between subsystem monitors, maintaining the Nucleus, changing the machine environment, changing the sequence of jobs to be run, assigning external storage devices to logical input/output functions, or changing the system configuration.

Operation of the Supervisor

After being called into core storage, the Supervisor processes System Monitor control cards until a Subsystem control card is recognized. The Supervisor then indicates the location of the appropriate Subsystem phase to the System Loader and releases control to

the System Loader; the System Loader positions the appropriate library unit, loads the subsystem phase, and releases control to it.

The Supervisor tests for operator interrupt immediately prior to reading each control card.

When the subsystem reaches an application outside its scope, the Supervisor is called in to take control. When all jobs are completed, i.e., when the $STOP card is recognized, the Supervisor executes a terminal procedure.

The Dump Program

The 7040/7044 Operating System provides a dump program that can be used by object programs, system programs, or machine operators. This program lists the operator console panel, certain symbolic unit information, and specified portions of internal and external storage. It is primarily designed as a general diagnostic and debugging aid for system programs that cannot continue because of error conditions. It is not intended to replace object program symbolic debugging tools, e.g., DUMP or PDUMP in the relocatable library.

A series of dump parameters, identified by five-digit message numbers, is assembled in the Dump program in the System Library.

By including the appropriate error number in the calling sequence to the Dump program, the programmer provides complete and informative error messages and selective storage dumps. Additional error numbers and error messages may be added by reassembly of the Dump Program. For details concerning this, see the publication *IBM 7040/7044 Operating System (16/32K): Debugging Facilities,* Form C28-6803.

Checkpoint and Restart

Checkpoint

The Checkpoint routine is a relocatable subroutine that may be incorporated into any program running under the Operating System. A call to the Checkpoint routine saves the status of the console panel registers, console sense switch settings, file control information, and core storage on a specified checkpoint device.

The checkpoint calling sequence is:

```
TSX     ICKPT,4
pfx     S.Sxxx,t,files
PZE     low,,high
```

If the prefix *pfx* is MZE, no unit positioning information will be saved. If the prefix *pfx* is PZE, *files* is the location of the first of two words describing the location of label information used by the program currently in control. These words have the following format:

```
PZE     loc,,lall
PZE     dli,,li
```

If there are no file control blocks, the field *files* must be zero. If the field *files* is zero, the only units that are repositioned are those with bit 2 on in the Symbolic Units Table and reservation code 00-70 in the system control blocks. The symbol *loc* is the location of the first file control block, *lall* is the total length of all file control blocks, *dli* is the displacement of the label information within each file control block, and *li* is the length of one file control block. *S.Sxxx, t* designates the symbolic unit assigned as the checkpoint device; if it is zero, the system checkpoint device S.SCK1 is used. The symbols *low* and *high* are, respectively, the lower and upper limits of the core storage locations to be saved; if *low, high* is zero, the limits in S.SCOR are saved. If the designated checkpoint device is unattached, the request will be ignored. *It is not possible to restart from a checkpoint if the core storage limits specified are below IBORG or above S.SEND.*

Restart is usually accomplished by the machine operator, but special restarts can be initiated by a program. The Restart routine in the Nucleus uses the information in the checkpoint record to restore the computer to the conditions existing when the checkpoint was taken and causes the machine to resume processing from that point.

A checkpoint that does not retain unit positioning information (because of an interrupting program that requires use of core storage only) does not permit repositioning of input/output devices upon restart. The programmer requesting such a checkpoint must backspace the checkpoint unit over *a sufficient number of physical records to include two logical records before restart to accomplish a restart.* He must also supply a transfer address to which the Restart program may return. The procedures followed by the Checkpoint routine to take this type of checkpoint are:

1. Halt all input/output activity pertinent to the routine for which the checkpoint is being taken.
2. Write a checkpoint identification record.
3. Save the contents of the panel registers.
4. Save the contents of core storage.

The procedures usually followed by the Checkpoint routine, although not necessarily in the order given, are:

1. Halt all input/output activity pertinent to the routine for which the checkpoint is being taken.
2. Write a checkpoint identification record, saving pertinent Nucleus data.
3. Save the unit positioning information.

4. Save the contents of the panel registers. The contents of the accumulator and the MQ are not saved.

5. Save the contents of core storage.

6. Type a restart code (the prefix must be SIX).

If there is no unit corresponding to S.SCK1, a request for checkpoint on this unit will be ignored. If the program is taking checkpoints on S.SCK1, it cannot take snapshots. If it is taking snapshots, it cannot take checkpoints on S.SCK1.

Restart Program

The Restart program performs the following actions:

1. Locates the checkpoint device and position from the restart code.

2. Checks the core storage limits to be restored against S.SCOR limits. If S.SCOR limits are exceeded, the restart is terminated and a code 20902 error message is typed.

3. Checks the compatibility of the restart input/output data with current input/output data found in the Symbolic Units Table, the unit control block, and the system control block, and notifies the operator of unit discrepancies. Checks labels, if applicable. Positions devices that may be repositioned.

4. Notifies operator of the switch settings at checkpoint time.

5. Brings in the panel record to set up the restore portion of the Restart routine, which is located in the Nucleus.

6. Restores the appropriate monitor job data.

7. Transfers to S.SRST, the restore portion of Restart.

8. Restores the contents of core storage within the specified limits by reading the core-storage-load record from the checkpoint file on the restart device.

9. Restores the contents of the panel registers.

10. Transfers to the location specified by the Restart routine. (This location is usually the point from which the transfer to the Checkpoint routine was made.)

A special restart involving only the restoration of the panel and core storage to the checkpoint condition may be initiated by a program. The program taking the checkpoint must supply certain information to the phase of the Restart routine located in the Nucleus. It must provide the address to which transfer is to be made after restoration is complete, and it must supply information for restoring the panel and for reading the core storage load record. Before transferring to this phase, the checkpoint device must be positioned to the record that is to be read. Program-initiated restart then requires only steps 7-10 in the above list. No devices are repositioned.

Information on Restart procedures is in the publication *IBM 7040/7044 Operating System (16/32K): Operator's Guide*, Form C28-6338.

The Input/Output Control System

The Combined Monitor and the System Monitor use the 7040/7044 Input/Output Control System, which is a set of routines that performs input/output functions for an object program. Among the facilities provided by the Input/Output Control System are:

1. Blocking and deblocking logical records
2. Labeling data files
3. End-of-reel and end-of-file handling
4. Performing all input/output operations
5. Detecting and correcting errors
6. Scheduling operations on the channels

See the publication *IBM 7040/7044 Operating System (16/32K): Input/Output Control System*, Form C28-6309, for a detailed description of the Input/Output Control System.

Refer to Figure 8 for an illustration of the overlaying of unused portions of the Input/Output Control System by subsystem or object programs.

The 1401 Input/Output Control Program

The 7040/7044–1401 Input/Output Control Program permits the input/output devices on the 1401 on channel A to be used as if they were on the 7040/7044. Input/output equipment attached through an on-line 1401 is referred to as being on channel S. This program stays in a ready loop waiting for the 7040/7044 to select the 1401. When that occurs, the select instruction is brought into the 1401 and decoded to perform the proper input/output function.

The 1401 does not attempt to correct errors; the 7040/7044 is notified of all errors.

Input/Output Unit Assignment

This section describes the input/output unit assignment techniques available to the programmer and includes a discussion of the factors to be considered in using these techniques. All of the unit specifications described in this section may be used in the units field of a $FILE card or FILE pseudo-operation (MAP).

For the purpose of this discussion, an available unit is one that meets the following qualifications:

1. The unit is attached.
2. The unit is in ready status. (A test for ready status is performed only if the system is assembled with IFSNS SET 1. See note below.)
3. The reservation code assigned to the unit is 00. (Reservation codes are discussed later in this section.)
4. The unit is not a unit record device.
5. The format of a disk or drum unit corresponds to a standard format of the installation.
6. If the system is assembled with LABELS SET 2, the label associated with the unit must be standard and must have an expired retention period.

NOTE: IFSNS and LABELS are installation assembly parameters. More detailed information on them is in the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339.

Any search for an available unit begins with the first unit in the Symbolic Units Table and continues along the table until a unit is found that satisfies the requirement for that search.

Unit Assignment Techniques

There are five unit assignment techniques available to the programmer. The use of these techniques is governed by unit specifications on Operating System control cards. A description of each technique follows.

Symbolic Unit Reference Technique

This technique allows the programmer to designate units by means of any of the following specifications:

IN

This specification indicates that a system input unit is to be assigned. For a primary unit specification, S.SIN1 is assigned; for a secondary unit specification, S.SIN2 is assigned.

OU

This specification indicates that a system output unit is to be assigned. For a primary unit specification, S.SOU1 is assigned; for a secondary unit specification, S.SOU2 is assigned.

PP

This specification indicates that a system peripheral punch unit is to be assigned. For a primary unit specification, S.SPP1 is assigned; for a secondary unit specification, S.SPP2 is assigned.

CK1

This specification indicates that the system checkpoint unit is to be assigned.

LBx

This specification indicates that a system library unit is to be assigned. If x is 1, S.SLB1 is assigned; if x is 2, S.SLB2 is assigned.

Uxx

This specification indicates that utility unit xx is to be assigned. Any two-digit number from 00 through 99 may be used for xx.

RDa[y]
PRa[y]
PUa[y]

These specifications indicate the assignment of a unit record device: a card reader (RD), a printer (PR), or a card punch (PU). The channel, a, to which the device is attached may be either A or S. The channel A interface, y, may also be specified as 1, 2, or 3. If y is omitted, interface 1 is assumed.

*

This specification is valid in a secondary unit field only. It indicates that the secondary unit for a file is to be the same physical unit that is assigned as the primary unit for the file. This allows multi-reel tape files to be processed on one unit; when the end of a reel is reached, the program pauses to permit the mounting of a new tape. However, disk or drum units are not re-used when the end of the medium is reached.

NONE

This specification, in a primary unit field, indicates that no unit is to be assigned. In a secondary unit field, NONE has the same effect as an asterisk (*).

"Any Unit" Reference Technique

This technique allows the programmer to designate units through any of the following specifications:

T—to indicate that any available tape unit is to be used
D—to indicate that any available disk or drum unit is to be used
U—to indicate that any available unit (tape, disk or drum) is to be used

When the "any unit" references are used, units are assigned on any channel, although the use of channel A is avoided, if possible.

Intersystem Reservation Technique

Every system unit or utility unit that is included in the system has a two-digit code, called a reservation code, associated with it. The reservation code for a unit is placed in the system control block for that unit. The code identifies the unit or indicates the functions for which it can be used. The codes that can appear in the reserve status field of a system control block and their meanings are:

CODES (DECIMAL)	MEANING
00	The unit is unreserved and not in use.
01-20	The unit has been reserved by the user as an intersystem unit. It is assigned only when specific reference is made to its intersystem reservation code or to its symbolic unit designation.
21-49	These codes are not specified at present.
50	The unit is unreserved but in use as a system work unit. It may be assigned for use by object programs when it is made available by the Loader (IBLDR).
51-55	These codes are not specified at present.
56	The unit is unreserved but it is in use by the system or the object program. This unit is assigned only if specific reference is made to its symbolic unit designation.

CODES (DECIMAL)	MEANING
57	The unit is unreserved but in use by a priority program.
58	The unit is unreserved but in use by a permanent program.
59	The unit is in use as a system library unit (S.SLBx).
60	The unit is in use as a system input unit (S.SINx).
61	The unit is in use as a system output unit (S.SOUx).
62	The unit is in use as a system punch unit (S.SPPx).
63	The unit is in use as the system checkpoint unit (S.SCK1).

The intersystem reservation technique uses the codes 01 through 20. The programmer may assign one of these codes to a unit and thus reserve it for carrying over file assignments from application to application within a job. The same intersystem reservation code may not be assigned to more than one unit at the same time. Intersystem reservation codes may be assigned only to those units that have codes of 00 at the time the intersystem code is to be assigned.

The programmer may assign an intersystem reservation code to a particular unit by designating an = Iyy on a control card which allows this specification. The yy is a two-digit decimal number from 01 through 20. Some of the control cards which permit this designation are: the $OPEN card, $IBJOB card (Copy option), $OEDIT card, $FILE card, the Sort SYSTEM card, and the Edit control cards.

Once an intersystem code has been assigned, it may be used to identify the particular unit to which it has been assigned. The programmer specifies Iyy in the variable field of a control card calling for a unit designation, and the unit chosen is that which has been assigned the code yy. Some of the control cards on which this specification may be used are: the $OPEN card, $CLOSE card, $SWITCH card, $CHANNEL card, $IBJOB card (Copy option), $IEDIT card, $OEDIT card, $FILE card, $RELOAD card, the Sort SYSTEM card, and the Edit control cards.

An intersystem reservation code remains assigned to a particular physical unit until the programmer specifies that it should be released or until a $JOB card or a $RESTORE card is encountered.

Label Search Technique

If an installation uses standard labels, the programmer may use the label search technique to have units assigned for his input and output files. This technique allows the pre-mounting of input files. The control cards on which units may be designated by means of label search parameters include the $FILE card, the $RELOAD card, the Sort SYSTEM card, and the $ASSIGN card for the Update program.

INPUT LABEL SEARCH

To use label searching for input files, the programmer instructs the operator to place the file on any unit not currently in use. The programmer may optionally designate that the unit be on a particular channel. He uses a label search parameter as the unit designation on the control card. The label search parameter specifies the type of device on which the file is located and may specify the channel on which the device is located. The programmer uses a label control card to indicate the labeling information associated with the file (the particular label control card that is used varies with the application). The system assigns the unit for the file by searching among all the devices of the specified type on the specified channel (if any) until the label is found that contains the appropriate information.

The format of the label search parameter for input files is:

d[c]LIN

d – This letter is replaced by the device type, which may be any of the following:

T–to indicate a tape unit
D–to indicate a disk or drum unit
U–to indicate any unit (tape, disk, or drum)

c – This letter, if specified, is replaced by a character identifying a physical channel (A, B, C, D, or E) or a symbolic channel (V, W, X, Y, or Z). If a physical channel is designated, devices on that channel are searched. If a symbolic channel is designated, devices on the physical channel associated with that symbolic channel are searched first. If the label is not found, devices on the other channels are searched.

If no channel is specified, devices on all channels are searched until the desired label is found.

LIN – These characters must be included to indicate that the label search is for an input file.

OUTPUT LABEL SEARCH

To use label searching for output files, the programmer uses a label search parameter as the unit designation on the control card. The label search parameter specifies the type of device on which the file is to appear and may specify the channel upon which the device is located.

If the system is assembled with LABELS SET 1, the first available device will be selected as the output device. In this case, labels will not be checked. If the system is assembled with LABELS SET 2, the labels will be checked and only a unit having a label with an expired retention period will be assigned.

The format of the label search parameter for output files is:

d[c]LOU

d – This letter is replaced by the device type, which may be any of the following:

T–to indicate a tape unit
D–to indicate a disk or drum unit
U–to indicate any unit (tape, disk, or drum)

c – This letter, if specified, is replaced by a character identifying a physical channel (A, B, C, D, or E) or a symbolic

channel (V, W, X, Y, or Z). If a physical channel is designated, devices on that channel are searched. If a symbolic channel is designated, devices on the physical channel associated with that symbolic channel are searched first. If a label is not found, devices on other channels are searched.

If no channel is specified, devices on all channels are searched until the desired label is found.

LOU – These characters must be included to indicate that the label search is for an output file.

DEFERRED LABEL SEARCH

The point at which a label search is made is governed by the mounting option specified on the $FILE card associated with the file. If the option specified is MOUNT or READY, the search is performed by the Loader (IBLDR) or the Reload program during object program loading. If the option specified is DEFER, the search is performed when the file is opened. In this case, a label search subroutine is included with the object program and is called by the Input/Output Control System when it is needed. This procedure is known as deferred label searching. Further information about deferred label searching can be found in the publication *IBM 7040/7044 Operating System (16/32K): Input/Output Control System,* Form C28-6309.

Symbolic Channel Assignment Technique

This technique allows the programmer to refer to symbolic channels which have been previously defined by a $CHANNEL control card. The method for defining symbolic channels is described in the section, "System Monitor Control Cards," in which the $CHANNEL card is discussed.

Once a symbolic channel has been defined, the programmer uses variable unit references to request that available units on symbolic channels be assigned. This type of reference has the following general form:

[k]cn

k – the type of device that is to be assigned. This may be:
T – to indicate that a tape unit is to be assigned
D – to indicate that a disk or drum unit is to be assigned
Dm – to indicate that a unit on symbolic module *m* is to be assigned
U – to indicate that any unit is to be assigned

If the device type is not specified, U is assumed.

c – the symbolic channel with which the device is associated. The symbolic channel designations are any of the alphabetic characters V, W, X, Y, or Z. Any symbolic channel designated by a variable unit reference should have already been defined in a $CHANNEL card.

n – may be any number from 1 through 10. This is the order in which the device is chosen from the available units on the symbolic channel. Devices are chosen from available units in the order in which they appear in the Symbolic Units Table.

An example of a variable unit reference is TV3. It indicates that the third available tape unit on symbolic channel V is to be assigned for a file.

The following points should be kept in mind when variable unit references are used:

1. As a given stage in the processing of a job, a variable unit reference applies to a particular physical unit. As a job progresses, various units may become unavailable. Thus, at a later stage in processing, the same variable unit reference may apply to a physical unit different from the physical unit to which it previously applied.

2. If the order of availability specified in a variable unit reference is greater than the number of units available on the symbolic channel specified, a unit is selected on some other channel.

3. If an appropriate primary unit cannot be assigned, the job is terminated.

The Use of Unit Assignment Techniques

The variety of unit assignment techniques available allows the programmer a choice in indicating how units are to be assigned for input and output files. For each unit that is to be assigned, any one of the five techniques may be specified; the programmer need not specify the same technique for all units. It is even permissible for the primary and the secondary unit specifications on a $FILE control card or in a FILE pseudo-operation (MAP) to indicate two different unit assignment techniques. The rules followed by the Loader (IBLDR) in assigning units, particularly the order of assignment, will affect the programmer's choice of technique.

Order of Assignment

All units for a single Loader application are assigned by the Loader before the object program is loaded. (Prior to this time, any desired channel relationships should have been established by means of a $CHANNEL card.) All of the unit assignment specifications are evaluated as a group when units are about to be assigned.

When more than one technique of unit assignment has been specified for the units that are being assigned, the order in which units are assigned is:

1. All units that have been specified by symbolic unit reference.

2. All units that have been specified through their intersystem reservation codes. Units specified by Iyy are assigned before units specified by IyyR (codes to be released).

3. All units that are specified by input label search. If the search fails to locate a unit with the requested input file label, a message will be typed identifying the file to be mounted.

NOTE: At this point, the Loader makes any reservation code changes required for the units thus far assigned. For all units with intersystem codes that are to be released, the Loader sets the code to 00. Each assigned unit with a code of 00 is then assigned either an intersystem code (if one has been specified for it) or a "unit in use" code, which is the decimal number 56.

Adjusting the codes at this point prevents the units from also being assigned by any subsequent methods. Duplicate assignments may have already occurred, however. For example, if s.SU04 has been assigned the intersystem code 02, two files, one specifying s.SU04 and the other specifying I02, are both assigned to the device attached as s.SU04. A symbolic unit specification and an input label search specification may also result in duplicate assignment if the symbolic unit (for example, s.SU06) has no code assigned to it (making it available for input label search) and the input file happens to be on the device attached as s.SU06. An intersystem specification and an input label search specification will never cause duplicate assignment since units reserved by intersystem codes are not included in label search.

The assignment of units then continues as follows:

4. Units that are specified by output label search.

NOTE: If a request for an output label search designates a particular channel, but there are not enough units of the specified type on that channel, the unit request is said to have "spilled." The channel designation is then ignored and a search is made later for a unit of the specified type.

The search for a "spilled" primary unit is made after all other requests for output label search have been processed. The search for a "spilled" secondary unit is made when the "any unit" references are processed.

5. Units that are specified by variable unit references.

   a. Units for which a particular device type has been specified are assigned before those units for which no device type has been specified.
   b. Disk or drum units for which a symbolic module number has been specified are assigned before those without such a specification.

NOTE: If a variable unit reference specifies the *n*th available device on a symbolic channel, but there are fewer than *n* devices available on the physical channel associated with that symbolic channel, the unit request is said to have "spilled." The variable unit reference is then treated as if it were an "any unit" reference of the same device type. The Loader does not assign a device to the unit at this time, but processes any remaining variable unit references. The "spilled" request is processed when the "any unit" references are processed.

The last units to be assigned are:

6. Units specified by the "any unit" references (U, T, or D) and units for which the requests were "spilled."

   a. Primary units are assigned first. The primary units assigned are those specified by "any unit" references and by "spilled" variable unit references. Units for which a particular device type (T or D) is specified are assigned before those for which a general unit type (U) is specified. At this point, if there are any primary unit requests (output label search, variable unit references, or "any unit" references) that have not been satisfied because the Loader could not find enough available units, a message is typed, indicating the number of units needed along with the device type and physical channel requirements of each unit. If possible, the operator should provide available units to meet the requirements (by placing units in ready status, for example). If available units are not provided, the job is terminated.
   b. Secondary units are assigned after all primary units have been assigned. The secondary units assigned are those specified by "any unit" references, by "spilled" variable unit references, and by "spilled" output label search requests. Units for which a particular device type (T or D) is specified are assigned before those for which a general unit type (U) is specified. If there is no unit available for assignment as a secondary unit, the physical unit assigned as the primary unit for a file is also assigned as the secondary unit.

When output label search, variable unit references, or "any unit" references are used, as the Loader assigns a unit for a file, it assigns to the unit a reservation code. If the unit specification included = Iyy, the specified intersystem code is assigned. Otherwise, the "unit in use" code, which is the decimal number 56, is assigned.

Additional Factors

The programmer should be aware of several points concerning the manner in which unit assignment options are processed.

1. If a unit has been assigned for an application as a result of a variable unit reference, the programmer may not assume that the same variable unit reference, used in a subsequent application within the job, will result in the same unit being assigned. For example, a specification of TV5 calls for the fifth available tape unit on channel v. The tape unit found will be the fifth one available at the time the search is made by the Loader. A program loaded subsequently involves a different search by the Loader, and the fifth unit available at this time may or may not be the same unit found before.

A method for guaranteeing that the same unit be assigned in the later application is that of assigning an intersystem code to the unit. For example, a valid specification for the first application would be TV5 = I03 (if the intersystem code 03 is not already assigned to

another unit). The specification for the subsequent application need only be 103 to cause the same unit to be assigned.

2. Operation of label search for output units varies, depending on the setting of the assembly parameter LABELS, which is an installation option. (Assembly parameters are discussed in the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339.)

For a LABELS SET 1 system, since output header labels are not read and checked, a label search request for an output unit results in the assignment of the first available unit of the specified type. Therefore, since this assignment can result in the selection of a unit containing an input file for a subsequent application, the user should make certain that any such units are out of ready status or are assigned reservation codes. (If out of ready status, the units must be returned to ready status before they can be considered available for assignment.)

For a LABELS SET 2 system, since output header labels are unconditionally read and checked, a label search request for an output unit results in assignment only when a unit of the specified type contains a standard header label with an expired retention period. Input files are protected by labels with unexpired retention periods.

A label search for an input unit always involves checking each of the following control block fields, unless the field contains all zeros: file serial number, reel sequence number, label date, and file identification.

Label searching techniques can be used when an output file from one application is to be used as an input file in a subsequent application in the same job. In the earlier application, the programmer can specify output label search to have a unit assigned for the output file. Then, in the subsequent application, an input label search can be used to assign the unit on which the file is located. Any combination of the fields listed above can be specified in the control block for the input file. (This method is advised only for use with LABELS SET 2.)

Another method would be to request, in the earlier application, that the unit assigned as a result of the output search be reserved as an intersystem unit. (This is done by adding = Iyy to the label search parameter.) In the later application, the programmer can specify the intersystem reservation code to cause the same unit to be assigned for the input file. While a unit is assigned an intersystem reservation code, it is not considered to be available. This method, therefore, allows the file to remain on a device that is in ready status, without the risk of the unit being chosen as a system work unit or being assigned unintentionally (by means of a search for an available unit) during an intermediate application.

3. Any job will be terminated if the Loader cannot locate enough units to satisfy the primary unit specifications for object program files. If the primary unit for a file has been assigned, but there is no unit available for assignment as a secondary unit, the device assigned as the primary unit will also be considered as the secondary unit.

Secondary unit specifications (other than * or NONE) are subject to all the rules for order of assignment. Consequently, the secondary unit for a file is assigned before the primary unit for that file whenever the specification for the secondary unit has priority in order of assignment over the specification for the primary unit. However, if no unit is available when the primary is to be assigned, the job will be terminated.

Job termination might have been avoided if the primary unit had been assigned before the secondary unit, since the secondary unit requirement could have been fulfilled by the primary unit. Therefore, if any doubt exists as to the number of units available for a job, care should be taken that secondary unit specifications do not interfere with primary unit assignment.

4. In general, for jobs in which actual device types (T or D) are specified, the use of general unit types (U) should be avoided. As a result of a unit request being "spilled," it is possible that unit requirements will not be met although it seems that enough devices of the proper types are available for program execution.

For example, consider the following $CHANNEL control card:

```
1                    16
$CHANNEL             X(4-10U)Y(1T)Z(4-5T)
```

Assume that at the time that the $CHANNEL card is processed, there are 11 available tape and disk units on channel B, one available tape unit on channel A, and four available tape units on channel C. The channel requirements are met; that is, there are three channels, one channel has at least four units, another channel has at least one tape unit, and a third channel has at least four tape units. The total unit requirement of 16 units, six of which must be tape units (10U + 1T + 5T), is also met. The following relationship is then established:

Channel X = Channel B
Channel Y = Channel A
Channel Z = Channel C

The $FILE cards for a program that is loaded subsequently contain unit specifications for UX1, UX2, UX3, . . . UX10, TY1, TZ1, TZ2, . . . TZ5. The order of assignment indicates that all specifications for tape units are to be

processed first. The tape unit on channel A is assigned for TY1; the four tape units on channel C are assigned for TZ1 through TZ4. When TZ5 is to be assigned, there is no available tape unit on channel C. The request is "spilled"; that is, TZ5 is treated as the "any unit" reference, T, and is saved for later evaluation.

Ten units on channel B are then assigned for UX1 through UX10. The units are assigned in the order in which they appear in the Symbolic Units Table, without regard to device type. Therefore, when the units have been assigned, the unit that remains unassigned on channel B may be either a tape unit or a disk unit.

When the "any unit" reference, T, (converted from TZ5) is processed, a search is made for an available tape unit. The only available unit is the one unassigned unit on channel B. If this unit is a disk unit, it cannot be assigned to fulfill a T specification. Unit requirements will, therefore, not be met.

Introduction

The Processor has facilities for translating source languages into binary programs and for loading and executing translated programs. Input to the Processor is a deck consisting of all the cards necessary for a given processor application. A processor application is the basic unit processed by a Processor at any one time; it is preceded by a $IBJOB card and continues until the next $IBJOB or $IBSYS control card. A processor application may be a segment of a job, since a job (previously defined) may contain one or more processor applications together with other types of applications if desired. A deck for a processor application might consist of load-time control information, one or more source language decks (FORTRAN, COBOL, and MAP), and/or relocatable binary decks from some previous compilation.

An application could be processed in one of the following ways: (1) compile or assemble only, (2) compile or assemble and load, (3) compile or assemble, load, and execute, or (4) compile or assemble, load, execute, and debug.

The Processor consists of eight sections:

The Processor Monitor, which is dominant within the Processor. It supervises the execution of the compilers, the assembly program, and the Loader. As the supervisory portion of the Processor, it provides communication with the System Monitor, and it converts the load time control cards to a binary form and saves this information for the Loader.

The FORTRAN Compiler, which processes programs written in a subset of the FORTRAN IV language — a scientifically oriented language — and produces input to the Macro Assembly Program.

The COBOL Compiler, which processes programs written in the COBOL language — a commercially oriented language — and produces input to the Macro Assembly Program.

The Macro Assembly Program, which processes (1) programs written in the MAP language — a machine-oriented language with macro facilities — and (2) the internal language programs produced by the COBOL and FORTRAN Compilers. The Macro Assembly Program produces a binary program deck in relocatable format from each compilation. This deck can retain enough symbolic content to enable communication between separately compiled program decks.

The Loader, which combines program decks produced by the Macro Assembly Program, the load-time information saved by the Processor Monitor, and any requisite library subroutines, to form one executable machine-language program. It loads separately assembled program decks, allocates storage for common data and input/output buffers, and, optionally, provides a listing of the core storage allocation. The Loader transfers control to the object program if the programmer has specified execution of the program.

The Loader CHAIN feature provides a way to run programs that exceed a single core storage load by forming a multiphase program consisting of a main link and one or more dependent links that are processed to form several core storage loads.

The Loader COPY feature provides a way to produce and save absolute object programs for subsequent loading and execution by the Reload Program.

The Debugging Processor, which processes the load-time debugging requests associated with a MAP or FORTRAN program. The Debugging Processor analyzes the requests in a debugging deck, calls in special debugging subroutines that are executed with the object program, and edits the dumps that have been produced by the debugging subroutines. The dumps are edited after the object program has been executed.

The Reload Program, which loads absolute object programs from a file produced by the Loader.

The Subroutine Library, which contains control information and routines to be loaded into core storage if required for object time execution.

Figure 16 shows the operation of the Processor components on source language programs.

The Input/Output Control System

The Processor uses the highest level of the Input/Output Control System. In addition, the following levels of the Input/Output Control System are available to the user through the IOCS options that he may specify on the $IBJOB card for his processor application:

Input/Output Operations 1 (IOOP1) is the lowest level available to Processor users. It includes IOEX, which provides trap supervision. IOOP1 provides the unit synchronizer, device select and error correction routines, and overlapped input/output and processing for 729 and 7330 Magnetic Tape Units, 1301 Disk Storage, 1302 Disk Storage, and 7320 Drum Storage. The user may communicate directly with IOEX; however, the object program may not overlay IOOP1.

Input/Output Operations 2 (IOOP2) provides device selection, error correction routines, and overlapped input/output and processing for unit record equip-

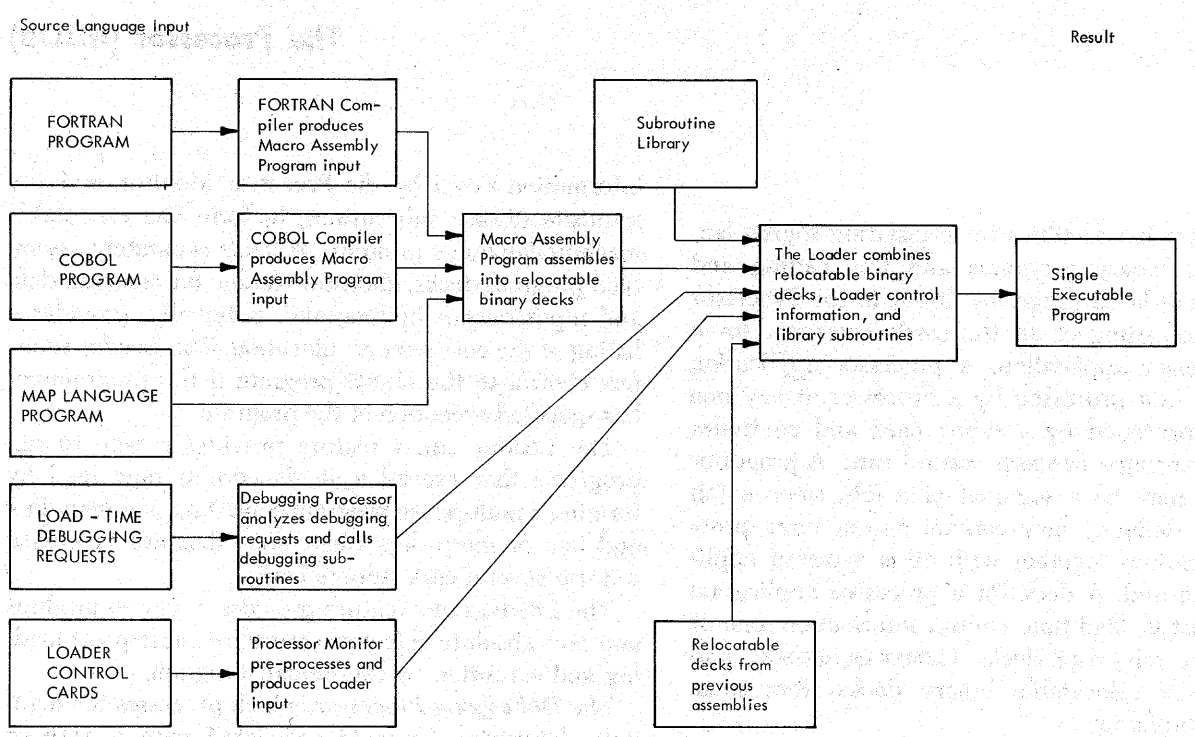


Figure 16. Operation of the Processor on Source Language Programs

ment and telecommunications devices. The IOOP2 level includes the IOOP1 level.

The Input/Output Label System (IOLS) provides automatic label handling routines and reel switching routines. The IOLS level includes IOOP2.

The Input/Output Buffering System (IOBS) is the highest level and provides record blocking and deblocking routines, and buffer supervision. Checkpoint facilities are provided. The IOBS level includes IOLS.

A further discussion of the Input/Output Control System facilities is in the publication *IBM 7040/7044 Operating System (16/32K): Input/Output Control System*, Form C28-6309.

Communication with the System Monitor

The Processor Monitor is part of the combined monitor core storage load. The Processor maintains communication with and uses some of the facilities of the System Monitor.

The Nucleus that is maintained by all Operating System monitors is checked by the Processor Monitor for the availability of input/output units at the initialization of any application and before releasing control to one of the compilers, the assembler, or the Loader.

The System Loader, which is part of the Nucleus, is used for locating and loading system phases.

Processor Source Languages

The Processor includes programs that translate Macro Assembly Program, COBOL, and FORTRAN source language programs and debugging requests.

THE MAP LANGUAGE

The MAP language is a symbolic machine language. In addition to providing symbolic operation codes for machine instructions, the MAP language contains a number of pseudo-operations for data generation, storage allocation, symbol definition, and file description. Certain pseudo-operations enable the MAP programmer to refer to programs that are compiled or assembled independently. The Macro Assembly Program permits the definition of macro-operations and the use of macro-instructions in the source program. When a macro-instruction is encountered, it is expanded according to the sequence of instructions used to define the corresponding macro-operation. In general, any element of an instruction in a macro-operation may be defined as variable, allowing flexibility in the use of macro-instructions.

Further information concerning coding in the MAP language is in the publication *IBM 7040/7044 Operating System (16/32K): Macro Assembly Program (MAP) Language*, Form C28-6335.

THE FORTRAN LANGUAGE

The 7040/7044 FORTRAN IV language and its associated compiler allow the user to communicate with the IBM 7040/7044 Data Processing System in a language more concise and familiar to him than the 7040/7044 machine language itself. This substantially reduces the training required to program, as well as the time con-

sumed in writing programs and in eliminating errors from them. Further information on coding in the FORTRAN language is in the publications *IBM 7040/7044 Operating System (16/32K): FORTRAN IV Language,* Form C28-6329, and *FORTRAN,* Form F28-8074.

THE COBOL LANGUAGE

The COBOL language and its associated compiler permit the solving of a variety of commercial data handling, inventory, and accounting problems. Whereas the FORTRAN language is designed to facilitate solution of mathematical and scientific problems, the COBOL language is oriented to commercial applications involving the processing of large amounts of data.

Further information concerning coding in the COBOL language is in the publications *IBM 7040/7044 Operating System (16/32K): COBOL Language,* Form C28-6336, and *COBOL,* Form F28-8053.

THE DEBUGGING LANGUAGE

The Debugging language is similar to FORTRAN IV. It enables the users of FORTRAN IV and MAP to request dynamic dumps of both specified areas of core storage and machine registers during execution of the object program.

Compile-time debugging requests (associated with a COBOL program) are processed by the COBOL Compiler. Further information on coding of debugging requests is in the publication *IBM 7040/7044 Operating System (16/32K): Debugging Facilities,* Form C28-6803.

Core Storage Allocation

During compilation, assembly, and loading, the following Operating System components are in core storage:

1. The Nucleus
2. IOEX
3. IOOP1
4. IOOP2
5. IOLS
6. IOBS
7. The current Processor component phase, which may include the Input and Output Editors

During execution of the object program, the following Operating System components are in core storage:

1. The Nucleus
2. IOEX
3. IOOP1
4. The object program
5. In addition, the following may be in core storage, depending on which level of IOCS the object program is using:
   a. IOOP2
   b. IOLS
   c. IOBS

Figure 17 indicates the approximate core storage arrangement of the major components of the Operating System during a processor application.

Object Program Origin: Object program origin is assigned to the first available non-storage-protected location after the level of IOCS specified on the $IBJOB card. IOOP2 and portions of IOLS may be storage protected, depending upon the size of the Nucleus and IOOP1. Object programs will not overlay storage protected areas.

Introduction to the Processor Monitor

The Processor Monitor controls (1) transition between processor applications and (2) the flow of a processor application through the Processor according to control card specifications. The Processor Monitor provides communication with the System Monitor and supervises the loading of the various parts of the compilers, assembler, and Loader.

Processor Application Control Options

Typical input to a processor application might consist of one or more source language programs (FORTRAN, COBOL, and MAP) intermixed with relocatable binary program decks from some previous compilation process. Several types of processor applications follow:

Compile or Assemble Only: The source language decks are processed by the compilers and the assembler, which produce a corresponding relocatable binary program deck for each source language deck.

Compile or Assemble, Load: The necessary compilations are performed and the relocatable decks are stacked, under Monitor control, onto a single unit for input to the Loader (IBLDR). Then, the Loader combines the stacked relocatable decks into a single absolute machine-language program that is ready for execution. The Map, Logic, and/or Copy options are performed as specified on the $IBJOB card.

This type of application is used to obtain a map of assigned core storage (MAP), a list of defined coding sections together with references to the (LOGIC or DLOGIC), and/or a copy of the absolute object program that is produced by the Loader (COPY). In all cases, the Loader is required.

Compile or Assemble, Load, and Execute: The necessary compilations are performed; the relocatable binary decks are stacked and formed into a single absolute object program. Control is then transferred to the object program, which ultimately returns control to the system.

If the input to the Processor includes load-time debugging requests (for FORTRAN or MAP source language programs), the procedure differs slightly. Before the Loader is called, the Debugging Preprocessor analyzes the debugging requests and ensures that the debug-

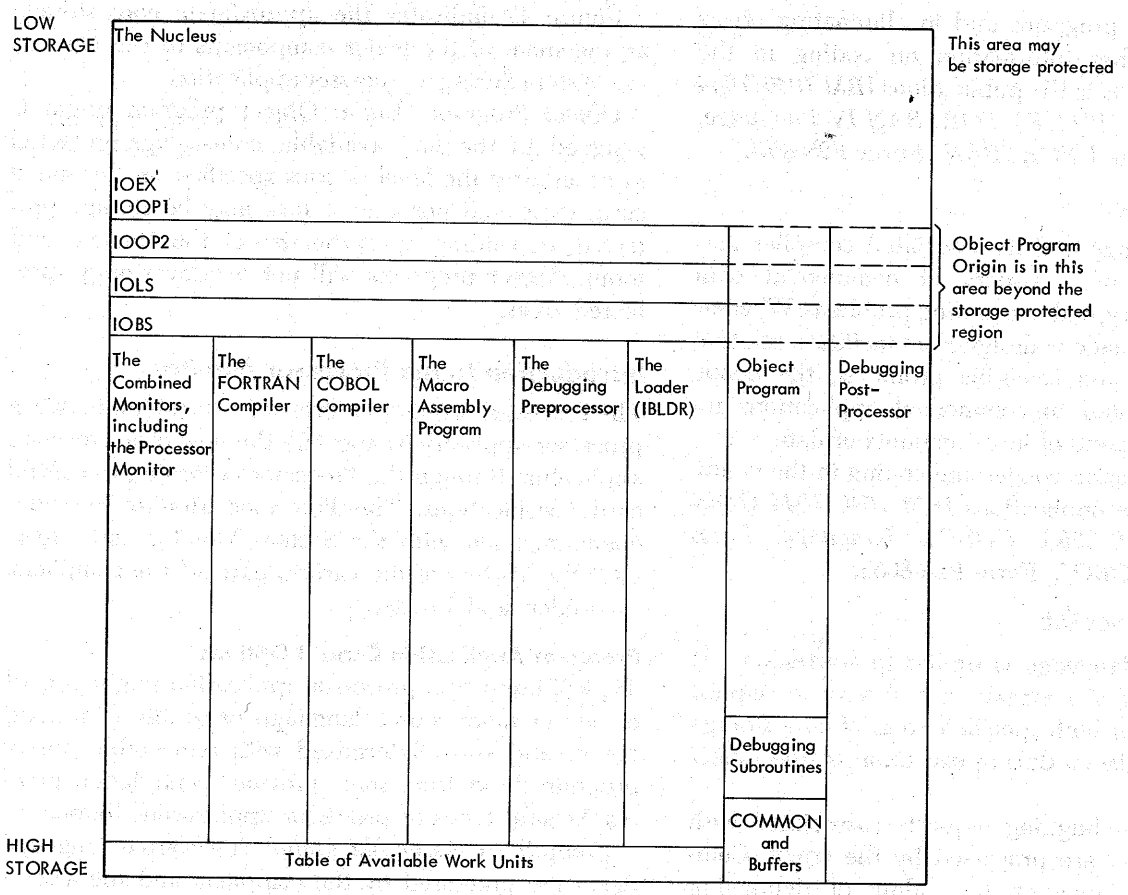


Figure 17. Storage Allocation for the Processor

ging subroutines will be loaded with the object program. Once the object program has been executed and control has been returned to the system, the Debugging Postprocessor is called by the system to edit the dumps produced by the debugging subroutines.

Input to the Processor may consist of relocatable binary program decks only. In this case, only the Loader is required. It accepts its input directly from the system input unit, eliminating the need for a stacked load file. The remainder of the procedure is the same as described above.

Organization of the Processor Monitor

The Processor Monitor is part of an integrated monitoring system that also includes the System Monitor, the Editor Monitor, and a short routine, IMSRT, which loads the Sort Monitor. The Processor Monitor consists of the Preprocessor, the Input and Output Editors, and control card processing routines.

THE PREPROCESSOR

The Loader control cards, $FILE, $LABEL, $POOL, $NAME, $USE, and $OMIT, are processed by the Preprocessor in the Processor Monitor. These control cards are described in the section on the Loader. They must appear immediately after the $IBJOB card for a processor application (or after the $CHAIN card for a CHAIN application) and before any source language or relocatable binary program decks for the processor application.

The Preprocessor scans the variable fields of the Loader cards and produces and saves binary information for the Loader.

THE INPUT AND OUTPUT EDITORS

The Input and Output Editors are used by all processor sections to read from the system input unit (S.SIN1) and to write on the system output file (S.SOU1). The Macro Assembly Program uses the Punch Editor to write punch card output. These editors are incorporated in absolute form into every component on the System Library that uses them. The Input and Output Editors are independent of each other and are in the Subroutine Library. (That is, any program can call one without calling the other.) These editors use the

Input/Output Buffering System (IOBS) of IOCS. These routines are described in the section "Introduction to the Subroutine Library (IBLIB)."

CONTROL CARD PROCESSING ROUTINES

These routines are not available to the user. They are described in the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339.

Symbolic Units Required by the Processor

The Processor operates as a labeled system in which all system units except the library unit can be labeled. A system input unit, a system output unit, and a peripheral punch unit are required for the operation of the Processor. Any object program that uses these units must either use the Input and Output Editors or adhere to all the procedures governing the control of these units. These procedures include adhering to established block sizes, ensuring proper positioning of the system units, and maintaining label conventions.

Three utility units are required as work units for the operation of the Macro Assembly Program (IBMAP), the FORTRAN Compiler (IBFTC), and the Loader (IBLDR). Four utility units are required for the operation of the COBOL Compiler (IBCBC). One additional utility unit, which is used for the load file, is required for applications in which compilation (or assembly) and loading take place.

The Processor Monitor locates a sufficient number of work units for the compilers and assembler. Units that are reserved by the programmer are not chosen as work units. The Processor Monitor then stores the address of the Symbolic Units Table entry for each unit in a table of available work units. The compilers and assembler refer to this table when choosing a work unit.

Figure 18 shows the flow of data and the use of input/output units during a processor application.

Application Processing

The Processor Monitor supervises processor application by using control cards in the input deck. It receives control from the System Monitor when the Supervisor encounters a $IBJOB card. The Processor Monitor functions as follows:

1. From information contained on the $IBJOB card, the Processor Monitor determines which Processor components are required, sets up a list of valid control cards for the application, and forms the phase dictionaries (Abbreviated Table of Contents) for the required sections. These phase dictionaries are part of the System Library Table of Contents, which contains loading information for each phase of the System Library. The Table of Contents is arranged in order by sections and by phases within a section.

2. After processing any load-time control information cards supplied, the first $IBFTC, $IBCBC, $IBMAP, or $IBLDR card is read in and the Processor Monitor passes the location of the load information for the appropriate section to the System Loader (S.SLDR) in the Nucleus. The System Loader loads the first phase of the section required and transfers control to it.

3. The phase transfers control to the System Loader when the phase is complete. The System Loader then loads the next phase and transfers control to it.

4. When the last phase of either the FORTRAN or COBOL compiler is complete, the System Loader provides for automatic transition, using the phase dictionary, to the Macro Assembly Program. Output from the Macro Assembly Program and decks following a $IBLDR card are stacked on the load file.

5. When all phases required for a compilation or assembly are complete, control is returned to S.SRET, which determines if the next card is valid for this processor application. If it is valid, the appropriate section is loaded, using the System Loader; if the card is invalid, the combined monitors are called.

6. The Loader (IBLDR) generates an absolute binary object program in a form that is acceptable to the System Loader. As much of the program as possible is retained in storage with IBLDR; the overflow is dumped onto a utility unit. The Loader (IBLDR) loads the internally stored portion of the program and then gives control to the System Loader to load the overflow, if any. After loading has been completed, control is given to the object program. *Object programs must ultimately return control to the Post-Execution routine,* S.JXIT, using a TRA S.JXIT instruction; S.JXIT, which is called from the Subroutine Library, then closes files on all units and performs the specified rewind and unload options on all utility units. S.JXIT returns control to S.SRET.

S.SRET returns control to the System Monitor, unless load-time debugging is included in the application. In this case, S.SRET transfers control to the Debugging Postprocessor, which edits dump output. Upon completion, the Postprocessor transfers back to S.SRET, which then returns control to the System Monitor.

End of Data

Data stacked on the system input unit with the control cards creates a special problem. The object program must recognize the end of data, or it must recognize a $ card and pass the $ card to the Nucleus area S.SAVE by using a routine such as the one following. It must not treat subsequent control cards as data. (See Figure 19.)

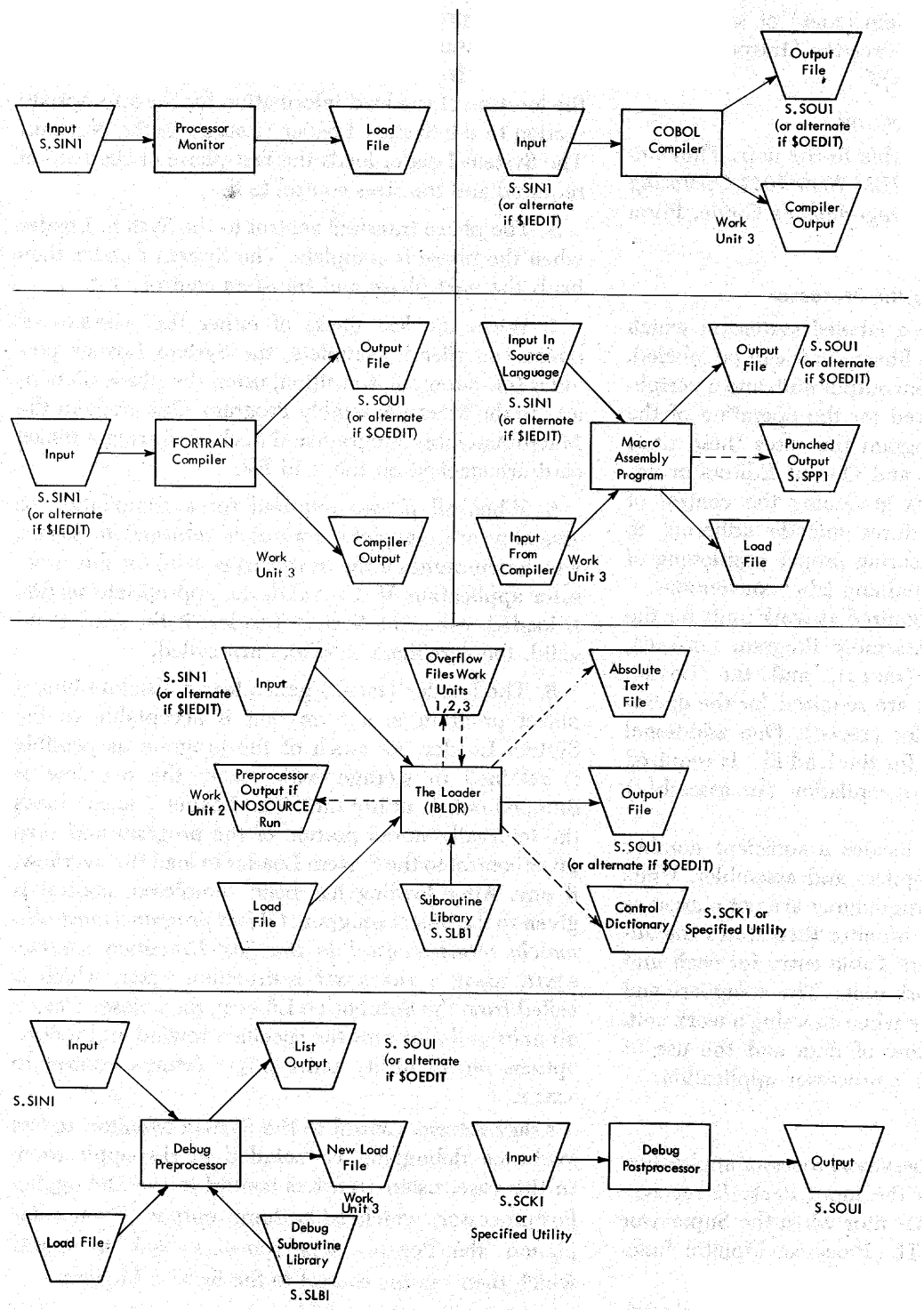


Figure 18. Flow of Input/Output Data During a Processor Application

```
        CLA     x
        TSX     S.SCCR,4
        TMT     14
        TSX     S.SCCR,4
        MSM     S.SCDI
```

where x is the location of the following coding:

```
x       PZE     S.SAVE,,*+1
        BSS     14   (The $ card must be here.)
```

Control Cards

After receiving control from IBSYS, or at the initialization of any processor application, the Processor Monitor examines the units tables in IBNUC and from them assigns the proper input/output units for the files used by the system. Then, during the processor application, the actions of the Operating System are controlled by the following control cards:

CONTROL CARD	FUNCTION
$ID	Causes control to pass to installation accounting routines.
$*	Serves as a comment card.
$PAUSE	Allows for operator action.
$IBSYS	Indicates that the next application falls outside the scope of the Processor and that control must be returned to the System Monitor.
$IBJOB	Initiates a new processor application.
$IBFTC	Indicates that a source program in FORTRAN language is to be processed.
$IBCBC	Indicates that a source program in COBOL language is to be processed.
$CBEND	Indicates the end of a COBOL source deck.
$IBMAP	Indicates that a MAP symbolic language source program is to be processed.
$IBDBC	Indicates that debugging of a COBOL program is to be performed by the COBOL Compiler in this application.
$IBDBL	Indicates that debugging of a FORTRAN or MAP program is to be performed by the Debugging Processor in this application. The debugging requests follow this card.
$DEND	Indicates the end of a deck of debugging requests for the Debugging Processor.
$IBREL	Indicates that all succeeding decks for this processor application are relocatable binary decks and are not to be stacked on the load file.
$IBLDR	Indicates that a relocatable binary deck follows.
$ENTRY	Indicates the end of the information to be included in a single core storage load.
$FILE $LABEL $POOL $USE $OMIT $NAME	Contains information that must be processed by the Preprocessor, a part of Processor Monitor. The information is reduced to a binary form for the relocatable Loader, IBLDR. These cards are described in the section on the Loader.
$CHAIN $LINK $ENDCH	Used with the CHAIN feature to form a multiphase program consisting of a main link and one or more dependent links that are processed to form several core storage loads. These cards are described in the section on the CHAIN feature.
$RELOAD	Used to initiate loading of absolute object programs that were produced by the Loader.

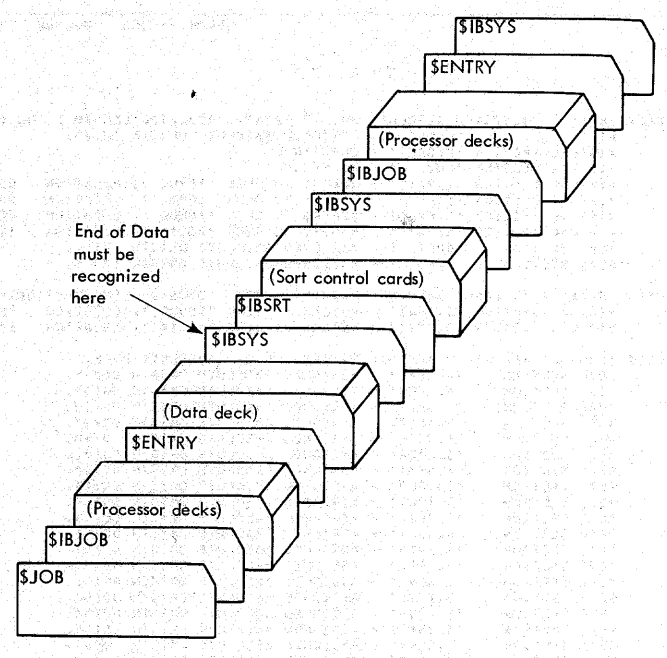


Figure 19. End of Data Recognition in the System Input File

SYSTEM MONITOR CARDS RECOGNIZED BY THE PROCESSOR MONITOR

Five control cards recognized by the System Monitor are also recognized by the Processor Monitor; however, their effect is the same in either case. These cards cause the Supervisor to be loaded:

$ID
$*
$PAUSE
$IBSYS
$IBJOB

The control cards, $ID, $*, or $PAUSE, may appear after the $IBJOB card and before any $IBMAP, $IBFTC, $IBCBC, or $IBLDR card for the application; however, they may not be intermixed with $FILE, $LABEL, $NAME, $USE, $OMIT, or $POOL cards.

PROCESSOR APPLICATION INITIALIZATION CARD

The $IBJOB Card: The format of this card is:

```
1           8           16
$IBJOB      progname    options
```

The $IBJOB control card must appear first in any application to be processed, and it can appear only once in each processor application. This card signals a new processor application and causes all control cards that follow it and that precede the next major control card to be processed by the Processor Monitor. The options on this card describe the manner in which an application is to be processed.

```
                                        IBLDR -- JOB   000000

                        L O G I C   M A P


DECK 'NAME  ' ASSIGNED ABSOLUTE ORIGIN 12247.  ADJUSTED LENGTH IS 00150.
     FILE          'DECDC  ' - ASSIGNED ABSOLUTE ORIGIN 12224.
     VIRTUAL SECTION 'BEGIN ' - UNDEFINED.
     VIRTUAL SECTION 'END   ' - UNDEFINED.
     VIRTUAL SECTION 'S.PUTL' - REFERS TO DECK 'IBNUC ', LOCATION   00164.
     VIRTUAL SECTION 'S.CLSE' - REFERS TO DECK 'IBNUC ', LOCATION   00167.
     VIRTUAL SECTION 'S.OPEN' - REFERS TO DECK 'IBNUC ', LOCATION   00160.
     VIRTUAL SECTION 'S.JXIT' - REFERS TO DECK 'POSTX ', LOCATION   12565.
     REAL SECTION    'OPFIL ' - ASSIGNED ABSOLUTE ORIGIN 12304.
     REAL SECTION    'BINOCT' - ASSIGNED ABSOLUTE ORIGIN 12376.

DECK 'BETA  ' ASSIGNED ABSOLUTE ORIGIN 12417.  ADJUSTED LENGTH IS 00146.
     VIRTUAL SECTION 'S.SDAT' - REFERS TO DECK 'IBNUC ', LOCATION   00211.
     VIRTUAL SECTION 'S.JXIT' - REFERS TO DECK 'POSTX ', LOCATION   12565.

DECK 'IBNUC ' ASSIGNED ABSOLUTE ORIGIN 00000.  ABSOLUTE DECK.
     REAL SECTION    'S.SUTL' - ASSIGNED ABSOLUTE ORIGIN 00055.
     REAL SECTION    'S.SLDR' - ASSIGNED ABSOLUTE ORIGIN 00135.
     REAL SECTION    'S.SRET' - ASSIGNED ABSOLUTE ORIGIN 00136.
     REAL SECTION    'S.SDMP' - ASSIGNED ABSOLUTE ORIGIN 00137.
     REAL SECTION    'S.SCCR' - ASSIGNED ABSOLUTE ORIGIN 00140.
     REAL SECTION    'S.SIDR' - ASSIGNED ABSOLUTE ORIGIN 00141.
     REAL SECTION    'S.SRST' - ASSIGNED ABSOLUTE ORIGIN 00142.
     REAL SECTION    'S.SRUP' - ASSIGNED ABSOLUTE ORIGIN 00143.
     REAL SECTION    'S.SRPT' - ASSIGNED ABSOLUTE ORIGIN 00144.
     REAL SECTION    'S.SCEM' - ASSIGNED ABSOLUTE ORIGIN 00145.
     REAL SECTION    'S.XACT' - ASSIGNED ABSOLUTE ORIGIN 00146.
     REAL SECTION    'S.XPRT' - ASSIGNED ABSOLUTE ORIGIN 00147.
     REAL SECTION    'S.XPSE' - ASSIGNED ABSOLUTE ORIGIN 00150.
     REAL SECTION    'S.XOVA' - ASSIGNED ABSOLUTE ORIGIN 00151.
     REAL SECTION    'S.XOVD' - ASSIGNED ABSOLUTE ORIGIN 00152.
     REAL SECTION    'S.XDVA' - ASSIGNED ABSOLUTE ORIGIN 00153.
     REAL SECTION    'S.XDVD' - ASSIGNED ABSOLUTE ORIGIN 00154.
     REAL SECTION    'S.SCKT' - ASSIGNED ABSOLUTE ORIGIN 00155.
     REAL SECTION    'S.IOOP' - ASSIGNED ABSOLUTE ORIGIN 00156.
     REAL SECTION    'S.IOLS' - ASSIGNED ABSOLUTE ORIGIN 00157.
     REAL SECTION    'S.OPEN' - ASSIGNED ABSOLUTE ORIGIN 00160.
     REAL SECTION    'S.OPNL' - ASSIGNED ABSOLUTE ORIGIN 00161.
     REAL SECTION    'S.GETL' - ASSIGNED ABSOLUTE ORIGIN 00162.
     REAL SECTION    'S.GETB' - ASSIGNED ABSOLUTE ORIGIN 00163.
     REAL SECTION    'S.PUTL' - ASSIGNED ABSOLUTE ORIGIN 00164.
     REAL SECTION    'S.PUTB' - ASSIGNED ABSOLUTE ORIGIN 00165.
     REAL SECTION    'S.PLOC' - ASSIGNED ABSOLUTE ORIGIN 00166.
     REAL SECTION    'S.CLSE' - ASSIGNED ABSOLUTE ORIGIN 00167.
     REAL SECTION    'S.CLSL' - ASSIGNED ABSOLUTE ORIGIN 00170.
     REAL SECTION    'S.BSR ' - ASSIGNED ABSOLUTE ORIGIN 00171.
     REAL SECTION    'S.WEF ' - ASSIGNED ABSOLUTE ORIGIN 00172.
     REAL SECTION    'S.REW ' - ASSIGNED ABSOLUTE ORIGIN 00173.
     REAL SECTION    'S.FEOR' - ASSIGNED ABSOLUTE ORIGIN 00174.
     REAL SECTION    'S.CKPT' - ASSIGNED ABSOLUTE ORIGIN 00175.
     REAL SECTION    'S.SLVL' - ASSIGNED ABSOLUTE ORIGIN 00176.
     REAL SECTION    'S.SCOR' - ASSIGNED ABSOLUTE ORIGIN 00177.
     REAL SECTION    'S.SCSN' - ASSIGNED ABSOLUTE ORIGIN 00200.
     REAL SECTION    'S.SPND' - ASSIGNED ABSOLUTE ORIGIN 00201.
     REAL SECTION    'S.SCMX' - ASSIGNED ABSOLUTE ORIGIN 00202.
     REAL SECTION    'S.SPER' - ASSIGNED ABSOLUTE ORIGIN 00203.
     REAL SECTION    'S.SUBC' - ASSIGNED ABSOLUTE ORIGIN 00204.
     REAL SECTION    'S.SSBC' - ASSIGNED ABSOLUTE ORIGIN 00205.
     REAL SECTION    'S.SUNI' - ASSIGNED ABSOLUTE ORIGIN 00206.
     REAL SECTION    'S.SLTC' - ASSIGNED ABSOLUTE ORIGIN 00207.
     REAL SECTION    'S.SRCC' - ASSIGNED ABSOLUTE ORIGIN 00210.
     REAL SECTION    'S.SDAT' - ASSIGNED ABSOLUTE ORIGIN 00211.
     REAL SECTION    'S.SCLK' - ASSIGNED ABSOLUTE ORIGIN 00213.
     REAL SECTION    'S.SCIS' - ASSIGNED ABSOLUTE ORIGIN 00214.
     REAL SECTION    'S.SDEX' - ASSIGNED ABSOLUTE ORIGIN 00215.
     REAL SECTION    'S.SCUR' - ASSIGNED ABSOLUTE ORIGIN 00217.
     REAL SECTION    'S.SFAZ' - ASSIGNED ABSOLUTE ORIGIN 00220.
     REAL SECTION    'S.SSWI' - ASSIGNED ABSOLUTE ORIGIN 00221.
     REAL SECTION    'S.SFLG' - ASSIGNED ABSOLUTE ORIGIN 00222.
     REAL SECTION    'S.SAVE' - ASSIGNED ABSOLUTE ORIGIN 00246.
     REAL SECTION    'S.SCDI' - ASSIGNED ABSOLUTE ORIGIN 00264.
     REAL SECTION    'S.PGCT' - ASSIGNED ABSOLUTE ORIGIN 00265.
     REAL SECTION    'S.SHDR' - ASSIGNED ABSOLUTE ORIGIN 00266.
     REAL SECTION    'S.SSCH' - ASSIGNED ABSOLUTE ORIGIN 00273.
     REAL SECTION    'S.SSNS' - ASSIGNED ABSOLUTE ORIGIN 00274.
     REAL SECTION    'S.SCBL' - ASSIGNED ABSOLUTE ORIGIN 00274.
     REAL SECTION    'S.XTDT' - ASSIGNED ABSOLUTE ORIGIN 00275.
     REAL SECTION    'S.XSNS' - ASSIGNED ABSOLUTE ORIGIN 00276.
     REAL SECTION    'S.XLTP' - ASSIGNED ABSOLUTE ORIGIN 00300.
     REAL SECTION    'S.XSCH' - ASSIGNED ABSOLUTE ORIGIN 00301.
     REAL SECTION    'S.XTPS' - ASSIGNED ABSOLUTE ORIGIN 00302.
     REAL SECTION    'S.XCPS' - ASSIGNED ABSOLUTE ORIGIN 00303.
     REAL SECTION    'S.NAPT' - ASSIGNED ABSOLUTE ORIGIN 00304.
     REAL SECTION    'S.SFBL' - ASSIGNED ABSOLUTE ORIGIN 00305.
     REAL SECTION    'S.JNAM' - ASSIGNED ABSOLUTE ORIGIN 00307.
     REAL SECTION    'S.IAUN' - ASSIGNED ABSOLUTE ORIGIN 00310.
     REAL SECTION    'S.OAUN' - ASSIGNED ABSOLUTE ORIGIN 00311.
     REAL SECTION    'S.LOUN' - ASSIGNED ABSOLUTE ORIGIN 00312.
     REAL SECTION    'S.EOUN' - ASSIGNED ABSOLUTE ORIGIN 00313.
     REAL SECTION    'S.SRUS' - ASSIGNED ABSOLUTE ORIGIN 00314.
     REAL SECTION    'S.SPRP' - ASSIGNED ABSOLUTE ORIGIN 00326.
     REAL SECTION    'S.SLB1' - ASSIGNED ABSOLUTE ORIGIN 00327.
     REAL SECTION    'S.SLB2' - ASSIGNED ABSOLUTE ORIGIN 00330.
     REAL SECTION    'S.SIN1' - ASSIGNED ABSOLUTE ORIGIN 00331.
     REAL SECTION    'S.SIN2' - ASSIGNED ABSOLUTE ORIGIN 00332.
     REAL SECTION    'S.SOU1' - ASSIGNED ABSOLUTE ORIGIN 00333.
     REAL SECTION    'S.SOU2' - ASSIGNED ABSOLUTE ORIGIN 00334.
     REAL SECTION    'S.SPP1' - ASSIGNED ABSOLUTE ORIGIN 00335.
     REAL SECTION    'S.SPP2' - ASSIGNED ABSOLUTE ORIGIN 00336.
     REAL SECTION    'S.SCK1' - ASSIGNED ABSOLUTE ORIGIN 00337.
     REAL SECTION    'S.SU00' - ASSIGNED ABSOLUTE ORIGIN 00340.
     REAL SECTION    'S.SU01' - ASSIGNED ABSOLUTE ORIGIN 00341.
     REAL SECTION    'S.SU02' - ASSIGNED ABSOLUTE ORIGIN 00342.
     REAL SECTION    'S.SU03' - ASSIGNED ABSOLUTE ORIGIN 00343.
     REAL SECTION    'S.SU04' - ASSIGNED ABSOLUTE ORIGIN 00344.
     REAL SECTION    'S.SU05' - ASSIGNED ABSOLUTE ORIGIN 00345.
     REAL SECTION    'S.SU07' - ASSIGNED ABSOLUTE ORIGIN 00347.
     REAL SECTION    'S.SU08' - ASSIGNED ABSOLUTE ORIGIN 00350.
     REAL SECTION    'S.SU09' - ASSIGNED ABSOLUTE ORIGIN 00351.
     REAL SECTION    'S.SU10' - ASSIGNED ABSOLUTE ORIGIN 00352.
     REAL SECTION    'S.SU11' - ASSIGNED ABSOLUTE ORIGIN 00353.
     REAL SECTION    'S.SU12' - ASSIGNED ABSOLUTE ORIGIN 00354.
     REAL SECTION    'S.SU13' - ASSIGNED ABSOLUTE ORIGIN 00355.
     REAL SECTION    'S.SU14' - ASSIGNED ABSOLUTE ORIGIN 00356.
     REAL SECTION    'S.SU15' - ASSIGNED ABSOLUTE ORIGIN 00357.
     REAL SECTION    'S.SU16' - ASSIGNED ABSOLUTE ORIGIN 00360.
     REAL SECTION    'S.SU17' - ASSIGNED ABSOLUTE ORIGIN 00361.
     REAL SECTION    'S.SU18' - ASSIGNED ABSOLUTE ORIGIN 00362.
     REAL SECTION    'S.SU19' - ASSIGNED ABSOLUTE ORIGIN 00363.
     REAL SECTION    'S.SU20' - ASSIGNED ABSOLUTE ORIGIN 00364.
     REAL SECTION    'S.SU21' - ASSIGNED ABSOLUTE ORIGIN 00365.
     REAL SECTION    'S.SU22' - ASSIGNED ABSOLUTE ORIGIN 00366.
     REAL SECTION    'S.SU23' - ASSIGNED ABSOLUTE ORIGIN 00367.
     REAL SECTION    'S.SU24' - ASSIGNED ABSOLUTE ORIGIN 00370.
     REAL SECTION    'S.SU25' - ASSIGNED ABSOLUTE ORIGIN 00371.
     REAL SECTION    'S.SU26' - ASSIGNED ABSOLUTE ORIGIN 00372.
     REAL SECTION    'S.SU27' - ASSIGNED ABSOLUTE ORIGIN 00373.
     REAL SECTION    'S.SU28' - ASSIGNED ABSOLUTE ORIGIN 00374.
     REAL SECTION    'S.SU29' - ASSIGNED ABSOLUTE ORIGIN 00375.
     REAL SECTION    'S.SU30' - ASSIGNED ABSOLUTE ORIGIN 00376.
     REAL SECTION    'S.SU31' - ASSIGNED ABSOLUTE ORIGIN 00377.
     REAL SECTION    'S.SU32' - ASSIGNED ABSOLUTE ORIGIN 00400.
     REAL SECTION    'S.SU33' - ASSIGNED ABSOLUTE ORIGIN 00401.
     REAL SECTION    'S.SU34' - ASSIGNED ABSOLUTE ORIGIN 00402.
     REAL SECTION    'S.SU35' - ASSIGNED ABSOLUTE ORIGIN 00403.
     REAL SECTION    'S.SUTT' - ASSIGNED ABSOLUTE ORIGIN 00404.
     REAL SECTION    'S.UCBL' - ASSIGNED ABSOLUTE ORIGIN 00405.
     REAL SECTION    'S.UCBT' - ASSIGNED ABSOLUTE ORIGIN 01011.
     REAL SECTION    'S.SCBT' - ASSIGNED ABSOLUTE ORIGIN 01305.
     REAL SECTION    'S.SORG' - ASSIGNED ABSOLUTE ORIGIN 03000.
     REAL SECTION    'S.SP1D' - ASSIGNED ABSOLUTE ORIGIN 10000.
     REAL SECTION    'S.SP2D' - ASSIGNED ABSOLUTE ORIGIN 10000.
     REAL SECTION    'S.SLND' - ASSIGNED ABSOLUTE ORIGIN 10364.
     REAL SECTION    'S.SSND' - ASSIGNED ABSOLUTE ORIGIN 12224.
     REAL SECTION    'S.SEND' - ASSIGNED ABSOLUTE ORIGIN 77777.

DECK 'POSTX ' ASSIGNED ABSOLUTE ORIGIN 12565.  ADJUSTED LENGTH IS 00112.
     VIRTUAL SECTION 'S.SFLG' - REFERS TO DECK 'IBNUC ', LOCATION   00222.
     VIRTUAL SECTION 'S.SRET' - REFERS TO DECK 'IBNUC ', LOCATION   00136.
     VIRTUAL SECTION 'S.SCUR' - REFERS TO DECK 'IBNUC ', LOCATION   00217.
     VIRTUAL SECTION 'S.SFBL' - REFERS TO DECK 'IBNUC ', LOCATION   00305.
     VIRTUAL SECTION 'S.CLSE' - REFERS TO DECK 'IBNUC ', LOCATION   00167.
     VIRTUAL SECTION 'S.SCCR' - REFERS TO DECK 'IBNUC ', LOCATION   00140.
     VIRTUAL SECTION 'S.SIN1' - REFERS TO DECK 'IBNUC ', LOCATION   00331.
     VIRTUAL SECTION 'S.SSWI' - REFERS TO DECK 'IBNUC ', LOCATION   00221.
     VIRTUAL SECTION 'S.SOU1' - REFERS TO DECK 'IBNUC ', LOCATION   00333.
     VIRTUAL SECTION 'S.SIDR' - REFERS TO DECK 'IBNUC ', LOCATION   00141.
     VIRTUAL SECTION 'S.SPP1' - REFERS TO DECK 'IBNUC ', LOCATION   00335.
     REAL SECTION    'S.JXIT' - ASSIGNED ABSOLUTE ORIGIN 12565.
```

Figure 20. Portion of a LOGIC Listing Produced by Specifying the LOGIC Option on the $IBJOB Card

The content of columns 8-13 is the program name, which must consist of six or fewer alphameric characters.

The content of the variable field is:

[,{GO / NOGO}]

The Execution options: The GO option specifies that the processor application that follows is to be executed after it has been successfully loaded.

The NOGO option specifies that the processor application that follows is not to be executed. This option suppresses loading if neither LOGIC nor MAP (see below) is specified.

If neither GO nor NOGO is specified, GO is assumed.

[,{LOGIC / DLOGIC / NOLOGIC}]

The Logic options: The LOGIC option specifies that a detailed storage-allocation list of the object program is to be produced on the system output unit by the Loader (IBLDR). The list consists of the origin and extent of all object program control sections, including Subroutine Library sections. The presence of this option indicates that the object program is to be processed by the Loader. However, execution takes place only when the execution option specifies GO.

The DLOGIC option specifies that a detailed storage-allocation list of the input object program is to be produced on the system output unit by the Loader (IBLDR). The list indicates the origin and extent of all input object-program control sections and object-time files. This option is similar to the LOGIC option except that Subroutine Library sections are not included in the list.

The NOLOGIC option specifies that the LOGIC list is not to be produced.

If neither LOGIC, DLOGIC, nor NOLOGIC is specified, NOLOGIC is assumed.

A sample logic listing is shown in Figure 20. The name of each control section, including file control blocks, is listed, and the absolute location assigned to each is shown.

[,{MAP / NOMAP}]

The Map options: The MAP option specifies that a non-detailed storage allocation map of the object program is to be produced on the system output unit by the Loader (IBLDR). This list shows the origin and extent of all object program decks, including Subroutine Library decks and buffer pool assignments. The presence of this parameter causes the Loader to process an object program even if the execution option specifies NOGO.

The NOMAP option specifies that the storage map is to be suppressed.

If neither MAP nor NOMAP is specified, NOMAP is assumed. A sample MAP listing is shown in Figure 21; each program deck and its length are given.

[,{FILES / NOFILES}]

The File List options: The FILES option specifies that a list of object program files and their unit assignments is to be produced on the system output unit.

The NOFILES option specifies that the file list is to be suppressed.

If neither FILES nor NOFILES is specified, NOFILES is assumed.

[,{IOOP1 / IOOP2 / IOLS / IOBS}]

The Object-Time IOCS options: The IOOP1 option specifies that the minimum IOCS package is to be used with this object program. The origin assigned to the object program by the Loader is the first available non-storage-protected location after IOOP1.

The IOOP2 option specifies that the second level of IOCS is to be used with this object program. The origin assigned to the object program by the Loader is the first available non-storage-protected location after IOOP2. IOOP2 includes IOOP1.

The IOLS option specifies that the label package of IOCS is to be used with this object program. The origin assigned to the object program by the Loader is the first available non-storage-protected location after IOLS. IOLS includes IOOP2.

The IOBS option specifies that the IOCS buffering package is to be used with this object program. The origin assigned to the object program by the Loader is the first available location after IOBS. IOBS includes IOLS.

If neither IOOP1, IOOP2, IOLS, nor IOBS is specified, IOBS is assumed.

IOOP2 and portions of IOLS may be storage-protected, depending upon the size of the Nucleus and IOOP1. Object programs may not be loaded within the storage-protected area.

[,{SOURCE / NOSOURCE}]

The Program Stacking options: The SOURCE option specifies that the following processor application includes decks to be

```
                                    IBLDR -- JOB   000000

                        M E M O R Y   M A P

SYSTEM, INCLUDING ICCS                              00000 THRU  12223

FILE BLOCK ORIGIN                                   12224

     NUMBER OF FILES -      1

     1.  DECOC                      12224

OBJECT PROGRAM                                      12247 THRU  12676

     1.  DECK 'NAME  '              12247
     2.  DECK 'BETA  '              12417
     3.  SUBR 'POSTX '              12565

   (* - INSERTIONS OR DELETIONS MADE IN THIS DECK)

INPUT - OUTPUT BUFFERS                              77757 THRU  77776

UNUSED CORE                                         12677 THRU  77754
```

Figure 21. Sample MAP Listing Produced by Specifying the MAP Option on the $IBJOB Card

processed by the compilers and the assembler. The presence of this option with GO, MAP, or LOGIC causes all relocatable output from the compilers to be stacked on the load file along with the relocatable decks for the processor application. (Those relocatable decks following a $IBREL card are not stacked. See $IBREL below.)

The NOSOURCE option indicates that no source decks appear in the processor application to be processed. The Loader processes its input directly from the system input file if this option is used.

If neither SOURCE nor NOSOURCE is specified, SOURCE is assumed.

[, { DECK | NODECK }]

The Relocatable Deck options: The DECK option specifies that relocatable decks are to be produced by the compilers and the assembler. The presence of this option on the $IBJOB card does not override the DECK option on the compiler cards. The DECK option on the $IBJOB card assures that a punch file is available to the compilers and the assembler.

The NODECK option specifies that relocatable decks are not to be produced by the compilers and the assembler. If this option is chosen and DECK is specified on the compiler cards, no relocatable decks are produced.

If neither DECK nor NODECK is specified, DECK is assumed.

[, { COBOL | NOCOBOL }]

The COBOL options: The COBOL option specifies that this processor application includes a source deck to be processed by the COBOL Compiler (IBCBC). If the COBOL option does not appear and if a $IBCBC card is encountered by the monitor, the deck associated with the $IBCBC card will not be processed.

The NOCOBOL option specifies that this processor application does not include a deck to be processed by the COBOL Compiler (IBCBC).

If neither COBOL nor NOCOBOL is specified, NOCOBOL is assumed.

Note that the effect of the COBOL option is to ensure that the additional utility unit required by COBOL is available.

[, { COPY | COPY = Iyy | COPY = unit [=Iyy] | NOCOPY }]

The Copy options: The COPY option specifies that the Loader is to produce the absolute object-program file on an available unit. A message is typed to notify the operator of the unit chosen, and the unit is rewound and unloaded after the file is created.

The COPY=Iyy option specifies that the Loader is to produce the absolute object-program file on the unit that has been assigned the intersystem reservation code yy. Since the unit is not rewound, several programs within one job can be stacked on the unit.

The COPY=*unit* option specifies that the Loader is to produce the absolute object-program file on one of the following:

1. The utility unit, S.SUxx (COPY=Uxx)
2. The first available unit (COPY=U)
3. The first available tape unit (COPY=T)
4. The first available disk or drum unit (COPY=D)

The operator is notified of the chosen unit.

The COPY=*unit*=Iyy option is similar to the preceding option. The chosen unit is assigned the intersystem reservation code yy.

If the Loader cannot assign a specified Copy unit (for example, if the specification is invalid or if the unit is unavailable), the unit specification is disregarded. The option is then processed in the same way as the COPY option.

If the file on the Copy unit is to be labeled, the programmer may include a special $LABEL card with the other Preprocessor cards. The file name on the $LABEL card must be S.FBCP.

The NOCOPY option specifies that a copy of the absolute object-program file is not desired.

NOTES:

1. Use of the Copy feature does not delete execution of the copied program. It may not be used during an Edit run or when an application contains load-time debugging requests.
2. If assignment of an intersystem reservation code is not specified, the Copy unit will be reserved by the system and will not become available for other assignments until either a $CLOSE or a $JOB card is encountered.

In the following $IBJOB card, the NOGO and LOGIC options are specified in the variable field. Thus, execution is deleted and a detailed storage allocation list of the object program is produced on the system output unit. All of the underlined options are assumed except GO and NOLOGIC.

```
1               8               16
$IBJOB      PRONAM   NOGO, LOGIC
```

LOADER INPUT FILE CARD

$IBREL Card: The format of this card is:

```
1               8               16
$IBREL
```

This control card specifies that the decks following it for this application are all relocatable binary decks. This card is meaningful only in a processor application that contains both source language decks and relocatable binary decks. The deck name field and the variable field of this card are not significant. This card discontinues the stacking of the load file and causes the Loader to take its input from both the load file and the system input file.

The following is an example of a $IBREL card:

```
1               8               16
$IBREL
```

The relocatable decks following this card are not placed on the load file. The Loader will take its input for these decks from the system input file.

INPUT AND OUTPUT EDITOR CONTROL CARDS

The presence of Input or Output Editor control cards indicates that alternate units are to be used for input or output, respectively, for the compilers, the assembler, and the Loader. Either card overrides the effect of any card of the same type that may have preceded it in the application. The absence of these cards indicates that all Processor input is on the system input unit (S.SINX) and that all Processor output is to be produced on the system output unit (S.SOUX).

A $IEDIT or $OEDIT card cannot be placed following a $IBREL card or after a relocatable deck in a NOSOURCE application. The encounter of a $IBDBL card terminates $IEDIT control; the end of Loader processing automatically terminates both $IEDIT and $OEDIT control. The system input and system output units are then used for input/output.

$IEDIT Card: The format of this card is:

1	8	16
$IEDIT		variable field

The contents of the variable field are:

[{ IN / Uxx / Iyy [R] }]

The Input options: The IN option specifies that all compiler, assembler, and/or Loader input is on the system input unit.

The Uxx option specifies that all compiler, assembler, and/or Loader input is on the utility unit, S.SUxx. The unit should be reserved by the programmer to prevent its use as a compiler or assembler work unit.

The Iyy option specifies that all compiler, assembler, and/or Loader input is on the unit that has been assigned the intersystem reservation code yy. Appending an R to this specification indicates that the unit is to be released after use.

If none of these options is specified, IN is assumed.

[, { SRCH / NOSRCH }]

The File Position options: The SRCH option specifies that the alternate unit specified by the input option is to be searched for a compiler, assembler, or Loader control card whose deck name matches that of the corresponding control card on the system input file.

The NOSRCH option specifies that the alternate unit is positioned exactly at the beginning of the identifying control card of the desired deck and should not be rewound. In case the alternate unit is labeled and the desired unit is at the beginning, care should be taken to specify REWIND, so that label checking procedures will take place.

If neither SRCH nor NOSRCH is specified, NOSRCH is assumed.

[, { REWIND / NOREWIND }]

The Rewind options: The REWIND option specifies that the alternate unit indicated by the input option is to be rewound before it is examined for a matching deck.

The NOREWIND option specifies that the alternate unit indicated by the input option is not to be rewound before it is examined for a matching deck.

If neither REWIND nor NOREWIND is specified, NOREWIND is assumed.

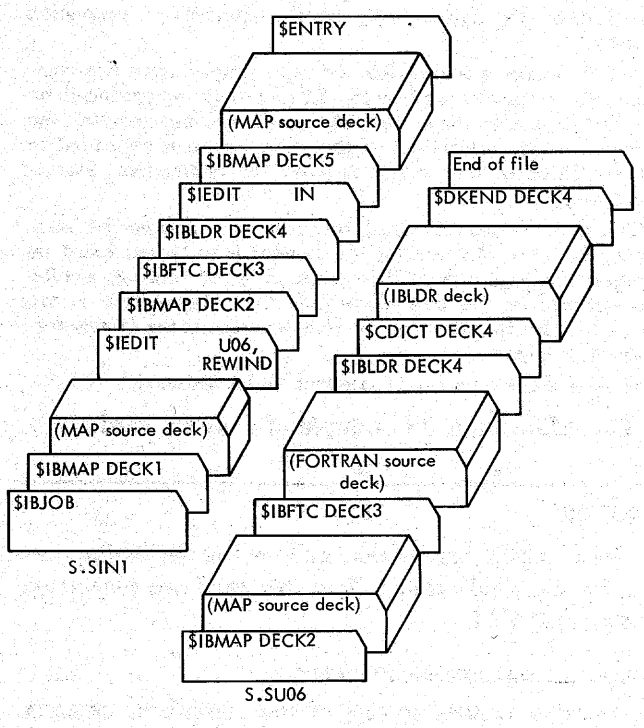
The Rewind options are not effective if IN is specified.

The following is an example of $IEDIT card:

1	8	16
$IEDIT		U06

This card indicates that utility unit U06 is used as an alternate input unit. Since no other option is specified, the standard NOSRCH option is chosen; thus, utility unit U06 will not be rewound.

An example of a deck on S.SIN1 that uses $IEDIT cards follows:



When processing this application, DECK1 is read in from the system input unit. DECK2, DECK3, and DECK4 are read in from utility unit U06, as indicated by the first $IEDIT card. The options for these decks are taken from the control cards on S.SIN1. DECK5 is read in from the system input unit, as indicated by the second $IEDIT card.

$OEDIT Card: The format of this card is:

1	8	16
$OEDIT		variable field

The contents of the variable field are:

[{ OU / Iyy / Uxx [=Iyy] / U [=Iyy] / T [=Iyy] / D [=Iyy] }]

The Output options: The OU option specifies that the print output from the compilers, the assembler, and/or the Loader is to be produced on the system output unit.

The Iyy option specifies that the print output from the compilers, the assembler, and/or the Loader is to be produced on the unit that has been assigned the intersystem reservation code yy.

The Uxx option specifies that the print output from the compilers, the assembler, and/or the Loader is to be produced on the utility unit, S.SUxx. The unit is rewound. The unit should be reserved by the programmer to prevent its use as a compiler or assembler work unit. If the =Iyy option is appended to this specification, the unit is assigned the intersystem reservation code yy.

The U option specifies that the print output from the compilers, the assembler, and/or the Loader is to be produced on the first available unit. A typed message notifies the operator of the unit chosen. If the =Iyy option is appended to this specification, the unit is assigned the intersystem reservation code yy.

The T option specifies that the print output from the compilers, the assembler, and/or the Loader is to be produced on the first available tape unit. A typed message notifies the operator of the unit chosen. If the =Iyy option is appended to this specification, the unit is assigned the intersystem reservation code yy.

The D option specifies that the print output from the compilers, the assembler, and/or the Loader is to be produced on the first available disk or drum unit. A typed message notifies the operator of the unit chosen. If the =Iyy option is appended to this specification, the unit is assigned the intersystem reservation code yy.

If none of these options is specified, OU is assumed.

The following is an example of a $OEDIT card:

```
1              8              16
$OEDIT                        U14
```

The compiler, assembler, and/or Loader listing output for the decks that follow this card are placed on utility unit U14.

COMPILER AND ASSEMBLER CARDS

Transfer of control to one of the compilers, IBCBC or IBFTC, or to the assembler, IBMAP, is directed by the control cards described below.

The deck name, which is composed of six or fewer characters, may appear, left-justified, in columns 8 through 13 of the $IBCBC, $IBFTC, or $IBMAP cards. The deck name is required if debugging is to be performed during the application. Compiler and assembler output is labeled with this deck name. (A $IBLDR card is generated with the appropriate deck name in the deck name field.)

$IBFTC Card: The format of this card is:

```
1              8              16
$IBFTC         deck name      variable field
```

When this card is encountered, the FORTRAN Compiler is called to process a FORTRAN source deck.

The contents of the variable field are:

```
[{ LIST   }]
[{ FULIST }]
[{ NOLIST }]
```

The List options: The LIST option produces an abbreviated three-column assembly program listing in the output file.

The FULIST option produces the standard one-column assembly program listing in the output file.

The NOLIST option suppresses the assembly program listing.

If none of these options is specified, NOLIST is assumed.

```
[,{ DECK   }]
[ {NODECK  }]
```

The Punch options: The DECK option causes a relocatable deck to be produced with the system punch output.

The NODECK option suppresses the production of a deck.

If neither option is specified, DECK is assumed unless NODECK is specified on the $IBJOB card.

```
[{ REF   }]
[{ NOREF }]
```

The Cross-Reference Table options: The REF option produces an alphabetic listing of all the symbols used in the deck, together with cross references to the statements in the deck which use each symbol. This cross-reference table appears immediately after the assembly program listing.

The NOREF option suppresses the listing of the symbolic cross-reference table.

If neither option is specified, NOREF is assumed:

```
[{ DD   }]
[{ SDD  }]
[{ NODD }]
```

The Debugging Dictionary options: The DD option causes the production of a full debugging dictionary. It contains all statement numbers and all symbols (both those specified by the programmer and those generated by the Compiler) in the FORTRAN program.

The SDD option causes the production of an abbreviated debugging dictionary. It contains all statement numbers and all variable names specified by the programmer in the FORTRAN program.

The NODD option suppresses the production of a debugging dictionary.

If none of these options is specified, NODD is assumed.

The following is an example of a $IBFTC card:

```
1              8              16
$IBFTC         BETA           LIST
```

Since LIST is specified, an abbreviated assembly program listing in a three-column format is produced in the output file for this deck.

$IBMAP Card: The format of the card is:

```
1              8              16
$IBMAP         deck name      variable field
```

When this card is encountered, the Macro Assembly Program is called to process a MAP source deck.

The contents of the variable field are:

```
[{ LIST   }]
[{ NOLIST }]
```

The List options: The LIST option produces an assembly program listing in the output file.

The NOLIST option suppresses the assembly program listing.

If neither option is specified, LIST is assumed.

```
[,{ DECK   }]
[ {NODECK  }]
```

The Punch options: The DECK option causes a relocatable deck to be produced with the system punch output.

The NODECK option suppresses the production of a deck.

If neither option is specified, DECK is assumed unless NODECK is specified on the $IBJOB card.

```
[,{ REF   }]
[ {NOREF  }]
```

The Cross-Reference Table options: The REF option produces an alphabetic listing of all the symbols used in the deck, together with cross references to the statements in the deck which use each symbol. This cross-reference table appears immediately after the assembly program listing.

The NOREF option suppresses the listing of the symbolic cross-reference table.

If neither option is specified, REF is assumed unless NOLIST is specified.

[,SYMSIZ=xxxxx]

The symbol xxxxx is a decimal integer signifying the number of locations to be reserved for the Macro Assembly Program Symbol Table.

The size specified for the symbol table must be at least as large as the sum of the numbers of symbols in the program, the number of unique literals, and the number of macro definitions. An increase in the size of the Symbol Table implies a comparable reduction in size of the Macro Definition Table and vice versa.

In the absence of this option, the Symbol Table is a predetermined size (set by an assembly parameter within the Macro Assembly Program). The option should only be used for programs containing either an abnormally large number of symbols or of macro definitions which would otherwise result in table overflow during assembly. The use of this option to reduce the size of the Symbol Table may result in an appreciable reduction in speed of assembly.

[, { RELMOD / ABSMOD }]

Assembly Relocation Mode option: The RELMOD option causes the assembly to be in relocatable mode.

The ABSMOD option causes the assembly to be in absolute mode. If neither RELMOD nor ABSMOD is specified, RELMOD is assumed.

[{ DD / SDD / NODD }]

The Debugging Dictionary options: The DD option causes the production of a full debugging dictionary. It contains all the symbols in the MAP source program.

The SDD option causes the production of an abbreviated debugging dictionary. It contains only those symbols specified by KEEP pseudo-operations in the MAP source program.

The NODD option suppresses the production of a debugging dictionary.

If none of these options is specified, NODD is assumed.

The following is an example of a $IBMAP card:

```
1          8          16
$IBMAP     GAMMA      ABSMOD, LIST
```

Since ABSMOD is specified, the assembly will be in absolute mode.

$IBCBC Card: The format of this card is:

```
1          8          16
$IBCBC     deck name  variable field
```

When this card is encountered, the COBOL Compiler is called to process a COBOL source deck.

The contents of the variable field are:

[{ LIST / FULIST / NOLIST }]

The List options: The LIST option produces an abbreviated three-column assembly program listing in the output file.

The FULIST option produces the standard one-column assembly program listing in the output file.

The NOLIST option suppresses the assembly program listing.

If none of these options is specified, LIST is assumed.

[, { DECK / NODECK }]

The Punch options: The DECK option causes a reloctable deck to be produced with the system punch output.

The NODECK option suppresses the production of a deck.

If neither option is specified, DECK is assumed unless NODECK is specified on the $IBJOB card.

[, { REF / NOREF }]

The Cross-Reference Table options: The REF option produces an alphabetic listing of all the symbols used in the deck, together with cross references to the statements in the deck which use each symbol. This cross-reference table appears immediately after the assembly program listing.

The NOREF option suppresses the listing of the symbolic cross-reference table.

If neither option is specified, NOREF is assumed.

[, SPACE]

The SPACE option causes the COBOL Compiler to attempt to produce an object program which occupies fewer storage positions, though it may require more time for execution.

The following is an example of a $IBCBC card:

```
1          8          16
$IBCBC     DELTA
```

Since the variable field is blank, all of the standard options are chosen, that is, LIST, NOREF, and DECK.

$CBEND Card: This control card is necessary to terminate a COBOL compilation. The format of this card is:

```
1          8          16
$CBEND     deck name
```

The deck name is optional on this card.

$IBLDR Card: The format of this card is:

```
1          8          16
$IBLDR     deckname   [date of assembly]
```

This card must precede every relocatable binary deck to be loaded. The Macro Assembly Program automatically produces a $IBLDR card preceding each relocatable binary deck.

Columns 8-13 must contain the deck name. A $IBLDR card produced by the Macro Assembly Program contains the deck name obtained from the $IBMAP, $IBCBC, or $IBFTC card. The date of assembly may be placed in columns 16-23 for the purpose of identification.

The following is an example of a $IBLDR card:

```
1          8          16
$IBLDR     SEGMT1     12/31/63
```

This card specifies that a binary deck (assembled December 31, 1963) with the deck name SEGMT1 follows.

$ENTRY Card: The format of this card is:

1	8	16
$ENTRY		variable field

This card is used by the Loader to delimit a core storage load for an object program. It should follow the last deck for a core storage load. (This card also precedes a $LINK card in a CHAIN application. See the section "Loader (IBLDR) Chain Feature.") The variable field is interpreted by the Loader to determine the initial entry point for the program. When the Processor Monitor encounters this card, it performs one of the following operations:

1. If SOURCE, together with GO, MAP, LOGIC, DLOGIC, or one of the COPY options, is specified on the $IBJOB card, the Processor Monitor calls the Loader (IBLDR).
2. If SOURCE is specified on the $IBJOB card, but none of the above options is specified, the $ENTRY card is meaningless and is skipped.
3. If NOSOURCE is specified, this card is processed by the Loader.

The contents of columns 8 through 13 are not significant.

The content of the variable field is:

[{ externalname }]
[{ deckname }]

The variable field of this card is either blank or contains an external name or a deck name. If an external name is specified, the entry point for the program is the location assigned to that external name. If a deck name is specified, the entry point for the program is the standard entry point for the deck. For a MAP deck, the standard entry point for a deck is initially contained in the variable field of the END pseudo-operation. If the field is left blank, the entry for the program is the standard entry point for the first retained deck in the program.

The following examples of $ENTRY cards indicate their significance.

Example 1:

1	8	16
$ENTRY		

Example 2:

1	8	16
$ENTRY		EXNAME

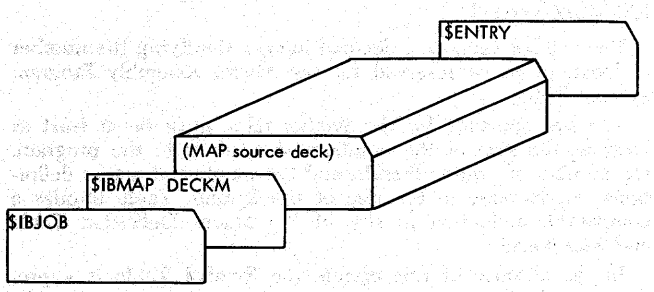
Example 3:

1	8	16
$ENTRY		DKNAME

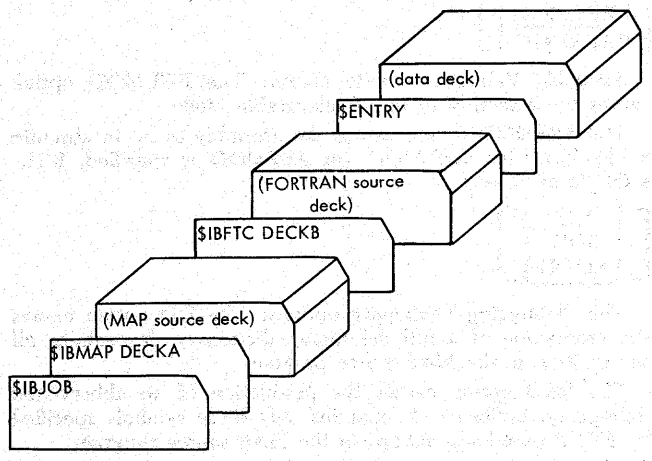
Since the variable field of example 1 is blank, the entry point for this program is the standard entry point for the first retained deck in the program. Example 2 specifies that the entry point for the program is the absolute location assigned to the EXNAME (external name) specified. Example 3 specifies that the standard entry point for the program is the entry point for the deck named in the variable field of the card.

Sample Processor Applications

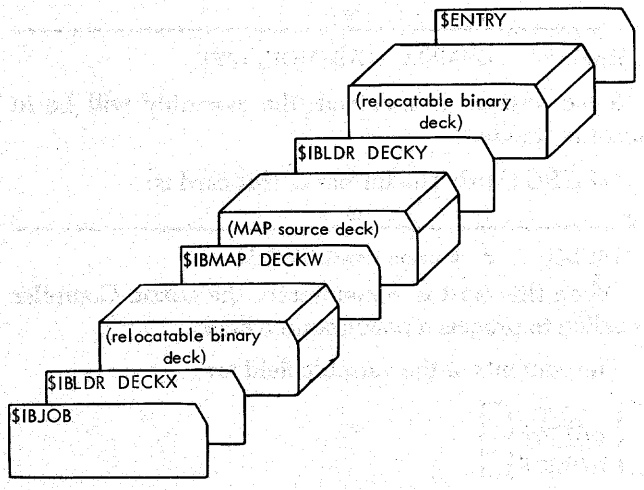
A sample processor application consisting of one MAP language deck follows. No data is on the system input file.



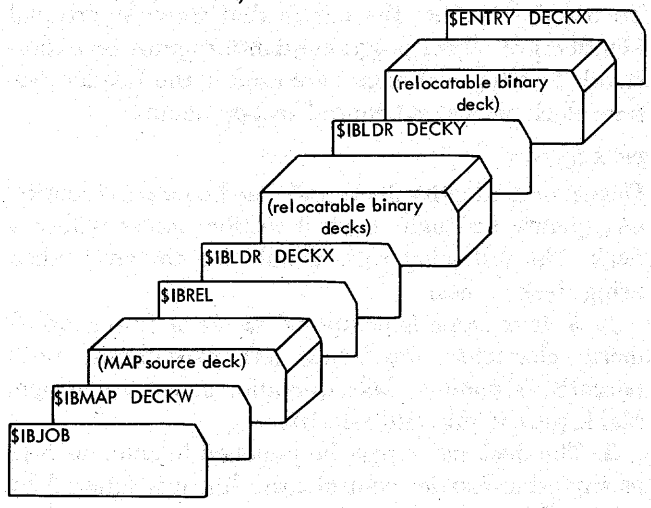
A sample processor application consisting of a MAP language deck and a FORTRAN language deck follows. Data is on the system input file.



A sample processor application consisting of a MAP language deck and some relocatable binary decks follows.

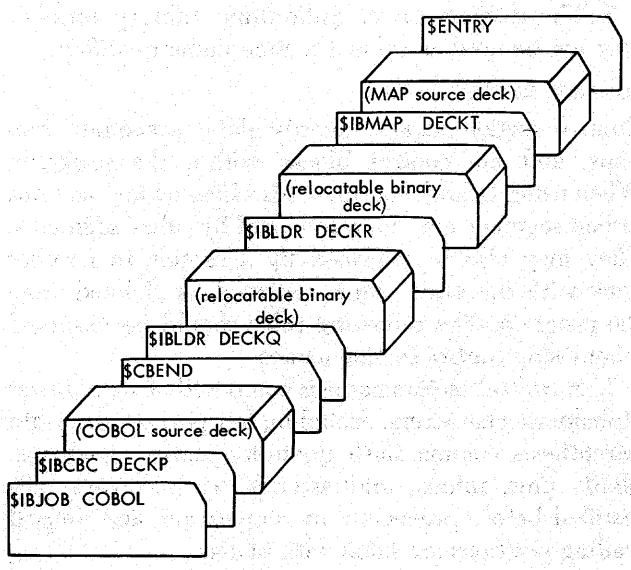


In the preceding processor application, the relocatable decks are read into storage and read out on the load file. The same processor application follows, showing the $IBREL card used to signal the Loader to take the relocatable input decks directly from the system input file, rather than stacking them on the load file. (The relocatable decks must be reordered last to use the $IBREL card.)

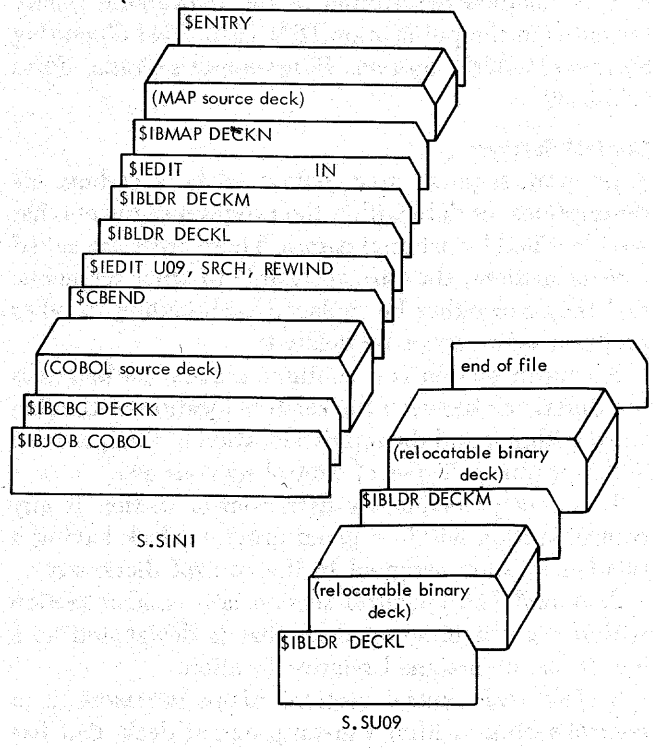


Specifying DECKX on the $ENTRY card ensures that the entry point for the program is the same in both cases. In addition, some $USE and/or $OMIT cards may be necessary when rearranging the decks within a processor application to ensure that the same control sections are used. See the section "Introduction to the Loader (IBLDR)" for further details.

The following is an example of a processor application, consisting of COBOL, MAP, and relocatable binary decks.



The following processor application also consists of COBOL, MAP, and relocatable binary decks. However, the relocatable binary decks are on a separate unit, U09, which may not be positioned correctly. $IEDIT cards are used to indicate this condition.



Introduction to the Loader (IBLDR)

This section discusses the fundamentals of the Loader (IBLDR), which is a Processor component. The Loader assigns storage, processes relocatable binary instructions, completes all cross references between program segments, and, upon completion, transfers control to an entry point in the program. The Loader can process one or more relocatable binary program *segments*, prepare one executable object program from these segments, and transfer control to the object program.

Program Decks

Each segment of the program is passed to the Loader as a separate, relocatable *program deck*. The program deck consists of all the cards contained between the $IBLDR control card and the $DKEND control card. It is Macro Assembly Program output from either this or a previous run, since a programmer can save program decks from one run to be incorporated in a later run. On a tape-oriented system, a program deck consists of a series of card images on tape. Any number of program decks can be run at one time. All of these decks constitute a processor application when they are executed together. A processor application can consist of one program deck or of many, some of which may operate

like closed subroutines or subprograms. Each program deck contains a *control dictionary* and a *relocatable* text. The control dictionary contains the information necessary for cross-referencing control sections; and the relocatable text contains data, procedure, and file text. A detailed description of the relocatable binary format is in the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339.

Control Sections

A program segment may contain areas of coding, file descriptions, or data within the program segments that are identified by external names. These areas are called *control sections;* they are accessible to other segments; and they can either be replaced with coding in other program segments or be deleted.

A control section is a contiguous area; its length is the difference between the relative location of the first word within it and the relative location of the last word within it plus 1. Types of control sections are:

1. A *real* procedure or data control section is any control section within a given program deck having a relative location assigned in the control dictionary.
2. A *real* (file) control section is a control section within a given program deck that is designated as a file. It has no assigned relative location.
3. The *real* control section, blank COMMON, is a control section, within a given program deck, that has a variable field designated as //. It is assigned an absolute location in high storage. (See the section "Storage Allocation.") All references to the // control section from decks loaded together will be adjusted to refer to high storage. No data will be loaded into blank COMMON.
4. External (*virtual*) control sections are the control sections that have no origin or length in the deck in which they are referenced. A virtual control section must be defined in another input deck or in the Subroutine Library as a real control section with the same external name.

The Loader recognizes that control sections are equivalent to one another by their identical names. Only one of each named reference item is included by the Loader, which adjusts all cross referencing to the retained item. Therefore, the user may refer symbolically in one program to the name of a control section in another program, and the Loader will perform the desired cross referencing.

Use of Loader Control Cards

The object time control information that is contained on the Loader control cards for the Loader is processed initially by the Preprocessor section of the Processor Monitor. The information is passed to the Loader and is used to modify the information in the program decks when they are processed. External names may be named or renamed at load time by using these Loader control cards. These cards are a convenient means of specifying the equivalence of names, file characteristics, and labeling procedures.

Loader Name Conventions

To use the Loader, the names that serve as external identifiers of object program quantities must be understood. Two types of names are used in the Loader System: deck names and control section names.

DECK NAMES

Deck names identify decks and may be used to identify, i.e., qualify uniquely, control section names within a deck. The following rules should be observed when using deck names:

1. A deck name is composed of six or fewer alphameric characters, excluding left parenthesis, right parenthesis, comma, slash, quotation marks, equal sign, blank, plus, minus, and asterisk.
2. The deck name may be punched in columns 8-13 of any other Loader control card, but it is ignored by the Loader. It is suggested that deck names be punched in all alphameric control cards of a processor application for visual identification of the decks.
3. Deck names in a processor application should be unique. Multiple use of a deck name will result in deletion of the second and subsequent identical deck names (and their corresponding decks) when they are encountered by the Loader.
4. The deck name may be punched in the variable field of the $NAME, $USE, and $OMIT cards to qualify a control section name. Action taken on the named control section is thus restricted to the deck named. In other decks, the control section with the same name remains unchanged.
5. The deck name of Subroutine Library routines may not be used as control-section name qualifiers.

CONTROL SECTION NAMES

Control section names identify data, procedure sections, and file control blocks within the program. When using Loader control cards, these named sections in one segment may be referenced by other segments. They may also be replaced by a section in another deck with the same name, renamed, or deleted from the program. The following rules should be observed when using control section names:

1. A control section name is composed of six or fewer alphameric characters, excluding left parenthesis, right parenthesis, comma, slash, quotation marks, equal sign, blank, plus, minus, and asterisk. It is always left-justified before processing or comparison, and unused trailing positions are filled with blanks.

2. The first real section with a given name that is physically encountered while loading is retained, and all succeeding occurrences of it are deleted unless explicitly excluded by a $USE card. All references to the given name are adjusted to refer to the storage assigned to the retained section.

3. *Explicit* inclusion of two control sections with the same name (by using deck name qualification on a $USE card) results in a multiple definition of that section; consequently, execution is not allowed.

4. Each control section that is referenced by text must be defined (assigned an absolute origin by the Loader) or execution will not be allowed. For example, if a reference were made to a section mentioned on an $OMIT card, but no other entry with the same name was encountered in any control dictionary, execution would be deleted.

5. All text references to control sections are made to a name in a control dictionary.

6. A subroutine on the library tape is automatically called if: (1) a name in the Subroutine Name Table is identical to that of an external control section and (2) no real control section with the same name appears in any of the retained object decks provided.

7. Control sections of library routines may *not* be renamed.

8. All control dictionary entry names (i.e. deck names, control section names, and ENTRY statements) must be unique to be retained. However, if an ENTRY statement has the same name as the deck it is contained in, all references will be made to the ENTRY and not the deck name.

Object Program Files

Object-program file control blocks are created by the Loader from file text that is generated either from $FILE cards by the Preprocessor or from FILE pseudo-operations by the Macro Assembly Program. The MAP programmer may describe his files by means of FILE pseudo-operations or $FILE cards. In general, the FORTRAN user can rely on the FORTRAN File routines (consisting of FILE pseudo-operation coding) to establish the relation between FORTRAN logical units and Operating System symbolic units.

Note, however, that the user may be required to modify file specifications if all of the following conditions exist:

1. The object program reads data from the system input unit without using the system Input Editor (JOBIN).

2. The object program is processed by the Loader (IBLDR) during an application for which the system input file is specified as a double-buffered file.

3. The object program is edited into the System Library in absolute format.

4. The absolute object program is loaded by the System Loader (S.SLDR) and executed during an application for which a card reader is assigned as the system input unit.

Under these conditions, the system input file is treated at execution time as though it were a double-buffered file. Since a file on a card reader should be processed with single-buffering, the card (usually a $IBSYS card) immediately following the object-program data on the system input unit may be lost (read but not processed). To avoid this, one of the following precautions should be taken:

1. At the time the object program is to be edited into the System Library, $FILE cards specifying single-buffering may be inserted with the program. There should be one $FILE card for each file assigned to the system input unit.

2. At the time the object program is to be executed, an additional $IBSYS card may be inserted immediately following the object-program data on the system input unit.

Even Storage

The Loader provides a technique for ensuring that an even storage location is assigned to specified data or instructions. This is necessary because the machine must store double-precision floating-point operands in successive locations, the first having an even machine location. Even storage for data or instructions is specified by using the EVEN pseudo-operation in the MAP language.

The EVEN pseudo-operation causes generation of an entry in the control dictionary, without a name. The Loader generates an AXT 0, 0 for an EVEN pseudo-operation if the current absolute location is odd.

Loader Diagnostics

The Loader diagnoses errors in the file descriptions, control section references, and storage allocation, and produces appropriate error messages off-line. These messages are self-explanatory.

Loader Control Cards

All of the Loader control cards in the following list are processed by the Preprocessor section of the Processor Monitor. These cards describe file and program loading modifications for an entire object program. They must appear before any source decks or relocatable binary decks and after the $IBJOB card for the processor application.

$FILE
$LABEL
$POOL
$NAME

$USE
$OMIT
$ETC

These cards may not be necessary for a processor application. For example, they are never needed when the $IBJOB card parameters are such that the relocatable Loader is not needed (that is, when no LOGIC, MAP, or GO options are used on the $IBJOB control card). These cards are used to:

1. Override file or label descriptions that appear in the source or relocatable programs for a processor application.
2. Modify the control section retention scheme used by the Loader. (In the control section retention scheme, the Loader uses the first control section that it encounters with a given name.)
3. Depart from the standard buffer assignment. The section, "Input/Output Buffer Allocation," contains further information about this subject.
4. Modify the names of data, procedure, or file control sections.
5. Provide a file description that is not assembled.
6. Delete control sections.
7. Label a file assembled as unlabeled.

The cards must appear in the following order. All out of order cards are not processed.

1. All $NAME, $USE, $OMIT, $POOL, and all $LABEL cards (except that a $LABEL card may immediately follow its associated $FILE card).
2. All $FILE cards. Each $FILE card may be immediately succeeded by a corresponding $LABEL card, if desired.

The publication *IBM 7040/7044 Operating System (16/32K): Input/Output Control System,* Form C28-6309, contains information on file procedures, label procedures, and buffer pools, resulting from $FILE, $LABEL, and $POOL cards, respectively.

$FILE Card: The format of this card is:

1	8	16
$FILE	deckname	'filename', options,...

This card overrides file specifications that appear (in the form of file text) for the file name in the decks to be loaded or provides file specifications for files that are not described by a FILE pseudo-operation.

The presence of any $FILE card for a processor application causes the Preprocessor to construct a file text and a control dictionary entry, which are passed to the Loader. If a $FILE card appears for any file in the processor application, the file description on the card overrides the file description in the file text, if any, for the named file. Thus, when a $FILE card is used, it must completely describe the file. An associated $LABEL card must accompany the $FILE card if the file is to be labeled when the job is executed.

The deck name is optional and does not qualify the file name.

The content of the variable field is:

'filename'

The 'filename' is an alphameric name of six or fewer characters that identifies the file. It must be enclosed by quotation marks. (IBM card code: 8-4). All options except unit assignment options may be entered in any order. Options are separated from the file name and from each other by commas.

[, unit-1, unit-2]

Unit Assignment options: Two symbolic units, unit-1 and unit-2, may be specified for each file. The unit-1 option is the primary unit, and the unit-2 option is the secondary unit used for unit switching. Unit specifications are described in the section titled "Input/Output Unit Assignment" in this publication.

[, { MOUNT / READY / DEFER }]

Mounting options: This option when used applies to unit-1 and unit-2. It should not be used with the corresponding option shown below. It specifies the type of message to be printed and the operator action required when an input/output unit is used. The MOUNT option causes the message to be printed accompanied by a stop to permit operator action before execution.

The DEFER option has the same effect as MOUNT, but the actions are delayed until the file is opened. If no mounting action is specified, READY is assumed. The READY option deletes the mounting messages and the halt for operator intervention. The COBOL Compiler assumes DEFER.

[, { MOUNT i / READY i / DEFER i }]

This option when used applies only to unit-i, where i=1 or 2. The action taken is the same as for the corresponding option shown above. If neither MOUNTi, READYi, nor DEFERi is specified for unit-1 or unit-2, READY is assumed.

[, CKFILE]

Checkpoint File option: The CKFILE option specifies that the file is a checkpoint file.

, BLOCK=xxxx

Block Size option: This option specifies the block size for the file. The letters xxxx represent the number of words per block. If this field does not appear, or if xxxx=0, the Loader will not assign this file to a buffer pool.

If the file contains Type 2 records, the size specified must allow space for the control word preceding each logical record. If the check-sum or block-sequence option is chosen (see below), the size must include space for the check-sum or block-sequence word.

[, { SINGLE / DOUBLE }]

Buffer options: The SINGLE option specifies that one buffer is to be assigned to the file.

The DOUBLE option specifies that two buffers are to be assigned to the file.

If neither SINGLE nor DOUBLE is specified, DOUBLE is assumed.

[, { REEL / REELS }]

Reel Handling options: These options do not apply to input files that have standard labels. The REEL option specifies that no reel switching will occur.

The REELS option specifies that reel switching will occur. If neither REEL nor REELS is specified, REELS is assumed.

[, {HIGH / LOW}]

File Density options: This field specifies the density at which the file is to be read or written. The density setting is included in the mount or ready message to the operator. The density switch on the console and the density key on the tape drive must be set as directed in the mounting message. If both settings are not made, unrecoverable errors will occur when input files are read, and output files will be written in the wrong density.

If neither HIGH nor LOW is specified, HIGH is assumed.

[, MIXED]

The MIXED option specifies that the file is composed of both BCD and column binary records in the mixed mode format. It is used for documentary purposes only.

[, {SEQ / NOSEQ}]

Block Sequence options: The SEQ option specifies that the block sequence word, which indicates the relative position of a physical record, is to be checked if the file is input, or formed and written if the file is output.

If NOSEQ is specified, the block sequence word will not be checked or written. The block sequence word, if used, is part of the physical record and must be provided for in the block size specifications for the file.

This field should be used only with files that are in binary mode.

If neither SEQ nor NOSEQ is specified, NOSEQ is assumed.

[, {CKSM / NOCKSM}]

Check Sum options: The CKSM option specifies that a check sum, which is the folded logical sum of all the data in a block, is to be checked if the file is input, or formed and written if the file is output.

If NOCKSM is specified, the check sum will not be checked or written. The check sum, if used, is located in the same word as the block sequence number. Provision must be made for this word in the block size specifications for the file.

This field should be used only with files that are in binary mode.

If neither CKSM nor NOCKSM is specified, NOCKSM is assumed.

[, {NOCKPT / CKAFLB / CKCKFL / CKLBFL}]

Checkpoint options: The NOCKPT option specifies that no checkpoints are associated with this file.

For input files, the CKAFLB specifies that a checkpoint record follows each header label and is to be passed over. For output files, CKAFLB specifies that a checkpoint record is to be written on the file following each header label.

For input or output files, the CKCKFL option specifies that a checkpoint record is to be written on the checkpoint file whenever a new reel is started on this file.

For input files, the CKLBFL option specifies: (1) that there is a checkpoint record, which is to be bypassed, at the beginning of this file and (2) that a checkpoint record is to be written on the checkpoint unit whenever a new reel is started in this file. This option is not allowed for output files.

For COBOL files, at least one file in the source program must have been specified in a RERUN...REEL clause (I-O-CONTROL paragraph, Option 5) for any other file to be changed from the no-checkpoint condition to CKAFLB for output files, to CKLBFL for input files, or to CKCKFL for an input or output file. CKAFLB may be specified without this restriction for an input file.

If neither NOCKPT, CKAFLB, CKLBFL, nor CKCKFL is specified, NOCKPT is assumed.

[, {PRINT / PUNCH / HOLD / SCRTCH}]

File Disposition options: The PRINT option specifies that the file is to be printed. At the end of an application, the file is rewound and unloaded.

The PUNCH option specifies that the file is to be punched. At the end of an application, the file is rewound and unloaded.

The HOLD option specifies that the file is to be saved. At the end of an application, the file is rewound and unloaded.

The SCRTCH option specifies that the file is to be rewound at the end of the processor application.

If neither PRINT, PUNCH, HOLD, nor SCRTCH is specified, SCRTCH is assumed.

[, {ADDLBL=exname / NSLBL=exname}]

Labeling options: This field specifies the external six-character name of a procedure control section that is to be entered by IOCS to process additional label fields (ADDLBL) or nonstandard labels (NSLBL).

Note that the presence or absence of a $LABEL card for this file determines whether or not the file is considered to be labeled.

[, LRL=xxxx]

Logical Record Length option: This entry specifies the length of a logical record in the file. The letters xxxx represent the number of words per record. If this entry is omitted, zero is assumed.

[, RCT=xxxx]

Record Count option: This option specifies the maximum number of logical records, xxxx, that may appear in a block. If this entry is omitted, zero is assumed.

For COBOL files, this option may be used only if the file is not assigned to a system unit. If the number of records per block is changed, the block size should be adjusted accordingly.

[, EOR=exname]

End-of-Reel option: This field specifies the external six-character name of a procedure control section that is to be entered by IOCS for end-of-reel processing.

[, ERR=exname]

Error Exit option: This field specifies the external six-character name of a procedure control section that is to be entered when IOCS detects an error.

[, {TYPE1 / TYPE2 / TYPE3}]

Record Type option: Type 1 specifies that the file contains fixed-length records or nonstandard variable-length records.

Type 2 and Type 3 specify that the file contains standard format variable-length records with control characters. The section, "Record Formats," under "Input/Output Buffering Systems (IOBS)" in the publication, *IBM 7040/7044 Operating System (16/32K): Input/Output Control System,* Form C28-6309, contains additional information.

If this field is omitted, TYPE1 is assumed.

[, EOF=exname]

End-of-File option: This field specifies the external six-character name of a procedure control section that is to be entered by IOCS for end-of-file processing.

```
1          8          16
$FILE                 'FILE1',U01,U02,BLOCK=28,LRL=28
```

The sample file card shown above indicates that the file named FILE1 is assigned to utility unit S.SU01. The alternate unit is assigned to utility unit S.SU02. All the standard options (underlined options) are assumed for the omitted options.

$LABEL Card: The format of this card is:

```
1          8          16
$LABEL     deckname   'filename', [ file serial number ]
                      [ reel sequence number ] [ date / days ]
                      [ identification ]
```

This card provides labeling information for the file. If the card is omitted, the file is assumed to be unlabeled. The $LABEL card is an exception to the variable field format; the fields that are present must appear in the order shown in the format. However, all fields except the first and last may be omitted, with omissions indicated by adjacent commas (,,). The last field is considered to be ten characters long with embedded blanks allowed. All labeling information must be on this card; $ETC cards are not allowed.

The deck name field is not significant.

'filename'

The 'filename' is an alphameric name of six or fewer characters, identifying the file. It must be enclosed by quotation marks (IBM card code: 8-4).

, [file serial number]

The file serial number is an alphameric field of five or fewer characters. If this serial number is present, standard input labels for this file are checked against it. Standard output labels for this file contain this serial number only if the reel sequence option specifies a reel number greater than 1. If the reel sequence number is not specified or is specified as 1, the file

serial number is taken from the label that is already present on the file.

,[reel sequence number]

The reel sequence number is a numeric field of four or fewer characters that specifies the reel sequence of the first reel to be processed in this file. If the field is omitted, the reel sequence number is assumed to be 1. The reel sequence number is adjusted at object time to reflect reel switching and is checked in standard input labels.

,[{date / days}]

Input Files: This field contains the creation date in the following format:

Y/D

where Y is a number of one or two digits indicating the year, and D is a number consisting of three or fewer digits indicating the day of the year. This field is checked against the creation date field in the label. If the field is left blank, the check is not made.

Output Files: This field contains the number of days the file is to be retained. Retention days are expressed as a number of four or fewer digits. If the field is blank, the retention period is set to zero.

,[identification]

The identification consists of ten alphameric characters following the last comma in the card. It is the file identification. Embedded blanks are permissible. If the file is input and this field is blank, the identification in the standard label is not checked. If the file is output and this field is blank, the standard label contains an identification field of ten zeros.

An example of a $LABEL card is shown below.

```
1          8          16
$LABEL                'FILE2' , , , 63/360, ident
```

This card indicates that the file named FILE2 was created on December 26, 1963. Since the reel sequence number has been omitted, it is assumed to be 1, and the file serial number is taken from the label already present on the file.

$POOL Card: The format of this card is:

```
1          8          16
$POOL      deckname   BLOCK=xxxx , BUFCT=xxx
                        [, 'filename']
```

This card designates those files that are to share common buffer areas, hereafter called pools. The effect of this card may be extended, if necessary, by $ETC continuation cards.

The deck name is optional and may be omitted.

The content of the variable field must appear in the order shown. It is:

BLOCK=xxxx

xxxx is a number that specifies block size of the buffer pool.

, BUFCT=xxx

xxx is a number that specifies the number of buffers to be assigned to the pool. This number should be at least equal to the maximum number of files that will be open simultaneously in the pool. BUFCT is assigned as specified. However, since IOCS uses a maximum of twice the number of files in the pool, the balance of the buffers, if any, will not be used.

The buffer count specified here overrides the $FILE card buffer options. Only the specified number of buffers are assigned to the pool. If a file requires a buffer during the running of a job and none is available, the job is terminated.

[, 'filename']

The remaining fields on the card are the names of the files that are to be included in the pool. Each file name is an alphameric name of six or fewer characters, enclosed by quotation marks.

An example of a $POOL card is shown below.

```
1          8          16
$POOL                 BLOCK=100,BUFCT=2,
                        'FILE1','FILE2','FILE5'
```

This card indicates that FILE1, FILE2, and FILE5 are to be assigned to the same buffer pool. The largest block size of any file in the pool is used. Two buffers will be assigned to the pool.

$NAME Card: The format of this card is:

```
1          8          16
$NAME                 { deckname (exname)=exname  }
                      { exname=exname            }
                      { deckname ('filename')='filename' }
                      { 'filename'='filename'     }
```

This card may be used to change the name of a file or control section. A name change is required when the same name has been used in different decks for two or more distinct files or control sections, in which case one of them must be renamed with a distinct name. This card may also be used when two different names are used to refer to the same file or control section, in which case one name is replaced by the other.

The content of columns 8-13 is not significant on this card.

Each entry in the variable field consists of two alphameric names separated by an equal sign (=). The name on the left consists of an external name that may be qualified by a deck name. This external name is replaced by the name to the right of the equal sign. If files are to be renamed, then the name must be enclosed by quotation marks.

If the external name on the left is not qualified, it is replaced by the name on the right wherever it occurs. If the name is qualified by a deck name, it will be replaced by the name on the right only in the deck named.

A single $NAME card may contain one or many entries, each serving to rename a control section. Successive entries must be separated by commas. $ETC cards may be used if the information will not fit on a single $NAME card.

Special Notes: Because name changes are processed first, the following special rules apply:

1. If the external name of a file or control section is changed in all decks by a $NAME card, the *new* name

must be used on any other Loader control cards that refer to the file or control section.

2. If the external name of a control section is changed by a $NAME card in one deck only, the old external name may be used on any other Loader control card.

The following are examples of $NAME cards:

Example 1:

```
1          8          16
$NAME                 DECK2(ROUT1)=JOBXV
```

Example 2:

```
1          8          16
$NAME                 DECK1('FILE')='SAVE'
```

Example 3:

```
1          8          16
$NAME                 LOOKUP=SCAN1
```

Example 4:

```
1          8          16
$NAME                 'FILE3'='LISTS'
```

The effect of the card in example 1 would be to change the name ROUT1 to JOBXV in DECK2 only.

The effect of the card in example 2 would be to change the file named FILE to SAVE in DECK1 only.

The effect of the card in example 3 would be to change the name LOOKUP to SCAN1 in all decks.

The effect of the card in example 4 would be to change the file name FILE3 to LISTS in all decks.

$USE Card: The format of this card is:

```
1          8          16
$USE                  deckname (exname),...
```

This card provides a method of specifying a particular data, procedure, or file control section that is used at execution time. The control section in the first deck encountered by the Loader is normally retained and all control sections with the same name in other decks are deleted, but this card may be used to retain a control section from any deck. All control sections with the same name in other decks will be deleted.

The content of columns 8-13 is not significant.

The entries in the variable field are alphameric literals. The first six or fewer characters of the entry are the deck name. The external name of the control section follows, consisting of six or fewer characters enclosed by parentheses.

A single $USE card may contain one or many entries, each serving to retain a control section. Successive entries must be separated by commas. $ETC cards may be used if the information does not fit on a single $USE card.

An example of a $USE card follows:

```
1          8          16
$USE                  DECK10(TABLE1),DECK5
                        (ENTRY4),DECK4(ROUTN3)
```

This card indicates that the control sections, TABLE1 in DECK10, ENTRY4 in DECK5, and ROUTN3 in DECK4, are to be used instead of the control sections having the same names that were loaded before this control section.

$OMIT Card: The format of this card is:

```
1          8          16
$OMIT                 { exname                  }
                      { deckname (exname)       },...
```

This card provides a method of deleting data, a procedure, or a file control section from a specific deck or from all decks in which this section appears.

The content of columns 8-13 is not significant on this card.

The entries in the variable field consist of alphameric names. The entry may be the external name (six or fewer characters) of a control section, in which case the control section will be deleted from all decks in which it occurs. Alternatively, the entry may be a deck name (six or fewer characters) followed by the external name of a control section enclosed by parentheses, in which case the control section will be deleted from the named deck only.

A single $OMIT card may contain one or many entries, each of which serves to delete a control section. Successive entries must be separated by commas. $ETC cards may be used if the information does not fit on a single $OMIT card.

Some examples of $OMIT cards follow:

Example 1:

```
1          8          16
$OMIT                 TABLE1
```

Example 2:

```
1          8          16
$OMIT                 DECK1(TABLE1),DECK2
                        (ROUTN3),DECK2(HALT4)
```

The first example indicates that TABLE1 is to be omitted in all decks. The second example indicates the following omissions: TABLE1 in DECK1 only, ROUTN3 in DECK2 only, and HALT4 in DECK2 only.

Input/Output Buffer Allocation

Since the amount of usable storage for input/output buffers can be determined only by the Loader, the Loader will:

1. Allocate buffers for object program files, if any, from storage not assigned for use by the system or the object program.
2. Allocate buffers to pools as specified, if $POOL cards are used, and assign the specified files to those pools.
3. Assign the remaining files to pools according to block size, and assign buffers to those pools according to file specifications.

GENERAL BUFFER ASSIGNMENT

If no $POOL cards are used, the rules for input/output buffer allocation are:

1. A different buffer pool is created for each distinct block size encountered. All files of the same block size are assigned to the same pool, whether they are input or output files.
2. The pool is given two buffers for each file for which the DOUBLE buffer option is specified in the file control block; and one buffer for each file having the SINGLE buffer option specified.
3. The storage used by each buffer pool is:
   a. One pool control word.
   b. One control word for each buffer in the pool.
   c. Block size plus two words for each buffer.

BUFFER ASSIGNMENT WITH $POOL CARDS

$POOL cards may be used to direct the assignment of files to certain pools.

1. All files mentioned on $POOL cards are assigned to the same buffer pool. No other files, not even those with block size equal to that of the pool, are assigned to the specified pool.
2. If a buffer count (BUFCT) is specified on a $POOL card, the pool will have exactly that number of buffers. The buffer count overrides any buffer options that may have been specified on the $FILE cards for the files in the pool. Since no check is made to assure that a sufficient number of buffers have been specified to accommodate all of the files that are opened simultaneously in the pool, care should be exercised in specifying the pool buffer count. If, when needed during execution, buffers are not available for a file, the job is terminated.
3. If no buffer count (BUFCT) is specified on the $POOL card, the number of buffers assigned to the pool will be the sum of the buffers specified by the buffer count options on the $FILE cards for all files in the pool.
4. If block size (BLOCK) is not specified on the $POOL card, the largest block size of any file assigned to the pool is used to allocate buffers.

Storage Allocation

The Loader allocates storage to the object program, as follows:

1. If the program refers to S.SLOC, then S.SLOC is assigned to the first nonstorage-protected location after the level of IOCS specified on the $IBJOB card, and a five-word block is placed in the succeeding locations.
2. File text for each file is formed into a 19-word file control block. All file blocks are located immediately after S.SLOC and the five-word block, or at the first nonstorage-protected location after the level of IOCS specified on the $IBJOB card. (File text from the Subroutine Library appears with the rest of the file text.)
3. The data and procedure text are formed into absolute text starting at the first location after the last word assigned to the file blocks.
4. Subroutine Library data and procedure text follow the data and procedure text formed from the input decks.
5. Blank COMMON is assigned storage immediately below the highest location available to the system (S.SEND).
6. Buffers are assigned immediately below blank COMMON.
7. Pool control words are created and are assigned locations immediately below the buffers.

Loader (IBLDR) Chain Feature

Multiphase Programming

It is a common programming problem that a program too large to fit into core storage must be executed as a sequence of smaller programs. The 7040/7044 Processor has the capability of processing such a programming application, consisting of several core storage loads or phases, by using the Chain feature of the Loader (IBLDR).

The user of the Chain feature specifies through control cards that the source language and/or relocatable input decks that follow a $IBJOB card are to be processed to form several component programs (links) that are loaded separately. Although each input deck has the same format that is used in an application consisting of only one core storage load, certain rules must be followed when coding a program for a Chain application. In addition, restrictions are placed on the order of the input decks in a Chain application.

When processing a Chain application, the Loader forms an absolute multiphase program and then places it on a utility unit. This program must consist of a main or controlling program, called the main link, and one or more links that are loaded subsequently, called dependent links. The main link remains in core storage at all times during execution.

Links may be coded in the FORTRAN, COBOL, or MAP languages. References from one link to another are accomplished through control dictionary entries.

DEFINITIONS

A *previous* dependent link is one whose input decks precede the input decks of another dependent link.

A *subsequent* dependent link is one whose input decks follow the input decks of another dependent link.

CROSS-REFERENCING

Cross-referencing among links is governed by the following:

1. A previous dependent link may never refer to a control dictionary entry that is defined in a subsequent dependent link.
2. The main link may not refer to a control dictionary entry that is defined in a dependent link.
3. A subsequent dependent link may refer to a control dictionary entry in a previous dependent link only if the coding for that section of the previous link is still in storage. The variable field of the $LINK card can be coded to insure that portions of previous links are not destroyed by a CALL to a subsequent link. It specifies the origin of this link and prevents any reference, by this or subsequent links, to all external names in previous links from this point on.
4. Dependent links may always refer to blank COMMON or external names in the main link because the main link cannot be destroyed by any dependent link. The main link remains in storage throughout execution.

FILES

The files for a Chain application must be defined with the main link; however, references to these files may occur in any link. Files that appear in subsequent links are deleted and a warning message is written.

SUBROUTINE LIBRARY REFERENCES

Subroutine library references may be made in any link. Each subroutine will become part of the link which refers to it, except when the subroutine is part of a previous link and the coding for that section of the previous link is still in core storage (as described in item 3 above). If this requirement is met, a link may refer to the subroutine in core storage, even though the subroutine is not made a part of the link that refers to it. The placing of a dependent link immediately following the main link (by omission of a deckname in the $LINK card) effectively destroys all other links previously called for the purpose of referring to subroutines, regardless of the actual physical position in core storage. When such a link or any subsequent link refers to a subroutine, this subroutine will become a part of the link referring to it. When the main link refers to a subroutine, the subroutine effectively becomes a part of the main link and cannot be destroyed.

Chain Programming Considerations

THE MAIN LINK

The following are requirements for the main or controlling link:

1. It must contain the definitions of all files used in the entire program.
2. It must contain the definition of the largest block of blank COMMON in the program.
3. It should contain a CALL to all dependent links in the order in which they are to be executed. This is done by a CALL to the CHAIN subroutine.
4. It should return control to the Processor Monitor when all processing is complete (using a TRA to S.JXIT).
5. It should contain the subroutines that are common to all dependent links, so that they will not be loaded with every link that references them. Library subroutines need not be referenced by a CALL statement in the main link; they may be named in an EXTERN pseudo-operation.

As long as the main link conforms to the above rules, it may perform any additional functions that the programmer desires; however, it may not refer to any locations within dependent links.

The CHAIN Subroutine: Dependent links are loaded and entered by using a CALL to the CHAIN subroutine in the main link. This reference to CHAIN in the main link causes the Loader to make the CHAIN subroutine part of the main link. This subroutine is located in the system Subroutine Library.

The calling sequence is:

```
1         8         16
          CALL      CHAIN(i)
```

where *i* is the link number; this number is determined by the order in which the links are stacked as input to the Loader (IBLDR), exclusive of the main link.

The CHAIN subroutine uses the System Loader (S.SLDR) to load and enter each link. Each dependent link returns control to the main link by means of a transfer to an entry point in the CHAIN subroutine (CHNXIT).

The following is an example of a main link coded in the MAP language that contains a CALL to three dependent links:

```
*MAIN OR CONTROLLING LINK
FILEA   FILE    U00,,BLOCK=50 FILE BLOCK FOR
        ETC     LGL=50,RCT=1 FILE 1
FILEB   FILE    U01,,BLOCK=60 FILE BLOCK FOR
        ETC     LGL=60,RCT=1 FILE 2
```

```
CALL 1  CALL     CHAIN(1)
        CALL     CHAIN(2)
        CALL     CHAIN(1)
        CALL     CHAIN(3)
        TRA      S.JXIT          TRA TO POST
*                                EXECUTION
*                                ROUTINES
        EXTERN   CHAIN
        EXTERN   S.FBIN,S.FBOU   FILE BLOCKS FOR
*                                SYSTEM UNITS
*                                FROM SR LIBRARY
        EXTERN   CONVRT          CONVERSION
*                                ROUTINE FROM
*                                SR LIBRARY
        EXTERN   JOBIN,JOBOU     INPUT AND OUT-
*                                PUT EDITOR
*                                FROM SR LIBRARY
        CONTRL   //              DEFINE BLANK
        USE      //              COMMON
        BSS      500
        USE      PREVIOUS        RETURN TO
*                                MAIN LINK
*                                LOC CNTR
        END      CALL 1
```

The Loader (IBLDR) will allocate core storage for the main link as shown in Figure 22.

DEPENDENT LINKS

Dependent links must conform to the following requirements:

1. No dependent link may overlay any part of the main link, including buffer areas, blank COMMON, and subroutines referenced by the main link.

2. No dependent link should CALL the CHAIN subroutine since this would destroy the return to the main link.

3. A dependent link may reference control dictionary entries defined in the main link. It may also reference control dictionary entries defined in any part of a previous dependent link if the associated sections of coding have not been destroyed by the appearance of a $LINK card that origins a link at a point lower in core storage. Thus, in the example in Figure 25 neither Link 2 nor Link 3 could reference external names in Link 1 (even if Link 3 were to be called immediately after Link 1). However, external names in Deck 1 of Link 2 can be referenced by Link 3.

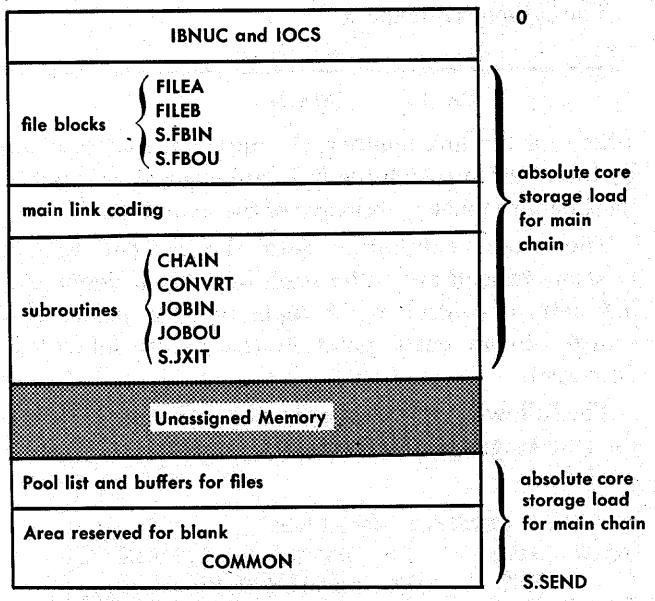


Figure 22. Core Storage Allocation for a Main Link

4. Links may be called in any order. Care must be taken to ensure that the portion of a dependent link that is referenced by a subsequently called link is not destroyed before the referencing link is called. This is especially important when more than one CALL is made to a link or when library subroutines are "shared" by two or more dependent links due to initial relative placement.

In the example in Figure 25, although Link 3 can refer to Deck 1 of Link 2, if the links were called in the order 1, 2, 1, 3, Link 1 might have replaced the section needed by Link 3. Since the order in which the links will be called cannot be determined by the Loader, this type of error will not be diagnosed or prevented. The responsibility for proper placement and cross-referencing in these instances rests solely with the programmer.

5. When a dependent link has finished processing, it must return control to the main link by means of a TRA to CHNXIT for MAP coded links, or a CALL CHNXIT for FORTRAN or COBOL coded links.

6. The TCD pseudo-operation may not be used in a MAP assembly in any link.

7. The entry point for any link may not be a SAVE; therefore, a FORTRAN subroutine subprogram may not contain the entry point for a link.

The following is an example of a dependent link coded in MAP:

```
*DEPENDENT LINK 1
LINK 1      TSX      S.OPEN,4    OPEN FILES
            PTW      FILEA
            TSX      S.OPEN,4
            PTW      FILEB
              .        .
              .        .
            CLA      A
              .        .
              .        .
            CALL     JOBIN       GET INPUT
              .        .
              .        .
              .        .
            CALL     JOBOU       WRITE OUTPUT
            TSX      S.CLSE,4    CLOSE ALL FILES
            PZE      FILEA
            TSX      S.CLSE,4
            PZE      FILEB
            CALL     INCLOS
            CALL     OCLOS
            TRA      CHNXIT      RETURN TO MAIN
*                                  LINK
```

```
         CONTRL   //          DEFINE BLANK
         USE      //          COMMON
A        BSS      25
         USE      PREVIOUS    RETURN TO
*                             DEPENDENT LINK
*                             LOC CNTR
         EXTERN   FILEA, FILEB
         EXTERN   CHNXIT,JOBOU,JOBIN
         END      LINK1
```

If this dependent link were processed following the main link previously described, the Loader would allocate more core storage, as shown in Figure 23.

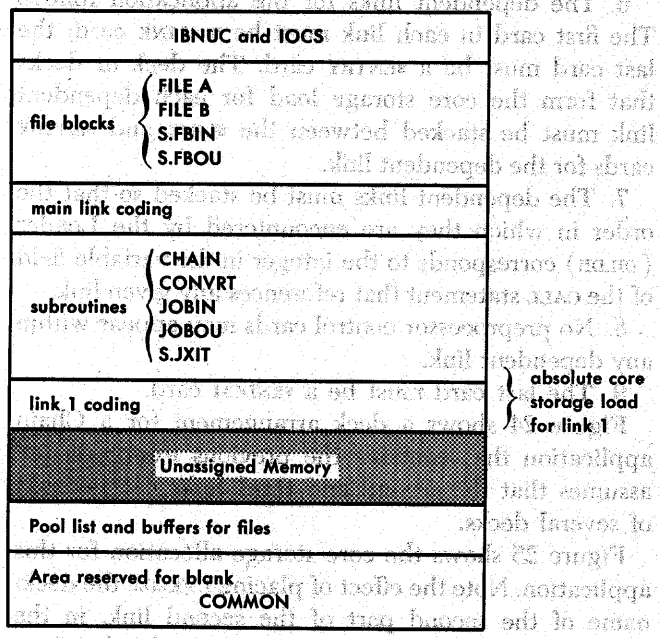


Figure 23. Core Storage Allocation for a Main Link and the First Dependent Link

Control Cards

$CHAIN CARD

The format of the $CHAIN card is:

1	8	16
$CHAIN	main name	variable field

This control card is required whenever the Chain feature is to be used in a Processor application. The card must be placed immediately after the $IBJOB card for the application.

The *main name* field must contain the name of the main link of the program. The name is composed of six or fewer alphameric characters, starting in card column 8.

The contents of the variable field are:

[{ Iyy
Uxx[=Iyy]
U[=Iyy]
T[=Iyy]
D[=Iyy] }]

The Chain Unit options: This specification allows the programmer to indicate the unit on which the Loader is to produce the absolute text for the object program. The Loader assigns the specified unit before units are assigned for the object program files. Each link of the object program is loaded from this unit as it is required.

Iyy refers to the unit that has been assigned the intersystem code yy.

Uxx refers to the system utility unit xx. If the programmer also specifies =Iyy, the unit is to be assigned the intersystem code yy.

U refers to the first available unit. If the programmer also specifies =Iyy, the unit is to be assigned the intersystem code yy.

T refers to the first available tape unit. If the programmer also specifies =Iyy, the unit is to be assigned the intersystem code yy.

D refers to the first available disk or drum unit. If the programmer also specifies =Iyy, the unit is to be assigned the intersystem code yy.

If none of these options is specified, no unit is assigned until all object program units have been assigned. The Loader then assigns the first available unit as the "chain unit." (In this case, the search for an available unit begins with work unit 1 and then continues in the usual manner, starting with the first unit in the Symbolic Units Table.)

The chain unit specification is of particular advantage to the FORTRAN programmer. Because of the procedures used to assign units for FORTRAN files, it is recommended that the chain unit be assigned first.

$LINK CARD

The format of this card is:

1	7	8	16
$LINK	x	linkname	deckname

This control card must precede each dependent link in the chain. The deck or decks that follow it and precede a $ENTRY card will be formed into one core storage load.

The content of column 7 is:

x

If the current link is to overlay the deck specified in the variable field, column 7 is blank. If the variable field is blank, this link will follow the main link. If the current link is to follow the deck specified in the variable field, column 7 contains an * (IBM card code 11-8-4). If the variable field is blank, this link will follow the last dependent link.

The content of columns 8-13 is:

linkname

The link name is an alphameric name for the link and is comprised of six or fewer characters.

The content of the variable field is:

deckname

The deck name is the name of a program deck in a preceding link. If column 7 is blank, the Loader will locate the current link starting at the absolute address assigned to the specified program deck. If column 7 is *, the current link will be located at the location following the last location in the specified deck. If both the variable field and column 7 are blank, this link will start at the location following the last location in the main link. If the variable field is blank and column 7 is *, this link will start at the location following the last location in the last dependent link.

$ENTRY CARD

The format of this card is:

1	8	16
$ENTRY	variable field	

This card must be the last card in every link. It is used to specify the entry point for any link. The appearance of this card in the input stack of a Chain application does not cause the Processor Monitor to call the Loader as it does in other processor applications.

The content of columns 8-13 is not significant.

The content of the variable field is:

[{ externalname }]
[{ deckname }]

The variable field of this card is either blank or contains a control dictionary name or a deckname. If a control dictionary entry is specified, the entry point for the link will be the location assigned to that external name. If a deckname is specified, the entry point for the link will be the standard entry point for the deck. If the field is left blank, the entry for the link will be the standard entry point for the first retained deck in the link. Any deckname or control dictionary name specified as the variable field must exist within that link.

$ENDCH CARD

The format of this card is:

1	8	16
$ENDCH		

This card must be the last card of a Chain application. When this card is read, the Loader is called if GO, MAP, or LOGIC has been requested.

Columns 8-13 and the variable field of this card are not significant.

Deck Arrangement Rules

The following rules govern the arrangement of a deck for a Chain application:

1. The first control card for the application must be a $IBJOB card; the parameters on this card have the same meaning as for any other processor application. (See the section "Introduction to the Processor Monitor," for a description of the format and function of this card.)

2. The second card in the deck must be a $CHAIN card. This card indicates that the decks that follow are to be formed into several links; it also signifies that the deck or decks which immediately follow this card, but which precede the first $ENTRY card, compose the main or controlling link for the application.

3. Any $FILE, $LABEL, $POOL, $NAME, $USE, and $OMIT cards for the main link must immediately follow the $CHAIN card.

4. The deck or decks that form the main link must appear next.

5. A $ENTRY card, indicating the entry point for the main link, must appear next.

6. The dependent links for the application follow. The first card in each link must be a $LINK card; the last card must be a $ENTRY card. The deck or decks that form the core storage load for each dependent link must be stacked between the $LINK and $ENTRY cards for the dependent link.

7. The dependent links must be stacked so that the order in which they are encountered by the Loader (IBLDR) corresponds to the integer in the variable field of the CALL statement that references any given link.

8. No preprocessor control cards may appear within any dependent link.

9. The last card must be a $ENDCH card.

Figure 24 shows a deck arrangement for a Chain application that contains the previous examples and assumes that the second and third links each consist of several decks.

Figure 25 shows the core storage allocation for this application. Note the effect of placing LNK2D2, the deckname of the second part of the second link, in the variable field of the $LINK card for the third link. When Link 3 is loaded, the Loader will locate the link as the absolute address of the second deck of Link 2.

Figures 26 and 27 show a Chain application that contains examples of storage allocations using $LINK card options. The same allocation would result if the card $LINK LINK3 LNK2D2 were replaced with the card $LINK* LINK3 LNK2D1. Note that Link 4 follows the subroutines for Link 3.

Execution

If the GO option is indicated on the $IBJOB card, the Loader (IBLDR), after forming a multiphase program and placing it on a utility unit, will clear storage and transfer control to the System Loader (S.SLDR), requesting that the main link be loaded.

The Reload Program

The Reload program is a subsystem under the Processor Monitor. It initiates the loading of absolute object programs that were previously produced by the Copy feature of the Loader. It eliminates the necessity of

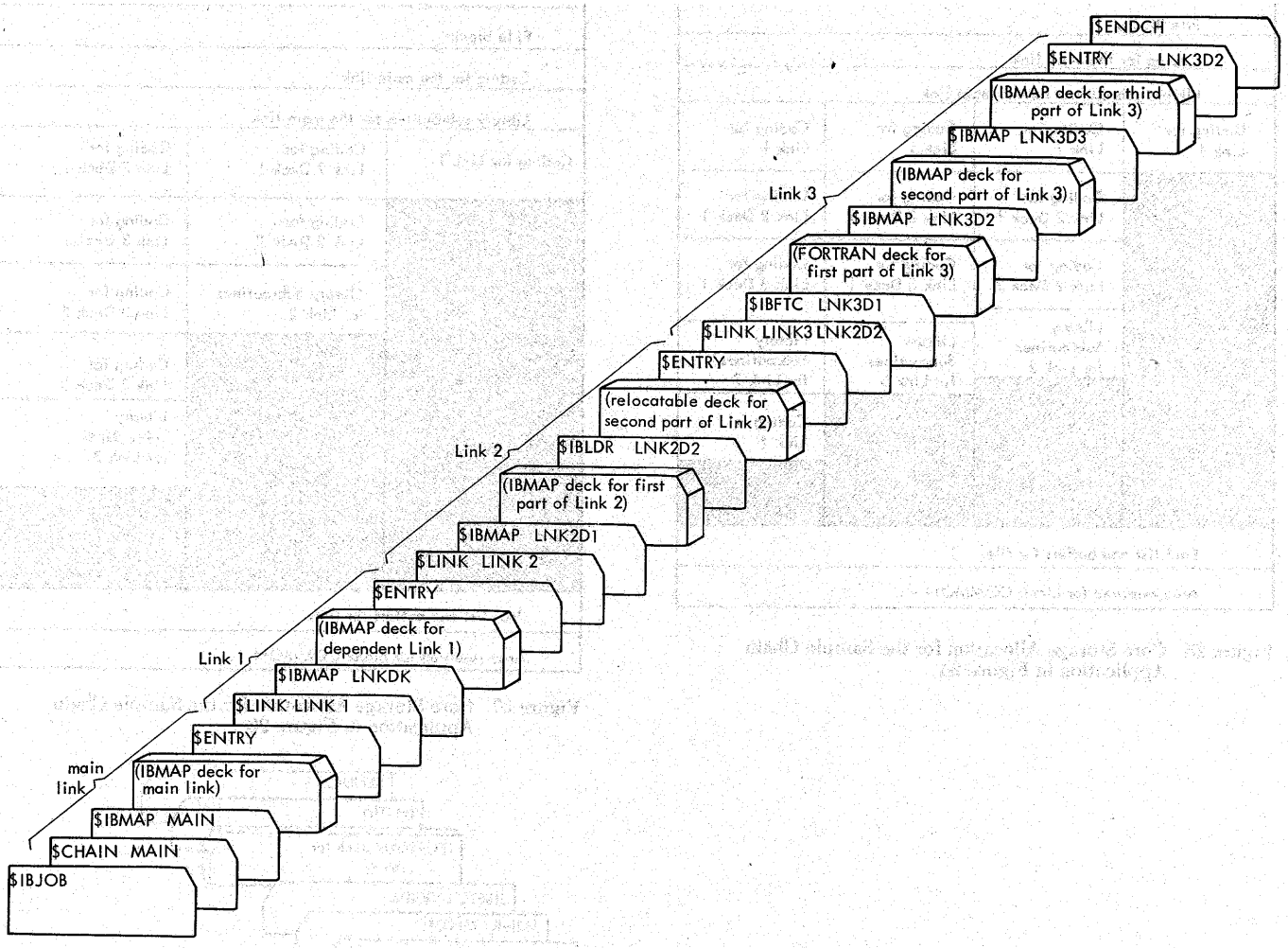


Figure 24. Sample Deck Arrangement

using the Loader to load frequently used programs (e.g., a payroll program). This is especially desirable in the case of Chain applications where load-time can be time-consuming.

Absolute Object-Program Files

When a Copy option is specified on the $IBJOB control card to indicate that a copy is to be performed, the Loader assumes immediate overflow of absolute text. When processing is complete, the Loader produces the absolute object-program file. This file consists of the appropriate table-of-contents record followed by the absolute text. Also included is a record with the information necessary to reload the absolute object program.

The absolute object-program file can be repeatedly used until the system Nucleus is modified. Such an occurrence would require a new absolute object-program file to be produced. If there are any changes in the Subroutine Library, the user may want to reassemble his programs to include these changes and then have a new object-program file produced.

Using the Reload Program

When it is desired to reload the absolute object program(s) contained on the file, use of the following control card signals the Processor to call in the Reload program:

```
1          8        16
$RELOAD    { Uxx   }
           { Iyy   } [,NAME=progname]
           { d[c]LIN }
                                  [,{ SRCH   }]
                                    { NOSRCH }

{ Uxx   }
{ Iyy   }
{ d[c]LIN }
```

The Unit options: The Uxx option specifies that the absolute object program is to be loaded from the utility unit, S.SUxx.

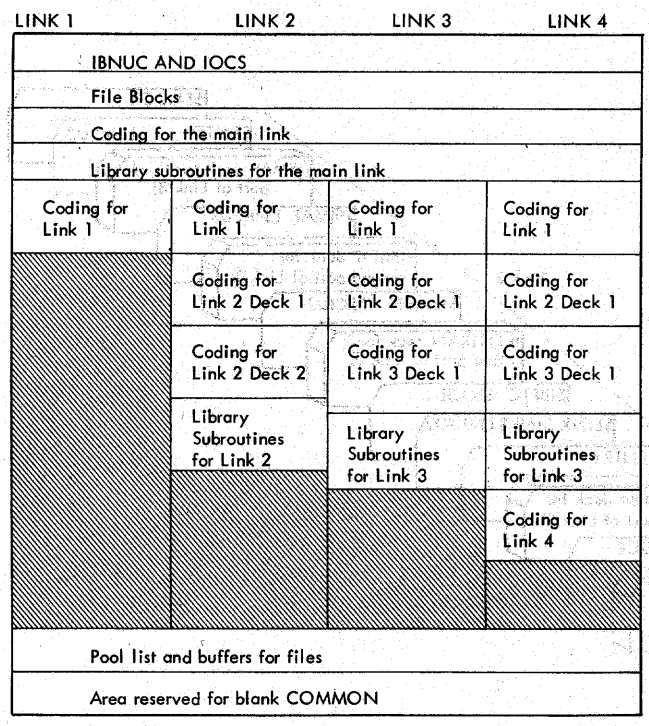


Figure 25. Core Storage Allocation for the Sample Chain Application in Figure 24

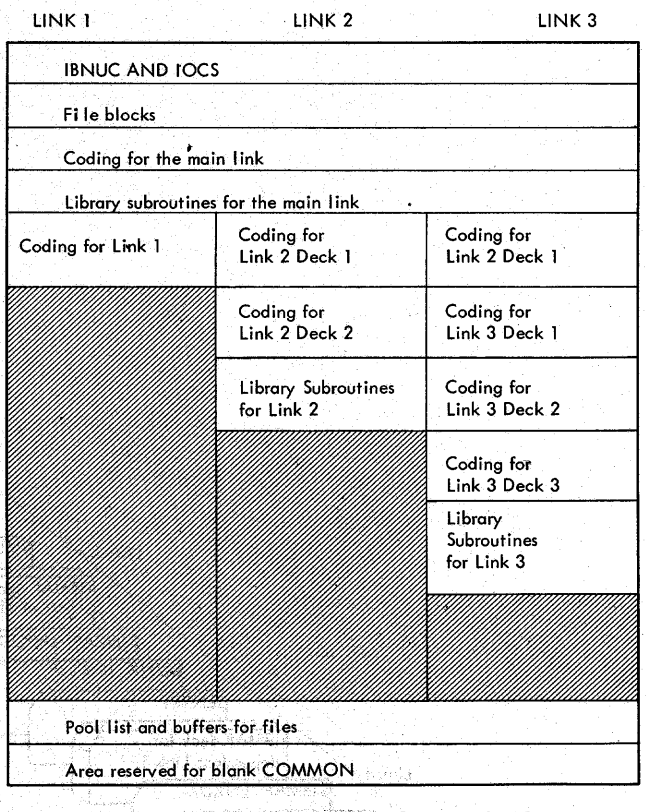


Figure 27. Core Storage Allocation for the Sample Chain Application in Figure 26

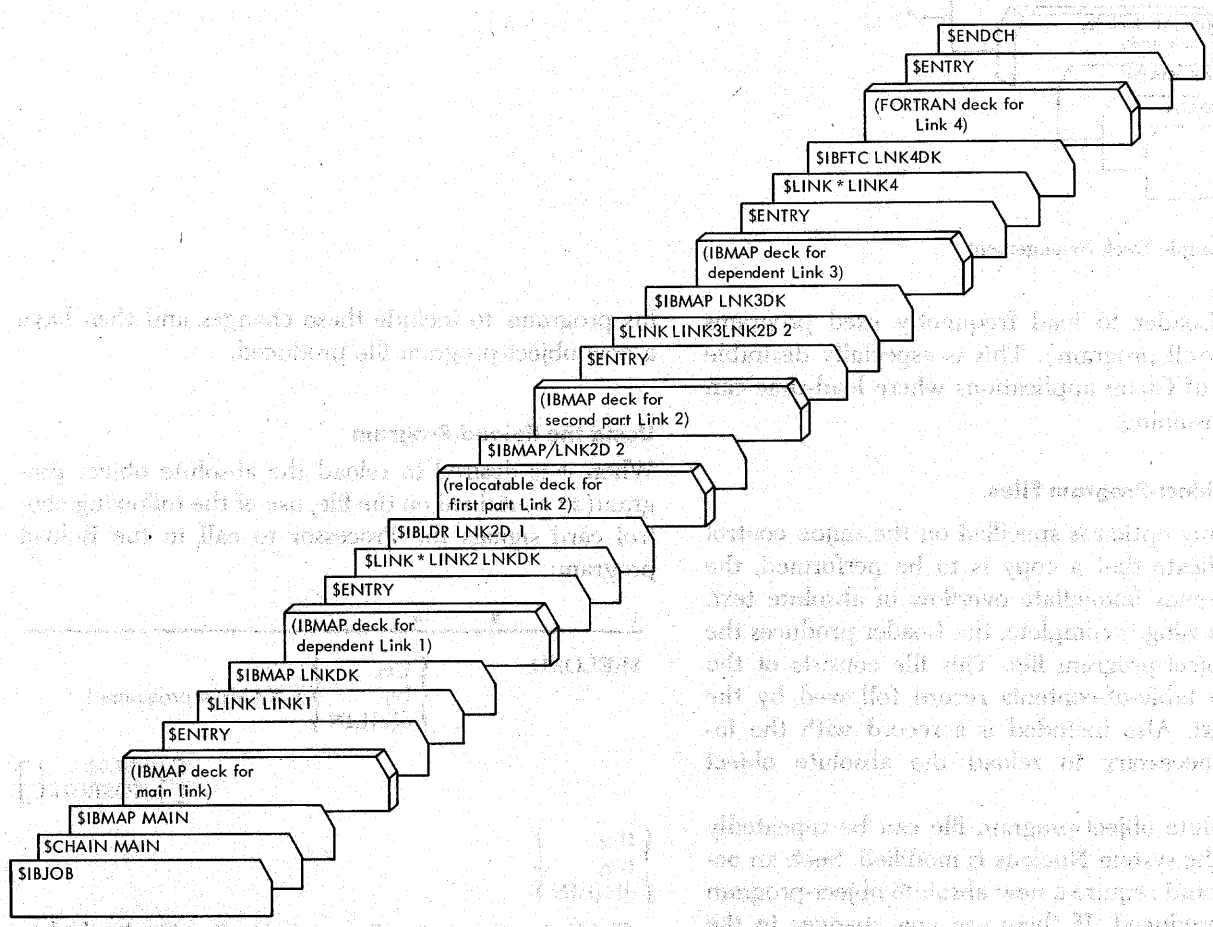


Figure 26. Sample Deck Arrangement Using $LINK*

The Iyy option specifies that the absolute object program is to be loaded from the unit that has been assigned the inter-system reservation code yy.

The d[c]LIN option specifies that a label search is to be performed to determine the unit from which the absolute object program is to be loaded. (The label search options are explained in the section titled "Input/Output Unit Assignment" in this publication.) If this option is specified, the $RELOAD card must be preceded by a special $LABEL card. The $LABEL card indicates the contents of the label associated with the absolute object-program file. The file name for this $LABEL card must be S.FBCP. The format of the $LABEL card is discussed in the section titled "Loader Control Cards" in this publication.

[,NAME=progname]

Progname is either the name of the main link in a CHAIN application at COPY time or it is the program name on the $IBJOB card at COPY time. This option must be taken if the SRCH option is specified on the $RELOAD card.

$$\left[,\left\{\begin{array}{c}\text{SRCH}\\ \underline{\text{NOSRCH}}\end{array}\right\}\right]$$

The Search options: The SRCH option specifies that S.SUxx is to be searched for the program indicated by the progname entry.

The NOSRCH option indicates that the absolute object program to be reloaded is at load-point on S.SUxx.

If neither SRCH nor NOSRCH is specified, NOSRCH is assumed.

The sample deck arrangements shown below illustrate the sequence of events in a typical Reload application.

```
1.  $DATE          051864
    $JOB           STACKING
    $OPEN          S.SU00=I15,REWIND
    $IBJOB    A    COPY=I15,MAP,NOGO,SOURCE
                   (program deck)
    $ENTRY
    $IBJOB    B    COPY=I15,MAP,NOGO,NOSOURCE
                   (program deck)
    $ENTRY
    $IBSYS
    $CLOSE         I15R,REMOVE
    $IBSYS
    $STOP
2.  $DATE          030964
    $JOB           PRODUCE TWO COPY TAPES
    $CLOSE         S.SU12,REWIND
    $CLOSE         S.SU05,REWIND
    $IBJOB    D    COPY=U05,MAP,DLOGIC,GO
    $IBSYS
    $CLOSE         S.SU05,REMOVE
    $IBJOB    E    COPY=U12,MAP,DLOGIC,GO
    $IBSYS
    $CLOSE         S.SU12,REMOVE
    $STOP
3.  $DATE          061664
    $OPEN          S.SU00=I10,REWIND
    $OPEN          S.SU01=I11,REWIND
    $OPEN          S.SU02=I12,REWIND
    $IBJOB         NOSOURCE
    $RELOAD        I10,NAME=A,NOSRCH
    $IBJOB         NOSOURCE
    $RELOAD        I10,NAME=B,SRCH
    $IBJOB         NOSOURCE
    $RELOAD        I11,NAME=D,NOSRCH
    $IBJOB         NOSOURCE
    $RELOAD        I12,NAME=E,NOSRCH
    $IBSYS
    $STOP
```

NOTE: The NOSOURCE option should be indicated on the $IBJOB control card when the Reload program is used. Any mounting messages that are required should be provided with a $PAUSE card preceding the $IBJOB card, or the file should be specified as DEFER.

Label Changing Procedure

Label information, which may vary from run to run for object program files, may be changed at reload-time by the following procedure:

1. Include all $LABEL cards necessary for the job immediately before the $RELOAD card. They will be processed by the Preprocessor and the information on them will be written in the file on work unit 1.
2. If $LABEL cards are included, the unit used for loading the absolute object program should be reserved by the programmer to prevent it from being chosen as work unit 1.

Execution

When the Reload program is called, it performs the following:

1. Processes the $RELOAD card
2. Initializes page heading cells
3. Finds the requested absolute object program
4. Reads and processes its information records and table of contents
5. Reads in the file control blocks
6. Processes the new label information and assigns input/output units
7. Initializes the Nucleus for the object program
8. Tests for program interrupt
9. Types the BEGIN message
10. Zeros storage
11. Transfers control to the System Loader

The System Loader then loads the object program.

If the System Loader finds that the device is incorrectly positioned, a permanent error occurs, since the device cannot be correctly positioned.

Programming Cross References

Central to the design of the Loader is the facility to have cross-references between decks, regardless of the original source language. One example of this is the sharing of file descriptive information discussed in the section, "Object Program Files."

An understanding of the MAP and FORTRAN languages is helpful in understanding this section. The general reader may skip the section.

Macro Assembly Program

Several MAP operations provide referencing between symbols that appear in several program segments that

are intended for separate assembly. When these operations are used within a program segment, they produce an entry in the control dictionary for that program deck. These control dictionary entries are used by the Loader to make the correct references between program segments and to assign the correct absolute address for all such symbols.

Figures 28 and 29 show parts of a program, consisting of two related program segments that reference each other.

REFERENCEABLE CONTROL SECTIONS

Two operations, CONTRL and ENTRY, may be used to designate a control section in a program segment as referenceable or replaceable by other program segments.

The coding in Figure 28 that begins with LEAST and ends with END may be defined as a control section by using the following CONTRL pseudo-operation:

```
RANGE    CONTRL   LEAST, END+1
```

The effect of this pseudo-operation would be to place the control section name RANGE, together with a length of nine locations, into the control dictionary for DECK1; this effectively defines the coding mentioned above as a control section with a referenceable name, RANGE.

The following is an example of the ENTRY pseudo-operation that is used to indicate that NEXT is an entry point in the coding in Figure 29.

```
         ENTRY    NEXT
```

The effect of this pseudo-operation would be to place NEXT together with the length zero in the control dictionary for Deck 2. NEXT is not necessarily the address of the first executable instruction of the deck. The END pseudo-operation specifies the first executable instruction of the deck as follows:

```
         END      name
```

where name is the nominal starting point of the program. The Loader will transfer control to *name* when this deck is specified on the $ENTRY card.

```
$IBMAP   DECK 1
  .        .
  .        .
  .        .
LEAST    CLA      LOCA
         CAS      LOCB
         CLA      LOCB
         TRA      ONE
ONE      CAS      LOC
         CLA      LOCC
         TRA      TWO
TWO      STO      LOW
END      TRA      NEXT
  .        .
  .        .
  .        .
RANGE    CONTRL   LEAST,  END+1
         EXTERN   LOCA, LOCB, LOCC, LOW, NEXT
```

Figure 28. Sample Program with Cross References Between Decks

```
$IBMAP   DECK2
  .        .
  .        .
  .        .
LOCA     DEC      0
LOCB     DEC      0
LOCC     DEC      0
LOW      DEC      0
  .        .
  .        .
  .        .
NEXT     TSX      S.GETL,4
         PZE      FIL1,,LOCA
  .        .
  .        .
  .        .
         TRA      RANGE
         ENTRY    LOCA
         ENTRY    LOCB
         ENTRY    LOCC
         ENTRY    LOW
         EXTERN   RANGE
         ENTRY    NEXT
```

Figure 29. Sample Program with Cross References Between Decks

REFERENCING CONTROL SECTIONS

The EXTERN pseudo-operation references other program segments. This operation specifies that the symbols in the variable field are used in this program segment but are not defined within it. These symbols are defined in other program segments.

The following is an example of the EXTERN pseudo-operation that is used to indicate that the symbols LOCA, LOCB, LOCC, LOW, and NEXT are not defined in Deck 1 (Figure 28), but are defined in other segments.

```
         EXTERN   LOCA,LOCB,LOCC,LOW,NEXT
```

FORTRAN

The FORTRAN statement, COMMON, is used for sharing data areas with other program segments. Linkages to other program segments can be made either by a CALL statement or by a function name usage.

Example: Referencing between common data areas. Figure 30 shows a simple case consisting of three adjacent data areas that are six, three, and five words long, respectively.

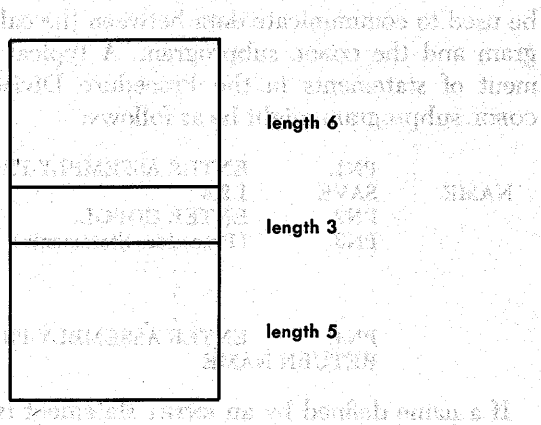


Figure 30. Adjacent Data Areas in Storage

The following FORTRAN coding defines such an area with a label DATUM:

```
COMMON/DATUM/ A(6), B(3), C(5)
```

In the MAP language, the following sequence also defines such an area:

```
X       BSS     6
Y       BSS     3
Z       BSS     5
```

If the coding sequences are part of FORTRAN and MAP programs that are to be run together, the addition of the following statement to the MAP sequences effects the required referencing:

```
DATUM     CONTRL   X,Z+5
```

This procedure ensures the equivalence of the statements, and only one such area will be provided by the Loader if the programs are loaded and/or run together.

COBOL

The program cross-reference facilities of the Loader (IBLDR) can be utilized in a COBOL program by using the CONTROL and FILE-REFERENCE statements in the SPECIAL-NAMES paragraph of the Environment Division and the ENTER verb in the Procedure Division. The Loader facilities enable the COBOL programmer to (a) define common data or procedure sections between two or more programs, (b) cause subroutines to be loaded from the Subroutine Library, (c) use the COBOL program as a main program linking to separately compiled routines or subprograms, (d) use the COBOL program as a subprogram, and (e) define common files between separately compiled but commonly loaded programs. Each of these procedures is described in detail below.

DEFINITION OF COMMON DATA AREAS OR PROCEDURE SECTIONS

The CONTROL statement in the SPECIAL-NAMES paragraph allows the user to specify that certain-reserved data-areas or blocks of instructions are considered to be control sections by IBMAP and the Loader. These control sections can be complete record descriptions in the Working-Storage Section or Constant Section of the Data Division, paragraphs or sections in the Procedure Division, or the out-of-line subroutines generated for a particular file. Record descriptions in the File Section *cannot* be designated as control sections since records are located in the buffers, rather than transmitted by IOCS, and consequently the compiler reserves no storage for these record areas.

The following restrictions must be observed in definitions of common data areas or procedure sections.

Data Areas: If a data-name is designated as a control section, the user must ensure that the size of the area reserved for it be identical to that of any other commonly named control sections to be loaded at the same time. The actual amount of space reserved by the Loader for a given control section is normally determined by the space occupied by the *first* appearance of a commonly named section. All other appearances of the named section are referenced to the beginning of the first one. Thus, portions of adjacent storage can be destroyed when storing into an area, or erroneous data can be extracted if each commonly named control section is not the same size.

Procedure Sections or Paragraphs: If two or more separately compiled procedure sections or paragraphs are given the same control-section name, the Loader retains only one in storage (normally, whichever one appears first at load time) and deletes the others. The Loader also adjusts all references to the address of the section actually loaded. To ensure that a return is made to the proper program after the control-section procedure has been executed, these procedures should either be entered by means of a PERFORM statement or ended with a GO TO statement. (The PERFORM and GO TO statement should refer to the COBOL procedure name and not the CONTROL name.) If a control-section procedure is referred to by a PERFORM statement, the entire procedure must be within the same control section to ensure proper execution.

If a COBOL *section-name* is designated as a control section, all other commonly named procedure control sections must occupy (when assembled) the same amount of storage. This restriction does not apply if only COBOL *paragraph-names* are designated as control sections, which is the normal case.

Out-of-line File Material: The out-of-line file material option should not normally be used unless the other commonly named control section is also out-of-line file material in a COBOL program. If this option is used, there should be a FILE-REFERENCE clause (Environment Division) giving the file the same external name in both programs; the FD entries and record descriptions (Data Division) should be the same, and

the OPEN specification (Procedure Division) should be the same.

If the out-of-line file material includes generated data areas, as for Type-2 OCCURS . . . DEPENDING ON . . . files or files assigned to PP or OU, these areas will be included in the control section.

LOADING SUBROUTINES FROM THE SUBROUTINE LIBRARY

Following an ENTER ASSEMBLY-PROGRAM statement in the Procedure Division, use of an EXTERN for a library-subroutine name, or a CALL to a library-subroutine entry point causes the named or called subroutine to be loaded with the compiled COBOL program.

Linking the COBOL Program to Separately Compiled Routines or Subprograms: Either a GO TO statement with an operand defined by a MAP-language EXTERN statement or a MAP-language CALL statement used after an ENTER verb can be used to link a COBOL program to separately compiled routines.

If the CALL statement is used, the normal return is to the statement following the CALL. This may be another CALL statement. If the CALL is the last MAP-language statement in an ENTER ASSEMBLY-LANGUAGE portion of the Procedure Division, the normal return is to the first COBOL statement following the ENTER ASSEMBLY-LANGUAGE portion (which must end with an ENTER COBOL statement).

Paragraph or section-names may be specified as alternate returns in the CALL statement. Any names defined in the COBOL program that are specified as arguments or returns of the CALL need not be operands of an ENTRY statement.

File names, COBOL literals no greater than six characters or ten digits, and level 01 or level 77 data names in the Working-Storage or Constant sections may be used as CALL arguments.

If the GO TO *name-defined-by-EXTERN* form is used to link to a subprogram, the return must be made to a procedure name defined by an ENTRY statement. Any data areas common to the main program and the subprogram(s) should be included in control sections. Data areas within a control section may be referenced by address-adjustment using the beginning location of the control section as a base.

COBOL Subprograms: Either a SAVE statement or a procedure name specified in an ENTRY statement must begin a subprogram written in COBOL. If the SAVE statement is used, the return to the calling program should be made by a RETURN statement specifying the name of the SAVE statement as its operand. Any of the alternate returns may be specified in additional RETURN statements.

No provision exists in COBOL for interrogating a list of arguments in the CALL statement that calls the subprogram. Data-names specified as control sections may be used to communicate data between the calling program and the COBOL subprogram. A typical arrangement of statements in the Procedure Division of a COBOL subprogram might be as follows:

```
        PN1.    ENTER ASSEMBLY-PROGRAM.
NAME    SAVE    1,2,4
        PN2.    ENTER COBOL.
        PN3.    (Procedure Statements)
                .
                .
                .
        PN4.    ENTER ASSEMBLY-PROGRAM.
        RETURN NAME
```

If a name defined by an ENTRY statement is used as the entry point for the subprogram, return to the calling program should be made by a GO TO *name-defined-by-EXTERN* statement form. Typical statements in the Procedure Division might be:

```
        PN1.    ENTER ASSEMBLY-PROGRAM.
          ENTRY PN3
          EXTERN CALLER
        PN2.    ENTER COBOL.
        PN3.    (Procedure Statements)
                .
                .
                .
              GO TO CALLER.
```

Defining Files Common to Two Programs: If two COBOL programs are to share the same file(s) when loaded together, each program must have:

1. A FILE-REFERENCE clause (SPECIAL-NAMES paragraph) providing a common reference-name for the file.
2. A CONTROL clause (SPECIAL-NAMES paragraph) for the out-of-line file material generated for the file (see "Out-of-Line Subroutines," above), with a common external name that is not the same as the FILE-REFERENCE reference-name.

If these requirements are met, only one buffer area (or set of areas, if double-buffered) and one file control block will be generated for each file shared by two programs. Also, only one set of linkages to IOCS subroutines and, if applicable, one set of generated data areas will exist for the file.

The descriptions of the file in the Environment Division, in the FD entry (including record descriptions), and in the OPEN statement must be the same in the two programs. If they are different, only the file information from the first program loaded will apply to the file and unpredictable results may ensue from the second program.

Compiler and Assembler Diagnostic Messages

The Error Editor section of the Macro Assembly Program processes all diagnostic error messages from the

Macro Assembly Program, the COBOL Compiler, and the FORTRAN Compiler.

If the assembler or compilers encounter a card with an error, the card is flagged with a letter, either W or E, in the program listing.

The W flag warns the programmer that the card may be in error; however, since the error, if any, is trivial, execution will be allowed. For example, the following FORTRAN statement, from which a comma has been omitted, would be flagged with a W:

```
GO TO K (17, 12, 19)
```

The correct statement is:

```
GO TO K, (17, 12, 19)
```

An E flag indicates a more serious error. Depending on the severity of the error, loading and/or execution may be deleted. In addition, at the end of the listing, an error message will be printed indicating the nature of the error in more detail. The format of these messages is as follows:

severity code statement number error number message

The severity code identifies the procedure taken for the error, as listed below:

0	Warning message only.
1	Mild error. Loading and execution of the object program are not affected.
2	Definite error. The object program is not loaded into core storage. However, the Loader does produce a storage-allocation map and/or a storage-allocation list, if these have been requested on the $IBJOB card by means of the Map and Logic options, respectively.
4	Serious error. The object program is not loaded. Options on the $IBJOB card that require the use of the Loader are not processed. The first error of severity level 4 halts the production of a binary deck and causes a $FAIL card to be punched. Assembly of the source program continues.
7	Catastrophic error. All processing of the source statements terminates. The assembly listing is incomplete.

The statement number identifies the flagged MAP statement to which the message corresponds; the error number uniquely identifies the message. The message specifies the nature of the error.

FORTRAN and COBOL error messages are correlated to the source program by internal serial numbers (ISN for FORTRAN; CSN for COBOL). Each input statement to FORTRAN and COBOL compilers is assigned an Internal Serial Number by the compiler. This number appears in the listing of the input statements.

Introduction to the Subroutine Library (IBLIB)

This section describes the Subroutine Library, which contains a set of relocatable subroutines for system and programmer use. These subroutines are available to the user through the Loader, which incorporated them, as required, in the object program at load time. At system assembly time, many of these subroutines are also incorporated in absolute form into the system components that use them.

Subroutines may be added to or deleted from the Subroutine Library by using the System Editor (for further information, see the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339). The System Editor accepts, for editing into the Subroutine Library, programs that are written in the FORTRAN, COBOL, or MAP languages, or programs that are in the relocatable binary format. The System Editor passes these programs to the appropriate processor.

The following types of subroutines are available in the Processor Monitor sections of the Subroutine Library:

1. The Input Editor
2. The Output Editor
3. The Punch Editor
4. The Snapshot Subroutine
5. The Checkpoint Subroutine
6. The Post-Execution Routine

A detailed description of these routines appears later in the text.

The following types of subroutines are available in the FORTRAN sections of the Subroutine Library:

1. Mathematical routines
2. System routines for input and output editing and for conversion of data under control of FORMAT statements
3. System routines for communications with the System Monitor and Processor Monitor for such items as standard diagnostic procedures, machine indicator testing, and loading control

FORTRAN input/output routines are discussed later in the text. FORTRAN mathematical routines are described in the publication *IBM 7040/7044 Operating System (16/32K): Subroutine Library (FORTRAN IV Mathematical Subroutines),* Form C28-6806. FORTRAN system routines are described in the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339.

The Input and Output Editors

The Input, Output, and Punch Editors provide the user with input, output, and punch file handling routines that can be incorporated in his object program

by using the standard CALL statement of the FORTRAN and MAP languages. If the Punch Editor is called, the Output Editor is also included.

The System Input File

The following CALL statement is used in the object program for input from S.SIN1:

```
CALL    JOBIN
```

This entrance to the Input Editor is used by the compilers, the assembler, the Loader, and the object program to get a logical record from the system input file or its alternate. The file is automatically opened, without rewind or labeling procedures, when the first CALL statement is made. A logical record is located in the input file buffer; if the record is acceptable, the Input Editor returns with the accumulator set to:

```
pfx     ia, , n
```

where:

if pfx
=PZE, the record is BCD.
=PON, the record is BCD with a $ in column 1.
=MZE, the record is binary.

ia
=the initial address of the record in the buffer.

n
=the number of words located (14 if the record is BCD, 27 or 28 if the record is binary).

The following is a list of conditions that cause the record to be unacceptable. In each case, the processor application (and the job) is terminated, and control is returned to the System Monitor.

1. A redundancy error on the system input file or its alternate, if the unit is not a card reader
2. An end of file on the system input file
3. An unsuccessful search for a matching $ card on the alternate input file (when SRCH was specified as an option on the $IEDIT card)

The following is the format of the CALL statement used by the object program to close the system input unit (S.SIN1):

```
CALL    INCLOS
```

This entrance is used by any system part except the System Monitor. It may be used by the object program to close the system input file and its alternate; however, this is not a final close. The compilers, the assembler, the Loader, and the object program must never close the system input file with an end-of-file procedure.

The System Output File

The following CALL statements are used in the object program for output to S.SOU1:

```
CALL    JOBOU (list) or
CALL    JOBOUL (list)
```

This entrance to the Output Editor is used by the compilers, the assembler, the Loader, and the object program to place output into the buffers for the system output file or its alternate. The file is automatically opened without rewind or label handling when the first CALL statement is made. The logical output is placed into the buffer in accordance with the user's specifications.

The symbol list is the initial address of a set of parameters that have the following form:

```
list    PZE     n
        pfx     A1, T1, M1
         .       .
         .       .
         .       .
        pfx     An, Tn, Mn
```

where:

n=the number of words in the list following the first word.

if pfx
=PZE, start a new line with a single space.
=PON, start a new line with a page restore.
=PTW, start a new line with a double space.
=PTH, start a new line with spacing controlled by first character of output line.
=MZE, continue the current line.

NOTE: When pfx PTH is used, the user has assumed responsibility for spacing, page overflow, and page count. The user can maintain the line count by decrementing the counter L.PGLN by the number of lines that are used for output. L.PGLN is initially set to 57_{10} and is reset when it decreases to zero.

A_j
=the location of the first word of the text.

if T_j
=0, the initial byte=0 and M_j is in words; and

if T_j
$\neq$0, the initial byte=T_j-1 and M_j is in characters.

M_j
=the word count or character count, depending on T_j.

In the above j may be 1 . . . n.

Page heading and page numbering are automatic and are printed at the top of every new page.

JOBOUL is used for writing complete lines. The first character of the line must be a blank. When JOBOUL is used, pfx MZE is invalid, t_j must be 0, and M_j must be the word count.

Page Heading: One field of 17 words, called PAGHD, and one field of 14 words, called SUBHD, are provided as entry points within JOBOU for page heading information. These fields must be initialized by the object program. The contents of S.SHDR are inserted into the first five words of PAGHD on the first entry to JOBOU.

Page Numbering: A one-word page number field, PGNUM, is provided within JOBOU for page numbering. This field is set to the address of S.PGCT on the first entry to JOBOU and is incremented by 1 before a heading is written. This incrementation is performed by the routine L.UPPG. This routine will take effect only if the

file has been opened by the Output Editor. The calling sequence to this routine is:

```
TSL     L.UPPG
```

The following CALL statement is used in the object program to close the system output file and its alternate:

```
CALL    OCLOS
```

This entrance is used by the compilers, the assembler, the Loader, and the object program to close the system output file and the alternate output file; however, this is not a final close. The compilers, the assembler, the Loader, or the object program must never close the system output file with an end-of-file procedure.

The System Punch File

The following CALL statement is used in the object program for output to S.SPP1:

```
CALL    JOBPP (list)
```

This entrance is used by any system component and by the object program so that the output to be punched can be placed into the system punch file buffers or into the system output file buffers. The appropriate file is opened automatically. It is rewound if applicable and its label is checked if necessary.

list is the location of one parameter having the following form:

```
list    pfx     card
```

where:

pfx
= PZE, if the record is BCD.
= MZE, if the record is binary.

card
= the initial address of a block of 14 words if the record is BCD, or a block of 24 words if the record is column binary. Columns 73-80 of a column binary card will be taken from the field PPLBL, described below.

Sequencing of binary cards is provided automatically from the PPLBL field. PPLBL is an entry point within JOBPP that may be set by the user. It is an eight-character card label (BCD) and is left-justified. It is initially set to:

```
BCI     2,00000001bbbb
```

The first four bytes are assumed to be alphabetic, the next four numeric, and the last four blank. The Punch Editor increases the numeric bytes by 1 after each binary record is placed into the buffer.

The following CALL statement is used by the object program to close the system punch file:

```
CALL    PCLOSE
```

This entry is used by any system component and by the object program to close the system punch file. This action releases buffers only. The system components and object program must never close the system punch file with an end-of-file procedure.

User Punch Routine: An object program writing on S.SPP1 but not using the Punch Editor, in an installation where system punch output may be combined with system print output, must test the Punch File Open bit (bit 17 in location S.SFLG). If the Punch File Open bit is one, the object program must open S.SPP1 without repositioning. If that bit is zero, the object program must:

1. Open S.SPP1 with repositioning and label checking.
2. Set the Punch File Open bit to one.

The Snapshot Subroutine

A snapshot routine is provided in the Subroutine Library for recording the console and selected areas of core storage. This routine will be incorporated in any object program that calls it with the proper calling sequence. This useful program debugging facility and its calling sequence are described in the publication *IBM 7040/7044 Operating System (16/32K): Debugging Facilities,* Form C28-6803.

The Checkpoint Subroutine

A checkpoint routine is provided in the Subroutine Library so that restart may be accomplished after an interrupt in the processing of a job. This routine will be incorporated in any object program that calls it with the proper calling sequence. This routine and its calling sequence are described in the section, "Checkpoint," earlier in this publication.

The Post-Execution Routine

The Post-Execution routine is provided in the Subroutine Library to ensure return to System Monitor control. Every object program upon completion must transfer to this routine with the calling sequence, TRA S.JXIT.

FORTRAN Files

Constant Units

Any FORTRAN source program input/output statement that references a constant unit (for example, READ (1,10)A, where the reference is to the constant FORTRAN logical unit 1) causes the library File routine corresponding to that unit to be loaded with the object program. A FILE routine contains a Macro Assembly Program FILE pseudo-operation that will be generated into a 19-word file control block and its associated buffer(s) by the Loader (IBLDR). The File routine determines various file specifications, such as unit assign-

ment, block size, number of buffers, record type, and length. These File routines are described in the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339; file control blocks are described in the publication *IBM 7040/7044 Operating System (16/32K): Input/Output Control System,* Form C28-6309. The unit assignment specification establishes the correspondence between FORTRAN logical units and system units, as shown in Figure 31.

FORTRAN Logical Input/ Output Unit	Library FILE Routine	External FILE Name	System Unit Assignment	System Unit Description
0	F00	FTC00.	S.SU00	Utility 0
1	F01	FTC01.	S.SU01	Utility 1
2	F02	FTC02.	S.SU02	Utility 2
3	F03	FTC03.	S.SU03	Utility 3
4	F04	FTC04.	S.SU04	Utility 4
5	F05	—	S.SIN1	System Input Unit
6	F06	—	S.SOU1	System Output Unit
7	F07	—	S.SPP1	System Peripheral Punch Unit
READ	FRD	—	S.SIN1	System Input Unit
PRINT	FPR	—	S.SOU1	System Output Unit
PUNCH	FPC	—	S.SPP1	System Peripheral Punch Unit

Figure 31. Correspondence Between FORTRAN Logical Units, FORTRAN File Routines, and System Units

Variable Units

Any FORTRAN source program input/output statement that references a variable unit causes the UTV input/output subroutine and all File routines to be loaded with the object program. The following is an example of such an input/output statement:

```
WRITE    (I, 10)A
```

In this example, the FORTRAN Input/Output Logical Unit I varies during execution of the program. The UTV routine takes the value of the variable unit at the time the variable input/output statement is to be executed, and references the IOU Table to determine which File routine (hence, which system file) is required. The IOU has the following typical format:

```
*    INPUT/OUTPUT LOGICAL UNIT TABLE ADDI-
*    TIONS OR DELETIONS SHOULD BE MADE BE-
*    TWEEN IOU AND NFILES

IOU       PZE    FIL00.
          PZE    FIL01.
          PZE    FIL02.
          PZE    FIL03.
          PZE    FIL04.
          PZE    FIL05.
          PZE    FIL06.
          PZE    FIL07.
NFILES    PZE    *-IOU-1
```

IOU is the point of reference for the UTV routine. Thus, IOU + nn (where nn is the current value of the variable unit, e.g., nn = 0 in the WRITE example above) contains the address of the correct File routine entry point. In turn, FILnn contains the location of the file control block.

The value of nn may range, in the above example, from +0 to +7.

Modifying FORTRAN File Specifications

Any of the FORTRAN File specifications, including unit assignment, may be modified by the user in either of two ways:

1. Temporarily, by including a $FILE card with his object program having the same external file name as that of the library File routine. This $FILE card must list not only the desired modifications, but also all other specifications necessary to describe the file completely. If the file is to be labeled, an associated $LABEL card must accompany the $FILE card.

2. Permanently, by assembling a File routine having the same entry point as that of the library File routine and then replacing the library routine with his own in an edit run. This new File routine must include all specifications necessary to describe the file completely. If the user wishes to label the file, a LABEL statement should immediately follow the FILE statement in the routine.

Notes:

1. A FORTRAN file may be labeled by including a $LABEL card with the object program. This card need not be accompanied by a $FILE card.

2. If a FORTRAN file is specified as TYPE1, the Record Count (RCT) must be 1.

3. If the BACKSPACE statement is used, the Record Count (RCT) must be 1.

Modifying the IOU Table

The IOU Table is modified as follows:

Deletions: If the user wishes to reduce the amount of storage space required when the variable units are referenced, he may replace with a PZE 0 the PZE FILnn. word corresponding to any unused file. This procedure prevents loading of the file control block of FILnn. and its associated buffer(s).

Additions: If the user wishes to use additional files on a regular basis, he should insert a PZE FILnn. word corresponding to each additional file in the appropriate place in the IOU Table. He must also assemble a File routine having the entry point of FILnn., and then make an edit run.

Buffer Pools

Whether the source program input/output statements reference constant and/or variable units, the user may considerably reduce the storage space required for file buffers. Required storage space is reduced by including with his object program a SPOOL card that lists the external file names of the files to be assigned to a buffer pool, specifies the number of buffers to be included in the pool, and gives the block size of the buffers. A sufficient number of buffers must be specified to accommodate all files that will be open simultaneously in the pool, and the buffers must be of adequate size.

FORTRAN Subroutines

The following library subroutines are of concern to the user.

FORTRAN Input/Output Subroutines

The reader should be familiar with the section, "Input/Output Statements," in the publication *IBM 7040/7044 Operating System (16/32K): FORTRAN IV Language*, Form C28-6329.

The routines that provide communications between object program input/output requests and the Input/Output Buffering System (IOBS) are:

The BST Routine: This routine backspaces the designated file one physical record, if it is in BCD mode, or one FORTRAN record, if it is in binary mode.

BSTIO is the entry point called by the source program statement:

```
BACKSPACE (unit)
```

The MAP calling sequence generated by the compiler for the routine is:

```
TSX     BSTIO. , 4
PZE     (file name)[1]
```

The EFT Routine: This routine closes the designated file with the end-of-file procedure without a rewind.

EFTIO is the entry point called by the source program statement:

```
END FILE (unit)
```

The MAP calling sequence generated by the compiler for this routine is:

```
TSX     EFTIO. , 4
PZE     (file name)[1]
```

The IOS Routine: This routine supervises object program input/output and checks for invalid operations.

IOSUP. is the entry point from the RWD and RWB routines to prevent invalid write operations on the system input file and to ensure that the current file is open in the correct mode and type.

CKEND. is the entry point from either the RWD routine or the RWB routine that is used to check for a $ card on the system input file.

REOFX. is the entry point from IOBS upon reading end of file.

RERRX. is the entry point from IOBS for correction of errors.

SYSCK. is the entry point from the EFT and RWT routines to prevent invalid operations on the system input, output, and peripheral punch files.

The RWB Routine: This routine controls the input and output of binary records.

TSBIO. is the entry point for a binary read; it is called by the source program statement:

```
READ (unit) list
```

The MAP calling sequence generated by the compiler for this routine is:

```
TSX     TSBIO. , 4
PZE     (file name)[1]
```

STBIO. is the entry point for a binary write; it is called by the source program statement:

```
WRITE (unit) list
```

The MAP calling sequence generated by the compiler for this routine is:

```
TSX     STBIO. , 4
PZE     (file name)[1]
```

BNLIO. is the entry point for the binary input/output list; it effects the input/output of binary data items.

The MAP calling sequence generated by the compiler for a nonsubscripted input list item is:

```
TSL     BNLIO.
STO     location
```

The MAP calling sequence generated by the compiler for a subscripted input list item is:

```
TSL     BNLIO.
 .       .
(indexing computation)
 .       .
STO     location, tag
```

The MAP calling sequence generated by the compiler for an unsubscripted output list item is:

```
CLA     location
TSL     BNLIO.
```

The MAP calling sequence generated by the compiler for a subscripted output list item is:

```
(indexing computation)
 .       .
CLA     location, tag
TSL     BNLIO.
```

RLRIO. is the entry point for the end of the binary input list.

[1]The file name is FIL00. , FIL01. , . . . FILnn. for logical units 0, 1, . . . nn, respectively. If a variable unit is given, this field is filled in at object time by the UTV routine.

The MAP calling sequence generated by the compiler for this routine is:

```
TSX     RLRIO. , 4
```

WLRIO is the entry point for the end of the binary output list.

The MAP calling sequence generated by the compiler for this routine is:

```
TSX     WLRIO. , 4
```

The RWD Routine: This routine controls the input and output of BCD records and the conversion of alphameric data in accordance with FORMAT specifications.

One input record is read:

1. Immediately before execution or scanning of the FORMAT statement begins.
2. Upon encountering a slash anywhere within the FORMAT statement.
3. When the end of the FORMAT statement is reached, and items remain in the input/output list.

If n records are read by any of these methods with *no intervening processing*, that is, the input buffer is never referenced, the first n−1 records are effectively skipped.

One output record is written:

1. Upon encountering a slash anywhere within the FORMAT statement.
2. When the end of the FORMAT statement is reached, and items remain in the input/output list.
3. When the end of the input/output list is reached.

If n records are written by any of the above methods with *no intervening processing*, that is, nothing is placed in the output buffer, the last n−1 records will be blank.

TSHIO. is the entry point for a BCD read; it is called by a source program statement:

```
READ    (unit, format) list
```

The MAP calling sequence generated by the compiler for this routine is:

```
TSX     TSHIO. , 4
PZE     (file name)[1]
(format indicator)[2]
```

STHIO. is the entry point for a BCD write; it is called by the source program statement:

```
WRITE   (unit, format) list
```

The MAP calling sequence generated by the compiler for this routine is:

```
TSX     STHIO. , 4
PZE     (file name)[1]
(format indicator)[2]
```

HNLIO. is the entry point for the BCD input/output list; it produces the input/output of BCD data items.

The MAP calling sequence generated by the compiler for a nonsubscripted input list item is:

```
TSL     HNLIO.
STO     location
```

The MAP calling sequence generated by the compiler for a subscripted input list item is:

```
TSL     HNLIO.
.
(indexing computation)
.
STO     location, tag
```

The MAP calling sequence generated by the compiler for a nonsubscripted output list item is:

```
CLA     location
TSL     HNLIO.
```

The MAP calling sequence generated by the compiler for a subscripted output list item is:

```
(indexing computation)
.
CLA     location, tag
TSL     HNLIO.
```

RTNIO. is the entry point for the end of the BCD input list.

The MAP calling sequence generated by the compiler for this routine is:

```
TSX     RTNIO. , 4
```

FILIO. is the entry point for the end of the BCD output list.

The MAP calling sequence generated by the compiler for this routine is:

```
TSX     FILIO. , 4
```

Other entry points to this routine, which are concerned primarily with format conversion, are listed in the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide*, Form C28-6339.

The RWT Routine: This routine closes the designated file with the end-of-file procedure and a rewind.

RWTIO. is the entry point called by the program statement:

```
REWIND  (unit)
```

The MAP calling sequence generated by the compiler for this routine is:

```
TSX     RWTIO. , 4
PZE     (file name)[1]
```

The SLI Routine: This routine controls processing of lists containing nonsubscripted array names for input.

SLIIO. is the entry point for input of nonsubscripted arrays.

The MAP calling sequence generated by the compiler for this routine is:

[1]The file name is FIL00. , FIL01. , . . . FILnn. for logical units 0, 1, . . . nn, respectively. If a variable unit is given, this field is filled in at object time by the UTV routine.

[2]The format indicator may be: PZE (location of format instruction list) for a fixed format, or MZE (location of BCD format, , FMTSC.) for a variable format.

```
TSX     SLIIO., 4
PZE     array location, , number of items in
        the array
```

The SLO Routine: This routine controls processing of lists containing nonsubscripted names for output.

SLOIO. is the entry point for output of nonsubscripted arrays.

The MAP calling sequence generated by the compiler for this routine is:

```
TSX     SLOIO., 4
PZE     array location, , number of items in
        the array
```

The UTV Routine: This routine establishes correspondence between FORTRAN logical units and system units at object time. (See Figure 31.)

UTVAR. is the entry point called by a variable unit designation in the source program input/output statement.

The MAP calling sequence generated by the compiler for this routine is:

```
CLA     (logical unit value)
TSX     UTVAR., 4
```

Using the FORTRAN Input/Output Subroutines

The MAP programmer who wishes to use FORTRAN input/output library routines for reading and/or writing records should use the following method.

1. Call the initialization routine as specified under RWB/RWD. For example, for a BCD write, this step would consist of:

```
TSX     STHIO., 4
PZE     (file name)
(format indicator)
```

2. Set up an input/output list with successive entries to BNLIO, or HNLIO., or use routines SLI and SLO for unsubscripted array input/output. For the single unsubscripted BCD output item X:

```
CLA     X
TSL     HNLIO.
```

For output of an entire BCD array:

```
TSX     SLOIO., 4
PZE     array location, , number of items in
        array
```

3. End the input/output list as specified under RWB/RWD. A BCD output list would therefore terminate with:

```
TSX     FILIO., 4
```

4. In addition, when BCD input/output is desired, the programmer must provide a FORMAT statement:
   a. By including it as a series of out-of-line instructions which are of the type described in the section, "The FORTRAN Compiler," in the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339, under the RWD routine and the various conversion routines.
   b. By reading in the FORMAT statement at object time.
   c. By including within the program a BCD FORMAT, which is referred to in the initial input/output calling sequence as a variable FORMAT.

The programmer will find that the way of providing a FORMAT statement described in item 4a is extremely flexible. For example, variable group and field counts are possible by using the LXA instruction instead of the AXT on index registers 1 and 2 within the FORMAT statement. Also, tests may be made within the FORMAT statement to determine which part of the format to execute next. Branches outside the FORMAT statement for intermediate computation are possible, if care is exercised.

FORTRAN System Routines

These routines are not of general concern to the user and are described in the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339.

Update Facilities

It is often desirable to store program decks in the form of records on magnetic tape rather than to maintain large files of punched cards. The Update program provides a convenient means of creating and maintaining such a master file. This program is provided as part of the 7040/7044 Operating System and is intended to facilitate the maintenance of tape files of symbolic or binary program decks. It has, in addition to its primary update facilities, the capability of generating tape files from punched card decks and of listing system output tapes.

The options available to perform these file maintenance functions are:

Input Option: This option provides for the creation of a new master file from punched cards. Decks being placed on the master file may be serialized as this tape is being generated.

Update Option: This option provides for the deletion, insertion, modification, and renumbering of the program decks on the master file. The master file may be considered a "storage pool" of current binary and/or symbolic program decks. Facilities are provided to generate an input tape for the 7040/7044 Operating System from the decks contained in the storage pool.

Output Option: This option provides the facilities to list system output tapes on the system output unit (s.SOU1). Any records on the output tape intended for punch output will be processed by the System Punch Editor as output on the System Peripheral Punch (s.SPP1). This option is provided for the installation that does not have tape-to-card and tape-to-printer equipment available.

File Description

The files maintained by this program consist of 80-column symbolic and/or binary card images, which may be serialized in columns 73 through 80. All files created by the Update program are comprised of blocked Type 3 records; however, system control cards are unblocked. (A description of Type 3 records may be found in the publication, *IBM 7040/7044 Operating System (16/32K): Input/Output Control System,* Form C28-6309.) These files may be either unlabeled or labeled; if labeled, they must satisfy the installation label option.

The installation label option is specified by a system assembly parameter, LABELS. If the parameter constant is zero, no labels will be created or checked. A constant of 1 indicates that labeling is optional and that labels will be processed as indicated by the parameter in the $RUN control card. A constant of 2 indicates a label installation, and all files will be checked for labels. (See publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339.)

The maximum blocking factor for the output file is ten logical records per physical record. The maximum blocking factor for the input file is 15 logical records per physical record.

Transaction File

The user controls the operation of the Update program by means of the transaction file. This file contains the various control cards that direct the Update program, as well as data that is to be used for the update function. The transaction file is usually only one of many jobs on the system input tape. Therefore, after the update operation is completed, control is returned to the System Monitor to proceed to the next job. This makes it possible to utilize the output of the update run for the next job.

The transaction file is defined as all the cards that appear after the $EXECUTE UPDATE control card, up to and including the $ENDRUN control card. Although used in conjunction with an update operation, control cards outside of these limits are not part of the transaction file and are processed by other portions of the Operating System.

Level of Updating

Update action may be performed on two levels: the individual card level and the deck level.

A deck is defined as a series of cards with a name. The first card of the deck must contain a $ in column 1 and the name of the deck starting in column 8. The end of the deck is recognized when a $ control card with a different name is encountered. If the $ control card has a blank in column 8, and is not a $IBSYS or $STOP card, it is not recognized as indicating the end of the deck, but rather as a control card that is part of the deck.

The update action possible at the deck level is that of removing or replacing a deck on the old master file, or inserting a new deck onto the master file being created.

Update operations at the card level are accomplished by using the serial numbers that appear in columns 73 through 80 of the cards. (The format of COBOL statements reserves columns 1 through 6 for a serial number, columns 8 through 72 for text and columns 73 through 80 for program identification. In order to update a COBOL deck, the serial number used for modification must appear in columns 73 through 80. Therefore, to use the update facilities for a COBOL deck the program identification must be replaced by a sequencing number. This can be accomplished by using an input or an update run before performing any modifications.)

Modifications at the card level are accomplished in the following fashion:

1. The deck to be modified must be located on the old master file and serialized.
2. The input file (old master file) must be positioned to the deck to be modified.
3. The modifications to the desired deck must be on the transaction file in ascending order.
4. The modification cards are incorporated into the new master file in the following manner:
   a. If the serial number on the modification card matches a serial number on the old master file, the modification card is copied onto the new master tape and the card on the old master file is deleted.
   b. The last character of the serial number is used for insertion of new cards into an existing deck. If the serial number on the modification card falls between two that are already on the old master file, the modification card will be inserted between them.
   c. Blank serial numbers are permissible on modification cards. Such cards must, however, follow a modification card that does have a serial number. The blank serial number cards will be placed on the new master file, following the modification card with a serial number.
5. Numbering facilities are considered to operate within deck limits. Numbering will stop at the end of a deck if no "end of numbering" parameter is specified.
6. The delete facility also operates within deck limits.

Special Mode of Operation

To provide a degree of compatibility with the various other update programs in use, a special mode of operation is provided for the 7040/7044 Update program. This special mode of operation is accomplished by not using the $LOCATE control card. With the use of a $LOCATE control card, a deck is defined as starting with a $ control card containing the name of the deck starting in column 8. The end of the deck is recognized by encountering another $ control card with a different name or a $IBSYS or $STOP card. When the $LOCATE control card is not used, the entire file is considered to be one deck, starting with the first card image and extending to the end of file (or the trailer label). This mode of operation will cause the following message to appear in the comments produced at the end of the run:

```
WARNING NO DECK CONTROLS GIVEN.
ASSUMING ONE DECK ONLY.
```

LIMITATIONS OF THE SPECIAL MODE OF OPERATION

When using the special mode of operation, the following factors should be considered:

1. Since the input tape is considered to be one deck, all cards on the tape must be in sequence. Even though there are numerous program decks present, all the cards on the file must be in ascending order.
2. Although only one program on the file is to be changed, the whole file will be copied.
3. Because the file is considered to be one deck, the $NUMBER control card will cause the whole file to be numbered if a terminating parameter is not specified.
4. The $DELETE operation will continue across deck boundaries as long as the card serialization is in ascending order.
5. Once the update process has started in this special mode, it may not be returned to the normal operation.

Requesting the Update Program

The Update program is on the System Library (S.SLB1) file and is loaded under control of the System Monitor. The control card used to call the Update program is $EXECUTE UPDATE. When it is loaded, the Update program, using the facilities of the system Input/Output Control System, directs the operation of the update run requested. Upon completion of the update run, control is returned to the System Monitor.

The following sequence of control cards is used to initiate an update job:

CONTROL CARD	FUNCTION
$DATE	The system date card used with all jobs.
$JOB	The system job card stating the name of the job.
$EXECUTE UPDATE	The control card that initiates loading of the Update program.
$RUN	The first control card of the Update program run.

Units Used During an Update Run

The Update program makes use of the system library unit, the system input unit, the system output unit, and system utility units. Figure 32 lists the standard units used during an update run and indicates the purpose for which each is used.

The Update program assumes that output files will appear on specific utility units. The programmer may include a $OUTPUT card to specify a particular unit for output. If the assumed output unit(s) is not available, the Update program assigns the first available units for output and notifies the operator and the programmer of the units selected. If a unit switch is necessary, a secondary unit is also indicated by a message.

The Update program assumes the indicated units for the input files. The programmer may indicate that he wishes to use a different unit by specifying the unit on a $ASSIGN control card.

Symbolic Unit	Description
S.SLB1	System Library Unit
S.SU01	Old Master File (input)
S.SU04	Auxiliary Master File (input)
S.SU00	New Master File (output)
S.SU03	Auxiliary Master File (output)
S.SIN1	Transaction File
S.SOU1	Listing

Figure 32. Unit Assignment for an Update Run

Using the Update Program

All operations of the Update program are governed by control cards in the transaction file. The control cards used for an update job may be divided into three basic categories, as follows:

1. System control cards (e.g., $IBSYS, $SWITCH, $IBEDT, etc.) that are processed by programs other than the Update program and must be outside the transaction file.

2. Update control cards used for entire decks (e.g., $RUN, $OBTAIN, $LOCATE, $MESSAGE, $REWIND, $PLACE, and $ENDRUN). These cards are used to define the run type and its limits, position the input file, and direct the insertion, deletion, copying, and placing of entire decks. Once operation on a card level has started, these control cards are not recognized until after the end of the deck is reached.

3. Cards used to update within a deck (e.g., $NUMBER, $DELETE and modification cards). Modification cards are used to direct update action within a deck by means of the serial numbers in columns 73 through 80.

The control cards, $NUMBER, $DELETE, and modification cards, are the only ones that will be recognized at the card level. If they are encountered before any $LOCATE card has been encountered (the $LOCATE control card starts the within-deck processing routine), the single-deck-file mode of operation (described earlier in this section as the Special Mode of Operation) will be entered and the within-deck processing routine will begin at once. If they occur at any other position in the transaction file, while within-deck processing is not taking place (such as immediately after a $REWIND control card, which itself immediately follows a $LOCATE control card), they will be treated as errors.

The control cards used with the Update program are as follows:

CONTROL CARD	FUNCTION
$RUN	Initializes the Update program.
$OBTAIN	Requests supplementary listings.
$ENDRUN	Terminates transactions.
$ASSIGN	Assigns auxiliary input units.
$LOCATE	Positions the input tape and requests update action.
$NUMBER	Requests resequencing.
$DELETE	Requests deletion of serialized cards.
$PLACE	Makes selective insertion for the output auxiliary file.
$MESSAGE	Types a message with an optional pause for operator action.
$OUTPUT	Assigns output units.

Control Card Formats

The update control cards are prepared in the formats shown in the following paragraphs.

$RUN: This control card provides the information necessary to initialize the Update program. It specifies the type of run desired, blocking factors, and label specifications. The format of a $RUN control card follows:

1	6	14	17	23
$RUN	[Run types]	[blocking factor]	[Date]	[LABEL]

29	35	47	51	57	68
[Ser. No.]	[ID]	[Retention Cycle]	[Ser. No.]	[ID]	[Retention Cycle]

CARD COLUMNS	CONTENTS	DESCRIPTION
1-4	$RUN	Control card identification.
5		Not used.
6-13	(*Run Type*)	
	UPDATE	UPDATE specifies that an input file is to be read and an output file is to be created on S.SU00 unless S.SU00 is unavailable or a $OUTPUT card is included to specify a particular unit.
	EXTRACT	EXTRACT specifies that an input file is to be read and an auxiliary master file is to be created on S.SU03 unit, unless S.SU03 is unavailable or a $OUTPUT card is included to specify a particular unit.
	blank	Blank specifies that an input file is to be read and both an output file and an auxiliary master file are to be created on S.SU00 and S.SU03, unless these units are unavailable or a $OUTPUT card is included to specify particular units.

CARD COLUMNS	CONTENTS	DESCRIPTION
	INPUT	INPUT specifies that an output file is to be created on S.SU00 from decks on the transaction file (S.SIN1). A $OUTPUT card may be included to specify that a particular unit is desired for output.
	OUTPUT	OUTPUT specifies that an input file is to be read from S.SU01 (unless a $ASSIGN card has been included to specify a particular input unit), and either a listing is to be created on S.SOU1, a deck is to be punched on S.SPP1, or both. (This is the Output Option referred to in the introduction.)
	NOCOPY	NOCOPY specifies that no output file will be produced on tape. The purpose of this option is to speed up processing when only a listing or index is desired.
	GENERATE	This mode of operation must be used only when placing an Update job on a stacked-input tape. It is necessary in order to avoid the processing of Update control cards when it is only intended to place them on the output file. In this mode, the parameter STOP must appear on the $ENDRUN control card that ends the Generate run.
	SYSREL	SYSREL specifies that an input file is to be read from S.SU01 (unless a $ASSIGN card has been included to specify a particular input unit) and an output file is to be created on S.SU00 unless S.SU00 is unavailable or a $OUTPUT card is included to specify a particular output unit. This option is intended for use with the relocatable master tape supplied to users, which represents, in the form of binary decks and control cards, the entire operating system set up as an IBEDT run that is able to create a new S.SLB1. The purpose of this option is to distinguish the run so that certain system configurations can be requested that result in automatic deletion or retention of $ control cards on the relocatable master. The requests are made on the $OBTAIN control card. The usual update functions are also allowed in a SYSREL run.
14-15	[Blocking Factor]	The blocking factor is a two-digit number (or blank), from 01 to 10, that specifies the output blocking factor. This blocking factor will be used with both the new master file and the auxiliary master file. The input files, if blocked, need not have the same blocking factor, but they must not have a blocking factor greater than 15.
16	blank	
17-21	[date]	The date of the run is placed in this field as yyddd, where yy is year and ddd is the day of the year. For instance, December 31, 1963 would be 63365. This field is present solely for compatibility with the 1401 Update Program. It is neither used nor checked, the date being taken from the Nucleus S.SDAT cell.
22	blank	
23-27	[LABEL]	If the parameter LABEL is present, the Update program will check for labels on the input tapes and use them for multireel operations. If this field on the control card is left blank, no labels will be expected.
28	blank	
29-33	[Serial Number]	The number placed in this field will go into the label of the new master tape (five digits).
34	blank	
35-44	[Identification]	This information is placed in the identification field of the new master tape (ten characters).
45	blank	
46-49	[Retention Cycle]	The four digits of this field are placed in the retention field of the new master tape. This four-digit number specifies the number of days for which the file is to be retained. If this field is blank, 000 will be used.
50	blank	
51-55	[Serial Number]	The label information for the auxiliary master tape (output) is specified in columns 51 through 71 in the same fashion as it was supplied for the new master tape in columns 29 through 49.
56	blank	
57-66	[Identification]	
67	blank	
68-71	[Retention Cycle]	

$OBTAIN: The $OBTAIN control card is used to obtain several forms of supplementary data during an update run. This control card may be used as often as it is desired, but the options specified by the preceding $OBTAIN control card are superseded by the latest control card. The format of the $OBTAIN control card is as follows:

1	9	18	27	36
$OBTAIN	(Option-1)	(Option-2)	(Option-3)	(Option-4)

Run Type	LIST or PRINT	Option SUMMARY	INDEX	DECK
UPDATE SYSREL INPUT	All cards placed on the new master file will be listed. PRINT will cause only BCD cards to be listed.	All program control cards including $NUMBER and $DELETE will be listed and all insertions or deletions will be listed and flagged.	All $ control cards on the new master file will be tabulated.	No effect.
EXTRACT	All cards placed on the auxiliary master file will be listed. PRINT will cause only BCD cards to be listed.	Same as for UPDATE.	All $ control cards on the old master file are listed, except for those pertaining to decks that are being deleted or that follow the last extracted deck.	All decks on the auxiliary master file will be punched.
Blank	Same as for UPDATE.	Same as for UPDATE.	Same as for UPDATE.	Same as for EXTRACT.
OUTPUT	No effect.	No effect.	No effect.	No effect.
GENERATE	No effect.	No effect.	No effect.	No effect.

● Figure 33. Supplementary Output

Options 1, 2, 3, and 4 may be any of the five available supplementary output specifications (LIST, PRINT, DECK, INDEX, or SUMMARY), or they may be left blank. If no $OBTAIN control card is used, the SUMMARY option will be provided. The following list shows the options that are initially assumed for each type of run.

UPDATE	SUMMARY
SYSREL	SUMMARY
EXTRACT	SUMMARY
Blank	SUMMARY
INPUT	None
OUTPUT	None
GENERATE	None
NOCOPY	None

Figure 33 indicates the output obtained with each of the options, when used with various run types.

The following parameters may be used in any of the parameter fields of the $OBTAIN control card. They are recognized only if the run type on the $RUN control card is SYSREL.

[{16K / 32K}] Core Size Option: This option causes the Update program to prepare a relocatable master for a 16K or 32K system. If neither option is specified, 16K is assumed.

[{LABEL / NOLABEL}] Label Option: This option causes the Update program to prepare a relocatable master for labeled or unlabeled files. If neither option is specified, NOLABEL is assumed.

The input file used in a SYSREL run will contain $ control cards for all possible core size and label options. Each of these $ control cards will have an identifying code. These codes, placed in columns 70 to 72 of the appropriate $ control cards, are as follows:

LAB The card containing this code is required for a labeled system, regardless of core size.

16K The card containing this code is required for a 16K system.

32K The card containing this code is required for a 32K system.

16L The card containing this code is required for a 16K labeled system.

32L The card containing this code is required for a 32K labeled system.

When the SYSREL run is made, undesired options are automatically deleted on the basis of the code and the request specified on the $OBTAIN card. For example, if 16K and LABEL were specified on the $OBTAIN card then all cards on the input file with identifying codes 32K and 32L would be deleted by the SYSREL run.

Any $ control card that is deleted will be flagged in the Update summary with the words, DELETED BY UPDATE. The $OBTAIN card, therefore, becomes a means of making a mass deletion, not by referring to serial numbers, but on the basis of a particular property.

$ENDRUN: This control card is the last card of the transaction file, and if no parameter is present, the Update program is terminated. If a deck name is specified, the program continues to move data from the old master tape to the new master tape, up to and including the deck specified. When an OUTPUT run is being performed, a $ENDRUN control card is required, even though operation is terminated when the end of the main input file is encountered.

After the job being executed is completed, all the units used are rewound, and the input units are unloaded. However, input units that have been assigned intersystem reservation codes are only rewound, since the intersystem code indicates that the unit may be used later in the job. Any unit that has been chosen for output because an assumed unit was unavailable is also unloaded. The format of the $ENDRUN control card is as follows:

1 9

$ENDRUN [{deck name / STOP}]

STOP Necessary only when the Generate option is used, i.e., when GENERATE appears on the $RUN control card.

$ASSIGN: This control card is used to assign input units. It need not be used unless an update run requires input decks from a particular input unit. A $ASSIGN card may also be used to assign an intersystem reservation code to an input unit. If a $ASSIGN card is not used, input is assumed to be on S.SU01, and alternate input on S.SU04. After processing the desired decks from the input file, the $ASSIGN control card is used to switch to the second input file. It is possible to have two input files open at any time but only one at a time may be used to read and move data to the output file. The file not in use will stay positioned where it was when the input files were switched. If more than two input files are required during a run, the main file is mounted on S.SU01 (primary unit), while the others are mounted wherever desired and assigned in turn as the alternate input file. When returning to an alternate input file, only the last alternate used will retain its position. The format of the $ASSIGN card is as follows:

```
1          9          18         23         29
$ASSIGN [primunit] [OPEN]  [file type]  [label option]
35                    58          65
[label information]   [reel option]  [secunit]
```

CARD COLUMNS	CONTENTS	DESCRIPTION
1-7	$ASSIGN	Control card identification.
8		Not used.
9	[primunit]	The primary unit for the file being designated as the input file. It may have one of the following forms: *System Utility Unit options:* {S.SUnn Unn[=Iyy] nn[=Iyy]} The system utility unit nn is used for the input file and optionally assigned intersystem reservation code yy. (Intersystem reservation codes are explained in the section "Input/Output Unit Assignment.") MAIN S.SU01 is used for the input file. ALT S.SU04 is used for the input file. *Intersystem Unit option:* Iyy The unit to which intersystem reservation code yy has been assigned is used for the input file. *Label Search option:* d[c]LIN[=Iyy] A search is to be made for a unit with a specific label for this job. The contents of the labels are specified in columns 35 through 57 of the $ASSIGN card. (Label search parameters are explained in the section "Input/Output Unit Assignment.") If =Iyy is added to the label search parameter, the unit found is assigned the intersystem reservation code yy.
18	[OPEN]	This field may contain OPEN or be left blank. If OPEN is present, the file specified by primunit is closed with rewind. The file control block is set according to the other options on this card, and the file is opened and selected as the input file to be read. *If this field is blank, any subsequent fields on this card are ignored.* In this case, the file (determined by primunit) is checked to insure that it is already open and, if so, is selected as the input file to be read. If this file is not open, the $ASSIGN card is ignored.
23	[file type]	This field indicates the record type for the file. It may be TYPE1, TYPE2, TYPE3, or blank. If blank, Type 3 will be assumed.
29	[label option]	This field indicates whether the file is labeled or unlabeled. The field may contain LABEL or NOLAB, or should be left blank. The field may be in conflict with the installation label option.
35	[label information]	This field is made up of the following subfields: Columns 35-39— file serial number Columns 41-50— file identification Columns 52-56— creation date If these fields are not blank, the contents are placed in the corresponding field of the file control block. If any of them are blank, zeros are placed in the corresponding field. The information is used for label checking.
58	[reel option]	May be REEL, REELS, or blank. If blank, then REEL is assumed. REELS must be specified if unit switching is to occur.
65	[secunit]	The same options apply here that apply to the primary unit. If this field is blank, the secondary unit is set equal to the primary unit.

$REWIND: This control card is used to rewind the current input tape. The format of the $REWIND control card is as follows:

```
1
$REWIND
```

CARD COLUMNS	CONTENTS	DESCRIPTION
1-7	$REWIND	Control card identification.

$LOCATE: The purpose of the $LOCATE control card is to perform insertions, deletions, replacements, or removal of decks when updating a master file. It is also used to position the input tape to a deck that requires modification. By using the (blank) run type option on the $RUN control card two master files can be created. All the decks located will be placed on the new master file, and selected decks will be placed on the auxiliary master file. The decks to be placed on the auxiliary master file must be indicated by the EXTRACT parameter on the $LOCATE control card.

It is possible to make update runs without using the $LOCATE control card. In this case, the file will be considered one deck, even though there are many decks on the file. This mode of operation is described earlier in this section as the "Special Mode of Operation." The format of the $LOCATE control card is as follows:

```
1          9          18        27          36
$LOCATE    deck       [action][EXTRACT]  [remove to ]
           name                          [deck name ]
```

CARD COLUMNS	CONTENTS	DESCRIPTION
1-7	$LOCATE	Control card identification.
8		Not used.
9	deck name	This parameter must always be used. It either specifies the name of the deck to which the input file is to be positioned or, for an INSERT action, it specifies the name of the deck to be inserted. As the input file is being positioned, the decks from the old master file are copied onto the new master file.
18	action [INSERT]	This action causes the deck on the transaction file following the $LOCATE control card to be placed on the new master file. The old master file must have been positioned to the point of insertion by a preceding $LOCATE control card.
	blank	When the action field is left blank, the $LOCATE control card serves two purposes: if a $DELETE, $NUMBER, or modification card follows, the indicated action will be performed on the deck specified on the $LOCATE card; if another $LOCATE card follows, the deck on the old master file is copied onto the new master file.
	[REPLACE]	The REPLACE action causes the deck on the old master file to be effectively deleted and the deck on the transacton file, following the $LOCATE control card, to be placed on the new master file.
	[REMOVE]	The REMOVE action causes the specified deck to be deleted (not copied). If the "remove to deck name" parameter is used, a series of decks will be deleted.
27	[EXTRACT]	This parameter causes the deck to be placed on the auxiliary master file. If EXTRACT had been a parameter on the $RUN control card, it need not be used on the $LOCATE control card. If both a new master file and an auxiliary master file are being generated, the EXTRACT parameter is required on the $LOCATE control card.
36	[remove to deck name]	If more than one deck is to be removed, the "remove to deck name" parameter specifies the last deck to be removed. Starting with the deck specified in the "deck name" field of the $LOCATE control card, all decks up to and including the one specified in this field are deleted.

$NUMBER: This control card is used to sequence new decks or to resequence existing decks. Before the numbering process can take place, the input file must be positioned to the deck requiring the action. This is accomplished with a $LOCATE control card. (Refer to the preceding paragraphs for a description of the $LOCATE operation.)

The $NUMBER control card can specify deck sequencing in two fashions:

1. Begin numbering at a specified card in the deck, and continue until the end of the deck.

2. Begin numbering from the point at which the file is positioned, and terminate at a specified card in the deck.

At least one parameter must appear on the $NUMBER control card.

The format of the $NUMBER control card is as follows:

```
1          9                18                27
$NUMBER[initial serial no.] [from serial no.] [to serial no.]
```

CARD COLUMNS	CONTENTS	DESCRIPTION
1-7	$NUMBER	Control card identification.
8		Not used.
9	[initial serial number]	This parameter specifies the initial serial number to be used on the new master file. If this field is left blank, the serial number of the previous card on the old master will be used.
18	[from serial number]	This parameter specifies where to start the sequencing action. The numbering begins on or after the serial number specified in this field. If there is no record having the specified serial number, sequencing starts at the lowest number greater than the "from serial number."
27	[to serial number]	Sequencing is terminated at the serial number specified by this parameter. If no "to serial number" is specified, sequencing continues until the next $ control card is encountered.

NOTE: For an INPUT run, only the first parameter is used. The cards are expected to have no serial numbers, since they are being sequenced for the first time. Sequencing is terminated as soon as a $ control card is encountered.

$DELETE: This control card is used to request deletion of serialized cards. It may be used to remove one or more serialized cards from a deck. To delete one card, only the "from serial number" parameter is necessary. To delete a series of cards, both the "from" and "to" serial number parameters are required. Deletion starts at or after the first serial number specified and continues up to the second one specified. If the first parameter is blank, deletion will start at the current position of the input file and continue up to the "to serial number" or, if that serial number is not on the old master file, deletion will continue to the serial number closest to but less than the "to serial number."

The format of the $DELETE control card is as follows:

```
1          9                  18
$DELETE    (from serial no.)  (to serial no.)
```

CARD COLUMNS	CONTENTS	DESCRIPTION
1-7	$DELETE	Control card identification.
8		Not used.
9	[from serial number]	Deletion starts with this serial number or, if this particular serial number is not in the deck, with the serial number closest to and greater than the specified serial number.
18	[to serial number]	Deletion terminates with this serial number or, if it is not present in the deck, with the number closest to and less than the specified serial number.

NOTE: Binary decks may be renumbered or individual binary cards replaced, inserted, or deleted in the same manner as symbolic decks are modified.

$PLACE: During an update run, it is sometimes desirable to put system control cards on the auxiliary master file, but not on the new master file. This can be accomplished with the $PLACE control card. All cards following the $PLACE control card and before the next update control card will be placed onto the auxiliary master file, but not on the new master file. The format of the $PLACE control card is as follows:

```
1
$PLACE
```

No parameters are used with the $PLACE control card.

$MESSAGE: The $MESSAGE control card allows the user to relay a comment to the operator during an update run. When the $MESSAGE card is encountered, the contents of columns 13 through 72 will be printed on the console typewriter. In addition, if columns 13 through 17 contain the parameter PAUSE, a halt for operator action will occur. Pressing START will allow the program to continue. $MESSAGE control cards may not be used in the middle of transactions to a given deck. They may be used before the first $LOCATE card and/or after all transactions to a given deck have been processed. $MESSAGE control cards may not be used in the "special mode of operation." The format of the $MESSAGE control card is as follows:

```
1          13
$MESSAGE [PAUSE] or [message to be printed]
```

$OUTPUT: The $OUTPUT control card is used to specify output unit assignments. If this card is not used, the Update program uses assumed units or, if the assumed units are unavailable, finds available units. Secondary units are used for automatic unit switching. When the $OUTPUT card is encountered anywhere in the transaction file, the file on the currently assigned output unit is closed. If the currently assigned output unit was chosen because an assumed output unit was unavailable, it is rewound and unloaded. The format of the $OUTPUT card is as follows:

```
1          9                 18 27                36
$OUTPUT    unit assignment     unit assignment for
           for new master      auxiliary new master
```

CARD COLUMNS	CONTENTS	DESCRIPTION
9	unit	Primary unit for new master file.
18	unit	Secondary unit for new master file.
27	[unit]	Primary unit for auxiliary new master file.
36	[unit]	Secondary unit for auxiliary new master file.

The unit specifications may be any of the following:

System Utility Unit Options:

S.SUnn Unn[=Iyy] nn[=Iyy]	System utility unit S.SUnn is to be used for the output unit, and, if desired, assigned the intersystem reservation code yy. An intersystem code should be assigned if the programmer wishes to reserve the unit for later use in this job.

Intersystem Unit Option:

Iyy	Output is to appear on the unit that has been assigned the intersystem reservation code yy.

Label Search Option:

d[c]LOU[=Iyy]	A search is to be made for a unit with an expired label and the unit is to be optionally assigned the intersystem reservation code yy. (Label search parameters and intersystem reservation codes are discussed in the section "Input/Output Unit Assignment.")

Variable Unit Option:

[d]cn[=Iyy]	This is the standard variable unit reference indicating that the nth available unit of type d located on symbolic channel c is to be assigned for output. The unit is to be optionally assigned the intersystem reservation code yy. (Variable unit references are discussed in the section "Input/Output Unit Assignment.")

"Any Unit" Options:

T[=Iyy]	The first available tape unit is to be assigned as the output unit and optionally assigned the intersystem reservation code yy.
D[=Iyy]	The first available disk or drum unit is to be assigned as the output unit and optionally assigned the intersystem reservation code yy.
U[=Iyy]	The first available unit (tape, disk, or drum) is to be assigned as the output unit and optionally assigned the intersystem reservation code yy.

Error Detection and Warning Messages

The Update Program checks for error conditions that may appear during program operation. If the error is serious, the job is terminated, the error message is listed on the system output unit (s.sou1), and the following message is typed on the console typewriter: "UPDATE DISCONTINUED DUE TO ERRORS. JOB SKIPPED." If the error is not serious, a warning message is saved until the end of the update run and then is listed on the system output unit (s.sou1) after the run history.

The following messages are possible:

ERROR IN READING TRANSACTION FILE.

The input editor is unable to read from S.SIN. This error causes job termination.

$RUN CARD NOT USED PROPERLY.

The $RUN card must be the first card in the deck that controls the update facilities. This error causes job termination.

INVALID UNIT REFERENCE SPECIFIED ON $XXXXXX CARD.

A parameter on a $ASSIGN or $OUTPUT control card (specified by XXXXXX) is invalid. This error causes job termination.

$LOCATE HAS INVALID ACTION SPECIFIED.

The action parameter is not in the authorized list. This error causes job termination.

$ASSIGN CARD IGNORED. UNIT ALREADY ASSIGNED.

The unit requested for assignment is currently being used. The job is not terminated.

---------- IS GIVEN ON $RUN NOT VALID.

The parameter printed in the dashed space is not allowed. This error causes job termination.

BLOCK FACTOR SET TO MAX BY PROGRAM.

If the blocking factor is left blank or greater than the maximum, the blocking factor is set by the program. This is a warning message; the error does not terminate the job.

BLOCK SEQUENCE ERROR. IGNORED.

IOCS checks the input file for block sequence error. If any are found, the user will be informed by this message; the error does not terminate the job.

CHECKSUM ERROR. IGNORED.

IOCS found an error while reading the input file; the error does not terminate the job.

BLOCK SEQUENCE AND CHECKSUM ERRORS. IGNORED.

IOCS found an error while reading the input file. The error does not terminate the job.

PERMANENT READ REDUNDANCY.

IOCS is not able to get next record from input file. This error causes job termination.

BUFFER OVERFLOW.

An improper blocking factor has been used or an attempt has been made to write a record that is too large. The job is terminated.

REQUEST FOR MORE WORDS THAN IN BUFFER.

This error will occur, most likely, when an input file used for update has no end of file mark and the records that follow are too large to be read. The job is terminated.

SOURCE ORDER ERROR. ---------- AFTER ----------.

If the tape presented for update has a sequence error, the user will be informed of the condition by this message. The dashed lines are filled in with the serial numbers where the error occurred. If the user has used the $NUMBER improperly to resequence the new master and a sequence error results, then the word SOURCE will be replaced by the word UPDATE. The job is not terminated.

RENUMBERING OVERLAP AT ----------. NEW NUMBER USED.

The request for renumbering has come after the desired number has been placed on the new master. This job is not terminated.

NUMBER CARD WITH NO SERIALS. IGNORED.

A request for numbering with no parameters is ignored. The job is not terminated.

WARNING NO DECK CONTROLS GIVEN. ASSUMING ONE DECK ONLY.

This message is given if the user has not used the $LOCATE card to position the input tape; this mode of operation is allowed. That is, the input tape is considered to be one deck, no matter how many decks are actually on the tape. The job is not terminated.

WARNING ADDITIONAL DECK INSERTED AFTER ----------.

This message is given when more than one deck is inserted with one $LOCATE control card. It also appears on a generate run to indicate how many decks are being placed on the new master. The job is not terminated.

END OF MEDIUM ON OUTPUT UNIT.

If the end of reel exit is taken by IOCS during a write operation, this message will appear. The job is terminated.

INVALID OPTION ON $OBTAIN. IGNORED.

A parameter on the $OBTAIN card is not in the authorized list; the job is not terminated.

UPDATE CARDS ARE IMPROPER ON OUTPUT RUN. IGNORED.

Update control cards were placed in the transaction file and have no meaning on an output run; the job is not terminated.

INVALID RECORD DURING OUTPUT. ALL SUCH SKIPPED.

When listing an output tape created by the system, all control characters that cannot be interpreted will cause this message; the job is not terminated.

$DELETE WITH NO SERIAL NUMBER. IGNORED.

No parameters were specified on the $DELETE control card; the job is not terminated.

UNEXPECTED CHANGE IN MODE.

The record read is in a mode different from that indicated by the previous record control character. The job is not terminated.

PERMANENT WRITE ERROR.

IOCS unable to write current record; the job is terminated.

END OF REEL REACHED BEFORE LOCATING DECK ----------.

Request to locate a deck resulted in searching a whole reel without finding the deck. The name of the deck requested was either misspelled, nonexistent on the input, or overlooked when the request was made. The job is terminated.

NOT ENOUGH AVAILABLE UNITS FOR THIS RUN

Units must be in ready status in order to be considered available. This error terminates the job.

Iyy SPECIFIED ON $XXXXXX CARD, WAS ALREADY ASSIGNED.

The user has specified that a unit be assigned the intersystem reservation code yy (by means of the =Iyy option) and this code has already been assigned to another unit.

UNIT S.SUxx, SPECIFIED ON $XXXXXX CARD HAS ALREADY BEEN RESERVED AND CANNOT BE ASSIGNED Iyy.

The user has specified that the intersystem reservation code yy is to be assigned to a unit that already has a different intersystem reservation code assigned to it. The new code is not assigned.

FILE ---------- HAS BEEN OPENED AS UPDATE OUTPUT

This message is printed to inform the user that a file has been opened when no $OUTPUT card has been used to specify a particular unit for the file. The following file names may appear in place of the dashed lines:

OUTPT The main output file
AUXOU The auxiliary output file

The following messages will appear in the summary to inform the user of specific units used as output during an update run:

UPDATE OUTPUT ON XXXX (XXXX=2201=channel B tape unit 1)

INPUT LABEL SEARCH, FILE FOUND ON XXXX
OUTPUT LABEL SEARCH, FILE FOUND ON XXXX

Summary of Records Processed

The Update program will print out a summary of the records processed. Included in the summary will be a count of logical records written on the new master file. Also listed will be a count of deletions and a count of insertions. There will not be a count of extracted records. The summary, therefore, indicates the volume of information on a tape and also serves as a cross check of records processed during successive Update runs. (That is, the number of records read from the Update run master file must be equal to the number of records written on the master file during the previous Update run.)

Planning an Update Run

System Control Cards

System control cards are an important facet of the update run considerations. In planning an update run, the input file must be considered in relation to what system control cards precede and follow the various program decks. The output file, depending on its use, will also require system control cards.

If an auxiliary master file is being produced using the EXTRACT option, system control cards that are not on the input file may be placed on the auxiliary master file by means of the $PLACE control card.

For an update run where a new master file is being created, system control cards may be inserted by the following means:

1. Use a modification card identical to the last card in the deck after which the system control cards are to be placed.
2. Place the system control cards behind this serialized card.
3. During the update run, the serialized modification card will replace the last card of the deck specified, and the system control cards will be processed as unserialized modification cards.

If decks are maintained as a storage pool with no $IBJOB cards, output files produced from them can still be utilized by using $IEDIT to supply the required control cards from S.SIN.

Selection of the Run Type

During a typical Update run, the first parameter on the $RUN control card may be EXTRACT, UPDATE, or blank. If the parameter is left blank, then both an extract file and a new master file will be created. The situations in which the various types of runs are selected are outlined in the following text.

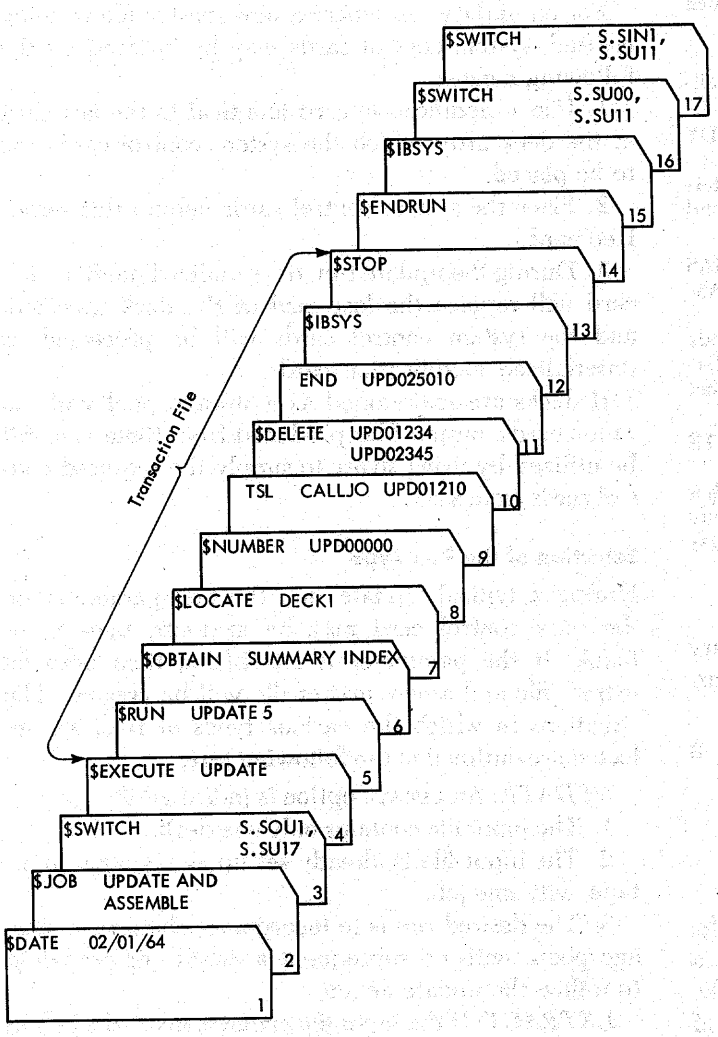
UPDATE: An UPDATE option is indicated if:

1. The input file contains only one deck.
2. The input file is already set up as a system input tape, with one job.
3. The desired run is to incorporate changes to storage pools, with no subsequent assembly or execution to follow the update action.

EXTRACT: If the input file contains many decks and only one is desired for modification and assembly, the programmer should use the EXTRACT option.

BOTH: When a new master file and a system input tape intended for processing are both desired, the "blank" option for the $RUN control card should be used.

EXAMPLES: The following examples, Figures 34 through 39, are intended as an aid for planning jobs using the Update program. They show only some of the more basic functions of the Update program and should not be considered as the limit of the facilities available.

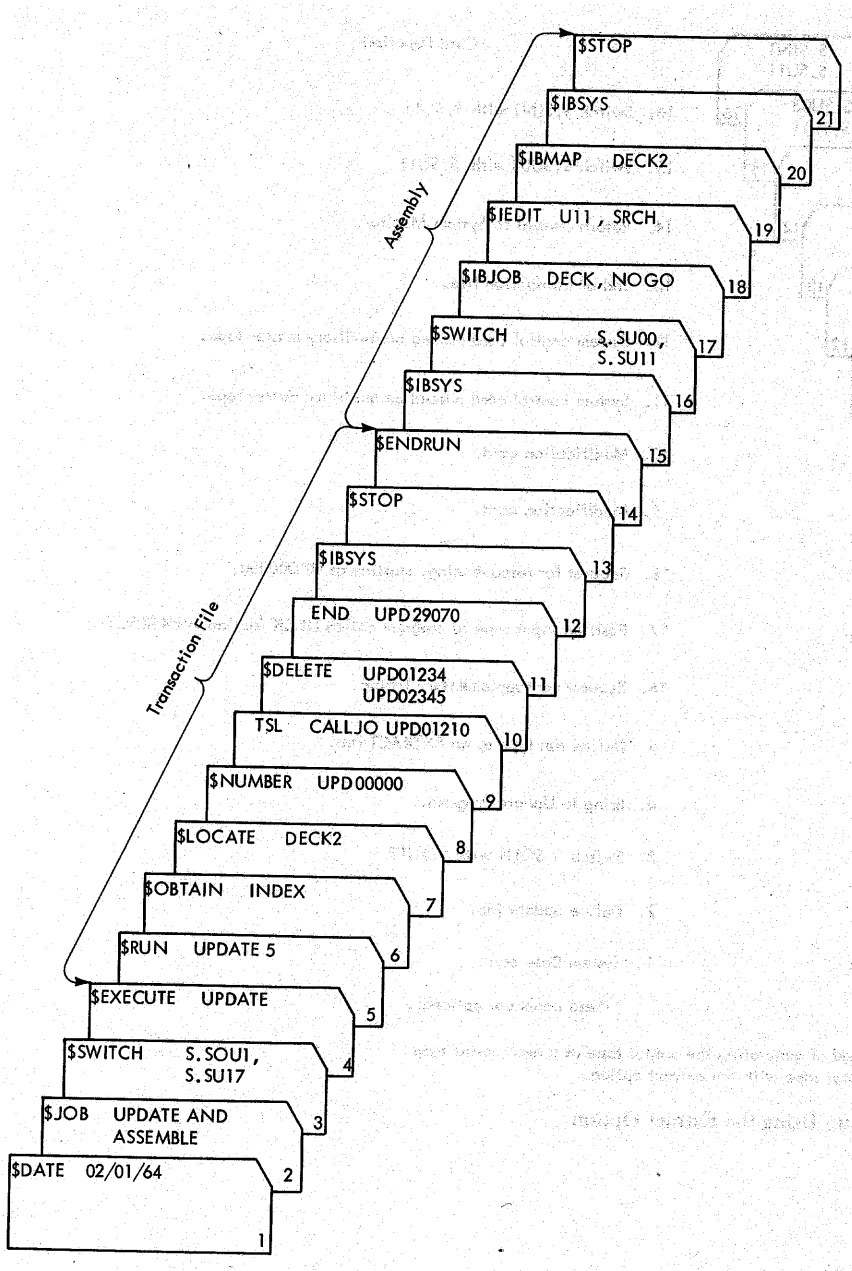


Card Function

17. Switch S.SIN1 with S.SU11.
16. Switch S.SU00 with S.SU11.
15. Return control to System Monitor.
14. End of transaction file.
13. System control card placed on new master file.
12. System control card placed on new master file.
11. Modification card.
10. Delete cards from UPD01234 to UPD02345.
9. Modification card.
8. Request for resequencing, starting at UPD00000.
7. Locate deck to be modified.
6. Request supplementary listings.
5. Specify run type.
4. Bring in Update Program.
3. Switch S.SOU1 with S.SU17.
2. System Job card.
1. System Date card.

This example shows a deck setup used to update a single deck and prepare it for assembly. In this case, the old master file is assumed to have the $IBJOB card preceding the symbolic deck to be updated and assembled. Because processing is to stop after assembly of this deck, the $IBSYS and $STOP control cards are added to it. The illustration of the deck shows the cards used for the update function and two switch cards to assign the system tapes in preparation for the assembly.

Figure 34. Deck Setup for an Update Run



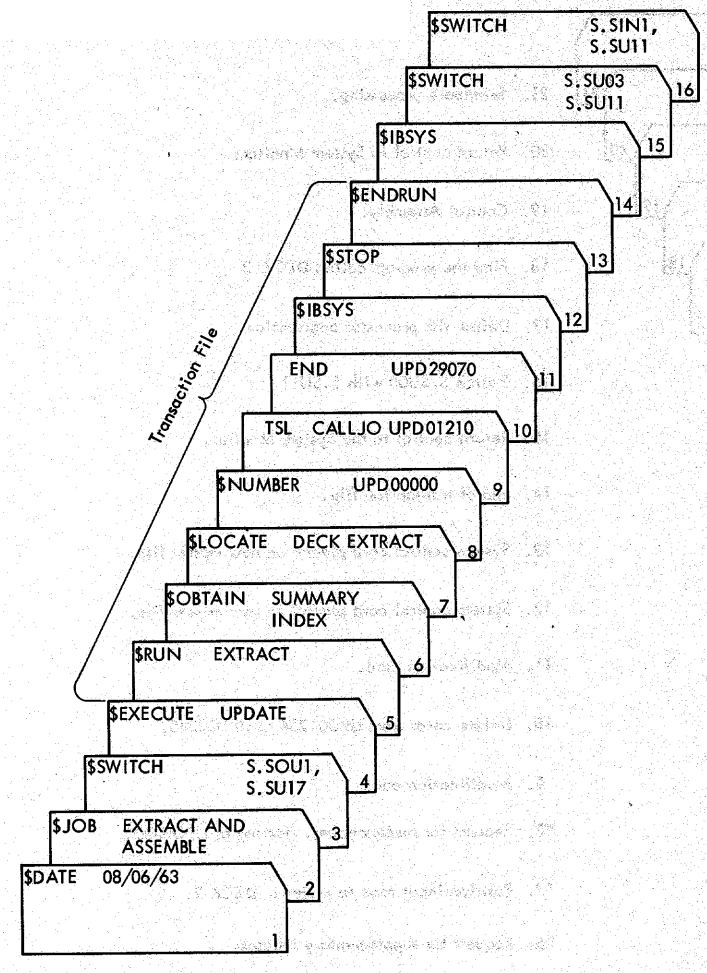
Card Function

21. Terminate processing.
20. Return control to System Monitor.
19. Control Assembly.
18. Find the program called DECK 2.
17. Define the processor application.
16. Switch S.SU00 with S.SU11
15. Return control to the System Monitor.
14. End of transaction file.
13. System control card placed on new master file.
12. System control card placed on new master file.
11. Modification card.
10. Delete cards from UPD01234 to UPD02345.
9. Modification card.
*8. Request for resequencing, starting at UPD00000.
*7. Position input tape to program DECK 2.
*6. Request for supplementary listings.
5. Defines run type (Update).
4. Bring in Update Program.
3. Switch S.SOU1 with S.SU17.
2. Define update job.
1. System Date card.

*These cards are optional.

This example shows the deck setup used to modify a single deck, prepare it for assembly by the insertion of system control cards, and assemble it. In this case, the old master file does not have a $IBJOB control card preceding the program deck, so the $IEDIT control card is used to supply this. The deck illustrated will perform the update and the assembly of the update deck.

Figure 35. Deck Setup for an Update and Assembly Run

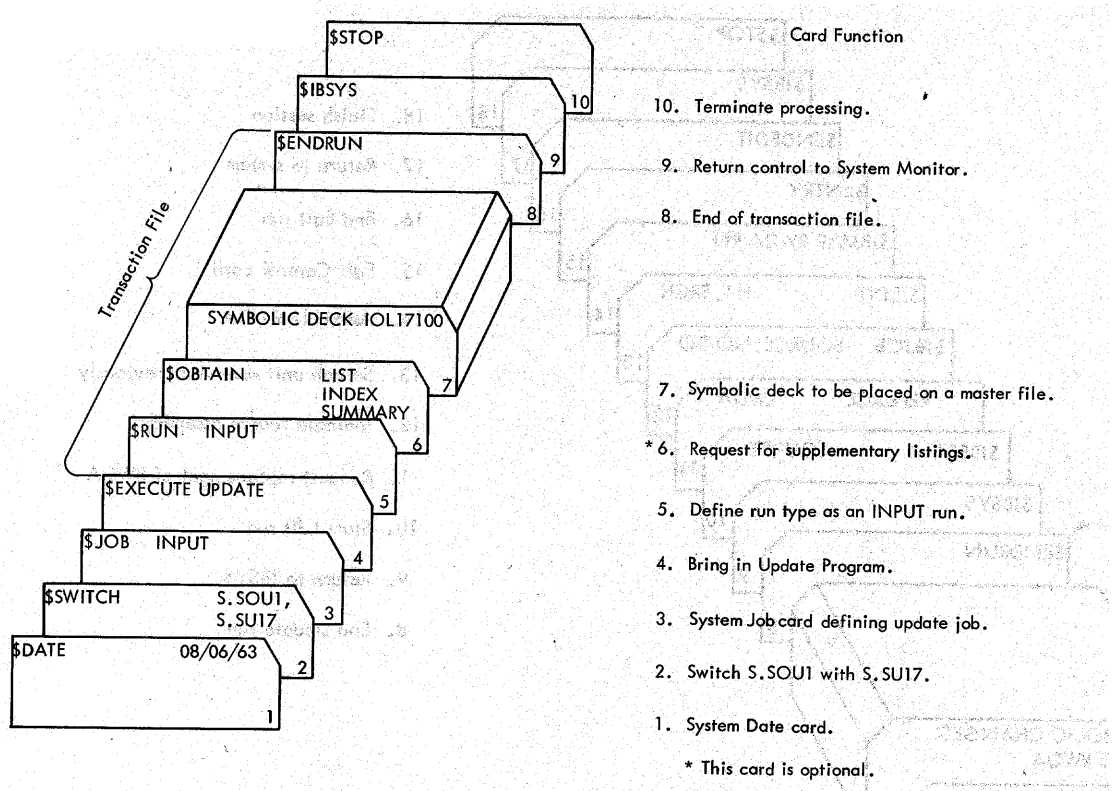


Card Function	
16.	Switch S.SIN1 with S.SU11
15.	Switch S.SU03 with S.SU11
14.	Return control to System Monitor.
13.	End of transaction file.
12.	System control card placed on auxiliary master tape.
11.	System control card placed on auxiliary master tape.
10.	Modification card.
9.	Modification card.
*8.	Request for resequencing, starting at UPD00000.
*7.	Position input tape to program called DECK (action is EXTRACT).
*6.	Request for supplementary listing.
5.	Define run type as an EXTRACT run.
4.	Bring in Update Program.
3.	Switch S.SOU1 with S.SU17.
2.	Define update job.
1.	System Date card.

*These cards are optional.

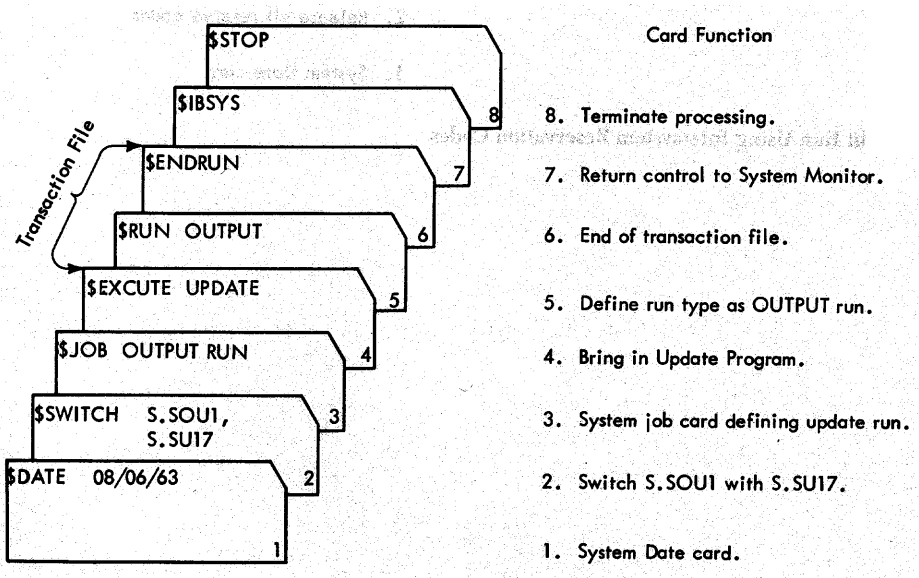
This example is similar to the one in Figure 36, except that instead of generating the output tape as a new master tape with the update option, the output is placed on the auxiliary master tape with the extract option.

Figure 36. Deck Setup for an Update and Assembly Run Using the Extract Option



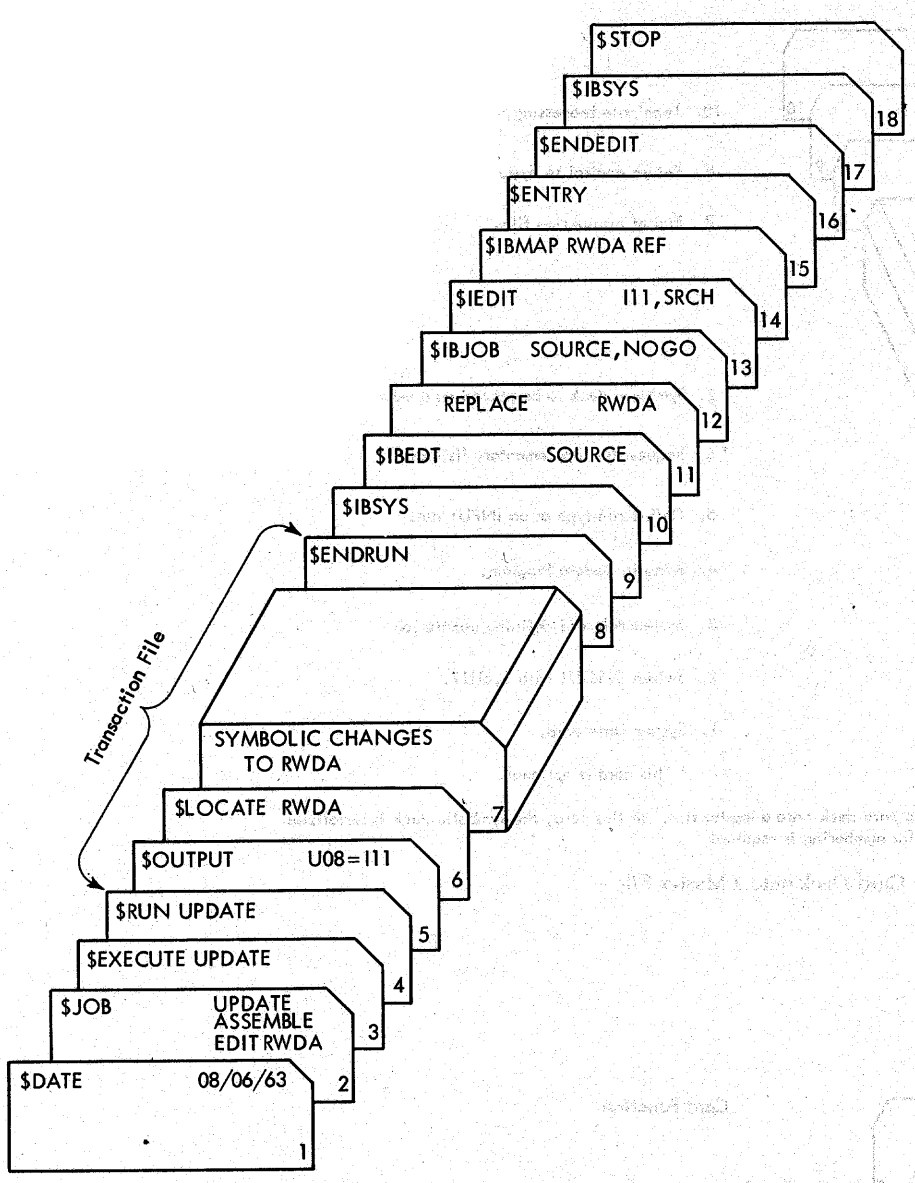
This example illustrates the cards used to put a symbolic card deck onto a master file. In this case, the symbolic deck is serialized (refer to the accompanying listing), so that no request for numbering is required.

Figure 37. Deck Setup for Placing a Symbolic Card Deck onto a Master File



This example shows the deck arrangement for listing an output tape on the system output unit.

Figure 38. Deck Setup for Listing an Output Tape



Card Function

18. Finish session
17. Return to system
16. End Edit run
15. Edit Control card
14. Assemble RWDA
13. Search unit reserved previously
12. Indicate source assembly
11. Request replacement of RWDA
10. Start Edit run
9. Return to IBSYS
8. End Update run
7. Change deck
6. Locate deck for later use
5. Select Output unit and reserve
4. Select Update run
3. Enter Update phase
2. Release all reserve codes
1. System Date card

Figure 39. Deck Setup for Update Followed by Edit Run Using Intersystem Reservation Codes

Monitored Utility Programs in the IBM 7040/7044 Operating System (16/32K)

Introduction

The 7040/7044 Operating System (16/32K) includes a subsystem of monitored utility programs. Five of these perform various utility functions for 1301 Disk Storage, 1302 Disk Storage, or 7320 Drum Storage. The subsystem also provides a program for listing information contained on disk storage, drum storage, magnetic tape, or punched cards. This chapter describes the six monitored utility programs and the methods for using them.

A number of utility programs that are independent of the 7040/7044 Operating System (16/32K) are also available. Information concerning these programs may be found in the publication *IBM 7040/7044 Utility Programs*, Form C28-6317.

The reader should be familiar with the publications *IBM 1301, Models 1 and 2, Disk Storage and IBM 1302, Models 1 and 2, Disk Storage with IBM 7040 and 7044 Data Processing Systems*, Form A22-6768, and *IBM 7320 Drum Storage with 7040 and 7044 Systems*, Form A22-6793.

The Utility Monitor

The 7040/7044 Operating System utility programs operate under the control of a Utility Monitor. The various subroutines common to all of the utility programs are incorporated into the Utility Monitor, saving library positioning and loading time. The Utility Monitor, which is controlled by the System Monitor, is loaded by the System Loader (S.SLDR). The System Loader also loads each utility program requested and passes control to it.

The system units used by the Utility Monitor are:

1. The system library unit (S.SLBX), from which the Utility Monitor and the utility programs are loaded
2. The system input unit (S.SINX), from which the control cards are read by the Utility Monitor
3. The system output unit (S.SOUX), on which the system output is written by the Utility Monitor

The Utility Programs

The following monitored utility programs are included in the 7040/7044 Operating System (16/32K):

1. The Device Print Program
2. The Format Track, Home Address, and Record Address Generator
3. The Load Disk/Drum Program
4. The Dump Disk/Drum Program
5. The Restore Disk/Drum Program
6. The Clear Disk/Drum Program

The arrangement of control cards for an application using the Dump Disk/Drum Program and the Restore Disk/Drum Program is shown in Figure 40. When the $EXECUTE IBUTL card is recognized, the System Loader (S.SLDR) loads the Utility Monitor and passes control to it.

When the Dump Disk/Drum parameter card is recognized by the Utility Monitor, the System Loader loads the Dump Disk/Drum Program from the system library unit (S.SLBX) and transfers control to it.

After the Dump Disk/Drum Program has been executed, control is returned to the Utility Monitor, which recognizes the Restore Disk/Drum parameter card. The System Loader then loads the Restore Disk/Drum Program from the system library unit and passes control to it.

The $IBSYS card returns control to the System Monitor (IBSYS).

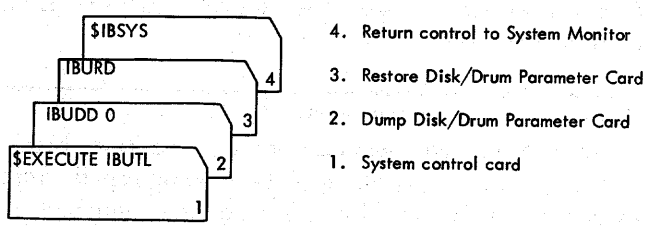


Figure 40. Sequence of Control Cards Used for a Utility Run

Messages to the Operator

Error messages and messages requesting operator intervention are typed whenever necessary. A list of messages associated with the monitored utility programs is contained in the publication *IBM 7040/7044 Operating System (16/32K): Operator's Guide*, Form C28-6338.

Control Cards Used with the Utility Programs

Three types of control cards are used with the utility programs: system control cards, parameter cards, and extension cards.

SYSTEM CONTROL CARDS

System control cards ($EXECUTE and $IBSYS) provide communication between the System Monitor and the Utility Monitor. The $EXECUTE card transfers control to the Utility Monitor. The format of the $EXECUTE card is as follows:

```
1                                16
$EXECUTE                         IBUTL
```

Any $ control card returns control to the System Monitor. The format of the $IBSYS card is as follows:

```
1
$IBSYS
```

PARAMETER CARDS

Parameter cards perform a dual function: they provide communication between the Utility Monitor and the utility programs, and they contain the specifications used by the utility programs to accomplish the utility function. The format of the parameter card is as follows:

```
1          7  8                              78
IBUxx      c  Specifications.....
```

where c = zero or blank.

EXTENSION CARDS

Extension cards provide a means of continuing parameter card specifications. Unless otherwise indicated, any number of extension cards may follow a parameter card. When extension cards are used, the interrupted field in the parameter card *must* end with a comma followed by a blank. The field is continued in column 8 of the extension card. The format of an extension card is as follows:

```
1          7  8                              78
IBUxx      c  Specifications (cont.).....
```

where c = any BCD character other than zero or blank.

The parameter cards can be distinguished from extension cards only by the character punched in column 7. Parameter cards must have either a zero or a blank column 7, while extension cards must have any BCD character other than zero or blank in column 7. A suggested method of distinguishing between the two is to punch a zero in column 7 of the parameter card and to punch numbers (1, 2, 3, . . .) in column 7 of the succeeding extension cards.

The Device Print Program

The Device Print Program produces a listing of the information contained on IBM 1301 Disk Storage, IBM 1302 Disk Storage, IBM 7320 Drum Storage, magnetic tape, or punched cards. Either binary or BCD information may be listed. The listing, which is produced on the system output unit, may be in any one of five formats. Figure 41 gives an example of each of the available formats.

Any file or group of files on the storage device may be selected for listing. Within a file, any record or group of records may be selected. Input files may be labeled. Unit switching is not performed. Labels are not checked by the program to ensure that the file is actually labeled. If labels are specified, files are printed or skipped in multiples of three. Also, if checkpoint records are included (each constituting a separate file), an adjustment is not made by the program, and skipping or printing of files still proceeds by multiples of three of the number of data files specified.

The Device Print Program has a maximum buffer length of $(4096)_{10}$ words. This maximum can be altered by changing the assembly parameter SIZE and reassembling the Device Print Program.

USING THE DEVICE PRINT PROGRAM

An IBUUD parameter card is used to transfer control from the Utility Monitor to the Device Print Program. This control card specifies the system unit from which the data is to be obtained, the input mode, the format of the output, and the location of the information to be written on the output unit.

CONTROL CARD FORMAT

The format of the parameter card used by the Device Print Program is (extension cards may be used when s.snnn is attached as a nonsequential device):

CARD COLUMNS	CODING	DESCRIPTION
1-5	IBUUD	Control card identification.
6		Not used.
7		Blank or zero.
8-13	S.Snnn	System unit containing data records to be printed.
14	/	Field separator.
15	0 or 1	Input mode: 0 for binary information; 1 for BCD information.
16	1, 2, 3, 4, or 5	Output format. 1. Octal—8 words per line. 2. BCD—16 words per line. 3. Octal with mnemonics—8 words per line. 4. Octal with BCD—8 words per line. 5. Octal with BCD and mnemonics—8 words per line.
17	/	Field separator.
18	0 or 1	Label: 0 for nonlabeled input files; 1 for labeled input files.
19	/	Field separator.
Either		
20-71	A, B/C, D for sequential input	A is the starting record (first, second, etc.) of starting file B. C and D are the terminating record and file, respectively. A maximum of four digits each is allowed for A and C. (See Figure 42.) Only one such

```
00000  300031000105  076000000004  -060000000104  -060000000077  076200000000  -054000000112  006100000106  050000000077
00010  060100000101  002000000101  300005000101   000000000000   000000000000  000000000000   000000000000  000000000000
```

Format 1 Octal--Eight Words Per Line

```
00000  H0I0157 0004000014000000*7S0000N-00100/001650000*6100110+0011H0501100000000000000000000000000000000
```

Format 2 BCD--Sixteen Words Per Line

```
00000  300031000105  076000000004  -060000000104  -060000000077  076200000000  -054000000112  006100000106  050000000077
       TXH           ENK           STQ            STQ            RDS           RCHB           TCOB          CLA
00010  060100000101  002000000101  300005000101   000000000000   000000000000  000000000000   000000000000  000000000000
       STO           TRA           TXH
```

Format 3 Octal With Mnemonics--Eight Words Per Line

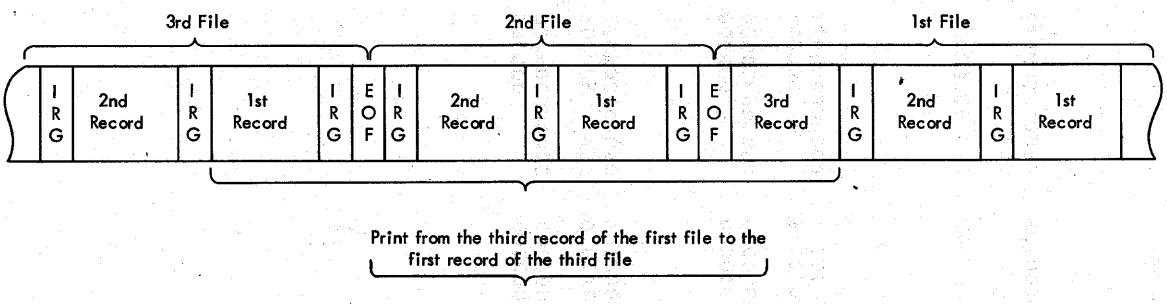
```
00000  300031000105  076000000004  -060000000104  -060000000077  076200000000  -054000000112  006100000106  050000000077
          H0I015        7 0004        000014         00000*         7S0000         N-0010         0/0016        50000*
00010  060100000101  002000000101  300005000101   000000000000   000000000000  000000000000   000000000000  000000000000
          610011        0+0011        H05011         000000         000000        000000         000000        000000
```

Format 4 Octal With BCD--Eight Words Per Line

```
00000  300031000105  076000000004  -060000000104  -060000000077  076200000000  -054000000112  006100000106  050000000077
       TXH  H0I015   ENK  7 0004   STQ  000014    STQ  00000*    RDS  7S0000   RCHB  N-0010   TCOB  0/0016  CLA  50000*
00010  060100000101  002000000101  300005000101   000000000000   000000000000  000000000000   000000000000  000000000000
       STO  610011   TRA  0+0011   TXH  H05011         000000         000000        000000         000000        000000
```

Format 5 Octal With BCD And Mnemonics--Eight Words Per Line

Figure 41. Device Print Program Output Formats




```
1        8
IBUUD    S.Sxxx/02/0/3, 1/1, 3
```

Figure 42. Sequential Device Print Program Input Parameters

CARD COLUMNS	CODING	DESCRIPTION
		specification is allowed for each IBUUD control card. Each additional specification must be placed on a separate IBUUD control card. If A, B=0, 0 the device will not be rewound and the number of records and files specified in C, D will be printed. If B/C, D=0/0, 0 the device will be backspaced the number of records specified by A and this number of records will be printed unless an End-of-File is encountered.
72-80		Not used.
Or		
20-71	ttttc for disk or drum input ttttc tttc ttc tc	Track address(es). Each address is followed by a connector that indicates the relationship of the track preceding the connector to the track following it. (This is shown in Figure 43.) A maximum of 50 addresses is allowed for each IBUUD control card and all of its extension cards. t-tttt is a track address (0-399 for drum; 0-9999 for disk). c can be: 1. A comma to indicate individual tracks; 2. A hyphen to indicate sequential tracks; 3. A blank to indicate the last track to be printed.
72		Blank.
73-78		Not used.

```
1        8
IBUUD    S.Snnn/12/0/32, 45, 230-243
```

Write the BCD information from S.Snnn, as follows: Output format 2; the file is unlabeled; write from tracks 32 and 45, and 230 through 243.

Figure 43. Nonsequential Device Print Program Input Parameters

The Format Track, Home Address, and Record Address Generator

This program can be used to write format tracks and/or home address identifiers and record addresses for 1301 Disk Storage, 1302 Disk Storage or 7320 Drum Storage.

The Format Track Generator generates, in core storage, the fields of characters used to write a format track and writes them on the format tracks for the cylinders specified in the Format Track parameter card.

The Home Address and Record Address Generator generates and writes home address identifiers and record addresses on one or more data tracks of disk storage or drum storage. When home address identifiers and record addresses are to be written on more than one data track, the data tracks must have identical format tracks. Record areas are filled with a character that is specified on the parameter card.

The maximum number of words available for recording information on the track depends on the number of record areas that can be defined for one data track and on the number of words per record area.

A record area to be formatted must be one word larger than the input record if the device is attached for single record operation.

The formulas for computing the maximum number of words available for recording data on the tracks are given below. The length of the home address identifier and the length of the record address are assumed to be of standard length.

Where:

M=the maximum number of words available for recording information on the track, and

N=the number of record areas per track.

Then, for the 6-bit mode, the maximum number of words available for recording data on the track is computed as follows:

1301 Disk: M = 465 − [19(N-1)/3]
1302 Disk: M = 974 − [25(N-1)/3]
7320 Drum: M = 530 − [19(N-1)/3]

Any fractional part of M should be disregarded.

USING THE FORMAT TRACK, HOME ADDRESS, AND RECORD ADDRESS GENERATOR

Three types of parameter cards are used with the Format Track, Home Address, and Record Address Generator. (The functions of these cards are summarized in Figure 44.) The parameter cards are processed by sets. A set is defined as all cards having the same set identification (columns 73-78 of the control cards). An error encountered in any card in a set causes all the following cards in the set to be ignored.

Writing Format Tracks: An IBUxF parameter card is used for writing format tracks. The following information is supplied to the program through this control card:

1. Control card identification (IBUxF), where x =
   F for 1301 Disk Storage
   H for 1302 Disk Storage
   G for 7320 Drum Storage
2. Symbolic unit designation (S.Sxxx) of the disk or drum storage unit
3. Numbers (0-9 for drum; 0-249 for disk) of the cylinders to be given formats (although there is only one format track for drum storage, cylinders 0-9 may be specified, thus providing some compatability with disk storage)
4. Number of record areas to be placed on the format track
5. Number of words in each record area

CONTROL CARD(S)	FUNCTION — WRITE:
IBUxF	Format Track Only
IBUxH (first card) IBUxF (second card) or IBUxH (first card) IBUxB (second card)	Home Address Identifiers and Record Addresses
IBUxB (first card) IBUxH (second card)	Format Track, Home Address Identifiers, and Record Addresses

● Figure 44. Use of Parameter Cards with Format Track, Home Address, and Record Address Generator

Writing Home Address Identifiers and Record Addresses: An IBUxH parameter card is used to write home address identifiers and record addresses. Before these can be written, however, the format of the tracks on which they are to be written must be supplied to the program. Two types of parameter cards may be used to identify the formats: the IBUxF control card, normally when writing format tracks, can be placed behind the IBUxH control card to supply the necessary information, or an IBUxB control card, which is similar to the IBUxF control card, may be placed behind the IBUxH control card.

The following information is supplied to the program through the IBUxH parameter card:

1. Control card identification (IBUxH), where x =
   F for 1301 Disk Storage
   H for 1302 Disk Storage
   G for 7320 Drum Storage
2. The addresses of the tracks on which the home address identifiers and the record addresses are to be written
3. The character to be used when filling the data records

Writing Format Tracks, Home Address Identifiers, and Record Addresses: To generate format tracks, home address identifiers, and record addresses, an IBUxB control card is used. This card performs essentially the same function as the control card used to write format tracks (IBUxF). It serves the added purpose of indicating to the program that all functions are to be performed (that is, the writing of format tracks, home address identifiers, and record addresses). This card is followed by an IBUxH control card containing the specifications for the home address identifiers and record addresses.

CONTROL CARD FORMATS

IBUxF Control Card: To generate a format track only, the IBUxF control card is used. The format of the IBUxF control card is as follows:

CARD COLUMNS	CONTENTS	DESCRIPTION
1-5	IBUFF (1301 Disk) IBUHF (1302 Disk) IBUGF (7320 Drum)	Control card identification.
6	W	Write checking will be performed if W is present.
7		Blank or zero.
8-13	S.Sxxx	Symbolic unit designation of the disk or drum storage unit that is to be given a format.
14	/	Field separator.
15-71	0-249 for disk 0-9 for drum	Numbers of the cylinders to be given formats. Each cylinder number is separated either by a comma (to indicate separate cylinders) or by a dash (to indicate a sequential series of cylinders). A field separator (/) follows the last cylinder number.
	01-63	A two-digit number, which designates the number of record areas to be placed in the format track, follows the field

CARD COLUMNS	CONTENTS	DESCRIPTION
		separator (/). A field separator also follows the two-digit number. This number will be used regardless of the number of records defined on the $ATTACH card.
	1-465 (1301 Disk) 1-974 (1302 Disk) 1-530 (7320 Drum)	A series of numbers separated by commas follow the last field separator. The numbers indicate the word length of each of the record areas. If the lengths of all the record areas are the same, only one number need be used.
72		Blank.
73-78	aaaaaa	Set identification. A six-character alphameric name assigned by the user to this set of control cards.
79-80		Not used.

The following is an example of the IBUFF control card:

```
1       8                                                    73
IBUFF  S.Sxxx/1-23, 26, 59/06/3, 6, 23, 99, 36, 25  SETONE
```

This control card is used to write a format track on the 1301 Disk Storage assigned as S.Sxxx. Cylinders 1-23, 26, and 59 are to be allocated six record areas each, with word lengths of 3, 6, 23, 99, 36, and 25 words. SETONE is the set identification.

IBUxH Control Card: To generate home address identifiers and record addresses only, the IBUxH control card is used, followed by either the IBUxF or the IBUxB control card. The format of the IBUxH control card is as follows:

CARD COLUMNS	CONTENTS	DESCRIPTION
1-5	IBUFH (1301 Disk) IBUHH (1302 Disk) IBUGH (7320 Drum)	Control card identification.
6	W	Write checking will be performed if W is present.
7		Blank or zero.
8	C	Filler character (may be any BCD character).
9	/	Field separator.
10-71	ttttc tttc ttc tc	Track address(es). Each address is followed by a connector, which indicates the relationship of the track address preceding the connector to the address following it. A maximum of 50 addresses is allowed for each IBUxH control card and all of its extension cards. t-tttt is a track address (0-399 for drum; 0-9999 for disk). c can be: 1. A comma to indicate individual tracks; 2. A hyphen to indicate sequential tracks; 3. A blank to terminate the field.
72		Blank.
73-78	aaaaaa	Set identification. A six-character alphameric name assigned by the user to the set of control cards.
79-80		Not used.

The cards in the following example may be used to write home address identifiers and record addresses.

```
1       8                                                    73
IBUFH  R/40-959, 1040-1079, 2360-2399                SETONE
1       8                                                    73
IBUFF  S.Sxxx/1-23, 26, 59/06/3, 6, 23, 99, 36, 25  SETONE
```

The control card IBUFH, followed by an IBUFF control card, is used to write home address identifiers and record addresses on the 1301 Disk Storage assigned as s.sxxx. Tracks 40 through 959, 1040 through 1079, and 2360 through 2399 (whose cylinders were previously given a format), with six data areas of 3, 6, 23, 99, 36, and 25 words, are to be given home address identifiers and record addresses. Data areas will be filled with the BCD character R.

IBUxB Control Card: To generate a format track, home address identifiers and record addresses, the IBUxB control card is used followed by the IBUxH control card. The format of the IBUxB control card is as follows:

CARD COLUMNS	CONTENTS	DESCRIPTION
1-5	IBUFB (1301 Disk) IBUHB (1302 Disk) IBUGB (7320 Drum)	Control card identification. Indicates that format tracks, home address identifiers, and record addresses are desired.
6-80	(Same parameters as for the IBUxF control card.)	

The cards in the following example may be used to write format tracks, home address identifiers, and record addresses.

```
1       8                                                    73
IBUFB  S.Sxxx/1-23, 26, 59/06/3, 6, 23, 99, 36, 25  SETTWO
1       8                                                    73
IBUFH  J/40-959,1040-1079, 2360-2399                 SETTWO
```

The IBUFB control card followed by an IBUFH control card, is used to write format tracks, home address identifiers, and record addresses on the 1301 Disk Storage assigned as S.Sxxx. Cylinders 1-23, 26, and 59 are to be allocated six data areas with word lengths of 3, 6, 23, 99, 36 and 25. Data areas will be filled with the BCD character J. Home address identifiers and record addresses will be written on tracks 40 through 959, 1040 through 1079, and 2360 through 2399. SETTWO is the set identification.

The Load Disk/Drum Program

The Load Disk/Drum Program transmits data obtained from disk or drum storage tracks, magnetic tape records, or cards from a card read punch to one or more consecutive record areas on specified data tracks of 1301 Disk Storage, 1302 Disk Storage, or 7320 Drum Storage. Any input record that exceeds the capacity of a record area is truncated; the excess characters or bits are not loaded. If a record area is greater than the input record, the unused positions of the record area are filled with blanks.

Neither truncating input records nor padding record areas is considered an error condition. The loading process is terminated when either all the designated track numbers are exhausted or all the specified input files have been loaded.

The Load Disk/Drum Program has a maximum buffer length of $(1000)_{10}$ words. This maximum can be altered by changing the assembly parameter TRKSZ1 and reassembling the Load Disk/Drum Program.

USING THE LOAD DISK/DRUM PROGRAM

Prior to loading information onto the disk or drum storage, the format track, home address identifiers, and record addresses must have been written. Depending upon the control card used, the following information must be provided to the Load Disk/Drum Program:

1. Control card identification (IBULD or IBULS)
2. Symbolic unit designation (s.snnn) of the disk or drum storage unit onto which the data is to be loaded.
3. Symbolic unit designation (s.smmm) of the unit containing the data to be loaded
4. Mode of the input data (BCD or binary)
5. Quantity of records to be loaded
6. Number of the track record at which loading is to begin
7. Tracks to be loaded
8. Number of system input unit files to be skipped before loading
9. Number of input files to be loaded

CONTROL CARDS

Two parameter cards may be used with the Load Disk/Drum Program, the IBULD control card and the IBULS control card. Either may be used if the device is attached for single-record operation or full-track mode. The IBULD control card must be used if the device is attached for random-access operations. The IBULS control card must be used if the device is attached for cylinder mode.

IBULD Control Card: This control card is to be used if the disk or drum storage is attached for random access, full-track mode, or single-record operation. The disk or drum storage must be attached for random-access operation when it is loaded if it is to be used as a random-access device. It must be attached for either full-track or single-record operation when it is loaded if it is to be used as a sequential-access device.

If the device was attached for this method of operation:	The IBULD card will load the device for use as a:
Random access	Random-access device
Full-track mode or Single-record operation	Sequential-access device

Thus, the method of operation specified at the time the device is attached determines whether the IBULD card will write on the disk or drum for random or sequential access. The format of the IBULD control card is as follows:

CARD COLUMNS	CONTENTS	DESCRIPTION
1-5	IBULD	Control card identification.
6	W	Write checking will be performed if W is present.
7		Blank or zero.
8-13	S.Snnn	Disk or drum to be loaded.
14	/	Field separator.
15-16	01-63	Number of track records per track to be loaded.
17	/	Field separator.
18-19	01-63	Track record at which loading is to start.
20	/	Field separator.
21-26	S.Smmm	Input device containing the data to be loaded.
27	/	Field separator.
28	0 or 1	0 indicates binary input; 1 indicates BCD input.
29	/	Field separator.
30-31	00-xx	Files to skip before loading data.
32	/	Field separator.
33-34	01-xx	Files to load.
35	/	Field separator.
36-71	ttttc tttc ttc tc	Track address(es): each address is followed by a connector that indicates the relationship of the track address preceding the connector to the address following it. A maximum of 50 addresses is allowed for each IBULD control card and all of its Extension cards. t-tttt is a track address (0-399 for drum; 0-9999 for disk). c can be: 1. A comma to indicate individual tracks; 2. A hyphen to indicate sequential tracks; 3. A blank to terminate the field.
72		Blank.
73-80		Not used.

The following is an example of the IBULD control card.

```
1       8
IBULD   S.Snnn/02/03/S.Smmm/0/00/01/2, 6, 7
```

This IBULD control card is used to load two records on each specified track starting at record number 3 of tracks 2, 6, and 7 of the 1301 Disk Storage s.snnn. Binary data is to be loaded from a magnetic tape unit designated as s.smmm, and no files are to be skipped before loading. Loading will be terminated when either the six record areas are loaded or an end of file is recognized on the input device.

IBULS Control Card: This card may be used for full track, single record, or cylinder mode. Unlike the IBULD control card, this card will in all cases write information on the disk or drum for sequential access.

The format track to be written for the disk or drum storage units is determined by the mode of operation that is used. For operation in single record mode, the format must provide a record area at least one word greater than the size of the input record. For operation in cylinder mode, the format must provide a record area of 465 words for 1301 Disk Storage, 974 words for 1302 Disk Storage, or 530 words for 7320 Drum Storage.

If the device was attached for this method of operation:	The IBULS card will load the device for use as a:
Full-track mode or Single-record operation or Cylinder mode	Sequential-access device

The format of the IBULS control card is as follows:

CARD COLUMNS	CONTENTS	DESCRIPTION
1-5	IBULS	Control card identification.
6		Blank indicates that an End-of-File will not be written. Any non-blank character indciates that an End-of-File will be written.
7		Zero or blank.
8-13	S.Snnn	Disk or drum to be loaded.
14	/	Field separator.
15-20	S.Smmm	Input device containing the data to be loaded.
21	/	Field separator.
22	0 or 1	0 indicates binary input; 1 indicates BCD input.
23	/	Field separator.
24-25	00-xx	Files to skip before loading data.
26	/	Field separator.
27-28	01-xx	Files to load.
29	/	Field separator.
30-71	iiiiic iiiic iiic iic ic	Disk or Drum Record numbers: each record number is followed by a connector that indicates the relationship of the record number preceding the connector to the record number following it. A maximum of 50 record numbers is allowed for each IBULS control card and all of its Extension cards. Consecutive records beginning with record numbers one must be specified if the disk or drum has been attached for cylinder mode. (i-iiii=1-32,767) c can be: 1. A comma to indicate individual record numbers; 2. A hyphen to indicate sequential record numbers; 3. A blank to terminate the field.
72		Blank
73-80		Not used.

Label Control Card for the Load Disk/Drum Program: Each control card that is to deal with labeled input must be preceded by a LABEL control card. If more than one LABEL control card precedes a control card, only the last one is used. The format of the LABEL control card is as follows:

CARD COLUMNS	CONTENTS	DESCRIPTION
1-5	IBULD or IBULS	Program identification.
6		Not used.
7		Blank.
8-12	LABEL	Label card identification.
13-15		Not used.
16	a	Is 1 if there is a checkpoint; Is 0 if no checkpoint.
17	b	Is 1 if multi-reel file; Is 0 if single-reel file.
18	c	Is 1 if block sequence numbers are to be checked; Is 0 if block sequence numbers are not to be checked.
19	d	Is 1 if a check sum is checked for each block read; Is 0 if no check sums are to be checked.
20	,	Field separator.
21-25	fffff	File serial number.
26	,	Field separator.
27-30	rrrr	Reel sequence number.
31	,	Field separator.
32-36	yyddd	Creation date, where yy is tens and units digits of the year and ddd is the number of the day in the year.
37	,	Field separator.
38-47	iiiiiiiii	Field identification.
48-80		Not used.

The Dump Disk/Drum Program

The Dump Disk/Drum Program writes all the information contained on one or more specified data tracks of 1301 Disk Storage, 1302 Disk Storage, or 7320 Drum Storage onto magnetic tape, disk storage, or drum storage. The operation is performed by a Read-Full-Track-with-Address instruction. Therefore, record addresses are included as part of the information to be recorded.

The program is intended for use with the Restore Disk/Drum Program, which restores to the disk or

drum storage all or selected portions of the information that has been recorded. (The format track for the affected area must be the same when the restore operation is performed as it was when the dump took place.)

USING THE DUMP DISK/DRUM PROGRAM

The following information should be included in the control card:

1. Program identification (IBUDD)
2. The designation for the disk or drum storage to be dumped
3. The system designation of the device to be written on
4. The tracks to be dumped
5. Identification for the block of information being dumped to be used during the restore operation

CONTROL CARD FORMAT

IBUDD Control Card: The format of the IBUDD control card is as follows:

CARD COLUMNS	CONTENTS	DESCRIPTION
1-5	IBUDD	Control card identification.
6		Not used.
7		Zero or blank.
8-13	S.Snnn	Disk or drum to be dumped.
14	/	Field separator.
15-20	S.Smmm	Primary output device for dumped tracks.
21	/	Field separator.
either		
22-28	S.Sxxx/	Alternate output device. If no alter-
29-71	(see below)	nate unit is specified, the primary unit is assigned.
or		
22-71	ttttc tttc ttc tc	Track address(es): each address is followed by a connector that indicates the relationship of the track address preceding the connector to the address following it. A maximum of 50 addresses is allowed for each IBUDD control card and all its extension cards. t-tttt is a track address (0-399 for drum; 0-9999 for disk) c can be: 1. A comma to indicate individual tracks. 2. A hyphen to indicate sequential tracks. 3. A blank to terminate the field.
72		Blank.
73-78	aaaaaa	Identification for use by the Restore Disk/Drum Program, where "a" is any BCD character except a comma, blank, or slash.
79-80		Not used.

When the same output device is specified on successive control cards, the program will not close the device after every control card. Instead, it will wait until there is a change in the control card field or until a non-dump control card is encountered. When there is a change in the output device and a non-dump control card is encountered, the current device will be closed with an end-of-data procedure and will be rewound. The following is an example of the IBUDD control card:

```
1       8                                   73
IBUDD   S.Snnn/S.Smmm/2, 5, 23              SETONE
```

The IBUDD control card is used to dump a specified disk storage area. In this example, tracks 2, 5, and 23 of the 1301 Disk Storage (s.snnn) are to be dumped onto a magnetic tape (s.smmm). SETONE is the identification.

Label Control Card for the Dump Disk/Drum Program: A LABEL control card should precede each control card or group of control cards when header and trailer labels are to be placed on the output device. The presence of a LABEL control card will result in closing the previous output device, if any. If more than one LABEL control card precedes a control card, only the last one is used. The format of the LABEL control card is as follows:

CARD COLUMNS	CONTENTS	DESCRIPTION
1-5	IBUDD	Program identification.
6		Not used.
7		Blank.
8-12	LABEL	Label card identification.
13-15		Not used.
16	a	Is 1 if multi-reel output; Is 0 if single-reel output.
17	,	Field separator.
18-22	fffff	File serial number.
23	,	Field separator.
24-27	rrrr	Reel sequence number. (rrrr should be greater than 1 to insure use of the file serial number.)
28	,	Field separator.
29-32	dddd	Retention period.
33	,	Field separator.
34-43	iiiiiiiii	File identification.
44-80		Not used.

The Restore Disk/Drum Program

The Restore Disk/Drum Program restores to 1301 Disk Storage, 1302 Disk Storage, or 7320 Drum Storage all or selected portions of the data previously recorded by the Dump Disk/Drum Program. It cannot be used to load any other information. New data can be placed onto disk or drum storage by using the Load Disk/Drum Program.

The information previously dumped is restored using the Write-Full-Track-with-Address operation. Therefore, only the home address is required for verification. If labels were specified when the Dump Disk/Drum Program was used, the input file is labeled. A LABEL control card must be used to affect proper processing of the labels during the restore operation. If the input to the program is a multi-reel file (unlabeled), an alternate unit must be specified. The

home-address identifiers and the format track must be the same as when the disk or drum storage was dumped.

USING THE RESTORE DISK/DRUM PROGRAM

The control card should contain the following information:

1. Program identification (IBURD)
2. Symbolic unit designation (s.snnn) of the disk or drum storage unit on which the data is to be restored
3. Symbolic unit designation (s.smmm) of the unit containing the data to be restored
4. Identification of data blocks to be restored

CONTROL CARD FORMATS

IBURD Control Card: The format of the IBURD control card is as follows:

CARD COLUMNS	CONTENTS	DESCRIPTION
1-5	IBURD	Control card identification.
6	W	Write checking of the data will be performed if W is present.
7		Zero or blank.
8-13	S.Snnn	Disk or drum storage to be restored.
14	/	Field separator.
15-20	S.Smmm	Primary input unit.
21	/	Field separator.
either 22-70	aaaaaa	Identification used by the Dump Disk/Drum control card(s). If more than one identification is used, they are separated by commas.
71		Blank.
72-80		Not used
or 22-27	S.Sttt	Alternate input device. If no alternate unit is specified, the primary unit is assumed.
28	/	Field separator.
29-71		(See card columns 22-70.)
72-80		Not used.

The following is an example of the IBURD control card:

1	8
IBURDW	S.Snnn/S.Smmm/aaaaaa, bbbbbb, cccccc

This IBURD control card is used to restore aaaaaa, bbbbbb, and cccccc to disk storage (s.snnn) from s.smmm.

Label Control Card for the Restore Disk/Drum Program: Each control card that is to deal with labeled input must be preceded by a LABEL control card. If more than one LABEL control card precedes a control card, only the last one is used. The format of the LABEL control card is as follows:

CARD COLUMNS	CONTENTS	DESCRIPTION
1-5	IBURD	Program identification.
6	,	Not used.
7		Blank.
8-12	LABEL	Label card identification.
13-15		Not used.
16	a	Is 0 if the file is single-reel; Is 1 if the file is multi-reel.
17	,	Field separator.
18-22	bbbbb	File serial number.
23	,	Field separator.
24-27	cccc	Reel sequence number.
28	,	Field separator.
29-33	ddddd	File creation date.
34	,	Field separator.
35-44	eeeeeeeeee	File identification.
45-80		Not used.

The Clear Disk/Drum Program

The Clear Disk/Drum Program clears record areas of designated data tracks of 1301 Disk Storage, 1302 Disk Storage, or 7320 Drum Storage. The record areas are filled with any specified BCD character. Home-address identifiers and record addresses are not disturbed. Two methods of clearing are provided:

1. One or more consecutive record areas on specified data tracks are filled with a designated BCD character. The number of record areas per track to be cleared and the number of the record at which the clearing is to begin must be specified. If disk/drum is attached in full-track mode, this method cannot be used.
2. All record areas on one or more specified tracks are filled with a designated BCD character. With this method of clearing, only the tracks to be cleared need be specified.

USING THE CLEAR DISK PROGRAM

The following information must be included on the control card:

1. Program identification (IBUCD)
2. System designation for unit being cleared
3. The track, nonsequential tracks, or series of tracks to be cleared
4. The BCD character with which the cleared areas are to be filled
5. If applicable, starting record number and number of record areas to be cleared

CONTROL CARD FORMATS

IBUCD Control Card: The format of the IBUCD control card is as follows:

Method 1:

CARD COLUMNS	CONTENTS	DESCRIPTION
1-5	IBUCD	Control card identification.
6	W	Write checking of data will be performed if W is present.

CARD COLUMNS	CONTENTS	DESCRIPTION
7		Zero or blank.
8-13	S.Snnn	Disk or drum to be cleared.
14	/	Field separator.
15	c	Filler character (BCD).
16	/	Field separator.
17	1	Method of clearing (Method 1).
18	/	Field separator.
19-20	01-xx	Number of track records per track to be cleared.
21	/	Field separator.
22-23	01-xx	Track record at which clearing is to start.
24	/	Field separator.
25-71	ttttc tttc ttc tc	Track address(es): each address is followed by a connector that indicates the relationship of the track address preceding the connector to the address following it. A maximum of 50 addresses is allowed for each IBUCD control card and all of its extension cards. t-tttt is a track address (0-399 for drum; 0-9999 for disk). c can be: 1. A comma to indicate individual tracks; 2. A hyphen to indicate sequential tracks; 3. A blank to terminate the field.
72		Blank.
73-80		Not used.

Method 2:

CARD COLUMNS	CONTENTS	DESCRIPTION
1-16		(Same as those for Method 1.)
17	2	Method of clearing (Method 2).
18	/	Field separator.
19-80		(See columns 25-80 in Method 1.)

The following is an example of the IBUCD control card (Method 1):

```
1       8
IBUCD   S.Snnn/B/1/03/04/26, 45, 230, 231
```

This IBUCD control card is used to clear three records starting at record 4 on tracks 26, 45, 230, and 231 of the 1301 Disk Storage (s.snnn).

Appendix A. Control Card Format Index

Refer to the given page reference for a complete description of each card.

System Monitor — Processor Control Cards

CARD FORMAT		PAGE REFERENCE
1 3		
$*	any text	OG-11
1 16		
$ATTACH	S. Sxxx, device, channel, number, type [,{n[, from, to] / dir}] [,{I57 / I58}]	OG-12
1 8		
$CBEND	deckname	PG-45
1 8 16		
$CHAIN	main name [{Iyy / Uxx[=Iyy] / U[=Iyy] / T[=Iyy] / D[=Iyy]}]	PG-57
1 16		
$CHANNEL	channel(f_1,f_2, . . . ,f_n)channel(f_1,f_2, . . . ,f_n). . .	PG-17 OG-15
1 16		
$CLOSE	{S.SPP1 / S.SUxx / IyyR} [, MARK] [,{REMOVE / REWIND}]	PG-16 OG-15
1 16		
$DATE	{mm/dd/yy / mmddyy}	OG-10
1		
$DEND		DB-9

CARD FORMAT		PAGE REFERENCE
1 16		
$DETACH	{S. Sxxx / device,channel,number,[, , dir]}	OG-11
1		
$ENDCH		PG-58
1 16		
$ENDEDIT	any text	SG-23
1 16		
$ENTRY	[{deckname / externalname}]	PG-58
1 16		
$EXECUTE	program name	PG-15
1 8 16		
$FILE	deck name 'file name',[primary unit], [secondary unit] [,{MOUNT / READY / DEFER}] [,{MOUNTi / READYi / DEFERi}]	PG-49
1 16		
$ETC	[CKFILE], BLOCK=xxxx[,{SINGLE / DOUBLE}] [,{REEL / REELS}] [,{HIGH / LOW}]	
1 16		
$ETC	[MIXED][,{SEQ / NOSEQ}][,{CKSM / NOCKSM}] [,{NOCKPT / CKAFLB / CKCKFL / CKLBFL}] [,{PRINT / PUNCH / HOLD / SCRTCH}]	
1 16		
$ETC	[{ADDLBL=exname / NSLBL=exname}][,LRL=xxxx] [,RCT=xxxx] [,EOR=exname]	
1 16		
$ETC	[ERR=exname] [,{TYPE 1 / TYPE 2 / TYPE 3}] [, EOF=exname]	

CARD FORMAT

PAGE REFERENCE

1 8 16

$IBCBC deck name [{ LIST / FULIST / NOLIST }] [, { DECK / NODECK }] [, { REF / NOREF }] [, SPACE] — PG-45

1 8 16

$IBDBC [name] location [, FATAL] [, { DUMP=OU / DUMP=Uxx }] [, MARKER=file name] — DB-6

1 16

$IBDBL [TRAP MAX=xxxxx] [, LINE MAX=xxxxx [, { FORTRAN / JOBOU / JOBOUL / IOBS=file name / IOOP=unit }]] [, DWU=unit]

DB-8

1 16

$IBEDT [{ S.SLB1 / S.SUxx[=Iyy] / Iyy[R] / NONE } [{ (S.SLB2) / (S.SUxx[=Iyy]) / (Iyy[R]) }]] — SG-21

1 16

$ETC [{ S.SUxx[=Iyy] / Iyy / IyyR=Iyy } [{ (S.SUxx[=Iyy]) / (Iyy) / (IyyR=Iyy) }]]

1 16

$ETC [LABEL(nn, mm, q, p)] [, { SOURCE / NOSOURCE }] [, MXBLK(nnnn)] [, NOMAP]

1 16

$ETC [EDTFIL(nn)] [, CORE(nnnnn)] [, MIN]

1 8 16

$IBFTC deck name [{ LIST / FULIST / NOLIST }] [, { DECK / NODECK }] [, { REF / NOREF }] [, { DD / SDD / NODD }] — PG-44

1 8 16

$IBJOB program name [{ GO / NOGO }] [, { LOGIC / DLOGIC / NOLOGIC }] [, { MAP / NOMAP }] [, { FILES / NOFILES }] — PG-39

CARD FORMAT		PAGE REFERENCE
$ETC	[{ IOOP1 / IOOP2 / IOLS / IOBS }] [,{ SOURCE / NOSOURCE }] [,{ DECK / NODECK }] [,{ COBOL / NOCOBOL }]	
$ETC	[{ COPY / COPY=Iyy / COPY=unit[=Iyy] / NOCOPY }]	
$IBLDR	deck name [date of assembly]	PG-46
$IBMAP	deck name [{ LIST / NOLIST }][,{ DECK / NODECK }][,{ REF / NOREF }] [, SYMSIZ=xxxxx][,{ RELMOD / ABSMOD }][,{ DD / SDD / NODD }]	PG-45
$IBREL		PG-42
$IBSRT	[{ NOLIST / NOTYPE }]	SM-18
$IBSYS		PG-15 OG-8
$ID	any text	OG-9
$IEDIT	[{ IN / Uxx / Iyy [R] }] [,{ SRCH / NOSRCH }] [,{ REWIND / NOREWIND }]	PG-42
$JEDIT		SG-32

CARD FORMAT		PAGE REFERENCE
1 16		
$JOB	any text	PG-15 OG-8
1 16		
$LABEL	'filename', [file serial number] , [reel sequence number] , [{ date / days }] , identification	PG-52
1 7 8 16		
$LINK x link name	deck name	PG-57
1		
$LIST		OG-9
1 16		
$NAME	{ deckname (exname) = exname / exname = exname / deckname ('file name') = 'file name' / 'file name' = 'file name' }	PG-53
1 16		
$OEDIT	[{ OU / Iyy / Uxx[=Iyy] / k=Iyy }]	PG-44
1 16		
$OMIT	[{ exname / deckname(exname) }] , . . .	PG-54
1 16		
$OPEN	{ S.SPP1 / unit=Iyy / S.SUxx / Iyy } [, REWIND]	PG-15 OG-14
1 16		
$PAUSE	any text	OG-10

CARD FORMAT		PAGE REFERENCE
1 16		
$POOL	BLOCK=xxxx ,BUFCT=xxx [,'filename']	PG-51
1 16		
$RELOAD	{ Uxx / Iyy / d[c]LIN } [, NAME=program name] [, { SRCH / NOSRCH }]	PG-59
1		
$RESTART	restart code	OG-10
1		
$RESTORE		OG-14
1		
$STOP		OG-10
1 16		
$SWITCH	{ S.Sxxx / Iyy [R] } , { S.Sxxx / Iyy [R] }	PG-16 OG-13
1 16		
$TIME	xxx	OG-11
1		
$UNITS		OG-15
1		
$UNLIST		OG-9
1 16		
$USE	deckname (exname), . . .	PG-53

Sort Control Cards

1 16		
$IBSRT	[{ NOLIST / NOTYPE }]	SM-18

CARD FORMAT	PAGE REFERENCE
END	SM-25
FILE, INPUT/nnnnn, BLOCKSIZE/nnnn [, {REELS/nn | REELS/X}] [, {MODE/B | MODE/D}] [, {LABEL/S | LABEL/N}]	SM-19
* [SERIAL/nnnnn] [, RLSEQ/nnnn] [, IDENT/xxHfile name] [, CKSUMS]	
* [BLKSEQ] [, CKPT] [, {REWIND | UNLOAD}]	
FILE, OUTPUT, {BLOCKSIZE/nnnn | BLOCKSIZE/U} [, {MODE/B | MODE/D}] [, {LABEL/N | LABEL/S}] [, SERIAL/nnnnn]	SM-20
* [RLSEQ/nnnn] [, IDENT/xxHfile name] [, RETAIN/nnnn] [, CKSUMS] [, BLKSEQ] [{REWIND | UNLOAD}]	
LABEL,IDENTIFICATION/xxxH where xxx≦120	SM-24
MERGE, FILES/(n1, n2, ...) [, ORDER/m] [, FIELD/(f1 [, {A | D}] , ...)] [, {SEQUENCE/C | SEQUENCE/S}]	SM-21
MODIFICATION, PROGRAM/PxMxx, UNIT/S.Sxxx	SM-27
OVERFLOW,BLOCKS/n [, REELS/nn] [, RLSEQ/nnnn]	SM-24

CARD FORMAT	PAGE REFERENCE
OPTION [,NOCKPT] [,EQUALS] [,CKSUMS] [,NODUMP] [,NOTAPE] [,NOEXTRACT]	SM-24
RECORD [, {TYPE/F / TYPE/V}] , LENGTH/([Lmin,] Lmax$_1$ [, Lmax$_2$] [, Lmax$_3$])	SM-20
* FIELD/(n [{B / C}] [{U / S}], ...)	
SORT, FILE/nnnnn, ORDER/ { m / (m1, m2) } , FIELD/(f1 [{A / D}] , ...) [{SEQUENCE/C / SEQUENCE/S}]	SM-21
SYSTEM, INPUT/ {channel / unit / (unit, unit, ...)} [, MERGE/ {channel / (channel, channel)}]	SM-22
* [OUTPUT/ {channel[=Iyy] / (channel=Iyy,=Iyy) / unit[=Iyy] / (unit[=Iyy], unit[=Iyy])}] [, CORE/(n1, n2, n3)] [, DISK/nnnn]	

Edit Control Cards

$IBEDT [{S.SLB1 / S.SUxx[=Iyy] / Iyy [R] / NONE} [{(S.SLB2) / (S.SUxx[=Iyy]) / (Iyy [R])}]]	SG-21
$ETC [{S.SUxx[=Iyy] / Iyy / IyyR=Iyy} [{(S.SUxx[=Iyy]) / (Iyy) / (IyyR=Iyy)}]]	
$ETC [LABEL(nn,mm,q,p)] [, {SOURCE / NOSOURCE}] [, MXBLK(nnnn)] [, NOMAP]	

CARD FORMAT		PAGE REFERENCE
$ETC	[EDTFIL (nn)] [,CORE (nnnn)][,MIN]	
$ENDEDIT		SG-23
$JEDIT		SG-32
AFTER	{phase / EOR}	SG-26
sysnam CALLS	phase1, phase2, . . .	SG-23
ETC	continuation of variable field of preceding card	SG-28
DUP	{S.Sxxx / Iyy [R]}, {S.Sxxx / Iyy [R]}, n, [inlabel] [, cdate] [, oulabel] [, rdays]	SG-26
INSERT	{phase / EOR}[, {S.SUxx / Iyy [R]}]	SG-24
LIBEND		SG-28
LIBE	[phase] [, format]	SG-28
MODIFY	phase [, {S.SUxx / Iyy [R]}]	SG-25

CARD FORMAT			PAGE REFERENCE
load address	OCT	patch1, patch2, . . . , patchn	SG-29
	REMARK	any text	SG-27
[sysnam]	REMOVE	{ phase / EOR }	SG-25
	REPLACE	phase [, { S.SUxx / Iyy [R] }]	SG-24
	REWIND	{ S.Sxxx / Iyy [R] }	SG-27

Update Program Control Cards

1	9(6)	18(4)	23(5)	29(5)	35, 41, 52	58(5)	65(6)	PAGE REFERENCE
$ASSIGN	[primary unit]	[OPEN]	[{ TYPE1 / TYPE2 / TYPE3 }]	[{ LABEL / NOLAB }]	[label information]	[{ REEL / REELS }]	[secondary unit]	PG-77
$DELETE	[from serial number]	[to serial number]						PG-79
$ENDRUN	[{ deckname / STOP }]							PG-76
$EXECUTE	UPDATE							PG-73

CARD FORMAT					PAGE REFERENCE
	1	9(6)	18(7)	27(7)	36(6)	
$LOCATE	deckname	[{INSERT / REMOVE / REPLACE}]	[EXTRACT]	[to deckname]		PG-78
	1	13(5)				
$MESSAGE		[PAUSE] or [message to be printed]				PG-79
	1	9(8)	18(8)	27(8)		
$NUMBER		[initial serial number]	[from serial number]	[to serial number]		PG-78
	1	9(7)	18(7)	27(7)	36(7)	
$OBTAIN		{LIST / PRINT / DECK / INDEX / SUMMARY / 16K / 32K / LABEL / NOLABEL}		any four of the options may be used		PG-75
	1	9, 18(6)		27, 36(6)		
$OUTPUT		unit assignments for new master: primary, secondary		unit assignments for auxiliary new master: [primary], [secondary]		PG-79
	1					
$PLACE						PG-79
	1					
$REWIND						PG-78

CARD FORMAT

1	6(8)	14(2)	17(5)	23(5)	29, 35, 46	51, 57, 68	PAGE REFERENCE
$RUN	[UPDATE / EXTRACT / INPUT / OUTPUT / NOCOPY / GENERATE / SYSREL]	bb	[yyddd]	[LABEL]	[label information new master]	[label information auxiliary master]	PG-74

Utility Control Cards

CARD FORMAT

1	16	PAGE REFERENCE
$EXECUTE	IBUTL	PG-88

1	8	PAGE REFERENCE
IBUCD[W]	Specifications	PG-96

1	8	PAGE REFERENCE
IBUDD	Specifications	PG-95

1	8	16	PAGE REFERENCE
IBUDD	LABEL	Specifications	PG-95

1	8	PAGE REFERENCE
IBUFB[W]	Specifications	PG-92

1	8	PAGE REFERENCE
IBUFF[W]	Specifications	PG-92

1	8	PAGE REFERENCE
IBUFH[W]	Specifications	PG-92

Appendix B. Control Card Check List

	Source language programs included:			Relocatable Binary	
	COBOL	FORTRAN	IBMAP	Programs	Comments
$JOB	X	X	X	X	One required at the beginning of each job
$ID	O	O	O	O	Transfers control to installation accounting routines
$*	O	O	O	O	Comments card
$PAUSE	O	O	O	O	Permits operator action
$IBJOB	X	X	X	X	Initiates an IBJOB application; one required for each processor application
$IBSYS	O	O	O	O	Next job segment will not be processed by IBJOB: control is passed to IBSYS
$IBFTC		X			Precedes each FORTRAN deck
$IBCBC	X				Precedes each COBOL deck
$CBEND	X				Follows each COBOL deck
$IBMAP			X		Precedes each MAP deck
$IBLDR				X	Precedes each relocatable binary program to be loaded
$ENTRY	X	X	X	X	Specifies location of initial transfer; initiates object program loading
$RELOAD	O	O	O	O	Reloads absolute program produced by IBLDR
$FILE	O	O	O	O	Provides file specifications; supersedes any deck specifications
$LABEL	O	O	O	O	Provides label information for files
$POOL	O	O	O	O	Designates files to share common buffer areas, i.e. pools
$USE	O	O	O	O	Specifies data, procedure or file sections to be used
$OMIT	O	O	O	O	Deletes file, data, or procedure sections
$NAME	O	O	O	O	Used to change control section or file names
$ETC	O	O	O	O	Continues variable fields of the above Preprocessor cards
$CHAIN	O	O	O	O	One required to initiate a CHAIN application
$LINK	O	O	O	O	One required at the beginning of a link deck
$ENDCH	O	O	O	O	One requred to terminate a CHAIN application

Notation: X necessary; O optional; blank does not apply.

$*	O	For comments
$IBSYS	X	One required to transfer control to the Supervisor
$ID	X	One required to transfer control to installation accounting routine
$JOB	X	One required at the beginning of each job
$IBSRT	X	One required at the beginning of each sort application
FILE (Input)	X	One required for each input file
FILE (Output)	X	One required for the output file
RECORD	X	One required to describe the record format
SORT	X	One required for each file to be sorted, if any
MERGE	X	One required for the files to be merged, if any
SYSTEM	X	One required to specify the 7040/7044 System environment
LABEL	O	Required for nonstandard labels only
MODIFICATION	O	For introducing modification programs only
OVERFLOW	O	For restarting a sort application after overflow
OPTION	O	Provides for additional sort application options
END	X	One required to indicate end of sort control cards

Notation: X necessary; O optional.

Appendix C. 7040/7044 — 1401 Auxiliary Programs

Input/Output Utility Program

The 7040/7044 — 1401 Input/Output Utility program provides a facility for producing, off-line, a stacked system-input file on magnetic tape, and for processing, off-line, a system-output tape or system-peripheral-punch tape to produce a printed listing and/or a deck of punched cards. These two program functions, called the Input Stacking function and the Output Print/Punch function, respectively, are selected and controlled by a control card and sense switch settings. The two functions are described separately below.

Machine Requirements

The 7040/7044 — 1401 Input/Output Utility program requires an IBM 1401 Data Processing System with the following:

At least 4000 positions of core storage
The Column Binary Feature (#1990)
Advanced Programming Features
The High-Low-Equal Compare Feature

The 1401 system must have the capability of attaching the following required input/output devices:

A minimum of one IBM 729 or 7330 Magnetic Tape Unit
One IBM 1402 Card Read Punch
One IBM 1403 Printer, Model 2

Two magnetic tape units are required to utilize all program options.

Input Stacking Function

The Input Stacking function is designed to eliminate the extensive and time-consuming use of the 7040/7044 on-line card reader attendant to loading programs and input data directly from card decks. The program performs an off-line card-to-tape operation with blocking of the input data as required by its varied format. Standard input for this function is a deck of stacked jobs; the deck can consist of both BCD cards and binary cards. A tape is produced that contains records in the format acceptable to the 7040/7044 Operating System (16/32K) for the system input file. If desired, a listing of the BCD records on this tape is also produced.

Basically, this program function reads card records, moves them to one of two separate work areas, depending on their mode, and fills the tape output area; and creates the control word which describes the Type-3 record so formed. See the publication *IBM 7040/7044 Operating System (16/32K): Input/Output Control System*, Form C28-6309, for a description of record types.

Input File

The input file consists of BCD and/or binary card decks, or of unblocked BCD card images on tape. If input to the stacking function is a stack of jobs for the Operating System, these card decks (or the equivalent on tape) must conform to the monitor requirements; that is, each deck must contain the appropriate $ control card. A $RUN control card must always accompany the input file; if the input is on cards, the deck is preceded by the $RUN card. (See the publication *IBM 7040/7044 Operating System (16/32K): Operator's Guide*, Form C28-6338, for a detailed description of this card.) Figure 45 shows a sample input file setup.

Output File

BLOCKING

The output file consists of blocked tape records in Type 3 format. The blocking for both BCD and binary records is determined by the blocking parameter in the $RUN control card. The maximum blocking factor possible is 10 BCD records per block, unless the program is reassembled for a 1401 with more than 4,000 positions of core storage. (See the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide*, Form C28-6339, for detailed information.) Binary card records will block to one-half of the BCD factor specified. There are, however, two major exceptions.

When it is determined that a card record is in the mode opposite to that of the previous record, the previous block of records is written on tape. This creates short length records, but it ensures that all records in a block are in the same mode.

The other exception concerns system control cards (cards with a $ in column 1). These card records will be unblocked, i.e., they will have a blocking factor of 1.

BCD Records: BCD records consist of 90 characters (15 words) per logical record as follows:

Positions 1 — 6
Control Word. This contains the number of characters in the logical record (90), including the control word, and the control code character for the next block; for example, 000904 or 000905. Both the 4 and 5 signify a BCD card record. The 5 also indicates a change in the mode of the next block.

Positions 7 – 86
Card Record. This is the 80-column BCD card image.
Positions 87 – 90
bbbb (ignored by the Operating System).

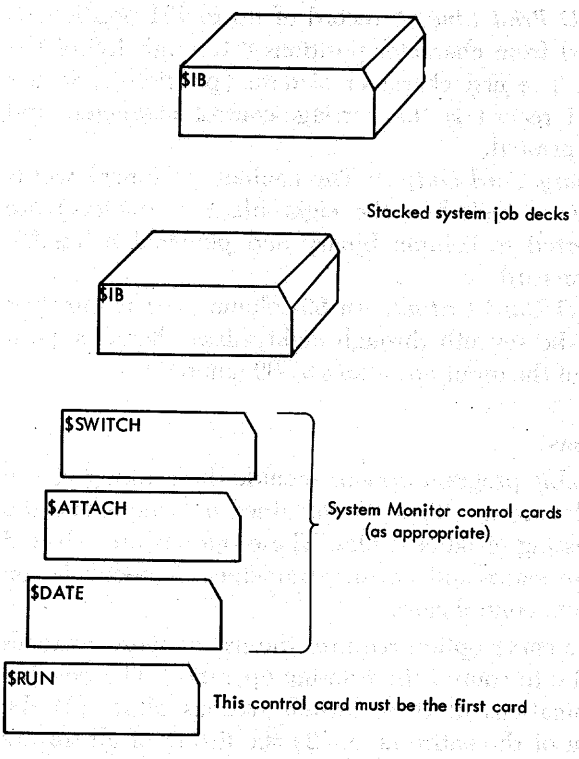


Figure 45. Sample Input File Set-Up for Stacking

Binary Records: Binary records consist of 29 words per logical record:

Control Word
One binary word containing the number of words in the logical record, *not* including the control word, the control code character for the next block, and other control information; for example, 500034200006 or 500034200007. The 6 and 7 signify a binary record. The 7 indicates a change in the mode of the next block. The prefix and tag (5 and 2) are ignored. They are included for compatibility with other systems.
Card Record
Words 2 – 29. This is the column binary equivalent of an 80-character binary card with blanks in the last eight character positions.

Labels: Both header and trailer labels conform to the standard 120-character tape label format. (See the publication *IBM Standard Tape Labels*, Form C28-8142.) Whenever label processing is specified, output trailer labels contain the record count and input header labels are printed (if input is on tape).

Options

Five program options are available:

1. If sense switch C is set ON, the program accepts an input file consisting of unblocked BCD card images on tape, instead of the usual card input file. In this case, however, the $RUN control card for the program must still be read in from the card reader. A tape mark on the input tape file serves to indicate the end of job. Labeled input tape files, however, may be multireel.
2. If sense switch B is set ON, each time a $JOB control card is read, the contents of the card are printed on a new page and the program then halts.
3. If sense switch E is set ON, a printed listing is produced, composed of all the $ control cards that are written on the output file. Each time a $JOB control card is read, the contents of the card are printed on a new page.
4. If sense switch F is set ON, a printed listing is produced, composed of all the BCD records that are written on the output file. Each time a $JOB control card is read, the contents of the card are printed on a new page.
5. Standard labeling information may be provided for the output file, in conjunction with the use of the LABEL option on the $RUN card.

Output Print/Punch Function

The Output Print/Punch function is an off-line aid for processing certain 7040/7044 Operating System (16/32K) output tapes. This function is designed expressly for tapes which are produced on the S.SOU1 or S.SPP1 symbolic system units; these tapes may contain BCD print records, BCD punch records, and/or binary punch records. However, any tape file whose format conforms to one of those described below may be processed by the program. An optional feature is the facility for selectively printing the contents of the MAP Symbolic tapes.

The punching or printing operations are performed with control words (part of the input data) determining the type of output desired and the mode of the next block of records. The tape may contain blocks in both modes; however, all records within a block must be of the same mode.

The user has the option of processing either labeled or unlabeled tapes. Labeled tape files may be multireel.

Input File

Any tape file used as input to the program must conform to the following format specifications, except for the block size limit if the program is reassembled for

a 1401 with more than 4,000 positions of core storage. (See the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide*, Form C28-6339, for detailed information.) The data formats described are those of Type 3 output as produced through the 7040/7044 Input/Output Control System.

Blocking: A block (tape record) of BCD input may consist of up to 900 characters; this allows a maximum of six logical print records (full line) or ten BCD punch records. Binary input may be a maximum of five cards per block (870 characters). All records in a block must be in the same mode.

BCD Records: BCD records consist of a maximum of 138 characters (23 words) per logical record for a full line of printing, or 90 characters (15 words) in the case of a card image, made up as follows:

Positions 1-6

Control Word. This word contains the number of characters in the logical record (including the control word) in positions 3-5, and the control code character in position 6. The control code character indicates the operation to be performed upon the record and whether there is a change in mode for the next block. For example, in the control words 001202 or 001203, the 120 indicates the number of characters in the record, and both the 2 and 3 indicate a BCD print record. The 3 also shows a change in mode for the next block.

Positions 7-86 or 7-138

This is a record of up to 80 positions for a BCD card, or up to 132 positions for a BCD print line.

Positions 87-90

For BCD card records, these positions are included to fill out the record to an even multiple of six.

Binary Records: Binary records for punching column-binary cards have a maximum of 29 words per logical record as follows:

Control Word

This is one binary word containing the number of words in the logical record, *not* including the control word, the control code character for the next block, and other control information; for example, 50 00 34 20 00 06 or 50 00 34 20 00 07. Both the 6 and 7 signify a binary punch record. The 7 also indicates a change in the mode of the next block. The prefix and tag (5 and 2) are ignored. They are included only for compatibility with other systems.

Card Record

Words 2-29. This is the column-binary equivalent of an 80-character card with blanks in the last eight character positions to fill out the record to an even multiple of six.

Labels: All labels should conform to the standard 120-character label format. Header labels will be printed out, separated from the main listing.

Output Files

BCD Print Line: A record of up to 131 positions is printed from character positions 8 through 138 of the input. The first character of data (position 7) in the logical record is the carriage-control character, and is *not* printed.

Binary Card Output: The contents of binary words 2-29 (not including the eight blank characters) are converted to column binary and punched in an 80-column card.

BCD Card Output: An 80-column card is punched from the seventh through eighty-sixth character positions of the input (positions 87-90 ignored).

Options

Available program options include the printing of BCD punch records (Type 3) contained on a tape, and the processing of labeled files. These options are selected by the PRINT and LABEL parameters, respectively, on the $RUN control card.

The PRINT option requires the use of sense switches B and C to control the printing operation. The possible combinations of these switch settings allow (1) the listing of the entire tape, (2) the listing of all $IBMAP control cards only, or (3) the printing of selected MAP program decks. The BCD records in a mixed mode file are printed if sense switch E is set ON. All binary punch records that are read are ignored; that is, they are not punched.

The LABEL parameter on the $RUN card specifies that the file is labeled, and that header labels are to be printed for operator verification.

Under the standard OUTPUT type of run, several sense-switch control options are available:

1. If sense switch B is set ON, each $JOB control card read is printed on a new page and the program halts.
2. If sense switch C is set ON, printing is suspended. Only the punching of BCD and binary card records occurs.
3. If sense switch E is set ON, punching is suspended. Only a printed listing is produced.
4. If sense switch B is set ON in conjunction with sense switches C and/or E, all or part of a job can be skipped. (A job is defined as the group of records between two successive $JOB control cards.) Punching and/or printing of the current job is terminated, and when the next $JOB card is encountered, it is printed and the program halts for operator action. (See

the publication *IBM 7040/7044 Operating System (16/32K): Operator's Guide*, Form C28-6338, for detailed procedures.)

Map Symbolic Update Program

The 7040/7044 – 1401 MAP Symbolic Update program provides the programmer with an off-line facility for maintaining a master tape file of MAP symbolic programs for the 7040/7044 Operating System (16/32K). The program eliminates the necessity of keeping a card file of MAP program decks, as the programmer can modify or replace symbolic decks on an existing master tape, or add decks to and delete decks from this tape. The program also allows the user to prepare a 7040/7044 system input tape (S.SIN1) by selecting symbolic decks from the master file and placing them on an extract file tape, along with any necessary system control cards. The initial symbolic master tape may be generated most conveniently by use of the 1401 Input/Output Utility program (described elsewhere in this publication), especially if a large number of cards are involved. However, the update program may also be used for this function.

The operation of the program is controlled by nine $ control cards. Two of these, $RUN and $ENDRUN, must be used and must be the first and last cards, respectively, in the change file card deck. The other seven cards, whose use varies according to the requirements of a given run, are the $ASSIGN, $OUTPUT, and $REWIND cards which select and control the input and output tape devices, and the $PLACE, $LOCATE, $DELETE, and $NUMBER cards which control the processing of decks, portions of decks, and individual cards.

The possible inputs to the program are a master input file (tape), an alternate master input file (tape), and a change file (cards). Of these three, at least the change file must be present. The possible outputs of the program are an updated master file (tape), an extract file (tape), and an operator's log (listing). Of these three, at least the operator's log and one of the others will be present.

Basically, the user can direct the program, by control cards and correction cards, or by control cards alone, to update the master file tape, or to produce a system-input tape (extract file). Updating the master file may involve insertion, replacement, or deletion of whole program decks, or it may involve the modification of one or more program decks by insertion, replacement, or deletion of individual cards or groups of cards. This updating of the master file is actually done, of course, by producing a new master file as output, from a combination of the old master file and the change file, though one may logically speak of performing the above-mentioned operations upon the same master file.

Modifications within individual program decks are performed on the basis of symbolic-deck serial numbering, required of both the deck to be modified and the correction cards. These serial numbers must be punched in the card identification columns (73-80) of the original MAP program-deck cards and the correction cards. The serial numbers consist of a three-column (73-75) alphabetic name or "ident," and a five-column (76-80) sequence number. The last digit of the sequence number (column 80) should always be zero on the original program deck cards, i.e., should be non-zero only on insert cards of the correction deck, allowing up to nine insertions between original program cards. (Decks or portions of decks can be renumbered by the program; see the $NUMBER card description under "Control Cards.")

The required serial numbering restricts the original decks to 10,000 cards, but a deck can be divided into subdecks, each up to 10,000 cards in length, by varying the ident. This scheme may be desirable for reasons other than the size limitation, e.g., for identification of closed subroutines and program sections.

Production of a S.SIN1 file involves placing program decks and individual cards (7040/7044 system $ control cards) from the change file, and/or selected decks from the master (or alternate master) input file onto the extract file. Both production of the extract file, and updating of the master file, then, may involve reordering of decks from input to output. The programmer should be aware of certain principles underlying the operation of the update program, in order to avoid faulty runs which could result from the attempt to reorder decks.

The program always executes control-card instructions from the point at which the input file is stationed at that time. In addition, the program only searches forward (on the input tape) in seeking a match for the deck name specified by a control card. Thus, the programmer is required to know the order of decks by deck name on the input tape(s) and the current reference point on the tape(s). When an updating or extract run requires a reordering of deck sequence, the input master file tape (or alternate, as appropriate) has to be rewound, and in some cases, those programs already processed by previous control cards must be bypassed. See Figure 46 for an illustration of control card usage for an updating run involving reordering.

DEFINITION OF SYMBOLIC DECK

For the purpose of the update program, a symbolic deck is defined as a MAP language program deck, the

first card of which is a $IBMAP card with a deck name in columns 8-13 and the appropriate system parameters beginning in column 16. The last card of the deck contains the symbolic END statement. Symbolic decks in the change file that are to be used for updating must not be preceded or followed by any 7040/7044 control cards other than $IBMAP.

Machine Requirements

The 7040/7044 – 1401 MAP Symbolic Update program requires an IBM 1401 Data Processing System with the following:

At least 4000 positions of core storage
The Column Binary Feature (#1990)
Advanced Programming Features
The High-Low-Equal Compare Feature

The 1401 system must have the capability of attaching the following required input/output devices:

A minimum of two IBM 729 or 7330 Magnetic Tape Units
One IBM 1402 Card Read Punch
One IBM 1403 Printer, Model 2

Up to four magnetic tape units are required to utilize the EXTRACT and alternate-master-input capabilities of the program.

Input/Output Files

The files which the update program uses or creates are summarized in the following table:

TAPE UNIT	FILE
1	Old Master input
2	Updated Master output
3	Extract output (S.SIN1)
*	Change input
4	Alternate Master input
**	Operator's Log

*The change file is assigned to the 1402 Card Read Punch.
**The operator's log is produced on the 1403 Printer.

The master and extract tape files contain complete symbolic decks in Type-3 format. (See the publication *IBM 7040/7044 Operating System (16/32K): Input/Output Control System,* Form C28-6309, for record types.) The $ control cards are unblocked while the symbolic cards in each deck may be blocked. Any 7040 $ control cards other than $IBMAP which precede or follow complete MAP symbolic decks on the master input file will be ignored. These cards, however, may be inserted on the extract file from the change file.

The extract file can be used as the system input file or as a master or alternate master input file for another update and/or extract run.

The alternate master file may be used to merge additional decks with the master file (by means of the $ASSIGN card, described later in this appendix).

The change file contains all control cards for the run, symbolic changes for individual decks and/or complete MAP symbolic decks. Complete MAP symbolic decks on the change file, which are being inserted onto the master output or are replacing decks on the master input file, must not be preceded or followed by any 7040 $ control cards other than $IBMAP.

An operator's log, produced on the printer, identifies each $IBMAP record encountered by the program as input and produced as output. Out-of-sequence conditions within a master input deck, and input-tape labels (if any) are also printed.

Any tape file can be multi-reel. At end of reel, the operator mounts a new reel and presses START to continue processing.

BLOCKING

The blocking factor of five records per block specified in the following paragraphs is a maximum for the distributed version of this program, which is assembled for a 4K 1401. If the user has a 1401 which is larger than 4K, the program may be reassembled to take advantage of the extra core storage capacity through an increased maximum block size. (See the publication *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide,* Form C28-6339, for detailed information.)

Input Tape Files: Input tape files may be unblocked or blocked, up to five logical records per block (see "Logical Record" below). They must be in Type 3 record format, i.e., a file produced by a simple card-to-tape utility run is not acceptable.

Output Tape Files: The output tape files are blocked according to the blocking factor specified in logical records per block, on the $RUN control card, up to the maximum of five. Only the MAP symbolic card records are blocked; the $IBMAP control cards (and any other $ cards written on the extract file) are always unblocked. The output files are in Type 3 record format.

Logical Record: The BCD card records consist of the following 90 characters:

Positions 1-6
Control word. This contains the number of characters in the logical record (090), in position 3-5, and the control character for the next blocks (always 4), in position 6. The word equals 000904.
Positions 7-86
Card record. This is the 80-column BCD card image.
Positions 87-90
bbbb (ignored by Operating System).

The control character "4" in the control word indicates a BCD card-image record, with no change in mode for the following block.

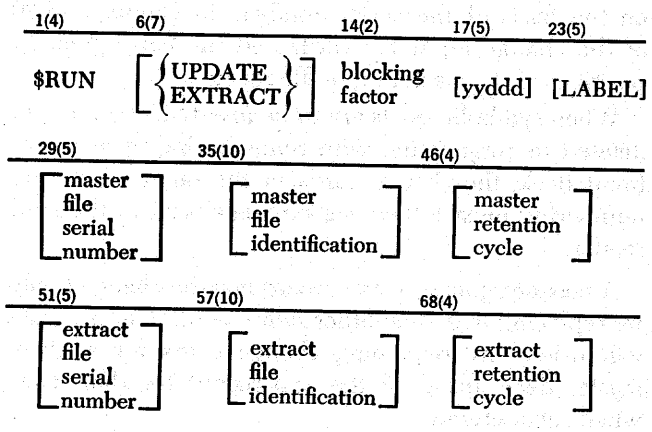
Control Cards

The control cards for this program are of the fixed-form type. Each field associated with a given card, whether required or optional, occupies a unique position on the card. In the card formats given below, the first number directly above a card field indicates the card column in which the field must begin. This number is followed immediately by another number in parentheses, indicating the maximum length of the field.

Parameters that are specified literally – that is, as they appear on the card – are given in capital letters; e.g., LABEL, as opposed to fields the contents of which are merely named. Also, fields which are optional or not essential to the running of the program are enclosed in square brackets. Where a choice of alternative parameters exist, they are enclosed in braces.

$RUN CARD

The format of the $RUN card is:



The $RUN card must be the first control card in the change file. The available options are:

[UPDATE / EXTRACT]

UPDATE specifies that *only* a new master file is to be created. EXTRACT specifies that *only* an extract file is to be created. Leaving the field blank specifies that both a new master file and an extract file are to be created.

blocking factor

Must be 2 digits, from 01 to 05, which specify the output blocking factor. This blocking factor will be used on both the master and the extract output files. The master input files need not be blocked the same, but must not exceed the maximum.

[yyddd]

Current date. This field is required for labeled files. If specified for unlabeled files, it will be printed on the operator's log only.

[LABEL]

Specifies labeled files. Leaving the field blank specifies unlabeled files. All input and output files must have the standard 120-character tape label, if label is specified. Input file labels will be printed on the operator's log. Labels of tapes to be used for output will be checked for expiration of retention period and the program will halt if the reel should not be used.

[file serial number]

A five-character alphameric file serial number. The file serial number in columns 29-33 is used in the header label of the new master file. The file serial number in columns 51-55 is used in the header label of the extract file. These fields may be omitted.

[file identification number]

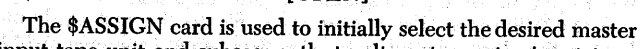
A ten-character alphameric file identification number. The file identification number in columns 35-44 is used in the header label of the new master file. The number in columns 57-66 is used in the header label of the extract file. These fields may be omitted.

[retention cycle]

A four-digit numeric field that gives the number of days the file is to be retained. The number in columns 46-49 is used in the retention period section of the new master file header label. The number in columns 68-71 is used in the extract file header label. These fields may be omitted, in which case zero is assumed.

$ASSIGN CARD

The format of the $ASSIGN card is:



The $ASSIGN card is used to initially select the desired master input tape unit and subsequently to alternate master input tape units. This card need not be used if no master input file is present during the run. When used, it must precede the first $LOCATE card that refers to the master input field or the $ENDRUN card if no such $LOCATE card exists. The two fields of the $ASSIGN card are issued as follows:

u

Either a 1 or 4 to indicate which master input tape unit is to be used for the operations that follow.

[OPEN]

The first reference to a new master input reel and the first reference to a new alternate master input reel must specify OPEN. Reels will be rewound, and input labels, if any, will be printed. Subsequent $ASSIGN cards should contain blanks in 18-21.

$OUTPUT CARD

The format of the $OUTPUT card is:

The $OUTPUT card is used to force an end-of-reel condition on either the master or extract output files after the current deck is completely processed. By this means, the splitting of a deck between two reels of tape can be avoided and the contents of a given output reel can be predetermined.

u

Either a 2 or 3 to indicate which output tape unit is to be closed. When the unit is closed, the standard end-of-file procedure is used and the tape is rewound and unloaded. The next reel mounted will be automatically opened.

$REWIND CARD

The format of the $REWIND card is:

1(7)
$REWIND

The $REWIND causes the currently-assigned master input tape to be rewound. It is used to change the order of symbolic decks on the master or extract output files.

$LOCATE CARD

The format of the $LOCATE card is:

1(7)	9(6)	18(7)	27(7)	36(6)
$LOCATE	deckname	[{INSERT / REPLACE / REMOVE}]	[EXTRACT]	[to deck-name]

The $LOCATE card copies the master input file up to, but not including, the deck identified by the six-character deckname. The program then performs the action indicated by the procedure options for one card deck. The available options are as follows:

[{INSERT / REPLACE / REMOVE}]

Procedure parameters, pertaining to entire program decks.
INSERT copies the card deck following the $LOCATE card from the change file onto the output file(s). The master input file is positioned to the first card following the deck identified by the deckname in columns 9-14 of the preceding $LOCATE card, or at the beginning of the file if no prior $LOCATE card had been encountered.

REPLACE copies the card deck from the change file onto the output file(s). The master input tape is positioned to the first card following the deck previously identified by the deckname in columns 9-14. The deck from the change file is identified on the output file by the deckname in columns 9-14.

REMOVE spaces the master input file to the first card following the deck identified by the deckname (columns 9-14) when columns 36-41 are blank, or to the first card following the deck identified by the deckname in columns 36-41. In the latter case, all decks up to and including the deck named in columns 36-41 will not be copied onto the new master.

If the procedure option is omitted, the deck identified by the deckname in columns 9-14 is copied with corrections, if any, onto the output file(s). Any change cards following the $LOCATE card are matched by serial number with the card records on the master input file. If the serial numbers match, the change card replaces the master card on the output file(s). Change cards that do not match are inserted on the output file(s).

The insert or replacement deck is on the change file. In either case, the deckname on the$IBMAP card following the $LOCATE card in the change file will be checked against the deckname in the $LOCATE card, and a mismatch will cause a halt.

[EXTRACT]

This parameter causes the deck specified (with or without changes) to be placed on the extract file, when a new master is being created, so long as the $RUN card has not specified update only. It is interpreted in combination with the output option specified on the $RUN card as follows:

$RUN CARD	$LOCATE CARD	OUTPUT FILE(S)
EXTRACT	(ignored)	Extract only
UPDATE	(ignored)	Master only
(both)	EXTRACT	Extract and Master

If this option is omitted and the $RUN card has not specified EXTRACT, the deck will be written only on the new master file.

In conjunction with the REMOVE option *and* a deckname in columns 36-41, extraction of all decks within the range of the REMOVE will take place subject to the conditions tabulated above.

[to deckname]

This parameter is specified only in conjunction with the REMOVE option, which is shown above.

Remarks on the Use of Procedure Options and Symbolic Change Cards: All symbolic change cards, $NUMBER cards, and $DELETE cards (discussed below) that are applicable to a given master input symbolic deck must follow the $LOCATE card for that deck in the card reader. In addition, they can only be used to update symbolic decks already existing on the master input file. They cannot be used to alter an incoming change file deck that is being inserted onto the master output file or is replacing a deck on the master input file.

No additional control card is required to insert changes or replace records in a master input deck. Change cards following a $LOCATE card which does not have INSERT, REPLACE, or REMOVE specified will be inserted or will replace records in a master input deck on the basis of the serial numbers in columns 73-80 of the change input file cards and the corresponding positions of the master input file records.

When symbolic cards are to be inserted, replaced, or deleted in conjunction with renumbering of a master input deck, the change cards in the range of the renumbering must follow the $NUMBER card in the card reader.

A master input MAP END record may be changed only by replacing it with another END record. The program will insert any remaining change cards for a given master deck ahead of the END record for that deck, when encountered.

A $LOCATE card with a deckname only, followed by another 1401 $ control card (other than $NUMBER or $DELETE), will copy up to and including the named deck. The same is true for $LOCATE cards with only deckname and EXTRACT specified.

To space over decks on the master input file without copying them, a $LOCATE card with the REMOVE option specified must be used. The deckname in columns 9-14 specifies the first deck, and the deckname in columns 36-41 (if any) specifies the last deck to be spaced over.

$DELETE CARD

The format of the $DELETE card is:

1(7)	9(8)	18(8)
$DELETE	from serial number	[to serial number]

The $DELETE card causes the deletion of a portion of a symbolic deck on the master input file. The range of cards to be deleted is specified as follows:

from
serial
number

The serial number of the first or only record in the master input deck to be deleted. The contents of this field are matched to columns 73-80 of the records on the master input file.

[to
serial
number]

The serial number of the last master input record to be deleted. If this field is left blank, only one record will be deleted.

$PLACE CARD

The format of the $PLACE card is:

```
1(6)
$PLACE
```

All cards following the $PLACE card and preceding the next 1401 $ control card in the change file are copied onto the extract file, unless the $RUN card has specified update. This allows the production of a new system input file incorporating any system control cards.

$NUMBER CARD

The format of the $NUMBER card is:

```
1(7)       9(8)       18(8)        27(8)
$NUMBER    new        [ from   ]   [ to     ]
           serial     [ serial ]   [ serial ]
           number     [ number ]   [ number ]
```

The $NUMBER card is used to renumber all or a portion of a symbolic deck on the master input file. Its three fields are used as follows:

new
serial
number

The new ident and sequence number to be used. The first three characters are the ident (alphabetic) and the last five characters (numeric) are the sequence number. The last digit should be zero; this number will be incremented by 10 on each subsequent output record within the range of the renumbering.

[from
serial
number]

The serial number of the first record in the master to be renumbered. The contents of this field are matched to columns 73-80 of the records on the master input file. If blank, the program will begin numbering at the current record.

[to
serial
number]

The serial number of the last record to be renumbered in the master input deck. If this field is blank, the master input deck will be renumbered up to and including the END record.

The first record to be renumbered (columns 18-25) is given the new serial number (columns 9-16) and subsequent records are renumbered consecutively from the new number.

The serial numbers are punched in the card identification field (card columns 73-80) of all correction cards and are in the same relative positions in the tape record. The serial numbers have the following format:

IDENT	SEQUENCE NUMBER	INSERT NUMBER
Columns 73-75	76-79	80

The ident (columns 73-75) is alphabetic. Only the numeric sequence portion (columns 76-79) is incremented by one. Thus, the subdeck identified by columns 73-75 is limited to 10,000 cards (before insertions).

$ENDRUN CARD

The format of the $ENDRUN card is:

```
1(7)       9(6)
$ENDRUN    [deckname]
```

The $ENDRUN card must be the last control card in the change file. The deckname in columns 9-16 causes the master input file to be copied up to and including the deck identified by the deckname. If the master input tape reaches end-of-reel before the deckname is found, the program will halt to allow the operator to change reels.

If the deckname field is left blank, the run is terminated (except for trailer-label processing, if any) after the current deck is completely processed. There must be only one $ENDRUN card used in a run.

EXAMPLE OF CONTROL-CARD USAGE

Figure 46 illustrates the use of control cards for a sample updating run involving reordering, correction, and insertion of program decks.

Order of Decks on Master Input File (Tape 1):

1. PAY603 symbolic deck (payroll program)
2. INV900 symbolic deck (inventory program)
3. PRC450 symbolic deck (production control program)

Desired Order of Decks on Updated Master File (Tape 2):

1. PRC450 symbolic deck
2. INV900 symbolic deck including corrections
3. PAY100 symbolic deck (new payroll program)

Contents of Change File

```
                  Card Columns
1          9           18                  36
$RUN UPDATE        04  63091
$ASSIGN    1           OPEN
$LOCATE    PAY603      REMOVE              INV900
$LOCATE    PRC450
$REWIND
$LOCATE    PAY603      REMOVE
$LOCATE    INV900
(Correction cards in sequence by serial numbers)
$LOCATE    PAY100      INSERT
(Symbolic deck for PAY100)
$ENDRUN
```

Figure 46. Sample Update Run with Re-ordering

Appendix D: COBOL Error Messages

The list below classifies the COBOL error messages according to phase name and purpose. Note, however, that a message belonging to one phase may be used by another phase.

MESSAGE NUMBERS	PHASE NAME AND PURPOSE
1-100	IBMAP Phase B, output phase
101-349	IBMAP Phase A, scan phase
350-450	IBMAP Dictionary Reduction, assignment phase
501-600	IBMAP Interface, compiler interface
1100-1199	IBFTC Scan
1200-1299	IBFTC Storage Allocator
1300-1399	IBFTC Arithmetic and Logical Translator
1400-1499	IBFTC Indexing Analyzer
1500-1549	IBFTC Instruction Generator
1550-1599	IBFTC Indexing Generator
2000-2099	IBCBC Phase I, language reduction
2100-2199	IBCBC Phase II, syntax analysis
2200-2299	IBCBC Phase III, data reduction
2300-2399	IBCBC Phase IV, procedure generation

In the messages described below, the following conventions are used:

1. CSN(n) refers to the source language COBOL statement number that appears in the lefthand column of the source program listing produced by the Compiler.
2. 'A1', 'A2', and 'A3' are the variable parameters that appear in the error messages produced by the compilation. In many cases, the meaning of these parameters is clear from the text of the message; where it is not clear, an explanation is given.
3. In messages where the variable parameter can be a data-name, it can also be a literal or a figurative constant, if applicable.
4. A data-name used as a parameter will be followed by a code for error messages 2300 and above. These codes are described below:

N, C	Numeric computational
N, D	Numeric display
A	Alphabetic
RPT	Report
AN	Alphanumeric
GRP	Group

2000 CSN (n) RECORD 'A1' CAUSES TABLE OVERFLOW. RECORD IS SEGMENTED.
Explanation: 'A1' is the name of the entry causing overflow. A record or section in an area of the source program is very large. If a qualifying name falls within a different segment from the name being qualified, the name may be incorrectly defined.
Action:
1. Shorten record or section in which 'A1' is stated.
2. Restate qualification so that no references to 'A1' or any name defined below 'A1' depends on a qualifier of higher level than 'A1'.
3. Restate entry names for remainder of record, using only unique names.

2001 CSN (n) NAME EXCEEDS 30 CHARACTERS. NAME IS TRUNCATED.
Action: Shorten or correct name.

2002 CSN (n) SYMBOL NOT FOLLOWED BY BLANK. BLANK ASSUMED.
Action: Correct statement, if necessary.

2003 CSN (n) NON-NUMERIC CHARACTER IN NUMERIC LITERAL. INTERVENING BLANK ASSUMED.
Explanation: A sequence of numeric characters including a decimal point is terminated at the first non-numeric character after the decimal point. This error may result from failure to allow a space separation.

2004 CSN (n) NON-NUMERIC LITERAL CONTINUATION MUST BEGIN WITH A QUOTE. QUOTE ASSUMED PRECEDING FIRST NON-BLANK CHARACTER.

2005 CSN (n) NON-NUMERIC LITERAL DOES NOT TERMINATE WITH A QUOTE. ASSUME LITERAL TERMINATES IN CARD COLUMN 72.

2006 CSN (n) NON-NUMERIC LITERAL EXCEEDS 120 CHARACTERS. LITERAL TRUNCATED.

2007 CSN (n) PICTURE CLAUSE EXCEEDS 30 CHARACTERS. PICTURE CLAUSE IS TRUNCATED.

2008 CSN (n) INVALID USE OF 'A1'. 'A1' DELETED.
Explanation: 'A1' is either a plus (+) or a minus (−) sign immediately following a plus or minus sign.

2009 CSN (n) BLANK SHOULD SEPARATE 'A1' AND 'A2'. BLANK ASSUMED.
Explanation: 'A1' and 'A2' each may be any special character or name in COBOL.

2010 CSN (n) INVALID CHARACTER FOUND IN SOURCE STATEMENT. REPLACED BY SPACE.

2011 CSN (n) 'A1' OUT OF SEQUENCE. CONDITION IGNORED.
Explanation: 'A1' is the number of the card found out-of-sequence.

2012 'A1' SHOULD BE $CBEND. $CBEND ASSUMED PRECEDING THIS CARD.
Explanation: 'A1' is the contents of columns 1-6 of a system control card ($ in column 1).

2013 CSN (n) NUMERIC LITERAL EXCEEDS 18 DIGITS. LITERAL TRUNCATED.

2014 CSN (n) 'A1' SHOULD NOT BE IN MARGIN A. MARGIN B ASSUMED.
Explanation: 'A1' is a COBOL word or operator. Margin A is reserved for special headers, section and paragraph names.

2015 CSN (n) USE OF 'A1' IS INVALID IN THE 'A2' DIVISION. WORD IS DELETED.
Explanation: 'A1' is a COBOL word. 'A2' is the division in which 'A1' has been found. Each COBOL word is restricted to one or more divisions. The message indicates either incorrect structure or the accidental use of a COBOL word as a name.

2016 CSN (n) REDUNDANT 'A1' DIVISION CARD. CARD IGNORED.
Explanation: 'A1' is the name of the redundant division.

2017 CSN (n) ILLEGAL USE OF 'A1' IN 'A2' STATEMENT. 'A1' DELETED.
Explanation: 'A1' is an extraneous element. 'A2' is the COBOL word that begins the statement.

2018 CSN (n) 'A1' IS NOT A BEGINNING WORD OF A 'A2' ENTRY. ALL SYMBOLS DELETED UNTIL NEXT VALID ENTRY IS FOUND.
Explanation: 'A1' is a name, a COBOL word, or a symbol. 'A2' is the division name (Environment or Data). 'A1' does not begin any recognizable clause, statement, or header in the 'A2' division.

2019 CSN (n) 'A1' IS IN THE WRONG SECTION OR PARAGRAPH. ASSUMED CORRECT.
Explanation: 'A1' is an Environment Division COBOL word. This clause does not appear in the proper Environment Division section or paragraph.

2020 CSN (n) DUPLICATE 'A1' CLAUSE SPECIFIED. DUPLICATE CLAUSE DELETED.
Explanation: 'A1' is a key COBOL word that appears twice in either a Data Division entry or in the Environment Division.

2021 OBJECT OR SOURCE COMPUTER OMITTED. ASSUMED PRESENT.
Explanation: Assume IBM 7040 or IBM 7044 was specified.

2022 CSN (n) 'A1' CLAUSES RENAMING TABLE OVERFLOW. NO FURTHER RENAMING FILES ARE RECOGNIZED.
Explanation: 'A1' is the file-name causing table overflow. All file-names specified with RENAMING clauses from 'A1' on will not be recognized as legitimate file-name definitions, and references to these files will result in undefined references.

2023 CSN (n) 'A1' HAS MORE THAN 6 CHARACTERS. FIRST 6 CHARACTERS ARE USED.
Explanation: 'A1' is a BCD name that must conform to the IBMAP restrictions for name formation.

2024 CSN (n) FILE NAME FOLLOWING FD IS OMITTED. FD ENTRY IS DELETED.

2025 CSN (n) PICTURE CLAUSE INCOMPLETE. PICTURE DELETED.
Explanation: The syntax structure of a PICTURE clause is incorrect.

2026 CSN (n) ILLEGAL FORMAT IN PICTURE. ASSUME PICTURE IS 'A1'.
Explanation: 'A1' is assumed to be the encoded PICTURE.

2027 CSN (n) 'A1' USED AS A DEFINING ENTRY MUST NOT BE SUBSCRIPTED OR QUALIFIED. QUALIFICATION OR SUBSCRIPTS DELETED.
Explanation: 'A1' is a name that appears as a defining entry in the Data or the Procedure Division.

2028 CSN (n) LEVEL NUMBER IS OMITTED OR INCORRECTLY STATED. ASSUME LEVEL NUMBER IS 49.
Explanation: A Data Division entry starts without a level number. A period may have been incorrectly positioned within the preceding entry.

2029 CSN (n) PERIOD OMITTED. ASSUME 'A1' 'A2' BEGINS NEW ENTRY.
Explanation: 'A1' is considered to be the level number. 'A2' is considered to be the data-name.

2030 CSN (n) 'A1' NOT FOLLOWED BY DATA NAME. ASSUME DATA NAME IS FILLER.
Explanation: 'A1' is the level number stated in the source program.

2031 CSN (n) 'A1' IS NOT DEFINED.
Explanation: 'A1' is a non-qualified data or procedure name. 'A1' is used, but no paragraph, section, or data-name definition exists in the source program.

2032 CSN (n) 'A1' IMPROPERLY QUALIFIED.
Explanation: 'A1' is a name that has one or more qualifiers. The definition of one or more of the qualifiers of 'A1' does not include 'A1'.

2033 SN (n) 'A1' CANNOT HAVE MORE THAN 49 QUALIFIERS. FIRST 49 USED.
Explanation: 'A1' is a lowest level data or procedure name at the point of usage.

2034 CSN (n) HIGHEST LEVEL QUALIFIER 'A1' IS NOT UNIQUELY DEFINED. 'A2' MAY NOT BE UNIQUE.
Explanation: 'A1' is the highest level qualifier of a name. 'A2' is the name that is being qualified. Duplicate definitions of 'A1' exist. If 'A2' appears within more than one definition of 'A1', an error condition exists.

2035 CSN (n) 'A1' IS NOT UNIQUE. FIRST DEFINITION USED.
Explanation: 'A1' is a non-qualified data or procedure name that has been defined twice.

2036 CSN (n) 'A1' IS NOT A RECOGNIZED IBMAP OP CODE. CARD DELETED.
Explanation: 'A1' appears following an ENTER ASSEMBLY-PROGRAM statement and is not one of the following: ENTRY, EXTERN, CALL, SAVE, RETURN, ETC.

2037 CSN (n) IMPROPER FORMAT FOR 'A1' STATEMENT. CARD DELETED.
Explanation: 'A1' appears following an ENTER ASSEMBLY-PROGRAM statement and is one of the following MAP operation codes: ENTRY, EXTERN, CALL, SAVE, RETURN.

2038 CSN (n) 'A1' IS NOT A PROPERLY FORMED IBMAP NAME. CARD DELETED.
Explanation: 'A1' is the label of an IBMAP statement or the operand of an ENTRY statement.

2039 CSN (n) 'A1' IS NOT A NUMERIC ELEMENT IN THE VARIABLE FIELD OF 'A2' STATEMENT. ELEMENT AND REMAINDER OF CARD IGNORED.
Explanation: 'A1' is the non-numeric element. 'A2' is a SAVE or RETURN MAP operation code.

2040 CSN (n) PARAGRAPH NAME DOES NOT PRECEDE 'A1' STATEMENT. PARAGRAPH NAME ASSIGNED.
Explanation: 'A1' is the first word of the statement. The paragraph name FILLER has been generated for a statement that required a paragraph name.

2041 CSN (n) COBOL WORD APPEARS IN ASSEMBLY-PROGRAM PORTION. ASSUME ENTER COBOL WAS STATED.
Explanation: An ENTER COBOL has been assumed and FILLER has been generated as a paragraph name.

2042 CSN (n) ENTER 'A1' NOT A VALID ENTER OPTION. ASSUME ENTER ASSEMBLY-PROGRAM WAS INTENDED.
Explanation: 'A1' is the operand of the ENTER verb.

2043 CSN (n) 'A1' RENAMING 'A2' IS INVALID. RENAMING IGNORED.
Explanation: 'A1' is a file-name that follows a SELECT in the Environment Division. 'A2' is a file-name that follows the RENAMING entry corresponding to the above SELECT.
1. A file that is renamed may not itself rename another file, or
2. 'A1' renaming 'A2' has been previously stated.

2044 CSN (n) DIVISIONS ARE INCORRECTLY ORDERED. 'A1' DIVISION DELETED.
Explanation: 'A1' is the division header that is out of order.

2045 'A1' IS AN INVALID ENTRY ON $IBCBC CARD. CONDITION IGNORED.
Explanation: 'A1' is the invalid option or symbol specified on the $IBCBC card.

2046 DECK NAME IS OMITTED FROM $IBCBC CARD. CONDITION IGNORED.
Explanation: The characters CBC will appear as the first three characters of file-names and CONTROL names generated by the Compiler.

2047 CSN (n) ETC CARD DOES NOT FOLLOW CALL. CARD DELETED.

2048 DECK SPECIFIED ON $IBCBC CARD BUT NOT ON $IBJOB CARD. NO DECK TAKEN.

2049 CSN (n) 'A1' OPTION HAS BEEN OMITTED. ASSUMED PRESENT.
Explanation: 'A1' is a required option.

2050 CSN (n) 'A1' IS NOT LEGAL IN 'A2' STATEMENT. CONDITION IGNORED.
Explanation: 'A1' is an extraneous symbol in an 'A2' statement. 'A2' is the beginning word of the statement.

2051 FIRST CARD AFTER $IBCBC IS NOT A DIVISION HEADER. ALL CARDS IGNORED UNTIL DIVISION HEADER IS ENCOUNTERED.

2052 CSN(n) ALPHANUMERIC LITERAL CONTAINS NO CHARACTERS. LITERAL ASSUMED TO BE SINGLE BLANK CHARACTER.

2053 CSN (n) NUMERIC LITERAL USED AS ARGUMENT IN ASSEMBLY-PROGRAM CALL STATEMENT EXCEEDS 10 DIGITS. LITERAL TRUNCATED.

2054 CSN (n) NON-NUMERIC LITERAL USED AS ARGUMENT IN ASSEMBLY-PROGRAM CALL STATEMENT EXCEEDS 6 CHARACTERS. LITERAL TRUNCATED.

2055 CSN (n) WARNING. NO CORRESPONDING SUBFIELDS.

2056 CSN (n) SYNTAX ERROR IN CORRESPONDING STATEMENT. STATEMENT DELETED.

2100 CSN (n) 'A1' IS NOT A BEGINNING WORD OF AN 'A2' ENTRY. ALL SYMBOLS DELETED UNTIL NEXT VALID ENTRY IS FOUND.
Explanation: 'A1' is a COBOL word. 'A2' is a division name (Data or Procedure). No recognizable COBOL word has been found as the beginning of a clause in the Data or Procedure Division.

2101 CSN (n) ILLEGAL USE OF 'A1' IN 'A2' CLAUSE. CLAUSE DELETED.
Explanation: 'A1' is a COBOL word. 'A2' is VALUE or APPLY. No legitimate option is indicated following the APPLY clause in the Environment Division or the VALUE OF clause in an FD entry.

2102 CSN (n) 'A1' OMITTED IN 'A2' STATEMENT. 'A1' ASSUMED.
Explanation: 'A1' is a required word in the 'A2' option. 'A2' is the beginning word of an option.

2103 CSN (n) ILLEGAL USE OF 'A1' IN 'A2' STATEMENT. 'A3' DELETED.
Explanation: 'A1' is an item that was improperly stated in an 'A2' clause. 'A2' is the beginning word of an option. 'A3' is the portion of the option that is deleted from the statement.

2104 CSN (n) ILLEGAL USE OF 'A1' IN 'A2' STATEMENT.
Explanation: 'A1' is an item that is improperly stated in an 'A2'. 'A2' is the beginning word of an option.

2105 CSN (n) EXTRANEOUS SYMBOL 'A1' FOLLOWING 'A2' STATEMENT. ALL SYMBOLS DELETED UNTIL NEXT VALID ENTRY IS FOUND.
Explanation: 'A1' is an extraneous symbol appearing after a complete clause. 'A2' is the beginning word of the clause.

2106 CSN (n) OPERAND OF 'A1' STATEMENT OMITTED.
Explanation: 'A1' is the beginning word of an incomplete statement.

2107 CSN (n) 'A1' STATEMENT INCOMPLETE. STATEMENT DELETED.
Explanation: 'A1' is RERUN or RECORD. Either no file assignment follows a RERUN clause, or the integer specifying the length of the RECORD CONTAINS clause has been omitted.

2108 CSN (n) EXTRANEOUS 'A1' PARENTHESIS. 'A2' PARENTHESIS 'A3'.
Explanation: 'A1' is RIGHT or LEFT. 'A2' is LEFT or RIGHT. 'A3' is 'INSERTED' or 'DELETED'.

2109 CSN (n) 'A1' DOES NOT BEGIN A SENTENCE. ALL SYMBOLS DELETED UNTIL NEXT VALID ENTRY IS FOUND.
Explanation: 'A1' is not a COBOL word.

2110 CSN (n) 'A1' FOLLOWED BY 'A2' IMPLIES END OF SENTENCE. PERIOD ASSUMED.
Explanation: The Compiler has detected a condition that implied the end of a sentence, but no period has been found.

2111 CSN (n) NON-NUMERIC LITERAL FOLLOWING 'ALL' GREATER THAN ONE CHARACTER. TRUNCATED TO ONE CHARACTER.

2112 CSN (n) SUBSCRIPT COUNT EXCEEDS THREE. RIGHT PARENTHESIS ASSUMED BEFORE 'A1'.

2113 CSN (n) SUBSCRIPT MISSING AFTER LEFT PARENTHESIS. INTEGER 1 ASSUMED FOR SUBSCRIPT.

2114 CSN (n) ILLEGAL USE OF 'A1' AS A SUBSCRIPT. INTEGER 1 ASSUMED FOR SUBSCRIPT.
Explanation: 'A1' is not a data-name or a numeric literal.

2115 CSN (n) ILLEGAL USE OF 'A1' AS A SUBSCRIPT. 'A1' DELETED.
Explanation: 'A1' is an alphanumeric literal.

2116 CSN (n) ILLEGAL FORMAT IN 'A1' STATEMENT. 'A2' DELETED.
Explanation: 'A1' is IF, COMPUTE, or PERFORM. 'A2' is not a legal symbol.

2117 CSN (n) 'A1' STATEMENT CAUSES OVERFLOW. 'A2' LEVELS OF IMBEDDED PARENTHESES ASSUMED.
Explanation 'A1' is PERFORM, IF, or COMPUTE. 'A2' is the maximum number of levels of parentheses that can be accommodated. Provision is made for 'A2' levels of parentheses within a formula. When this number is exceeded, 'A2' levels are assumed until the number of levels falls below 'A2'; i.e., pairs of left and right parentheses are ignored until the parenthetical level falls below 'A2'.

2118 CSN (n) 'A1' AND 'A2' IS AN ILLEGAL COMBINATION IN A 'A3' STATEMENT.

2119 DESCRIPTION OF FILE 'A1' INCOMPLETE. 'A2' STATEMENT OMITTED OR IMPROPER.
Explanation: 'A1' is a file-name. 'A2' is one of the following:
CLOSE. A CLOSE statement was not specified for the file.
ASSIGN. A unit was not assigned for the file.
VALUE. A VALUE clause does not exist for a file with standard labels.
DATA. A DATA RECORDS clause has not been specified for the file.

MULTIPLE. More than one unit has been specified, but MULTIPLE REEL was not specified.

2120 DESCRIPTION OF FILE 'A1' INCOMPLETE. 'A2' STATEMENT OMITTED. 'A3' ASSUMED.
Explanation: 'A1' is a file-name.
If 'A2' is CONTAINS, 'A3' is SIZE.
A RECORD CONTAINS has not been specified for the file. A size of 18 is assumed.
If 'A2' is LABEL, 'A3' is STANDARD or OMITTED.
a. The LABEL RECORDS clause is omitted. Assume STANDARD LABEL, since VALUE clauses are present.
b. LABEL RECORDS clause is omitted and a VALUE clause is not present. Assume LABEL RECORDS is omitted.
If 'A2' is SELECT, 'A3' is SELECT.
If 'A2' is MULTIPLE, 'A3' is MULTIPLE.

2121 'A1' MAY NOT BE SPECIFIED FOR FILE 'A2' DESIGNATED AS 'A3'. CLAUSE IGNORED.
Explanation: 'A2' is a file-name.
If 'A1' is RERUN, 'A3' is OUTPUT.
An APPLY RERUN-RECORDS clause has been specified for an output file. This clause applies only to input files and is ignored for output.
If 'A1' is READ, 'A3' is OUTPUT.
A READ statement has been specified for an output file. This is an invalid operation and the READ is ignored.
If 'A1' is CHECKPOINT-UNIT, 'A3' is INPUT.
The ON CHECKPOINT-UNIT specification is missing for the RERUN clause on an input file.
If 'A1' is LABEL, 'A3' is SYSTEM UNIT.
LABEL RECORDS may not be specified for a system unit.
If 'A1' is INPUT, 'A3' is OUTPUT.
A file assigned to S.SOU or to S.SPP is specified as OPEN INPUT.
If 'A1' is OUTPUT, 'A3' is INPUT.
A file assigned to S.SIN is specified as OPEN OUTPUT.
If 'A1' is BINARY, 'A3' is OUTPUT.
A file assigned to S.SOU is specified as RECORDING MODE IS BINARY.

2122 NO READ STATEMENT SPECIFIED FOR FILE 'A1' DESIGNATED AS INPUT. CONDITION IGNORED.
Explanation: 'A1' is a file-name.

2124 'A1' AND 'A2' SPECIFIED FOR 'A3' IS AN ILLEGAL COMBINATION OF UNIT ASSIGNMENT. 'A2' DELETED.
Explanation: 'A1' is unit name 1. 'A2' is unit name 2. 'A3' is a file-name. Unit 1 and unit 2 refer to invalid combinations of utility units. system units, and/or no units.

2125 OPERAND OF 'A1' CLAUSE SPECIFIED FOR FILE 'A2' IS ILLEGAL. 'A3' IS ASSUMED.
Explanation: 'A1' is RECORD. 'A2' is a file-name. 'A3' is the number of machine bytes.
1. Record length for a file on S.SIN,S.SOU, or S.SPP is not valid. Lengths of 28 words for S.SIN and S.SPP and of 22 words for S.SOU are assumed.
2. Record length is less than 18 characters. A record length of 18 characters is assumed.

2126 'A1' SPECIFIED FOR FILE 'A2' IS AN ILLEGAL UNIT ASSIGNMENT. 'A3' ASSUMED.
Explanation: 'A1' is unit designation. 'A2' is a file-name. 'A3' is NONE.

2127 DUPLICATELY DEFINED 'A1' CLAUSES FOR 'A2'. CLAUSE DELETED.
Explanation: If 'A1' is CONTROL, 'A2' is a file-name, a data-name, or a procedure name.
A duplicate CONTROL clause has been specified for a file-name, a data-name, or a procedure name.
If 'A1' is FILE-REFERENCE, 'A2' is a file-name.
A duplicate FILE-REFERENCE clause has been specified for a file.

2129 CSN (n) USE OF 'A1' IN 'A2' STATEMENT ILLEGAL. STATEMENT DELETED.
Explanation: If 'A1' is a COBOL word, 'A2' is VALUE or LABEL.
a. The literal that should be specified in the VALUE clause has been omitted or is unrecognizable.
b. The LABEL RECORDS clause is incorrect.

If 'A1' is an external name, 'A2' is a MAP operation code.
An invalid duplication of an external six-character name appears in a MAP statement.
If 'A1' is a COBOL name, 'A2' is ENTRY.
A procedure name is not specified as an entry.
If 'A1' is a MAP label, 'A2' is RETURN.
A label in MAP is not specified as the operand of a RETURN.

2130 END DECLARATIVES STATEMENT MISSING. CONDITION IGNORED.
Explanation: PROGRAM-START has been specified in the Environment Division.

2131 END DECLARATIVES STATEMENT MISSING. NO PROGRAM START CAN BE INFERRED.

2132 CSN (n) RECORD NAME 'A1' IS NOT NAMED IN DATA RECORDS CLAUSE. CONDITION IGNORED.
Explanation: 'A1' is the data-name of an 01 entry, following an FD entry, that does not correspond to any of the record names specified in the DATA RECORDS clause. It is assumed that 'A1' was named in the DATA RECORDS clause.

2133 CSN (n) OPERAND OF 'A1' CLAUSE SPECIFIED FOR FILE 'A2' IS ILLEGAL. 'A3' ASSUMED.
Explanation: 'A1' is the name of an improperly used clause. 'A2' is a file-name.

2134 CSN (n) DUPLICATELY DEFINED 'A1' CLAUSES FOR 'A2'. CLAUSE DELETED.
Explanation: See explanation for message 2127.

2135 CSN (n) DESCRIPTION OF FILE 'A1' INCOMPLETE. 'A2' STATEMENT OMITTED. 'A3' ASSUMED.
Explanation: See explanation for message 2120.

2136 CSN (n) 'A1' NOT FOLLOWED BY PARAGRAPH NAME. CONDITION IGNORED.
Explanation: 'A1' is a section name. A paragraph name does not immediately follow 'A1'.

2137 PROCEDURE DIVISION DOES NOT START WITH SECTION OR PARAGRAPH NAME. NAME PROVIDED.

2140 CSN (n) 'A1' OMITTED for 'A2' 'A3'. 'A4' DELETED.
Explanation: If 'A1' is a procedure name, 'A2' is USE, 'A3' is statement, and 'A4' is statement.
If 'A1' is CHECKPOINT-UNIT, 'A2' is INPUT, 'A3' is FILES, and 'A4' is RERUN clause *or* 'A2' is ALL, 'A3' is FILES, and 'A4' is INPUT RERUN.
If 'A1' is SELECT, 'A2' is ASSIGN, 'A3' is clause, and 'A4' is clause.
If 'A1' is a file-name, 'A2' is the beginning words for FD entry clauses, 'A3' is clause, and 'A4' is clause.

2141 CSN(n) COMPILER OVERFLOW CAUSED BY LENGTH OF A 'A1' STATEMENT.
Explanation: 'A1' is the COBOL word OPEN, GO, COMPUTE, IF or DATA RECORDS.
1. A single OPEN statement may not contain more than 20 file-names.
2. A single GO TO . . . DEPENDING ON statement may not contain more than 95 procedure-names.
3. The number of logical operators (AND, OR) in a single IF statement or in the UNTIL portion of a PERFORM statement may not exceed 99.
4. Error checking of the consistency between the names in the DATA-RECORDS clause and in the record descriptions under an FD entry takes place only if there are 50 or fewer data records for that FD entry.
5. Within the COMPUTE, IF, and the UNTIL portion of a PERFORM statement, 360 is the maximum number of elements that can be passed over before an operation within a formula can be evaluated. Due to the hierarchy of operations involved, the pattern in the first formula below would cause overflow at the 51st level of parenthesization and the pattern in the second formula would cause overflow at the 180th level:

A B*C**(A$_1$B*C$_1$**(...
A (A$_1$ (A$_2$ (A$_3$ (...

When the number of elements exceeds 360, all subsequent elements will be deleted until any of the following are found:

a. The end of the statement
b. The words AND or OR, within the IF and the UNTIL portion of a PERFORM statement
c. The word AFTER, within the UNTIL portion of a PERFORM statement

NOTE: Within the IF and the UNTIL portion of the PERFORM statement, the levels of parenthesization are limited to 50. Within the COMPUTE statement there is no limit on parentheses.

6. Within a formula to be evaluated by the COMPUTE, IF, and the UNTIL portion of a PERFORM statement, an additional limitation exists. If the elements are to be passed over, as explained under Limitation 5, include literals or subscripted variables, overflow occurs when:

$$\sum_{n=1}^{n=A} (2+{}^{Cn}/_{6})+2N+NN+3S+D>900$$

where A = the number of alphanumeric literals
C = the number of characters in each alphanumeric literal
N = the number of numeric literals, single or double-precision, used as subscripts or not
NN = the number of double-precision numeric literals
S = the number of subscripted variables.
D = the number of data-names used as subscripts

When such overflow occurs, subsequent literals and subscripted variables will be deleted. Normal processing will be resumed when the hierarchy of operation allows some portion of the formula containing subscripts or literals to be evaluated.

2142 CSN (n) S.SIN IS SPECIFIED BOTH IN AN ACCEPT STATEMENT AND AN ASSIGN CLAUSE.

2201 CSN (n) FIRST LEVEL NUMBER IS NOT AN 01 OR 77 ENTRY. 01 ASSUMED.
Explanation:
1. An acceptable level number does not follow section header or FD entry.
2. A 77 entry is not followed by another 77 entry or an 01.
The level number is changed to 01.

2202 CSN (n) SECTION HEADER MISSING. WORKING-STORAGE ASSUMED.
Explanation: Section name does not immediately follow section header. Working-storage section is assumed.

2203 CSN (n) LEVEL 77 ENTRY APPEARS OTHER THAN AT THE BEGINNING OF A WORKING-STORAGE OR CONSTANT SECTION. ASSUMED LEVEL 'A1'.
Explanation: 'A1' is the level number that is assumed.
1. If the level number appears in a file section following an FD entry, it is changed to 01.
2. If the level number appears anywhere other than in a file section following an FD entry, it is considered an elementary item of the current record.

2204 CSN (n) OCCURS CLAUSE ON LEVEL 77 ENTRY. OCCURS CLAUSE DELETED.

2205 CSN (n) REDEFINES CLAUSE FOR LEVEL 77 ENTRY IN CONSTANT SECTION. REDEFINES CLAUSE DELETED.

2206 CSN (n) CONDITION NAME IN CONSTANT SECTION. SITUATION IGNORED.

2208 CSN (n) INCORRECT USE OF REDEFINES. REDEFINES DELETED.
Explanation: Redefined data-name is not acceptable.

2209 CSN (n) SIZES OF REDEFINED AND REDEFINING AREAS ARE NOT EQUAL. REDEFINING SIZE USED.
Explanation: For other than level 01 definition, sizes of redefined and redefining areas do not agree. Redefining area size used.

2210 CSN (n) LEVEL 01 REDEFINES CLAUSE IN FILE SECTION. REDEFINES IGNORED.

2211 CSN (n) SIZE CLAUSE AND PICTURE DISAGREE. SIZE CLAUSE IGNORED.

2212 CSN (n) SIZE SPECIFIED AS GREATER THAN 32,767 CHARACTERS. SIZE ASSUMED TO BE 1 WORD.

2213 CSN (n) SUBFIELD USAGE DISAGREES WITH GROUP USAGE. GROUP USAGE TAKES PRECEDENCE.

2214 CSN (n) USAGE AND CLASS DISAGREE. ASSUME USAGE IS DISPLAY.
Explanation: Class and usage on same level do not agree. Usage is changed to conform to class.

2215 CSN (n) INVALID APPEARANCE OF BEGINNING-LABEL OR ENDING-LABEL. NOT TREATED AS LABEL DESCRIPTION.
Explanation: BEGINNING-LABEL and ENDING-LABEL can be used as data-names only on level 01 entries following an FD entry.

2216 CSN (n) OCCURS CLAUSE INTEGER IS NOT A VALUE BETWEEN 1 and 32,767. INTEGER ASSUMED TO BE 1.

2217 CSN (n) MORE THAN THREE LEVELS OF OCCURS CLAUSES IN HIERARCHY. EXTRA OCCURS CLAUSE DELETED.

2218 CSN (n) OCCURS CLAUSE ON LEVEL 01. OCCURS DELETED.

2219 CSN (n) INCORRECT VARIABLE FOR OCCURS DEPENDING OPTION. DEPENDING OPTION DELETED.
Explanation: Variable for OCCURS DEPENDING option is not a proper data-name.

2220 CSN (n) SIGNED CLAUSE IS STATED FOR NON-NUMERIC ITEM. SIGNED CLAUSE DELETED.

2221 CSN (n) POINT LOCATION CLAUSE IS STATED FOR NON-NUMERIC ITEM. POINT LOCATION CLAUSE DELETED.

2222 CSN (n) 'A1' CLAUSE INVALID ON GROUP LEVEL. CLAUSE DELETED.
Explanation: 'A1' is the first word (SIGNED, SYNCHRONIZED, POINT LOCATION, PICTURE, BLANK WHEN ZERO) of a clause that may not be stated for a group-level item.

2223 CSN (n) INCORRECT OPTION ON SYNCHRONIZED CLAUSE. ASSUME SYNCHRONIZED RIGHT.
Explanation: SYNHRONIZED is not followed by LEFT or RIGHT. RIGHT is assumed.

2224 CSN (n) POINT LOCATION CLAUSE INCORRECTLY USED. CLAUSE DELETED.
Explanation: INTEGER, LEFT, or RIGHT is missing from the POINT LOCATION clause. The POINT LOCATION clause is ignored.

2225 CSN (n) REDUNDANT POINT LOCATION CLAUSE (PICTURE IS GIVEN). CLAUSE IGNORED.
Explanation: PICTURE and POINT LOCATION clauses were given for the same entry. POINT LOCATION clause is ignored.

2226 CSN (n) CLASS DISAGREES WITH CLASS ON GROUP LEVEL. CLASS FROM GROUP LEVEL TAKES PRECEDENCE.

2227 CSN (n) OPERAND OF 'A1' CLAUSE OMITTED. CLAUSE DELETED.
Explanation: 'A1' is the first word of a Data Division clause that is improperly formed. CLASS is omitted.

2228 CSN (n) INVALID PICTURE CONFIGURATION. ASSUME PICTURE IS 'A1'.
Explanation: 'A1' is the assumed PICTURE. The inconsistent or invalid characters are changed to 9, A, or X, depending on CLASS.

2229 CSN (n) PICTURE AND CLASS DISAGREE. ASSUME PICTURE IS 'A1'.
Explanation: 'A1' is the assumed PICTURE.

2230 CSN(n) INVALID TEXT IN 'A1' CLAUSE. TEXT DELETED UNTIL NEXT VALID ENTRY.
Explanation: 'A1' is the first word of a clause that is improperly formed.

2231 CSN (n) PICTURE IS MISSING A NUMERIC CHARACTER POSITION. ASSUME PICTURE IS 9.

2232 CSN (n) BLANK WHEN ZERO IMPROPERLY USED. CLAUSE DELETED.
Explanation: BLANK WHEN ZERO clause is given when a report field cannot properly result from the description of the data entry.

2233 CSN (n) 'ZERO' DOES NOT FOLLOW 'BLANK'. BLANK WHEN ZERO ASSUMED.

2234 CSN (n) CONDITIONAL VARIABLE ON GROUP LEVEL. LEVEL 'A1' ASSUMED FOR DATA NAME FOLLOWING THE CONDITION NAMES.
Explanation: Conditional variable is not an elementary item. Conditional variable is changed to an elementary item by changing the level number of the first non-conditional name to that of the conditional variable.

2235 CSN (n) LEVEL 88 HAS EXTRANEOUS CLAUSES. EXTRANEOUS CLAUSES DELETED.

2236 CSN (n) 'A1' SECTION HEADER OUT OF ORDER. SECTION HEADER IGNORED.
Explanation: Section header is out of place. Section header, other than FILE SECTION, is processed correctly; FILE SECTION header is ignored.

2237 CSN (n) WORD 'SECTION' MISSING FROM SECTION HEADER. ASSUMED PRESENT.

2238 CSN (n) SIZE OF NUMERIC FIELD EXCEEDS 18 DIGITS. SIZE REDUCED TO 18 DIGITS.
Explanation: The field is truncated.

2239 CSN (n) SIZE AND PICTURE MISSING FROM ELEMENTARY ITEM. ASSUME ONE WORD FOR ITEM.

2240 CSN (n) FD ENTRY DOES NOT FOLLOW FILE SECTION HEADER. WORKING-STORAGE SECTION IS ASSUMED.

2241 CSN (n) ILLEGAL USE OF VALUE CLAUSE. VALUE CLAUSE DELETED.

2242 CSN (n) DEPENDING ON VARIABLE FIELD IMPROPERLY PLACED. VARIABLE FIELD ASSUMED TO BE TALLY.
Explanation: Data-name specified as option does not appear in the proper place.

2243 CSN (n) NON-ALPHABETIC CHARACTERS IN VALUE CLAUSE FOR ALPHABETIC FIELD. ASSUME VALUE IS SPACE.

2244 CSN (n) NON-ZERO NUMBERS IN VALUE CLAUSE FOR SCALED POSITIONS. ASSUME VALUE IS ZEROS.

2245 CSN (n) MINUS SIGN IN VALUE CLAUSE FOR UNSIGNED FIELD. SIGN DELETED.

2246 CSN (n) RECORD NAME FOR WRITE IS NOT PART OF AN OUTPUT FILE.

2247 CSN (n) ILLEGAL TYPE 2 FILE DESCRIPTION. ALL RECORDS BUT FIRST IGNORED FOR FILE PROCESSING.
Explanation: There is more than one record description for a file, the first of which contains at least one OCCURS . . . DEPENDING ON clause.

2248 CSN (n) FD ENTRY DOES NOT CONTAIN ANY RECORD DESCRIPTION. CONDITION IGNORED.

2249 CSN (n) COMPUTATIONAL FIELD IN RECORD FOR BCD FILE. CONDITION IGNORED.

2250 CSN (n) OCCURS DEPENDING ON CLAUSE APPEARS IN A RECORD WHICH IS NOT FIRST FOR FILE. DEPENDING ON PART OF CLAUSE IS DELETED.

2251 CSN (n) SIZE DEDUCED FROM RECORD DESCRIPTION IS GREATER THAN THAT SPECIFIED

IN FD ENTRY. SIZE SPECIFIED IN FD ENTRY IS USED FOR DETERMINING FILE CHARACTERISTICS.

2252 CSN (n) RECORD TYPE SPECIFIED IN FD ENTRY DOES NOT AGREE WITH RECORD DESCRIPTION. TYPE SPECIFIED IN FD ENTRY IS ASSUMED.
Explanation: The RECORD CONTAINS clause is improper.

2253 CSN (n) RECORD FOR FILE ON IN, OU OR PP CONTAINS OCCURS DEPENDING CLAUSE. DEPENDING OPTION DELETED.

2254 CSN (n) CONDITION NAME DOES NOT HAVE A VALUE CLAUSE. VALUE IS CONSIDERED AS BLANKS OR ZEROS DEPENDING ON CLASS.

2255 'A1' IS NOT AN 01 LEVEL, BUT IS SPECIFIED IN A CONTROL STATEMENT. CONTROL STATEMENT DELETED.

2256 CSN (n) 'A1' IS A REDEFINING RECORD BUT IS SPECIFIED IN A CONTROL STATEMENT. CONDITION IGNORED.

2257 CSN (n) 'A1' IS IN THE FILE SECTION, BUT IS SPECIFIED IN A CONTROL STATEMENT. CONTROL STATEMENT DELETED.
Explanation: 'A1' is a record description entry.

2258 CSN (n) RECORD IN THE CONSTANT SECTION CONTAINS NO CONSTANTS. CONDITION IGNORED.
Explanation: None of the entries for a complete 01 record description in the constant section contains a VALUE clause.

2259 CSN (n) CONSTANT-SECTION RECORD ENTRY WITHOUT A REDEFINES CLAUSE HAS NO VALUE CLAUSE. BLANKS OR ZEROS ARE STORED ACCORDING TO CLASS.
Explanation: An entry within a record description in the constant section does not have a VALUE clause associated with it. (If none of these entries has associated VALUE clauses, only message 2258 will appear.)

2260 CSN (n) A VALUE CLAUSE IS STATED FOR A REPORT FIELD ENTRY. VALUE CLAUSE IS IGNORED.

2261 CSN (n) VALUE CLAUSE IS STATED FOR A FIELD UNDER AN OCCURS CLAUSE. VALUE CLAUSE IS IGNORED.

2262 CSN (n) FIGURATIVE CONSTANT DOES NOT AGREE WITH CLASS OF FIELD. BLANKS OR ZEROS ARE STORED ACCORDING TO CLASS.

2263 CSN (n) CLASS OF VALUE CLAUSE CONTRADICTS CLASS OF DATA ITEM. BLANKS OR ZEROS STORED ACCORDING TO CLASS OF DATA ITEM.

2264 CSN (n) INTEGER OR DECIMAL COUNT OF VALUE CLAUSE EXCEEDS ASSOCIATED COUNT IN PICTURE. ASSUME VALUE IS ZERO.

2265 CSN (n) SIZE OF ALPHANUMERIC LITERAL IN THE VALUE CLAUSE EXCEEDS SIZE OF FIELD. LITERAL TRUNCATED.

2266 CSN (n) ALL IN VALUE CLAUSE IS FOLLOWED BY A FIGURATIVE CONSTANT. CONDITION IGNORED.
Explanation: The figurative constant is stored.

2267 CSN (n) RECORD MARK IS SPECIFIED FOR A COMPUTATIONAL FIELD. ASSUME VALUE IS ZERO.

2268 CSN (n) RECORD MARK IS ASSOCIATED WITH FIELD WHICH IS NOT SPECIFIED AS ALPHANUMERIC. ASSUME VALUE IS SPACE.

2269 CSN (n) RECORD ASSOCIATED WITH SYSTEMS UNIT IS NOT OF SPECIFIED LENGTH. CONDITION IGNORED.

2270 CSN (n) ALL IN VALUE CLAUSE IS FOLLOWED BY NUMERIC LITERAL. BLANKS OR ZEROS ARE STORED ACCORDING TO CLASS.

2271 CSN (n) ALL IS FOLLOWED BY MORE THAN ONE CHARACTER. BLANKS OR ZEROS ARE STORED ACCORDING TO CLASS.

2272 CSN (n) FILE NAME IN WORKING-STORAGE OR CONSTANT SECTION. CONDITION IGNORED.

2273 CSN(n) OPERAND IN 'A1' CLAUSE IS NOT A POSITIVE INTEGER. CLAUSE DELETED.
Explanation: 'A1' is SIZE, POINT LOCATION, or OCCURS.

2274 CSN(n) SUM OF SIZES OF ELEMENTARY ITEMS IS GREATER THAN 32K BYTES. SIZE ASSUMED IS MODULO 32K.

2275 CSN(n) 'INTEGER 1 TO' PHRASE WITHIN OCCURS CLAUSE IS INCORRECT AND IS BEING IGNORED.

2276 CSN (n) OCCURS DEPENDING ON CLAUSE APPEARS IN A RECORD ASSOCIATED WITH UNIT RECORD EQUIPMENT. DEPENDING ON PART OF CLAUSE DELETED.

2277 CSN (n) RECORD ASSOCIATED WITH UNIT RECORD EQUIPMENT IS NOT OF SPECIFIED LENGTH. CONDITION IGNORED.

2300 CSN (n) 'A1' MAY NOT BE SUBSCRIPTED. SUBSCRIPT DELETED.
Explanation: 'A1' is a data-name that does not have an associated OCCURS clause.

2301 CSN (n) TOO MANY LEVELS OF SUBSCRIPTS FOR 'A1'. EXTRANEOUS SUBSCRIPTS IGNORED.
Explanation: 'A1' is a data-name.

2302 CSN (n) INSUFFICIENT NUMBER OF SUBSCRIPTS FOR 'A1'. ASSUME INTEGER '1' FOR MISSING SUBSCRIPTS.
Explanation: 'A1' is a data-name.

2303 CSN (n) USE OF 'A1' IS IMPROPER AS A SUBSCRIPT. ASSUME SUBSCRIPT VALUE IS '1'.
Explanation: 'A1' is a subscript name that is not an integral numerical elementary field.

2304 CSN (n) 'A1' MAY NOT BE USED AS AN OPERAND OF 'A2.'
Explanation: 'A1' is a data-name.
'A2' is a COBOL verb.

2305 CNS(n) 'A1' and 'A2' ARE IMPROPER PAIR FOR 'A3.'
Explanation: 'A1' is an operand.
'A2' is an operand.
'A3' is a COBOL verb.
NOTE: The statement 'IF DATA-NAME IS {POSITIVE / NEGATIVE}' is converted to 'IF DATA-NAME IS {GREATER THAN ZERO / LESS THAN ZERO}'.
Therefore, whenever the sign test is not permitted for the data-name specified, 'FIG CONSTANT (ZR)' will be substituted for parameter 'A2.'

2306 CSN (n) LOW-VALUE OR SPACES BEING MOVED TO NUMERIC FIELD. CONDITION IGNORED.
Explanation: LOW-VALUE or SPACES is invalid in a numerical field, but the move is done.

Glossary

The definitions in this glossary apply to these terms as they are used in all 7040/7044 Operating System publications. The reader may also refer to the IBM Reference Manual *Glossary for Information Processing*, Form C20-8089.

ABSMOD deck
: A deck with an absolute load location and absolute length (produced by specifying ABSMOD on the $IBMAP card). The actual origins are determined by the user.

absolute address
: The label or number permanently assigned to a specific storage location, device or register.

absolute origin
: An origin that is a specific machine location. It is usually assigned by the user.

absolute program
: A machine-language program in a format ready for loading directly into a specific area of core storage.

activity
: Use of an input/output device that makes both device and channel busy.

alternate unit
: A symbolic unit that is substituted for another symbolic unit as a result of programmer direction.

application
: Any single use of a subsystem or user's program. A job segment.

argument
: An independent variable.

assembler
: See assembly program.

assembly program
: The program that interprets a symbolic source program and produces from it a machine-language object program.

assigned symbol
: A symbol given a specific value by the programmer, using a symbol definition pseudo-operation.

block
: A group of data records, words, or characters. The size of a block may be limited by the amount of core storage available for buffers or by some inherent characteristic of a particular input/output device.

block length
: The total number of words or characters in one block.

blocking
: Records are blocked, or grouped together in a buffer, in order to increase the average length of the physical records being written, thus reducing the process time per record, and increasing the total number of records that can be written on one unit.

buffer
: An area of core storage that is used to temporarily store information during a transfer of information to or from an input/output device.

buffer control word
: The first word preceding a buffer. It is used to keep a record of the status of information within the buffer.

buffering system
: A set of routines to block and deblock data records and to perform buffer switching operations.

calling sequence
: The instructions and data that establish the linkage to a subroutine. The data provides the arguments and error routines needed by the subroutine.

CHAIN
: A feature of the relocatable program loader that provides a way to run programs that exceed a single core storage load by forming a multiphase program.

channel scheduler
: A routine that allocates usage of a data channel among the devices attached to that channel.

channel trap
: See interrupt.

checkpoint
: A recording of the current status of the computer to permit restarting the program later.

closed subroutine
: A subroutine that is a separate program. To use a closed subroutine, the programmer transfers control out of his main program into the subroutine. The subroutine terminates by transferring back to the main program. A closed subroutine is entered into the program only once, and can then be used as often as needed by coding in the main program a *calling sequence* that gives the name of the subroutine and the desired parameters.

COBOL
: COBOL (COmmon Business Oriented Language) is a programming language designed primarily for commercial data processing. It allows the user to describe the processing to be performed in terms similar to business English.

COBOL compiler
: A program that translates COBOL language statements into assembler input or machine-language programs.

comments field
: This optional field becomes the fourth field of a MAP symbolic instruction. When it is used it must be preceded by at least one blank to separate it from the variable field. This field is available for explanatory remarks, etc., as a convenience to the programmer. It is listed in the assembly listing, but has no other effect on program execution. See also remarks card.

compile
: To produce a machine-language program from a source-language deck. The language of a compiler is closer than symbolic or machine language is to the language of the problem.

compiler
: A program used to produce either assembler input or a machine language program from a source, or programming, language deck. It typically involves program analysis, tabulation of information, and generation of instructions by synthesis of tabulated information and use of skeleton or model routines. In the 7040/7044 Operating System, the Macro Assembly Program assembles the input generated by the compilers.

control card
: A punched card containing input data or parameters for initializing or modifying a generalized program for one specific application.

control dictionary
The portion of a program deck that contains symbolic information necessary to relocate and/or load the deck, including the names and locations of control sections.

control section
A sequence of instructions or data within a program that can be referred to from outside the program segment in which it is contained. A control section can be deleted or replaced with a control section from other program segments.

core storage
The form of high-speed storage using magnetic cores that is found in the central processing unit of a 7040/7044 Data Processing System.

data record
A collection of facts, numbers, letters, symbols, etc., that a program can process or produce.

data record format
The predetermined arrangement of the contents of a data record.

debugging
The process of detecting and correcting errors.

definition
See symbol definition.

dummy argument
See substitutable argument.

dynamic dump
A dump of selected areas of core storage during the execution of a program at intervals specified by the programmer. A snapshot is a form of dynamic dumping.

element
A symbol or an integer less than the fifteenth power of two.

end of data file
The condition that exists when all of the records in a data file have been read or written.

end-of-file trailer label
A record with identification and control data related to previous records of a file. Its first five characters are 1EOF.

end-of-medium indicator
The signal that tells a program that the physical end of a device has been reached.

end-of-reel trailer label
A record with identification and control data related to previous records of a file that extends to another reel. This is the last record on all but the last unit of a multi-reel file. Its first five characters are 1EOR.

entry point
A specific location in a program segment which other segments can reference.

event
Use of an input/output device which makes the device busy but does not make the channel busy.

expression
A single term, or two or more terms combined by the operators + and −.

external symbol
See virtual symbol.

externally initiated trap
An interruption of central processing unit operations due to an event in a device independent of any activity in the central processing unit.

file
A collection of related information.

file closing
The termination of input/output operations on a file. It often involves the preparation of end-of-file trailer labels and rewind operations.

file control block
An area of core storage in which the user's specifications for a file are stored. A file control block can be generated from file and label pseudo-operations, by the use of $FILE and$LABEL control cards at load time, or can be created by the programmer.

file mark
A special indication on an external storage device that informs the program that the end of data has been read on the device. A file mark is written by the input/output label system after the header label, after the checkpoint recording, if any, and before and after the trailer labels of output files.

file opening
The initialization of input/output operations on a file. This often involves verification of header labels.

FORTRAN
FORTRAN is a programming language designed primarily for scientific and technical applications.

FORTRAN compiler
A program that translates FORTRAN language statements into assembler input or machine language.

get
To obtain a single data record from an input file.

header label
A record containing common, constant, or identifying information for a group of records which follow. It usually contains a file identification, a creation date, and a retention period.

immediate symbol
A symbol that is assigned a specific value during the first pass of the assembly program.

initial start
Beginning of data processing upon loading a system into a computer.

initialize
To set an instruction, counter, switch, or address to a specified starting condition at a specified time in a program.

interface
A common boundary between two programs or two data processing system components.

internally initiated trap
An interruption of central processing unit operations due to a currently scheduled activity on a data channel.

interrupt
A break in the normal flow of a routine such that the flow can be resumed from that point at a later time. An interrupt is usually caused by a signal from a source external to the central processing unit.

intersystem unit
An input/output unit that is to be reserved so that information may be passed between job segments.

job
One or more applications specified by the programmer to be executed as a logical unit. A job is delimited by the $JOB card.

label
See header label, end-of-file trailer label, end-of-reel trailer label, and label processing.

label processing
Standard
The use of the input/output label system to verify or create a standard IBM label. The format of the standard, 120-character IBM label is given in the publication *IBM 7040/7044 Operating System (16/32K): Input/Output Control System*, Form C28-6309.

Nonstandard
The use of the input/output label system to read and write labels verified and created by the user's routines.

Additional
The use of the input/output label system to verify or create a standard IBM label and then, by user's routines, verify or insert additional information in the optional field of the standard label.

library
An organized collection of operating programs, subroutines, and data. See system library.

link
A portion of a program that is distinct core storage load implemented by using the chain feature of the relocatable program loader.

literal
A symbol or quantity in a source program that is itself data rather than a reference to data.

literal pool
An area within an assembled program segment where the literals used in a program section are stored. Within an area, duplicate literals may be eliminated.

load
To take information from auxiliary or external storage and place it into core storage.

load address
The absolute storage location into which a word or the first of a series of contiguous words is loaded.

location
Also referred to as storage location. See absolute address.

location counter
A counter that is incremented by one for each word the assembly program generates in the object program.

location field
See name field.

logical record
See data record.

machine language
The binary data that can be executed or used by the processing unit of the 7040/7044.

Macro Skeleton Table
A Macro Assembly Program internal table that contains the prototypes of all the macro-definitions in a program.

macro-definition
The specification of a macro-operation. This includes specifying the name of the macro-operation and the prototype cards, which indicate the fields which are to be fixed, and the fields which are to be variable (substitutable arguments).

macro-instruction
The symbolic instruction that causes the named macro-operation to be inserted in-line into the program segment. The macro-instruction contains the parameters that are to replace the substitutable arguments in the macro-definition.

macro-operation
An open subroutine with variable arguments that can be defined once and then used by coding a single instruction in the symbolic source program.

merge
To combine data records from two or more ordered files into one ordered file.

monitor
A program or routine to control operation of several other programs or routines.

name field
The first field of a MAP symbolic instruction, punched in card columns 1-6. It may contain a symbol by which other instructions can refer to the instruction named. The name field is also referred to as the location field.

non-data operation
Any use of an input/output device that does not involve the transfer of data.

Nucleus
That portion of the system monitor that remains in core storage at all times during use of the operating system to provide common data areas, pointers, tables, and routines.

null field
A subfield that is indicated as being present by a blank preceding a comma, consecutive commas, or a comma followed by a blank.

object program
The binary machine language program.

off-line
Pertains to operation of input/output devices or auxiliary equipment not under direct control of the central processing unit.

on-line
Pertaining to operation of input/output devices under direct control of a program being executed in the central processing unit.

open subroutine
Subroutines that are inserted into the normal sequence of a program. Each time an open subroutine is used by a program, all of the instructions of that subroutine must be repeated. A macro-operation is a type of open subroutine; although a programmer uses only one instruction to call a macro-operation, the coding generated by that instruction is included in the sequence of a program each time it is used.

operating system
A collection of monitors, subsystems, data control programs, and user's programs that permits uninterrupted computer operation during the processing of a variety of jobs.

operation field
The second field of a MAP symbolic instruction, punched in card columns 8-14. It contains the instruction mnemonic, pseudo-operation, or macro-operation code of the instruction. An asterisk may follow a machine code to indicate indirect addressing.

operation
A sequence of events and activities on a particular device, such as writing a record on magnetic tape and correcting any writing errors, punching a card, backspacing several tape records, or seeking, writing, and write checking a track on disk storage.

order of merge
The number of files that can be combined into a consolidated file during a merging operation.

origin
The address of the beginning of a program section.

overlapping data channel
A data channel that allows asynchronous operation of its input/output devices and program processing by the central processing unit.

overlay
The technique of using the same areas of internal storage during different stages of a run. It is used to replace all or part of a program that is no longer needed in internal storage with another part of the program.

parameter
A quantity to which arbitrary values may be assigned.

patching
Correcting or changing the coding by overlaying it with another instruction or group of instructions.

phase
A portion of a program that is loaded as a unit for subsequent execution.

phase dictionary
An abbreviated table of contents that contains the phase name and load addresses of the phases to be loaded for a given application.

physical record
See block.

pointer
A word giving the address of another core storage location. See also transfer vector.

primary unit
The symbolic unit on which a file begins.

priority program
A special processing routine to be executed at the completion of an input/output operation, after a specified time interval has elapsed or upon some other signal external to the program in control.

processor
A program that performs the functions of generations, assembly, and/or compilation, loading or execution supervision.

processor application
Job segments that are applications of a processor.

program
A group of related routines that solve a given problem.

program deck
The binary output deck produced by the macro assembly program from the input deck for one compilation or assembly run. The program deck includes all the cards between the $IBLDR card and the $DKEND card.

program scheduler
A facility in IOEX that allocates use of the central processing unit among programs in storage, based on priority.

program segment
The part of a program that is contained in a program deck.

programming system
A source language and the processor program that translates it into machine language.

prototype
The part of a macro-definition that establishes the format of the instructions generated by the macro-instruction.

pseudo-instruction
The symbolic instruction that causes the performance of a pseudo-operation.

pseudo-operation
Any operation available in the MAP language that is not an actual machine operation, special instructions, IOCS operation, prefix code, or macro-operation.

put
To place a single data record into an output file.

qualification
The process of uniquely identifying the symbols defined in a given section of a program segment by appending another symbol.

random access
To obtain data directly from any storage location regardless of its position with respect to the previously referenced information.

random processing
The treatment of data without respect to its location in external storage, and in an arbitrary sequence governed by the input against which it is to be processed.

record
A group of related facts or field of information treated as a unit and stored as a continuous sequence of words or characters separated by gaps.

reel
Used figuratively to represent any external storage medium.

relative origin
A symbolic origin that is assigned to a machine location by the loader.

RELMOD deck
A deck with a relative load address. The load address is determined at load time.

relocatable binary format
The format produced by an assembler and acceptable by a loader. It permits relocation of addresses and decrements by means of relocation bits.

relocatable program loader
A program that assigns absolute origins to relocatable subroutines, object programs, and data, assigns absolute locations to each of the instructions or data, and modifies the reference to these instructions or data.

relocatable subroutine
A sequence of machine instructions and data with a specific function. The load address is determined at load time.

remarks card
A card that is used to insert remarks in a listing.

reserved unit
A symbolic unit that is given a unique designation, that is not available for assignment to a file, and that may be used only by referencing that designation.

restart
To return to a previous point in a program and resume operation from that point.

routine
A sequence of machine instructions that carry out a specific function.

run
The execution of one or more programs that are linked to form one operating program.

select minus routine
A routine that examines the results of an input/output activity and determines if any error recovery is required. When it is entered from the channel scheduler, the sign of the accumulator is minus.

select plus routine
A routine that determines if a device is ready to be used and, if it is, prepares a select instruction, channel command, and device order to accomplish an input/output activity or error recovery procedure. When this routine is entered from the channel scheduler, the sign of the accumulator is plus.

select routine
A routine for a specific input/output device consisting of a select plus routine and a select minus routine.

sense data
Information from an input/output file control unit indicating error, unusual, or attention conditions.

sequential access
Obtaining data from an input/output device in a serial manner only.

snapshot
See Dynamic Dump.

sort
To place a file of records in order according to a specified sequence.

sort application
Job segments that are applications of the Generalized Sorting System.

sort run
Same as sort application.

source program
A symbolic program written in a problem oriented or symbolic language, such as the FORTRAN, COBOL or MAP languages.

statement
An environmental definition, data description, or procedure in the source language of a compiler or assembler.

storage
A general term for any device or medium capable of retaining information for later use.

storage load
See phase.

string
A sequence of items.

subaddress
A portion of an input/output device that is accessible through an order code. For disk storage units, the module number is the subaddress.

subroutine
A set of instructions, taken as a unit, that perform a specific programming task. Subroutines are commonly used for such phases of a problem as common mathematical procedures (e.g., finding a square root), converting data from one form to another, and error procedures. The subroutines can be used many times over, by one or several programs.

substitutable argument
A prototype card field in a macro-definition that is variable and is to be replaced with a parameter (quantity or symbol) when the macro-operation is used. It is also called a dummy argument.

subsystem
A major component of the 7040/7044 Operating System such as the Processor, the Generalized Sorting System, or the System Editor.

symbol
A character or combination of characters used to represent the absolute address of a storage location, an input/output device, or any other program parameter.

symbol definition
The process of assigning a value to a symbol.

symbolic language
The defined set of characters and the rules for combining them into meaningful communication that permits the programmer to represent the machine locations and instructions by recognizable names and symbols, e.g., the MAP language.

symbolic unit
A mnemonic in the symbolic units table, which refers to an input/output device. A symbolic unit may be assigned to an entire input/output device or to a portion of a device.

symbolic units table
An area of core storage that holds the addresses of the unit control blocks (ucbs) and system control blocks (scbs) for each symbolic unit.

system
A collection of components united to form a functional unit. It usually is used in these publications to refer to the 7040/7044 Operating System (16/32K). It may also refer to the 7040/7044 Data Processing System.

system control block
An area of core storage holding information relating to an operation on a symbolic unit defined by an ATTACH macro-instruction. These blocks are used by the input/output operations level of IOCS.

System Library
The organized collection of absolute programs, relocatable subroutines, and data that make up the operating system.

System Loader
The absolute program loader in the Nucleus.

system monitor
The part of an operating system that contains routines and needed for continuous system operation.

system unit
One of the following: S.SLBx, S.SINx, S.SOUx, S.SPPx, or S.SCK1.

term
A single element or two or more elements combined by the operators * and /.

trailer label
See end-of-file trailer label and end-of-reel trailer label.

transfer vector
A collection of core storage words containing information used to identify the position of routines whose location is out of the regular sequence of instructions.

trap
See interrupt.

trap word
The core storage location used to store the instruction counter and trap identification data.

unit control block
A nine-word area of core storage describing an input/output device connected to the computer. These blocks are generated during system assembly for use by the input/output executor.

unit record equipment
Input/output devices such as the card reader, card punch, and printer.

unit synchronizer
A routine that coordinates completion of input and output operations on a given unit with the calling program according to the specifications of the programmer.

utility unit
A unit that is available for use by the system or by the programmer for any purpose.

variable field
The third field of a MAP symbolic instruction format. It contains the address, tag, and decrement (or count) portions of a machine instruction, or whatever is specified for a pseudo-instruction or the parameters for a macro-instruction.

variable-length records
Records of a file in which the number of words in each record varies.

virtual symbol
A symbol in a program segment that is used in the variable field of an instruction, which never appears in the name field of an instruction in that program segment, and which is identified as being defined in another program segment.

Index

Printed in U.S.A. C28-6318-5

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601