File No. 7090-27
Form C28-6389-1

IBM Systems Reference Library

IBM 7090/7094 IBSYS Operating System

Version 13

IBJOB Processor

This publication describes the IBM 7090/7094 IBJOB Processor, a subsystem of the 7090/7094 IBSYS Operating System, Version 13. The IBJOB Processor, 7090-PR-929, translates programming languages. It consists of the following components:

Processor Monitor (IBJOB)—7090-SV-801
FORTRAN IV Compiler (IBFTC)—7090-FO-805
COBOL Compiler (IBCBC)—7090-CB-806
Macro Assembly Program (IBMAP)—7090-SP-804
Loader (IBLDR)—7090-SV-802
Subroutine Library (IBLIB)—7090-LM-803
Debugging Processor (IBDBL)—7090-PR-807

This publication is divided into three parts. The first part describes the procedures for the applications programmer to follow in using the system. The second part describes the operations of each component for use by the systems programmer. The third part contains the text of all the error messages for each component with the appropriate explanations.

Preface

This publication provides procedural information for using the IBM 7090/7094 IBJOB Processor in the following capacities: compiling, assembling, loading, execution, and debugging. The primary objective of this publication is to help the reader to use these capabilities efficiently. For this reason, the organization, layout, and reference aids (diagrams, flowcharts, appendixes, and cross-references) are designed to follow a job-processing sequence.

This material is divided into three parts. The first part contains instructions for the FORTRAN IV, COBOL, or MAP programmer. It gives the basic information required to run a job. Once the programmer has become familiar with this material, he can use the control card checklists in the appendixes for quick reference.

The second part contains a more detailed explanation of the operations and functions of the various components for the programmer who is responsible for modification and maintenance of the system.

The third part of the manual contains all the messages generated by the IBJOB Processor, with explanations for all messages that are not self-explanatory.

It is not necessary that each reader cover all of the material in this manual. For example, the FORTRAN IV programmer need only be concerned in the first part with the Processor Monitor, the FORTRAN IV Compiler, and the Subroutine Library sections. He can then find additional information in the respective sections of the second part of the publication.

The reader should be familiar with the publication that describes the particular language he is using. These publications are:

IBM 7090/7094 IBSYS Operating System, Version 13: FORTRAN IV Language, Form C28-6390.

IBM 7090/7094 IBSYS Operating System, Version 13: COBOL Language, Form C28-6391.

IBM 7090/7094 IBSYS Operating System, Version 13: Macro Assembly Program (MAP) Language, Form C28-6392.

The programmer who wants to use the IBJOB debugging facilities should read the publication *IBM 7090/7094 IBSYS Operating System, Version 13: IBJOB Processor Debugging Package,* Form C28-6393.

The MAP programmer should be familiar with the contents of one of the following publications: *IBM 7090 Principles of Operation,* Form A22-6528; or *IBM 7094 Principles of Operation,* Form A-22-6703. The MAP programmer who uses the Input/Output Control System should also read the publication *IBM 7090/7094 IBSYS Operating System, Verson 13: Input/Output Control System,* Form C28-6345.

All readers should be familiar with the contents of the publication *IBM 7090/7094 IBSYS Operating System, Version 13: System Monitor (IBSYS),* Form C28-6248.

The machine configuration required for the operation of the IBJOB Processor is described in Appendix G.

MINOR REVISION (July 1965)

This edition, Form C28-6389-1, is a reprint of Form C28-6389-0, incorporating changes released in the following Technical Newsletters:

FORM NOS.	PAGES	DATED
N28-0140-0	23, 33, 38, 39, 101, 104, 126	April 7, 1965
N28-0154-0	3, 4, 45, 46, 47, 48, 49, 107, 158, 159, 161, 176-203	May 18, 1965
N28-0138-0	4, 14, 17, 18, 19, 20, 22, 25, 26, 30, 35, 58, 62, 96, 106, 126-133, 141, 149, 150, 164, 165, 167, 173, 174, 201-206, 208	June 11, 1965

Form C28-6389-0 and the Technical Newsletters are not obsoleted.

Copies of this and other IBM publications can be obtained at IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM Corporation, Programming Systems Publications, Dept. D91, 1271 Avenue of the Americas, New York, N. Y. 10020

Contents

PART 1: PROGRAMMING FUNDAMENTALS

Introduction

The IBJOB Processor is one of several system programs operating under the IBM 7090/7094 IBSYS Operating System. These programs operate under the control of the first-level monitor, the System Monitor (IBSYS).

The IBJOB Processor is a versatile monitored system that can translate several source language program decks within a single job. It can compile, assemble, load, and execute program decks written in the FORTRAN IV, COBOL, and/or MAP languages. It can also load and execute previously assembled object program decks. In addition, program decks written in different programming languages can be combined with previously assembled decks to form a single executable object program. Finally, a debugging program aids in program checkout.

A program deck is a series of card images headed by a control card that calls a component to translate the deck, and ended by a control card that transfers control back to the IBJOB Monitor. Any number of program decks can be run at one time. Some may operate like closed subroutines or subprograms. All these decks grouped together can form a Processor application, which is the basic unit of work that can be performed by the IBJOB Processor. In processing an application one or more operations may be performed, such as compilations, assemblies, or the loading of relocatable programs that were previously assembled.

Figure 1 illustrates the operation of the IBJOB Processor on source language programs. The following seven components may be used in these operations:

1. The Processor Monitor (IBJOB), which is the supervisory component of the IBJOB Processor. The Monitor provides communication with the IBSYS Monitor, positions the system tape, brings the various components into storage according to processing requirements, and regulates the input/output phasing of the components.

The Processor Monitor reads control cards that specify the actions to be performed in a Processor application.

2. The FORTRAN IV Compiler (IBFTC), which compiles and assembles programs written in the FORTRAN IV language. It produces input to the Loader.

3. The COBOL Compiler (IBCBC), which compiles programs written in the COBOL language and produces input to the Macro Assembly Program.

4. The Macro Assembly Program (IBMAP), which processes MAP language source programs and MAP programs produced by the COBOL Compiler, and produces input to the Loader.

5. The Debugging Package, which allows the programmer to obtain dumps of specified areas of core storage and machine registers during program execution for the purpose of debugging a program with a minimum of programming effort. Two separate facilities are provided: compile-time debugging for COBOL programs (IBDBC) and load-time debugging (IBDBL) for FORTRAN IV and MAP language programs.

6. The Loader (IBLDR), which processes and combines several relocatable binary programs to form one absolute binary object program. The Loader loads separately assembled program segments combined with any required subroutines from the Subroutine Library, allocates core storage for common data and input/output buffers, generates necessary initialization sequences for program use of input/output operations, and provides a listing of core storage allocation.

7. The Subroutine Library (IBLIB), which contains a group of relocatable subroutines available for system and programming use. Subroutines may be edited through the Librarian.

Control Card Format

The following control card notation is used in the control card formats throughout this publication:

1. *Upper-case letters* must be punched exactly as shown.

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

3. *Braces* { } indicate that a choice of the contents is mandatory.

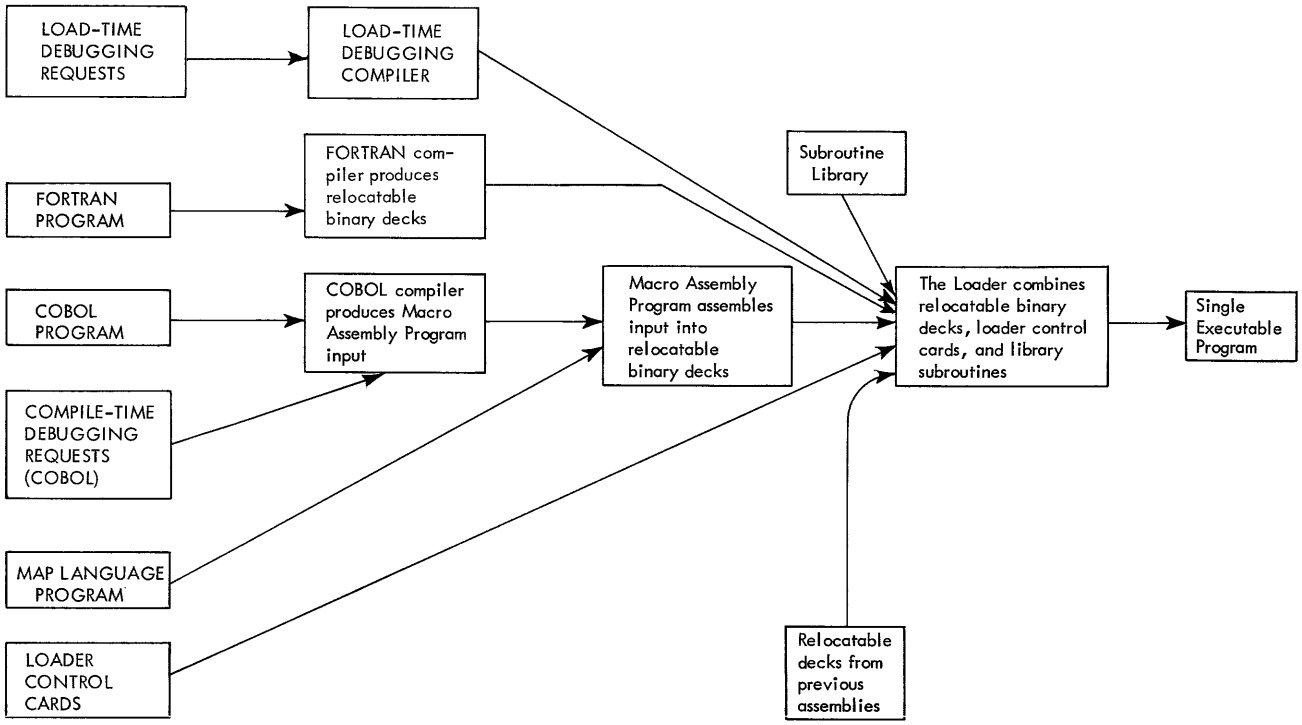


Figure 1. Operation of the IBJOB Processor on Source Language Programs

4. *Brackets* [ ] contain an option that may be omitted or included by the programmer.

5. *Options that are underlined* are the standard options. When no option is specified on a control card, the Processor uses the standard option.

6. *Commas* are used to separate options. *Embedded blanks* should not be used.

7. Except where noted specifically, the options on a control card may be punched in any order.

Core Storage Allocation

The core storage arrangement of the IBJOB Processor components is illustrated in Figure 2. The location of the object program during execution is also shown.

IBJOB Processor Dictionaries

The Loader is able to combine decks coded in different source languages by using dictionaries common to all the decks. Control dictionaries, assembled by the MAP Assembler and the FORTRAN IV Compiler, contain deckname entries and control section name entries. File dictionaries contain file names.

Deck Names

A deck name identifies a deck produced by either the FORTRAN IV Compiler or the Assembler from one source language. The programmer defines a deck name by punching it on a component control card (see "$IBFTC Card," "$IBCBC Card," and "$IBMAP Card"). A deck name may be used to identify or qualify control section names within a deck. The use of deck names and the rules for forming deck names are described under "Deck Name Rules."

Control Section Names

A control section name refers to an area of coding or data within a program deck. These areas, called control sections, can be referred to by other decks.

The COBOL and FORTRAN IV Compilers make up control section names for the programmer during processing. The MAP programmer designates control section names using the CONTRL and ENTRY pseudo-operations. The rules for creating MAP control section names are listed under "Control Section Rules." Control sections may be renamed, replaced, or deleted at load time (see "Loader Control Cards").

File Names

File names identify files used by a program. The MAP programmer creates file names using FILE pseudo-operations or the $FILE card. The FORTRAN programmer uses the FILE routines. The COBOL user describes a file by making a file description entry in the Data Division.

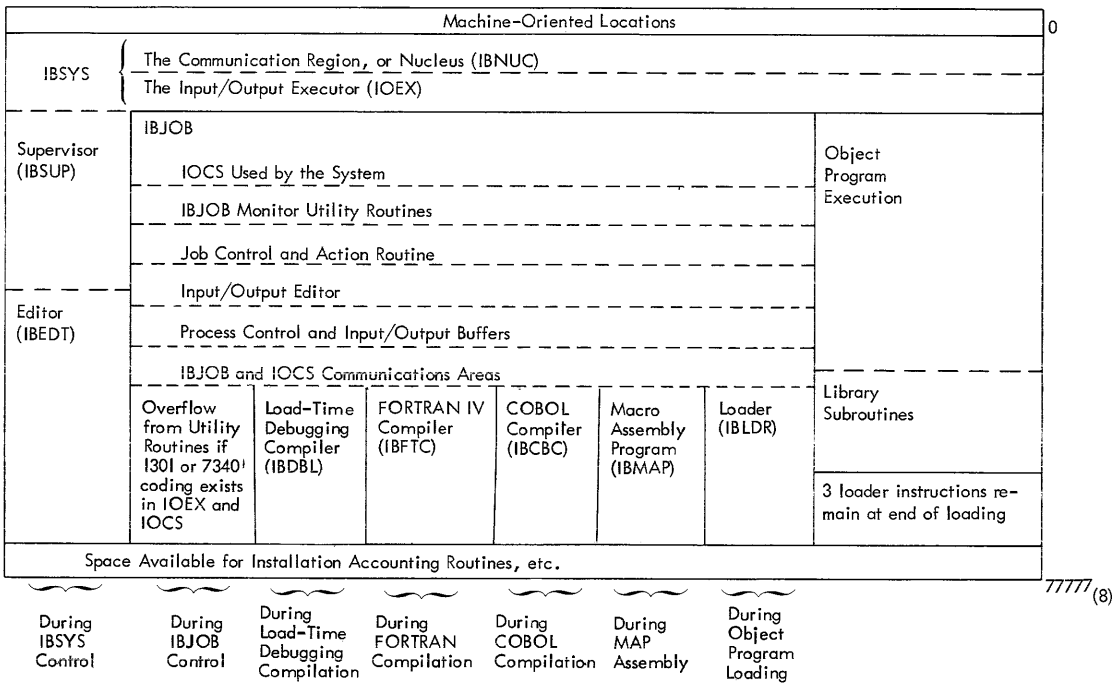


Figure 2. Core Storage Allocation of the Operating System Components

The rules for making up file names are listed in the section "File Name Rules."

Additional Index Register Mode

The IBJOB Processor always operates in the additional index register mode. It enters this mode upon receiving control from the IBSYS System Monitor, and it leaves the additional index register mode before giving control back to IBSYS. Compilers operating under control of the IBJOB Processor use the additional index register mode, and IBJOB Processor object programs are normally executed in this mode.

A compiled program may call a MAP language program that uses the multiple-tag mode. If this occurs, the MAP language must restore the additional index register mode, as well as all index registers used, before it returns control to the calling program.

When a MAP language program that uses the multiple-tag mode calls a compiled program, the compiled program automatically enters the additional index register mode. Thus, the instruction in the MAP language program immediately following the call to the compiled program should be to re-enter the multiple-tag mode.

Floating-Point Trap Mode

All programs compiled by the IBJOB Processor operate in floating-point trap mode. The floating-point trap routine is loaded with every object program.

The Processor Monitor is the major component of the IBJOB Processor. Its primary function is to control communication between the System Monitor (IBSYS) and the components of the IBJOB Processor. The Processor Monitor is called into core storage when the IBSYS Monitor reads a $EXECUTE card on which IBJOB is specified.

System Monitor Control Cards

The following IBSYS System Monitor control cards may be required for an IBJOB Processor application.

$JOB Card

The Processor Monitor transfers control to the resident portion of the System Monitor, called the Nucleus (IBNUC), when the Processor Monitor reads a $JOB card. If system units have not been reassigned or made unavailable during the last job and if a between-jobs interrupt condition does not exist, the Processor Monitor regains control and transfers control to the installation accounting routine. If there is no installation accounting routine, the $JOB card is listed on both the system printer and the system output unit, where it causes a page to be ejected before listing of the card, and the Processor Monitor retains control.

If units have been reassigned or made unavailable during the last job and/or if a between-jobs interrupt condition exits, the System Monitor retains control until a $EXECUTE card is read.

The format of the $JOB card is:

1	16
$JOB	any text

Columns 16 through 72 are normally used to identify a job and may contain any combination of alphameric characters and blanks. The information is printed in the program listing as punched, but it has no effect on the program except as a match when a $RESTART card is used. (See publication *IBM 7090/7094 IBSYS Operating System: System Monitor (IBSYS)*, Form C28-6248.) A deck name in columns 8-13 is printed in the listing, but has no effect on the program.

$EXECUTE Card

The $EXECUTE card specifies the IBSYS subsystem to be used in processing a program. It must precede a Processor application within a job if one of the following conditions exists:

1. The Processor application is the first unit of work to be performed within the job.
2. The previous Processor application resulted in execution of an object program.
3. Another subsystem was in control.

The Processor Monitor checks the subsystem name. If the name is IBJOB, no action is taken. If the name is anything other than IBJOB, this information is relayed to the IBSYS Monitor. The Processor Monitor then transfers control to the IBSYS Monitor.

The format of the $EXECUTE card is:

1	16
$EXECUTE	subsystem name

The subsystem name consists of six or fewer BCD characters beginning in column 16.

IBJOB Processor Control Card

The following control card is required for an IBJOB Processor application.

$IBJOB Card

The $IBJOB card must be the first control card read by the Processor Monitor for a given application. The options specified in the $IBJOB control card determine the manner in which an application is to be processed.

The format of the $IBJOB card is:

1	16
$IBJOB	[, options]

The options, which start in column 16, are described in the following text.

Execution Option

$$\left[,\left\{\begin{matrix}\underline{\text{GO}}\\ \text{NOGO}\end{matrix}\right\}\right]$$

GO—The object program is executed after it is loaded.

NOGO—The object program is not executed, even if it is loaded. If this option is specified, the object program is loaded only when LOGIC, DLOGIC, or MAP is specified in the $IBJOB card.

If neither GO nor NOGO is specified, the object program is to be executed (GO).

Logic Options

$$\left[,\left\{\begin{matrix}\underline{\text{NOLOGIC}}\\ \text{LOGIC}\\ \text{DLOGIC}\end{matrix}\right\}\right]$$

NOLOGIC—A cross-reference table is not wanted.

LOGIC—A cross-reference table of the program sections and of the system subroutines required for execution is generated. The origin and length of each program section and subroutine, and the buffer assignments, are also given. When this option is specified for a Subroutine Library editing execution, a listing

of the control section dependencies in the generated library is produced.

DLOGIC—A cross-reference table of the program sections and of the origin and length of each program section is generated. The system subroutines and buffer assignments are not given if this option is chosen.

If neither NOLOGIC, LOGIC, nor DLOGIC is specified, a cross-reference table is not generated (NOLOGIC).

MAP Options

[,{NOMAP / MAP}]

NOMAP—A core storage map is not wanted. This map is also called "memory map" and "load map."

MAP—A core storage map showing the origin and the amount of core storage used by the IBSYS Operating System, the object program, and the input/output buffers is generated. The file list and buffer pool organization are also given. When this option is specified for a Librarian execution, a listing of all subroutines in the generated library is produced. A sample core storage map is shown in Appendix F.

If neither NOMAP nor MAP is specified, a storage map is not generated (NOMAP).

File List Options

[,{NOFILES / FILES}]

NOFILES—A listing of the input/output unit assignments and mounting instructions to the operator are printed on-line.

FILES—A listing of the input/output unit assignments and mounting instructions to the operator are printed on-line and off-line.

If neither NOFILES nor FILES is specified, the list is printed only on-line (NOFILES).

Input Deck Options

[,{SOURCE / NOSOURCE}]

SOURCE—The application contains at least one compilation or assembly.

NOSOURCE—The application contains only relocatable binary program decks. These decks are loaded from the system input unit.

If neither SOURCE nor NOSOURCE is specified, it is assumed that a compilation and/or assembly is required in the application (SOURCE).

Input/Output Options

[, (IOEX / MINIMUM / BASIC / LABELS / FIOCS / ALTIO)]

IOEX—The object program uses the Input/Output Executor for trap supervision. The only IOCS routine available is for on-line printing.

MINIMUM—The Minimum level of IOCS is to be loaded with the object program. Internal files cannot be used. The following routines are available:

.OPEN
.CLOSE
.READ
.WRITE
.BSR
.READR } if IOCS has been assembled
.RELES } for disk or drum storage.

BASIC—The Basic level of IOCS is to be loaded with the object program. In addition to the Minimum level routines, Basic contains:

.BSF
.CKPT
.JOIN
.REW
.STASH
.WEF

LABELS—The Labels level of IOCS is to be loaded with the object program. In addition to the Basic level routines, the Labels level contains routines for label checking and preparation.

FIOCS—A reduced form of Minimum IOCS is to be loaded for use by a FORTRAN IV program.

ALTIO—The FORTRAN Alternate Input/Output package is to be loaded for use by a FORTRAN IV program.

The FORTRAN IV programmer can best use ALTIO when core storage is limited, input/output activity is low, or mixed mode files are required. Otherwise, FIOCS should be used. If the FORTRAN IV programmer chooses no option, the Minimum level of IOCS is loaded automatically.

MAP programmers can choose IOCS options according to the routines needed by their programs, although the Loader provides an additional check for all levels except ALTIO. Normally, if an object program requires a more comprehensive level of IOCS than that specified by the programmer, the Loader loads the required level. But if ALTIO is specified for a FORTRAN program, and the program requires a higher level of IOCS, ALTIO is still loaded, and an error message is generated.

Because the Loader automatically loads the correct level of IOCS, the COBOL programmer does not need to specify any level. The Loader also loads Random IOCS with the object program, if the program contains a reference to a Random IOCS routine.

The levels of IOCS are described in the publication *IBM 7090/7094 IBSYS Operating System: Input/Output Control System*, Form C28-6345. The two FORTRAN IV input/output packages, FIOCS and Alternate Input/Output, are described in the section "Subroutine Library Information."

Overlay Options

[,{FLOW / NOFLOW}]

FLOW—Execution of the object program is not permitted if the rules concerning references between links are violated. These rules are stated in the section "Overlay Feature of the Loader."

NOFLOW—Execution is allowed even though the rules governing references between links are violated.

If neither FLOW nor NOFLOW is specified, execution of the object program is not permitted when the rules governing references between links are violated (FLOW).

Component Control Cards

Each component operating within the IBJOB Processor has a unique control card that causes the Processor Monitor to load the component. These component control cards are:

FORTRAN Compiler	$IBFTC
COBOL Compiler	$IBCBC
Macro Assembly Program	$IBMAP
Debugging Compiler (Load-Time)	$IBDBL
Debugging Processor (Compile-Time)	$IBDBC
Loader	$IBLDR
Subroutine Library (Librarian)	$EDIT

Each of these cards will be described in detail in the respective component section of this publication.

End-of-File Card

The end-of-file card is an IBM 1401 utility program control card. The end-of-file card must be the last card in a Processor application. An end-of-file card is either a card with a 7 and 8 punch in card column 1 or any control card that causes a file mark to be written by a peripheral program.

Optional Control Cards

The following cards are optional System Monitor control cards frequently used in an IBJOB Processor application:

$IBSYS Card

The Processor Monitor prints the message

RETURNING TO IBSYS

on-line and transfers control to the IBSYS System Monitor when the $IBSYS card is read.

The format of the $IBSYS card is:

1
$IBSYS

$ID Card

The $ID card causes the Processor Monitor to transfer control to the installation accounting routine if one exists. The distributed version of the Operating System does not contain an installation accounting routine. Therefore, the only action that occurs is the listing of the $ID card.

The format of the $ID card is:

1	7
$ID	any text

Columns 7-72 may contain any combination of alphameric characters and blanks.

$STOP Card

The $STOP card transfers control to the System Monitor. In effect, the $STOP card defines the end of a deck of jobs.

The format of the $STOP card is:

1
$STOP

$PAUSE Card

The $PAUSE card causes the machine to halt temporarily for operator action.

The contents of the variable field of the $PAUSE card are printed on-line. The $PAUSE card allows the programmer to interrupt processing for specific operator action. When the $PAUSE card is used, the variable field should contain explicit instructions to the operator so that immediate action can be taken. Processing is resumed when the operator presses the START key.

The format of the $PAUSE card is:

1	16
$PAUSE	instructions to the operator

Columns 16-72 may contain any combination of alphameric characters and blanks.

$ Card*

The $* card is a comments card. The contents are printed on-line and off-line. No further action occurs.

The format of the $* card is:

1 3
$* any text

Columns 3-72 may contain any combination of alphameric characters and blanks.

$ENTRY Card

The $ENTRY card specifies the location of the initial transfer to the object program at execution time. The variable field contains an external name to which the initial transfer is to be made. If the $ENTRY card is omitted or if the variable field is blank, the initial transfer is either to the standard entry point of the first deck retained or to an entry point whose name is "....." (consists of six periods, the name compiled as the standard entry point to FORTRAN IV main programs).

The format of the $ENTRY card is:

1	16
$ENTRY	[{Exname / Deckname}]

where the variable field contains either an external name to which the initial transfer is to be made or a deck name, in which case the initial transfer is to the standard entry point of that deck.

A $ENTRY card is not needed when one of the following conditions exists:

1. The main program is a FORTRAN IV program.
2. The main program is processed first, and the desired entry point is the standard entry point of that program.

When a $ENTRY card is used, it must immediately follow the source deck. The $ENTRY card precedes either an end-of-file card or a $DATA card.

$DATA Card

The $DATA card is an IBM 1401 utility program control card. The $DATA card indicates the beginning of a data file on the input unit. An end-of-file card performs the same function and may be used in this capacity. The data file must be followed by an end-of-file card.

The format of the $DATA card is:

```
1
$DATA
```

$ENDREEL Card

The $ENDREEL card causes a reel switch involving the system input unit (SYSIN1) and the secondary input unit (SYSIN2). The $ENDREEL card must be preceded by an end-of-file card. This card must not appear in the middle of a data file.

The format of the $ENDREEL card is:

```
1
$ENDREEL
```

$POST Card

The $POST card causes the Processor Monitor to call the Debugging Postprocessor. This card is only used to restart a Processor Monitor application (1) that has failed during execution of the object program and (2) in which debugging information has been written on SYSCK2. The on-line message

DEBUG INFORMATION ON SYSCK2 (unit)

is printed indicating this condition.

The format of the $POST card is:

```
1
$POST
```

$IBREL Card

The $IBREL card indicates that no more compilations or assemblies follow on the system input unit (SYSIN1). The IBJOB Processor Monitor then reads the load file for the Loader until a file mark is read, at which time the input file on the system input unit is read until the end of the Processor Monitor application. In effect, the $IBREL card supplements the SOURCE option on the $IBJOB card. A program may have compilations and assemblies up to this point, but this card indicates that no more will follow. The Loader gains control when the $IBREL card is recognized.

The format of the $IBREL card is:

```
1
$IBREL
```

$TITLE Card

The $TITLE card causes the information contained in columns 16-72 to be printed as the heading for the next Compiler and/or Assembler output (preempting the normal or installation accounting routine heading).

The format of the $TITLE card is:

1	8	16
$TITLE	[NODAT]	any text

If the NODAT option is specified, a date is not printed. If the option is not specified, the date from the System Date word (SYSDAT) is printed.

Input/Output Control Cards

Input/Output Editor

The input/output editor, which is a part of the Processor Monitor, regulates the input/output operations of the IBJOB Processor. The input/output editor reads from the system input unit (SYSIN1) or from a unit specified by the programmer. Both punched output (written on the system peripheral punch unit [SYSPP]) and listing output (written on the system output unit [SYSOU1]) are prepared by the input/output editor. The programmer can specify a temporary alternate unit for the system output unit.

The input/output editor also writes the output from both compilers and reads the input for the Assembler and the Loader.

The IBJOB Processor uses the following system units:

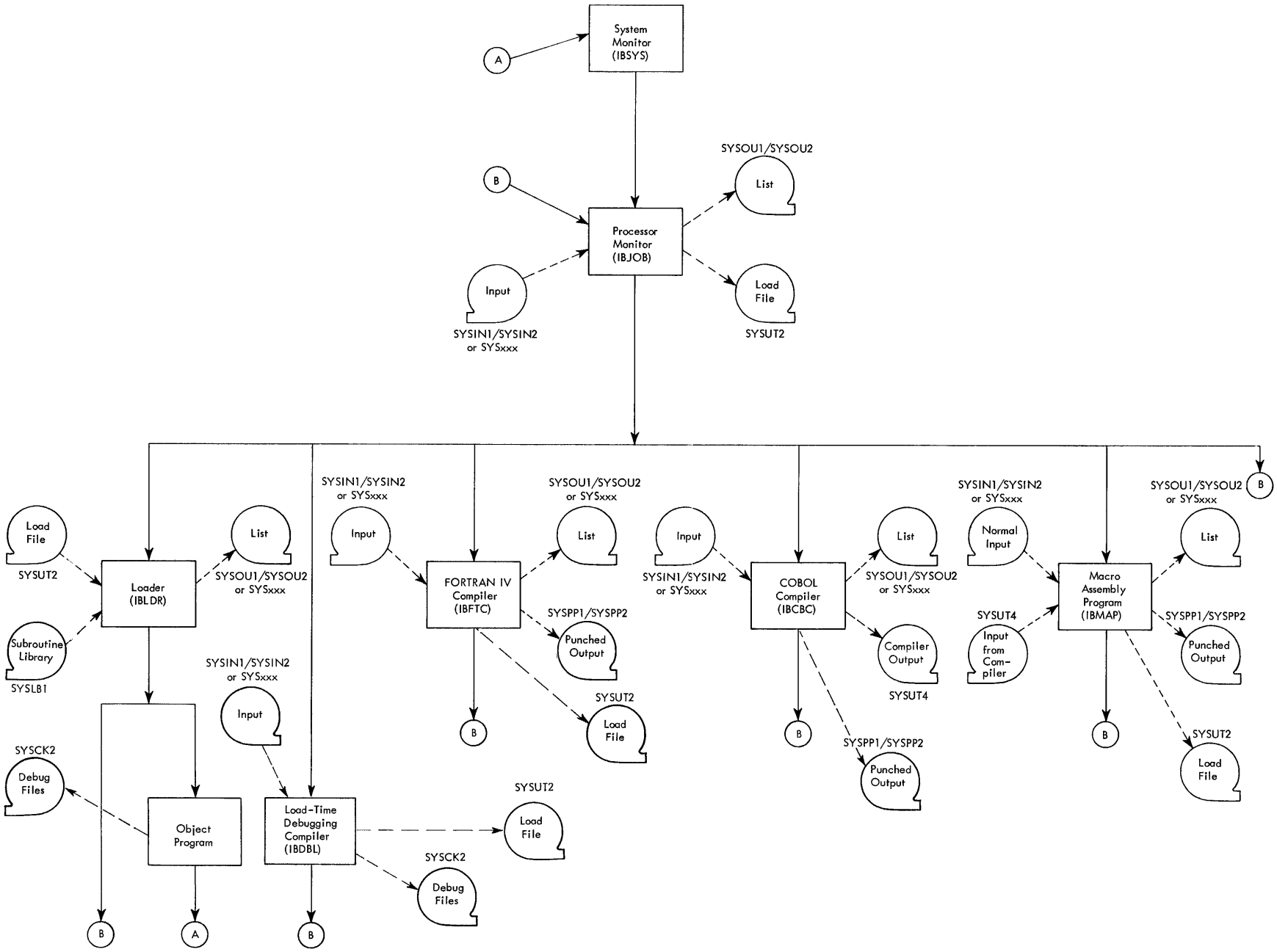
1. System input units (SYSIN1 and SYSIN2)
2. System output units (SYSOU1 and SYSOU2)
3. System peripheral punch units (SYSPP1 and SYSPP2)
4. System utility units (SYSUT1, SYSUT2, SYSUT3, and SYSUT4)

Figure 3 illustrates the flow of control and the input/output flow through the IBJOB Processor.

The control cards used to specify input/output configurations and formats are the $IEDIT and $OEDIT cards. When these control cards are used, they must precede the component control card of the deck that is affected by them. The specifications on the control card remain in effect either until the end of the application or until another $IEDIT or $OEDIT card changes the specifications. The standard specifications are used until one or both of these control cards change them. The formats of the $IEDIT and $OEDIT cards and explanations of their options are given in the following text.

$IEDIT Card

The $IEDIT card sets input specifications for the remainder of the application or until the next $IEDIT card is read.



The flow of control is designated by solid lines, whereas input/output flow is designated by dotted lines.

Figure 3. Flow of Control and Input/Output Flow Through IBJOB Processor

The format of the $IEDIT card is:

1	16
$IEDIT	[, options]

The options in the variable field are described in the following text.

Input Options

[,{SYSIN1 / SYSxxx}]

SYSIN1—The source, symbolic, or object program immediately follows the component control card on the system input unit (SYSIN1).

SYSXXX—The source, symbolic, or object program is on the specified alternate input unit. Only those system unit names not used by the IBJOB Processor may be used (SYSCK1, SYSCK2, SYSLB2, SYSLB3, SYSLB4).

If neither the system input unit nor an alternate input unit is specified, the input is read from the system input unit (SYSIN1).

NOTE: SYSCK2 is not available as an alternate input unit if debugging has been requested. SYSLB2, SYSLB3, and SYSLB4 may not be available if used to store system components.

Search Options

[,{NOSRCH / SRCHn / SCHFn}]

NOSRCH—The specified alternate input unit is positioned correctly.

SRCHn—Search through the designated number of files (n files) on the specified alternate input unit, for the source, symbolic, or object program whose deck name is the same as the deck name in the component control card.

SCHFn—Search the designated file (nth file) on the specified alternate input for the source, symbolic, or object program whose deck name is the same as the deck name in the component control card. This option cannot be used if the alternate input unit is disk storage or drum storage.

The n may be a one- or two-digit decimal number. If a comma or a blank immediately follows the SRCH or SCHF portions of the options, the number is assumed to be 1.

If neither NOSRCH, SRCHn, nor SCHFn is specified, the alternate input unit is not searched (NOSRCH).

Alter Options

[,{NOALTER / ALTER}]

NOALTER—There are no alter cards within the deck.

ALTER—There are alter cards in the deck.

If neither NOALTER nor ALTER is specified, it is assumed that there are no alter cards (NOALTER). For a description of the ALTER procedure, see "Altering An Input Deck."

$OEDIT Card

The $OEDIT card sets output specifications for the remainder of the application or until another $OEDIT card is read.

The format of the $OEDIT card is:

1	16
$OEDIT	[, options]

The options in the variable field are described in the following text.

Output Options

[,{SYSOU1 / SYSxxx}]

SYSOU1—The output listings for this deck are placed on the system output unit.

SYSXXX—The output listings for this deck are placed on the specified alternate output unit. Only those function names not used by the IBJOB Processor may be used (SYSCK1, SYSCK2, SYSLB2, SYSLB3, SYSLB4).

If neither SYSOU1 nor an alternate output unit is specified, the output is written on the system output unit (SYSOU1).

NOTE: SYSCK2 is not available as an alternate output unit if debugging has been requested. SYSLB2, SYSLB3, and SYSLB4 may not be available if used to store system components.

Assembler Prest Options

[,{NOPREST / PREST}]

NOPREST—A Prest symbolic deck is not wanted.

PREST—A compressed form of the symbolic input to the Assembler is written on the system peripheral punch unit. The deck produced is called the Prest deck. The FORTRAN Compiler does not produce input to the Macro Assembly Program; therefore, a Prest deck cannot be obtained for a FORTRAN compilation. If this option is specified for a FORTRAN compilation, it is ignored.

Programs in Prest format must be submitted in the following sequence:

```
$JOB
$EXECUTE        $IBJOB
$IBJOB          [options]
[component control card in BCD format ($IBMAP, $IBFTC, or $IBCBC) appropriate to the IBJOB component that processes the deck]
[deck in Prest format as punched out]
[end-of-file card]
$IBSYS
$STOP
```

In resubmitting decks in Prest format for processing, the programmer should reinsert any $-control cards used in the original source program, except the $CBEND card. For example, if two decks in a source program are separated by a $ORIGIN card, a duplicate $ORIGIN card

must be inserted between the corresponding Prest or Cprest decks.

Prest cards consist of series of field counts, string counts, and strings, which have been formed by scanning each field of an input card. A string consists of those characters in a field other than the leading blanks. Prest cards, generated in column-binary format, do not include control cards.

A field, and therefore a string, may not exceed 67_8. For example, if there are 72 (110_8) consecutive characters but never more than two consecutive blanks, the field would be coded as:

FIELD COUNT	STRING COUNT	STRING	FIELD COUNT	STRING COUNT	STRING
67_8	67_8	XXXXX...	21_8	21_8	XXXXX...

The characters 77 signify the end of the encoded input card. Trailing blanks on the input card are not encoded.

The following examples illustrate Prest output:

The input card

```
1              8
PRESbbbCLAbbbbbXYZb.....b
```

is encoded as:

04 04 P R E S 06 03 C L A 10 03 X Y Z 77...

The input card

```
1              8
bbbbABCbXYZbbDEFbb.....b
```

is encoded as:

20 14 A B C b X Y Z bb D E F 77.....

The following is an illustration of a column binary card containing 22 data words. Positions 4-7 distinguish a Prest from a Loader or relocatable deck.

WORD	POSITION	CONTENTS
1	S, 1	11 (examine bit 3)
	2	check-sum control bit 0=verify check sum 1= do not verify check sum
	3	0 (standard IBJOB Processor deck)
	4	1 ⎫ (Prest deck)
	5-7	111 ⎭
	8-12	01010
	13-17	word count (beginning with word 3)
	21-35	card sequence number
2	S, 1-35	logical check sum of word 1 and all data words on the card
3-24	S, 1-35	22 words containing either instructions or data from the program.

The contents of a column-binary card are described more fully in the "Loader Input" section of "Loader Information."

If neither NOPREST nor PREST is specified, the Prest deck is not generated (NOPREST).

Compiler Prest Options

[,{NOCPR / CPREST}]

NOCPR—A Prest symbolic deck of the source input to either the FORTRAN or the COBOL Compiler is not wanted.

CPREST—A Prest deck of the source input to either compiler is written on the system peripheral punch unit. Programs in Compiler Prest format can be submitted for processing in the same manner as Prest programs (see "Assembler Prest Options").

The format of cards which form a Compiler Prest deck is the same as for the cards which form an Assembler Prest deck.

If neither NOCPR nor CPREST is specified, a Prest deck is not generated (NOCPR).

If both PREST and CPREST are specified in the $OEDIT card that precedes a source deck, both compiler input and output are written, in that order, on the system peripheral punch unit in.

Altering an Input Deck

Any symbolic, source, or Prest input deck can be modified. To change an input deck, ALTER must be specified on the $IEDIT card, and Alter control cards must be used. These control cards are described later in the text.

If an alternate input unit is not specified on the $IEDIT card, it is assumed that the Alter control cards must follow a component control card and precede the input deck on the system input unit. Before the deck can be altered, the Alter control cards are moved to the system utility unit (SYSUT2). When the deck has been altered, the system utility unit (SYSUT2) is repositioned to be used for load file output.

If an alternate input unit is specified on the $IEDIT card, the input deck must be on the alternate input unit and the Alter deck must be on the system input unit. The input deck must be preceded by a component control card. The Alter deck must also be preceded by a component control card of the same type and with the same deck name.

The Alter feature does not produce an updated source or symbolic tape.

Alter Numbers

The contents of columns 73-80 of an input card are used as Alter numbers. An Alter number is generated before compilation or assembly when a Prest deck is requested as output. This generated number appears on the assembly or compilation listing, where columns 73-80 (identification field) of a card are normally printed. The numbers are a maximum of eight right-justified sequential digits with leading blanks.

If a source deck or symbolic deck is to be altered, the existing identification fields are used as Alter numbers. They are replaced on the listing with generated Alter numbers if a Prest deck is requested as output. This is necessary to enable alteration of the Prest deck.

Alter Control Cards

A source, symbolic, or Prest deck may be altered by using the following control cards:

1. To insert cards into a deck, a control card with the following format is used:

1	8	16
m	*ALTER	n1

Fields m and n are the contents of the identification field (columns 73-80) of a control card in the input deck, if the deck is a source or symbolic deck. If the input deck is a Prest deck, fields m and n1 are the generated Alter number. The first blank character appearing in the identification field indicates that all prior characters constitute field m. The characters remaining after the blank or blanks constitute field n1. In the identification field, field m is left-justified and field n1 is right-justified. If the identification field contains no blank characters, field m may be omitted or may consist of no more than the first six characters of the identification field. Field n1 then consists of the remaining characters that were not placed in field m. Fields m and n1 must have a total number of characters equal to the number of characters in the identification, excluding leading or embedded blanks. For example, if the identification in columns 73-80 is LABEL090, the format for this identification, on an Alter control card, could be in any of the following forms:

1	8	16
LABEL	*ALTER	090
LABEL0	*ALTER	90
LAB	*ALTER	EL090
	*ALTER	LABEL090

If the identification in columns 73-80 is LABELbb9, the format for this identification is:

1	8	16
LABEL	*ALTER	9

If there are embedded blanks in the identification, the Alter control card must have the preceding format. Cards following the Alter control card up to, but not including the next Alter control card, are inserted immediately before card mn.

2. To delete and/or insert cards in a deck, a control card with the following format is used:

1	8	16
m	*ALTER	n1, n2

Fields m and n1 are defined in item 1. Fields m and n2 are either the same as m and n1, in which case only card mn1 is deleted, or they identify a card following card mn1, in which case cards mn1 through mn2 are deleted. In addition, any cards following this Alter control card, up to but not including the next Alter control card, are inserted in place of the deleted cards.

3. To end the Alter deck, a control card with the following format is used:

1	8
	*ENDAL

This control card denotes the end of the Alter deck and must be the last control card in every Alter deck.

Sample Deck Format Using an Alternate Input Unit

Figure 4 shows the control cards that are necessary for the compilation and/or assembly and simultaneous execution of program decks located on both the system input unit and an alternate input unit.

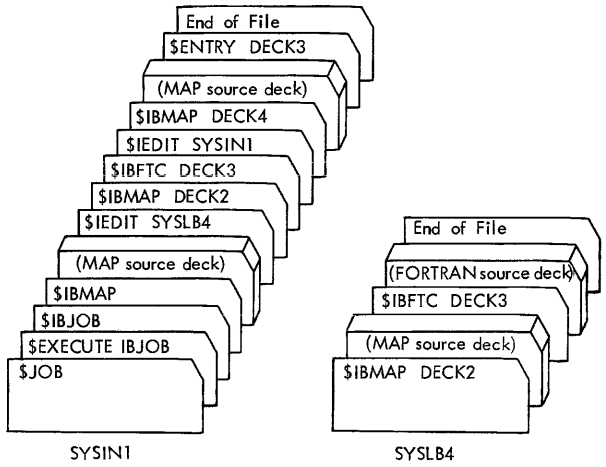


Figure 4. Sample Control Card Deck for Use of an Alternate Input Unit

After the $JOB, $EXECUTE, and $IBJOB cards have been read, the sequence of operations is:

1. The $IBMAP card is read from the system input unit (SYSIN1), and the MAP language deck is assembled.
2. The $IEDIT card, specifying the system Library unit (SYSLB4) as the alternate input unit, is read. This causes all input except the component control card to be read from the system Library unit (SYSLB4).
3. The $IBMAP card, specifying DECK2, is read from the system input unit (SYSIN1). The data in columns 1-15 on this $IBMAP card and on the corresponding $IBMAP card on the system Library unit (SYSLB4) is matched, and the MAP language deck on SYSLB4 is assembled.
4. The $IBFTC card, specifying DECK3, is read from the system input unit (SYSIN1). The data in columns 1-15 on this $IBFTC card and on the corresponding $IBFTC card (SYSLB4) is matched, and the FORTRAN IV language deck on the SYSLB4 is compiled and assembled.
5. The $IEDIT card, specifying the system input unit (SYSIN1), is read. This causes the IBJOB Processor to resume the reading of input from the system input unit (SYSIN1).

6. The $IBMAP card, specifying DECK4, is read from the system input unit (SYSIN1), and the MAP language deck is assembled.

7. The $ENTRY card, specifying DECK3, is read from the system input unit (SYSIN1). This indicates that control is to be transferred to the standard entry point of DECK3 when the object program is loaded.

8. The file mark is read on the system input unit (SYSIN1). Since the NOGO specification did not appear in the IBJOB card, the reading of the file mark causes the loading and execution of all program decks compiled and/or assembled by the IBJOB Processor during the application.

The FORTRAN IV Compiler translates programs written in the FORTRAN IV language, assembles these programs, and produces relocatable binary input to the Loader.

$IBFTC Card

The FORTRAN IV Compiler is called into core storage when the Processor Monitor reads a $IBFTC card. The $IBFTC card contains the name of the deck that will follow, output options (list and punch operations), and machine-oriented options that increase the efficiency of the object program.

The format of the $IBFTC card is:[1]

1	8	16
$IBFTC	deckname	[, options]

where deckname names the deck that follows. A deck name of six or fewer characters must be punched in columns 8-13. Characters that cannot be used in the deck name are parentheses, commas, slashes, quotation marks, equal signs, and embedded blanks.

The variable field starts in column 16. The options that may appear in this field are described in the following text.

List Options

[,{NOLIST / LIST / FULIST}]

NOLIST — A listing of the object program is not wanted.

LIST — A MAP language listing of the object program, three instructions per line, is generated. Only the relative locations and symbolic information are listed. See Figure 5 for an example of such a listing.

FULIST — A MAP language listing of the object program is generated, one instruction per line. This listing includes generated octal information and resembles the example in Figure 8.

If neither NOLIST, LIST, nor FULIST is specified, a listing of the object program is not generated (NOLIST). A listing of the source program, however, is always generated.

Punch Options

[,{DECK / NODECK}]

DECK — The object program deck is written on the system peripheral punch unit.

NODECK — A punched deck is not wanted.

If neither DECK nor NODECK is specified, the object program deck is written on the system peripheral punch unit (DECK). If Prest and Cprest decks have also been requested, the object deck is written last.

Instruction Set Options

[,{M90 / M94 / M94/2}]

M90 — The MAP language program uses only 7090 machine instructions. MAP language double-precision operations are simulated by system macros, and EVEN pseudo-operations are treated as commentary.

M94 — The MAP language program uses only 7094 machine instructions.

M94/2 — The MAP language program uses 7094 machine instructions. MAP language EVEN pseudo-operations are treated as commentary.

If neither M90, M94, nor M94/2 is specified, the MAP language program uses only 7090 machine instructions (M90). FORTRAN programmers should specify M94 if the machine they are using is a 7094 and M94/2 if the machine they are using is a 7094, model 2.

Index Register Options

[,{XR3 / XRn}]

XR3 — The MAP language program uses three index registers (1, 2, and 4).

XRn — The MAP language program uses n index registers (n is a number from 4 to 7).

If neither XR3 nor XRn is specified, the MAP language program uses only three index registers (XR3). FORTRAN programmers should specify the number of index registers available on the machine they are using.

```
BINARY CARD (NOT PUNCHED)
 00032          USE .PROL.          00032          USE .MAIN.          00071          USE .TRST.
 00071          USE .ERAS.          00073          USE .STOR.          00073  I       BSS 1,X
 00074  V       BSS 1,F             00032          USE .MAIN.          00032  29.     NULL
 00032  1A      NULL                00032  1AA     STZ **              00033  31.     NULL
 00033  2A      NULL                00033          AXC 1,1             00034  32.     NULL
 00034          STZ I               00035  2A1     FSCA I,1            00040  3A      NULL
 00040  3AA     CLA **              00041  3AB     FDP **              00042          XCA
 00043  3AC     FAD **              00044          FDP =2.
```

Figure 5. Listing of Object Program, Three Instructions per Line

[1] Note that the symbol table option is no longer available.

Debugging Dictionary Options

$$\left[,\left\{\begin{matrix}\underline{\text{NODD}}\\ \text{DD}\\ \text{SDD}\end{matrix}\right\}\right]$$

NODD — A debugging dictionary is not wanted.

DD — A full debugging dictionary is desired. All symbols used in the assembled program will appear in the debugging dictionary. In the case of a FORTRAN IV program, the debugging dictionary includes all statement numbers, all programmer-specified symbols, and all symbols generated by the FORTRAN Compiler.

SDD — A short debugging dictionary is desired. Only those symbols will appear in the debugging dictionary that are specified through KEEP pesudo-operations supplied by the MAP programmer. In the case of a FORTRAN IV program, the Compiler generates KEEP operations for statement numbers and programmer-specified symbols when SDD is chosen.

If neither NODD, DD, nor SDD is specified, a debugging dictionary is not generated (NODD).

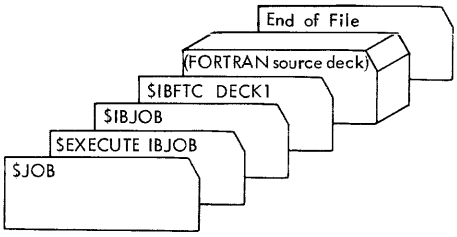


Figure 6. Sample Control Card Deck for One FORTRAN IV Compilation

Sample FORTRAN IV Deck Format

Figure 6 shows the control cards necessary for compilation and execution of one FORTRAN IV language deck.

Figure 7 shows the control cards necessary for compilation and execution of two FORTRAN IV language decks. When execution begins, control is transferred to the first instruction in the first deck. FORTRAN decks grouped together as shown permit phasing during FORTRAN IV compilation. (See "FORTRAN IV Compiler Information" for a discussion of phasing.)

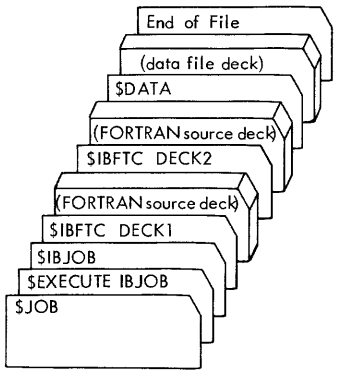


Figure 7. Sample Control Card Deck for Two FORTRAN IV Compilations

A data file for the object program follows the source language decks. The $DATA card that precedes the data file causes a file mark to be written when it is recognized by the peripheral input/output program. When the IBJOB Processor reads the file mark, loading and execution of the object program begin.

The COBOL Compiler translates programs written in the COBOL language and produces MAP language input to the Assembler.

$IBCBC Card

The COBOL Compiler is called into core storage when the Processor Monitor reads a $IBCBC card. The $IBCBC card contains the name of the deck that will follow, output options (list and punch operations), and machine-oriented options that increase the efficiency of the object program.

The format of the $IBCBC card is:

1	8	16
$IBCBC	deckname	[, options]

where deckname names the deck that follows. A deck name of six or fewer alphanumeric characters must be punched in columns 8-13. At least one character must be alphabetic or a period. Characters that cannot be used in the deck name are parentheses, commas, slashes, quotation marks, equal signs, hyphens, and embedded blanks.

The variable field starts in column 16. The options that may appear in this field are described in the following text.

List Options

$$\left[,\left\{\begin{matrix}\underline{\text{NOLIST}}\\ \text{LIST}\\ \text{FULIST}\end{matrix}\right\}\right]$$

NOLIST — A listing of the symbolic object program input to the Assembler is not wanted. Error messages produced by the Assembler are listed.

LIST — A MAP language listing of the symbolic object program input to the Assembler, three instructions per line, is generated. Only the relative locations and symbolic information are listed. The format shown in Figure 5 for FORTRAN is used also for COBOL.

FULIST — A MAP language listing of the object program is generated, one instruction per line. This listing includes generated octal information. An example of such a program listing is shown in Figure 8. Reading from left to right, the first 5 digits show the relative location of the instruction within the deck. (Note that the pseudo-operation NULL has no number, since it does not appear in the object program.) The next 12 digits are the instruction in octal form. The 5 bits following are relocation bits (see "Relocatable Binary Text" in the section "Loader Information"). Next follows the instruction in symbolic form. ENXXX numbers are "equivalent names," i.e., MAP symbols that the COBOL Compiler generates for names in the source program. GNXXX numbers are supplementary symbols needed to translate the program into MAP language.

If no option is specified, a listing of the source program is generated, but not of the program output to the Assembler (NOLIST).

Symbol Table Options

$$\left[,\left\{\begin{matrix}\underline{\text{NOREF}}\\ \text{REF}\end{matrix}\right\}\right]$$

NOREF — A sorted dictionary and a cross-reference table are not wanted.

REF — A sorted dictionary of the source language names and their associated equivalent name (ENXXX) numbers and a cross-reference table of the symbols used in the object program are generated. The following is an example of a cross-reference dictionary:

SOURCE PROGRAM NAME	EQUIVALENT NAME NUMBER
FIELD1	EN0255
FIELD2	EN0257
OUTPUT-FILE	EN0244, 0250
OUTPUT-RECORD	EN0251
WORK-RECORD	EN0254

If neither NOREF nor REF is specified, the dictionary and table are not generated (NOREF).

Punch Options

$$\left[,\left\{\begin{matrix}\underline{\text{DECK}}\\ \text{NODECK}\end{matrix}\right\}\right]$$

DECK — The object program input to the Loader is written on the system peripheral punch unit (SYSPP1).

NODECK — A punched deck is not wanted.

```
                        00070          EN0251 NULL
      C0070   0074 00 4 00232  10001  EN0252 TSX     GN0012,4
      00071   0441 00 0 00210  10001         LDI     SP+5
      00072   0604 00 0 02000  10011         STI     .CAREF
      00073   0441 00 0 00142  10001         LDI     PI+1            EN0244
      00074   0604 00 0 02400  10011         STI     .CBREF
      00075   0074 00 4 03000  10011         TSX     .CMPK3,4

BINARY CARD (NOT PUNCHED)
      00076   1 00006 1 06000  10011         TXI     .CANA1,1,6
      00077   0074 00 4 00232  10001         TSX     GN0012,4
```

Figure 8. Object Program Instructions Generated One Instruction per Line from a COBOL Program

If neither DECK nor NODECK is specified, the object program is written on the system peripheral punch unit (DECK).

Instruction Set Options

[,{M90 / M94 / M94/2}]

M90 — The MAP language program uses only 7090 machine instructions. MAP language double-precision operations are simulated by system macros, and EVEN pseudo-operations are treated as commentary.

M94 — The MAP language program uses only 7094 machine instructions.

M94/2 — The MAP language program uses 7094 machine instructions. MAP language EVEN pseudo-operations are treated as commentary.

At present, only the M90 option is used by the Compiler, regardless of the specification made.

Index Register Options

[,{XR3 / XR7}]

XR3 — The MAP language program uses three index registers (1, 2, and 4).

XR7 — The MAP language program uses seven index registers.

If neither XR3 nor XR7 is specified, the MAP language program uses only three index registers (XR3). COBOL programmers should specify the number of index registers available on the machine they are using.

Code Options

[,{INLINE / TIGHT}]

INLINE — The object program's computational and MOVE tasks are optimized for speed.

TIGHT — The object program's computational and MOVE generated coding is shorter, thereby conserving object-time core storage.

If neither INLINE nor TIGHT is specified, the object program's computational and MOVE tasks are optimized for speed (INLINE).

Tape Error Options

[,{IOEND / READON}]

IOEND — Errors in reading tape at object time cause irrecoverable error conditions.

READON — Errors in reading tape at object time are ignored. This option may be used to allow high-volume data processing to continue while ignoring low-volume error conditions.

If neither IOEND nor READON is specified, these errors cause irrecoverable error conditions (IOEND).

Collating Sequence Options

[,{COMSEQ / BINSEQ}]

COMSEQ — The object program uses the commercial collating sequence.

BINSEQ — The object program uses the binary scientific collating sequence.

If neither COMSEQ nor BINSEQ is specified, the object program uses the commercial collating sequence (COMSEQ).

Debugging Dictionary Options

[,{NODD / DD}]

NODD — A debugging dictionary is not wanted.

DD — A debugging dictionary is desired. A debugging dictionary helps in debugging a MAP program generated by the COBOL Compiler. The COBOL Compiler takes no action on this option except to pass it to the Assembler. The Assembler then produces a dictionary containing all IBCBC and IBMAP generated symbols.

If neither NODD nor DD is specified, a debugging dictionary is not produced (NODD).

$CBEND Card

Every COBOL source deck must be followed immediately by a $CBEND card.

The format of the $CBEND card is:

1
$CBEND

Debugging for COBOL Programs

The COBOL Compiler can also provide debugging aids during compilation of COBOL decks. This optional facility is described under "Compile-Time Debugging."

Sample COBOL Deck Format

Figure 9 shows the control cards necessary for compilation and execution of a single COBOL language deck.

Figure 10 shows the control cards necessary for compilation and execution of two COBOL language decks. When execution begins, control is transferred to the standard entry point of the first deck.

NOTE: Any number of COBOL source decks or MAP or FORTRAN source decks may appear between the $IBJOB and end-of-file cards.

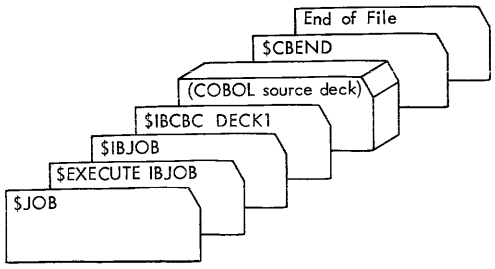


Figure 9. Sample Control Card Deck for One COBOL Compilation

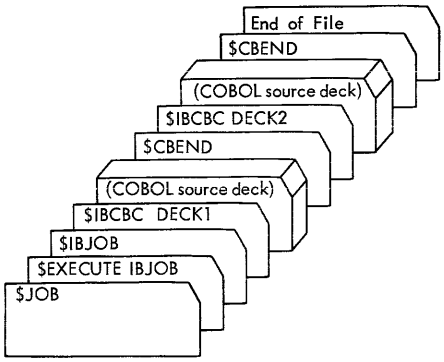


Figure 10. Sample Control Card Deck for Two COBOL Compilations Which Compile As a Single Job

The Macro Assembly Program (the Assembler) processes programs written in the MAP language as well as generated MAP programs that are output from the COBOL Compiler. The output from the Assembler can be either in relocatable binary form or in absolute binary form. The relocatable binary output is processed, if required, by the Loader. The Loader is used for processing and loading when either GO, LOGIC, DLOGIC, or MAP is specified in the $IBJOB card. An explanation of these options can be found in the section "$IBJOB Card." The object program, which is a result of assembling and loading, is composed of machine instructions that are generated by the Assembler, the input/output routines that are part of the Subroutine Library, and possibly the FORTRAN IV mathematical subroutines from the Subroutine Library. The use of the mathematical subroutines by the MAP programmer is described in the section "Subroutine Library (IBLIB)."

$IBMAP Card

The Assembler is called into core storage when the Processor Monitor reads a $IBMAP card. The $IBMAP card contains the name of the deck that follows, the type of assembly to be performed, output operations (list and punch options), and restrictions on the use of the MAP language in the deck that follows.

The format of the $IBMAP card is:

1	8	16
$IBMAP	deckname	[, options]

where deckname identifies the deck that follows. A deck name of six or fewer alphameric characters must be punched in card columns 8-13. Characters that cannot be used in the deck name are parentheses, commas, slashes, quotation marks, equal signs, and embedded blanks.

The options in the variable field, which starts in column 16, are described in the following text.

Card Count Option

[,count]

The card count option is an estimate of the number of symbolic language cards in the deck. The number may not exceed 32,767. If a card count is not specified, a count of 2,000 is assumed.

List Options

$$\left[,\left\{\begin{matrix}\underline{\text{LIST}}\\ \text{NOLIST}\end{matrix}\right\}\right]$$

LIST — A listing of the object program is generated.

NOLIST—A listing of the object program is not wanted.

If neither LIST nor NOLIST is specified, a listing of the object program is generated (LIST).

Symbol Table Options

$$\left[,\left\{\begin{matrix}\underline{\text{REF}}\\ \text{NOREF}\end{matrix}\right\}\right]$$

REF — A cross-reference table of the symbols used in the object program deck is generated. The deck listed in Figure 11 would result in the cross-reference table shown in Figure 12. In this table, under "Class," each

```
BINARY CARD (NOT PUNCHED)
       00000  0774 00 1 00011  10000  START  AXT     9,1
       00001  0500 00 1 00025  10001         CLA     NUMBER+10,1
       00002  0400 00 1 00026  10001  LOOP   ADD     NUMBER+11,1
       00003  2 00001 1 00002  10001         TIX     LOOP,1,1
       00004  0601 00 0 00025  10001         STO     SUM
       00005  000000000000     00010         CALL    DUMP(NUMBER,SUM+1,2)
       00005  0074 00 4 02000  10011
       00006  1 00003 0 01005  10011
       00007  0 00026 0 00006  10100
       00010  0 00000 0 00013  10001
       00011  0 00000 0 00026  10001
       00012  0 00000 0 00002  10000
       00013  000000000001     10000  NUMBER DEC     1,2,3,4,5,6,7,8,9,10
       00014  000000000002     10000
       00015  000000000003     10000
       00016  000000000004     10000
       00017  000000000005     10000
       00020  000000000006     10000
       00021  000000000007     10000

BINARY CARD (NOT PUNCHED)
       00022  000000000010     10000
       00023  000000000011     10000
       00024  000000000012     10000
       00025  200000000001     00001  SUM    BSS     1
       00026  000000000000     10000         *LDIR
       00027  235163212260     10000
                        00000  01111         END     START
```

Figure 11. Program That Would Result in Cross-Reference Table Shown in Figure 12

```
                CRTAB
        SYMBOL REFERENCE DATA

REFERENCES TO DEFINED SYMBOLS.

  CLASS  SYMBOL   VALUE   REFERENCES

         LOOP     00002   3
         NUMBER   00013   1,2,10
  LCTR    BLCTR
  QUAL    UNQS
  LCTR   //
         START    00000   30
         SUM      00025   4,11

REFERENCES TO VIRTUAL SYMBOLS.

         DUMP          2   5
```

Figure 12. Cross-Reference Table Resulting from Program Listed in Figure 11

symbol is classified by the types of defined symbols: ordinary symbols, location counter symbols, qualifying symbols, file names and symbols defined by BOOL, LBOOL, RBOOL, or SET pseudo-operations. Under "Symbol" each symbolic name defined in the program is listed. "BLCTR" (blank location counter) and "//" (blank COMMON) are location counter symbols supplied by the Assembler. "UNQS" is a qualifying symbol supplied by the Assembler. Under "Value" is shown the relative location assigned to each defined symbol within the deck. Under "References" are listed the relative locations at which the symbols are referred to. "References to Virtual Symbols" are the same as for defined symbols, except that "Value" is the number of the entry that the name occupies in the control dictionary for this deck after the preface entry.

NOREF — A cross-reference table is not wanted.

If neither REF nor NOREF is specified, a cross-reference table is generated (REF).

Punch Options

$$\left[,\left\{\begin{matrix}\underline{\text{DECK}}\\ \text{NODECK}\end{matrix}\right\}\right]$$

DECK — The object program deck is written on the system peripheral punch unit.

NODECK — A punched deck is not wanted.

If neither DECK nor NODECK is specified, the object program deck is written on the system peripheral punch unit (DECK).

System Symbol Options

$$\left[,\left\{\begin{matrix}\underline{\text{NOSYM}}\\ \text{MONSYM}\\ \text{JOBSYM}\end{matrix}\right\}\right]$$

NOSYM — No symbols are predefined by the Assembler.

MONSYM — The IBSYS Operating System symbols in the nucleus and Input/Output Executor (IOEX) communication regions and in the system unit function table, and the symbols SYSORG and SYSEND, are predefined by the Assembler. These symbols are defined as being in the qualification section "s.s" and, therefore, may be redefined at any point in the program. They are described in the publication *IBM 7090/7094 IBSYS Operating System: System Monitor (IBSYS)*, Form C28-6248.

JOBSYM — This option is effective only in an absolute mode assembly (ABSMOD). If it appears on a $IBMAP card that does not also specify ABSMOD, MONSYM is assumed. The IBSYS Operating System symbols, plus the IBJOB Processor system symbols used for subcomponent communication (see Appendix C), are predefined by the Assembler, as described for MONSYM.

If neither NOSYSM, MONSYM, nor JOBSYM is specified, all symbols referred to in an ABSMOD assembly must be defined by the source program (NOSYM).

Instructions Set Options

$$\left[,\left\{\begin{matrix}\underline{\text{M90}}\\ \text{M94}\\ \text{M94/2}\end{matrix}\right\}\right]$$

M90 — The Assembler generates only 7090 machine instructions. Double-precision operations are simulated, and EVEN pseudo-operations are treated as commentary.

M94 — The Assembler generates only 7094 machine instructions.

M94/2 — The Assembler generates only 7094 machine instructions, and EVEN pseudo-operations are treated as commentary.

If neither M90, M94, nor M94/2 is specified, the object program uses only 7090 machine instructions (M90).

Assembly Mode Options

$$\left[,\left\{\begin{matrix}\underline{\text{RELMOD}}\\ \text{SYSMOD}\\ \text{ABSMOD}\end{matrix}\right\}\right]$$

RELMOD — The object program is assembled in relocatable binary form.

SYSMOD — The object program, which has an absolute origin, is assembled in relocatable binary form.

ABSMOD — The object program is assembled in absolute binary form.

If neither RELMOD, SYSMOD, nor ABSMOD is specified, the program is assembled in relocatable binary form (RELMOD).

Parentheses Options

$$\left[,\left\{\begin{matrix}\underline{\text{NO( )}}\\ \text{( )OK}\end{matrix}\right\}\right]$$

NO( )—Parentheses should not be used in MAP symbols. If parentheses are used in a symbol in the location field, a warning message is printed, but assembly is permitted.

( )OK — Parentheses may be used in MAP symbols.

If neither NO( ) nor ( )OK is specified, parentheses should not be used in MAP symbols [NO( )].

Built-In Function Options

$$\left[,\left\{\begin{matrix}\underline{\text{NOMFTC}}\\ \text{MFTC}\end{matrix}\right\}\right]$$

NOMFTC — The built-in functions of the FORTRAN IV Compiler are not used by the object program.

MFTC — The built-in functions of the FORTRAN IV Compiler are used by the object program. These functions are described in the publication *IBM 7090/7094 IBSYS Operating System: FORTRAN IV Language,* Form C28-6390.

If neither NOMFTC nor MFTC is specified, the object program does not use the built-in functions (NOMFTC).

Debugging Dictionary Options

$$\left[,\left\{\begin{matrix}\underline{\text{NODD}}\\ \text{DD}\\ \text{SDD}\end{matrix}\right\}\right]$$

NODD — A debugging dictionary is not wanted.

DD — A full debugging dictionary is desired. All symbols used in the assembled program are included.

SDD — A short debugging dictionary is desired. Only those symbols that are specified by the MAP pseudo-operation KEEP appear in the debugging dictionary.

If neither NODD, DD, nor SDD is specified, a debugging dictionary is not produced (NODD).

Sample MAP Deck Format

Figure 13 shows the control cards necessary for the assembly and execution of a MAP language deck.

Figure 14 shows the control cards necessary for the assembly and execution of two MAP language decks. When execution begins, control is transferred to the standard entry point of the first deck.

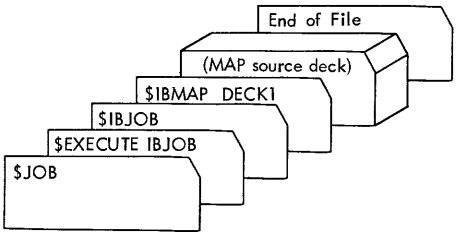


Figure 13. Sample Control Card Deck for One MAP Assembly

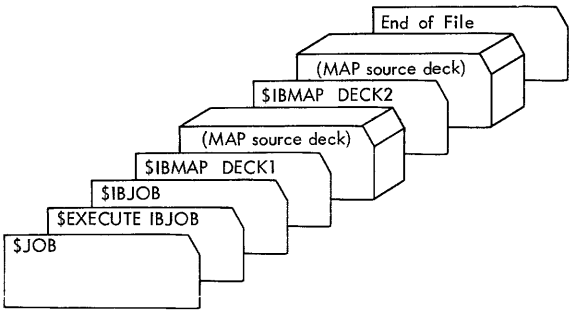


Figure 14. Sample Control Card Deck for Two MAP Assemblies

The Debugging Package enables the programmer to manipulate data, control processing, and obtain dumps of the contents of any locations in core storage at specified locations within his program. There is no limit on the number of requests that may be given for a single program. The publication *IBM 7090/7094 IBSYS Operating System, IBJOB Processor Debugging Package*, Form C28-6393, describes the procedure for coding debug requests.

This package provides the programmer with two types of debugging: compile-time debugging and load-time debugging.

Compile-Time Debugging

Compile-time debugging may be used with the COBOL Compiler at compilation to specify dumps at various points in a COBOL source program. Debug requests are similar to COBOL procedural statements and almost all procedural capabilities of the compiler may be used.

$IBDBC Card

The $IBDBC card heads each compile-time debug request. The $IBDBC card serves two functions: it identifies individual requests, and it defines the point at which the request is to be executed.

The format of the $IBDBC card is:

1	8	16
$IBDBC	[name]	location [, FATAL]

where name is an optional user-assigned control section name which permits deletion of the request at load time. This name must be a unique control section name consisting of up to six alphabetic and numeric characters, at least one of which must be alphabetic.

Location is the COBOL section-name or paragraph-name (qualified, if necessary) indicating the point in the program at which the request is to be executed. In effect, debug requests are performed as if they were physically placed in the source program following the section- or paragraph-name, but preceding the text associated with the name. Two $IBDBC cards in the same request packet may not refer to the same location.

FATAL, when specified, prevents loading and execution of the object program when an error of a level corresponding to the COBOL error level E or greater occurs within a debug request statement. If FATAL is not specified, a COBOL error of level E or less, when encountered within the procedural text of a debug request, does not prevent loading and execution of the object program. In this situation an attempt is made to interpret the statement. If the interpretation attempt is unsuccessful, the invalid statement is disregarded. If the request consists of more than one statement, only the invalid statement is disregarded.

The text of the debug request follows immediately after the $IBDBC card. The text may consist of any valid procedural statements conforming to the requirements of the COBOL language and format and the count-conditional statement. The only restriction on these statements is that they may not transfer control outside of the debug request itself. Display statements in a debug request are written on SYSOU1.

An end-of-file card or any $-control card terminates the compile-time debugging packet.

Load-Time Debugging

The load-time debugging facility allows FORTRAN IV MAP programmers to insert debug requests at load time that are to be executed with the object program. The language for load-time debug requests is derived from the FORTRAN IV language, with changes made for debugging purposes. All load-time requests for a particular Processor application are grouped together in what is called the *debugging packet*. The load-time debugging packet is placed immediately following the $IBJOB card at the beginning of the job deck, preceding the source and/or object decks.

$IBDBL Card

The $IBDBL card heads all load-time debugging packets and provides options governing the amount of debugging output.

The format of the $IBDBL card is:

1	16
$IBDBL	[, options]

The variable field starts in column 16. Columns 16-72 are scanned in full and therefore may contain blanks for legibility purposes. The options that may appear in this field are described in the following text.

Debugging Output Options

[, TRAP MAX=n_1] [, LINE MAX=n_2]

TRAP MAX = n_1 – All debugging action ceases after n_1 requests have been executed. The symbol n_1 is an integer. In octal, the number of requests may range from 01 to 077777. (Any integer containing a leading zero is recognized as an octal number.) If the number of requests is expressed in decimal, the range is from

1 to 32,767. Each time program execution is halted for debugging action, irrespective of any resultant action, an executed request is counted. If no TRAP MAX is specified, the debugging routines allow a maximum of 30,000 traps.

LINE MAX = n_2 – All dumping ceases after approximately n_2 print lines of debugging output have been written on tape; the postprocessor prints no more than n_2 lines. The symbol n_2 is an integer. In octal, the number of lines may range from 01 to 077777. (Any integer containing a leading zero is recognized as an octal number.) If the number of print lines is expressed in decimal, the range is from 1 to 32,767. If no LINE MAX is specified, the debugging routines allow a maximum of 1,000 lines.

The number of lines produced by a postmortem dump taken at the completion of execution is not included in the line count. To avoid the possibility of unlimited dumping during an uncontrolled loop, an approximation based upon LINE MAX is used to limit the number of words written on SYSCK2 during execution. This feature does not limit or prevent the uncontrolled loop.

Either or both of these options, in any order, may be used to control the amount of debugging output.

Message Listing Option

[, NOMES]

When specified, the NOMES option eliminates the object-time messages. Debugging output is unchanged, except that a preliminary list of the dumps and any TRAP MAX and LINE MAX messages are eliminated.

**DEND Card*

The *DEND card is the last card in the load-time debugging packet.

The format of the *DEND card is:

1

*DEND

Postprocessor Routines

The Postprocessor routines of the Load-Time Debugging Processor control the format of debugging output. This occurs following execution of the object program. If a Processor Monitor application fails during execution, the operator must initiate a restart using the $POST card. See the publication *IBM 7090/7094 IBSYS Operating System: Operator's Guide*, Form C28-6355, for descriptions of the $POST card and the restart procedure.

Sample Load-Time Debugging Deck Format

Figure 15 shows the control cards necessary for a Processor Monitor application that uses compile-time and load-time debugging.

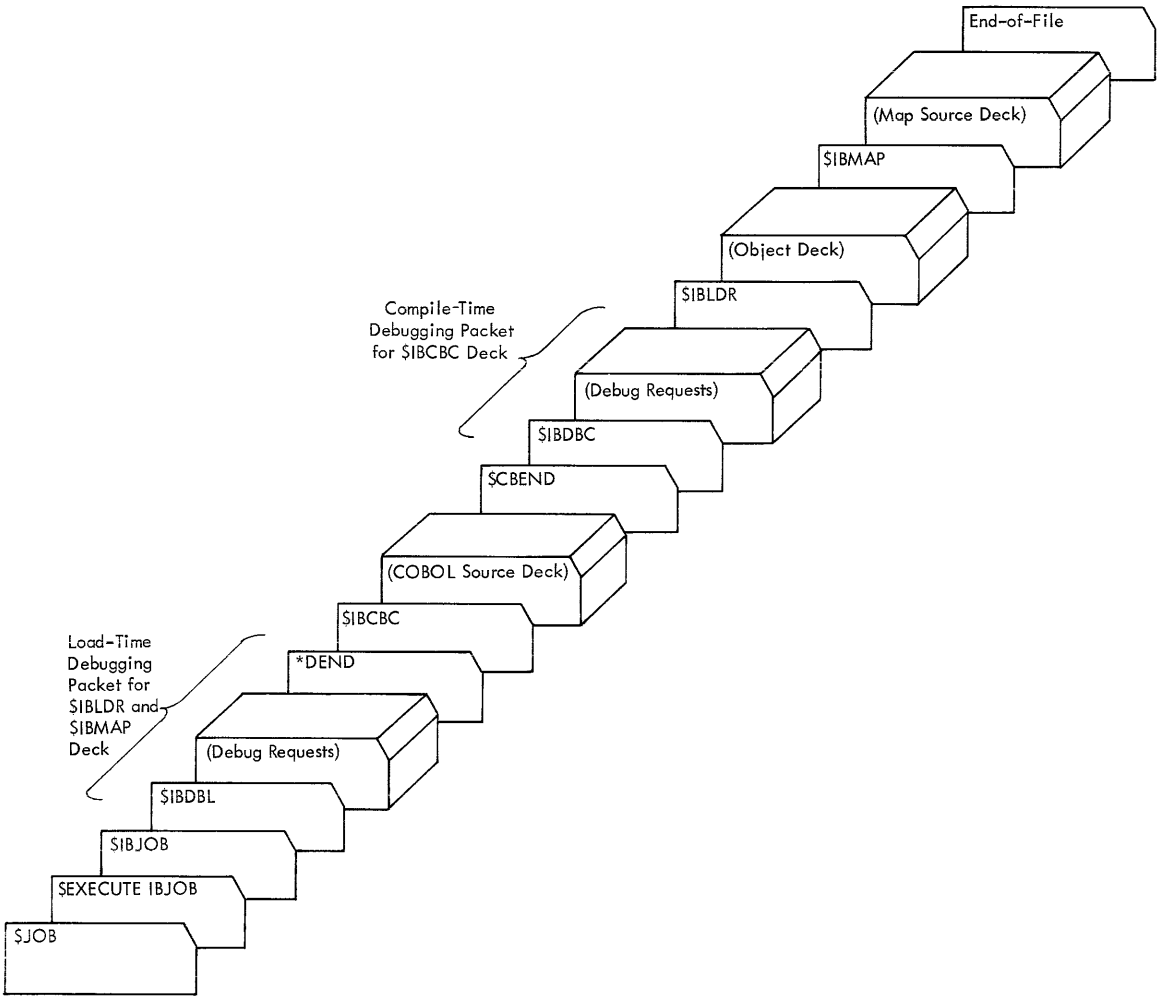


Figure 15. Sample Control Card Deck for Debugging

Relocatable Binary Decks

A deck produced by the Assembler or the FORTRAN IV Compiler for the Loader is in relocatable binary format. Such a deck, called an object deck, is written on input/output unit SYSPP1 to be punched out when the DECK option is specified on the control card for the component assembling the deck ($IBFTC, $IBCBC, or $IBMAP card). The $IBJOB card for the program within which the deck appears will also be punched out in BCD format.

The programmer can save compile and assembly time by using previously assembled object decks as part or all of a program that must be resubmitted for processing. In a case where the entire program is composed of object decks, the programmer can add control cards to the decks as follows and resubmit them for loading and execution:

```
$JOB
$EXECUTE        IBJOB
[$IBJOB card and decks as punched out]
[end-of-file card]
$STOP
```

NOTE: When an object deck follows another deck in a program, the $IBJOB card punched out with the object deck should be deleted.

However, in resubmitting binary decks, the programmer must supply again any $POOL, $GROUP, $LABEL, $ENTRY, $SIZE, $USE, $OMIT, $NAME, $ORIGIN, or $INCLUDE cards he used in the original source deck. The $ORIGIN and $INCLUDE cards must precede the decks to which they apply. In the following example the $JOB, $EXECUTE, $ORIGIN, $INCLUDE, $SIZE, end-of-file, and $STOP cards were inserted by the programmer:

```
1                       16
$JOB
$EXECUTE                IBJOB
[$IBJOB card and relocatable binary deck #1 as punched out]
$ORIGIN                 A
$INCLUDE                SIN [Subroutine from Subroutine
                        Library]
[relocatable binary deck #2 as punched out with $IBJOB card
deleted]
$ORIGIN                 A
[relocatable binary deck #3 as punched out with $IBJOB card
deleted]
$SIZE                   //=500
end-of-file card
$STOP
```

An object deck of load-time debugging reference tables cannot be obtained. For this reason, even when a job is submitted as an assembled object deck, any associated load-time debugging requests must be made as a source deck, and time for compiling the request packet must be expected.

NOTE: If the original source program did not request a debugging dictionary, the object decks punched out from the program cannot be used later with a load-time debugging request deck.

Column Binary Format

Column binary format is punched on a standard IBM card, 80 columns by 12 rows. Columns 73-80 are used for BCD identification only; they are not used by the Loader.

The remaining 72 columns are actually 24 sets of 3 columns each. Since each set of 3 columns has 12 rows, the set has 36 punch positions, corresponding to the 36 possible bits in a machine word.

The bits are read downward, starting with the first column to the left in a word. If the programmer divides the 12 rows in each column into groups of 3 bits each, he can read 4 octal numbers per column. The top bit in each group represents a 4, the middle bit a 2, and the last a 1.

For example, in Figure 16 column 19 reading downward shows 0-5-0-0 octal; column 20, 0-0-1-0; and column 21, 4-0-1-2. Put together to represent the contents of one word, the numbers are:

0 5 0 0	0 0	1	0 4 0 1 2
operation code	unused	tag	address

or

```
CLA     04012,1
```

The programmer will find it easier to read column binary when the cards are punched on an *IBM 704 Column Binary Card*, Form 121-N-2.

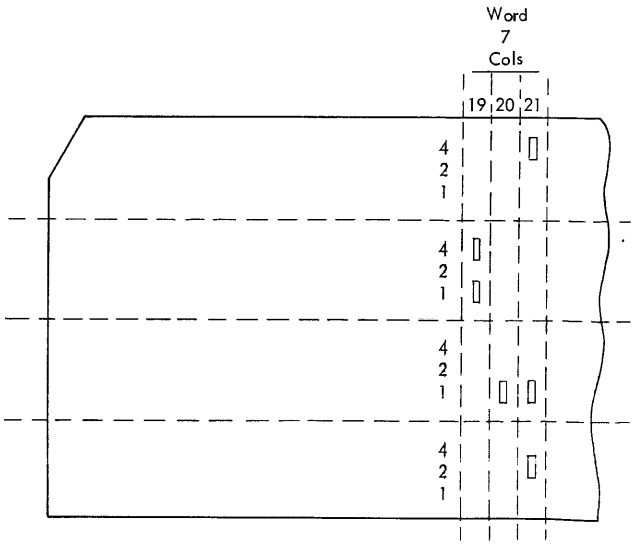


Figure 16. Binary Word Punched in Column Binary Format

The Loader processes relocatable binary program decks produced by the Assembler or the FORTRAN Compiler and combines any required subroutines from the Subroutine Library with the program decks. The program decks produced by the Assembler contain control information as well as the relocatable binary text of the program. This control information is of two types: information describing the file(s) to be used by the object program, and information that enables the Loader to resolve cross-referencing between sections of the program.

In addition to assigning absolute core storage locations to the relocatable binary text of the program and resolving cross-references, the Loader also allocates core storage for pools of input/output buffers and attaches files to the buffer pools. This is done automatically by the Loader, but a programmer can modify this procedure by using control cards.

The Loader processes one or more relocatable binary program decks, prepares one executable object program from these decks, and transfers control to the object program. In 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 can constitute a Processor application when they are grouped together. A Processor application can apply to one program deck or many, some of which may operate similarly to closed subroutines or subprograms.

A program deck may contain external names that refer to areas of coding or data within other program decks. These areas, called control sections, are accessible to other programs. 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 programmer may refer symbolically in one program to the name of a control section in another program, and the Loader performs the desired cross-referencing. External names may be designated using the MAP pseudo-operations CONTRL, ENTRY, and SAVE; they may be renamed, replaced, or deleted at load time, using Loader control cards.

Object Program Files

Input to the Loader that defines object program files comes from two sources: $FILE control cards and file dictionaries. These are normally produced by the Assembler in processing the FILE pseudo-operation. The Loader constructs file blocks from this information for the object program to use during execution.

The MAP language programmer uses FILE pseudo-operations for his file descriptions, whereas the FORTRAN user relies on the FILE routines that establish the relation between FORTRAN logical units and system units. The COBOL user describes a file by making a file description entry in the Data Division, and assigns units in the File-Control Paragraph of the Environment Division.

The file specifications generated by the Assembler on the $FILE card are described in the section "$FILE Card."

If a MAP language program contains FILE pseudo-operations but makes no reference to an IOCS Library subroutine such as .OPEN, IOCS is not loaded, and no buffer pools are attached to the files.

Loader Name Conventions

The use of alphameric literals as external identifiers of object program quantities is basic to the design and mechanization of the Loader. Three types of names are used in the Loader:

1. Deck names, which identify decks and may be used to identify or qualify control section names within a deck.
2. Control section names, which identify data and procedure sections within the program. When using Loader control cards, named sections in one deck may be replaced by a section with the same name in another deck. These sections may also be renamed or deleted from the program.
3. File names, which identify files within the program.

Deck Name Rules

The use of deck names and the rules for forming deck names are:

1. A deck name is composed of six or fewer alphameric characters, excluding parentheses, commas, slashes, quotation marks, equal signs, and embedded blanks.
2. In producing the binary deck, the Assembler places the contents of the deckname field of the Compiler and Assembler cards ($IBMAP, $IBCBC, and $IBFTC) in the deckname field (columns 8-13) of the BCD cards that delimit the major sections of a program deck. The names of the BCD cards are $IBLDR, $FDICT, $TEXT, $CDICT, $DDICT, and $DKEND.

3. The deck name may be punched in columns 8-13 of any other Loader control card, but it is ignored by the Loader.

4. The deck name defines a control section that encompasses the entire deck; therefore, the entire deck is a control section.

5. 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.

6. The deck names of routines in the Subroutine Library may not be used as control section name qualifiers.

7. A deck name may not be changed by a $NAME card. However, the control section with that name may be renamed by a $NAME card.

Control Section Rules

The use of control sections and the rules for forming control section names are:

1. A control section name is composed of six or fewer alphameric characters. Alphameric characters that cannot be used in the control section name are parentheses, commas, slashes, quotation marks, equal signs, and blanks. A control section name is always left-justified before processing or comparison, and unused trailing positions are filled with blanks.

2. A control section is a bounded section of coding; 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.

3. A real control section is any control section referred to in a deck that has a relative location assigned within that deck.

4. A virtual control section is any control section referred to in a deck that has no known origin or length in that deck. A virtual control section must be supplemented by a real control section with the same external reference name in either another input deck or in a deck in the Subroutine Library. If a virtual control section is not supplemented by a real control section, an error message is written on the system output unit.

5. Normally, if the six-character external reference names of two or more control sections are identical, the Loader retains the first control section encountered and deletes the control sections with duplicate names.

6. In the absence of explicit inclusion through $USE cards, the first real section with a given name that is physically encountered while loading is retained, and all succeeding occurrences of it are deleted. All references to the given name are adjusted to refer to the storage assigned to the retained section, including any ORG pseudo-operations that may have referred to the deleted section.

7. Explicit inclusion of two control sections with the same name (by using deck name qualifications on a $USE card) results in a multiple definition of that section. Consequently, an error message is written on the system output unit.

8. Each control section that is referred to by text must be defined (assigned an absolute location by the Loader) or execution will not be allowed. For example, if a reference is made to a section mentioned on a $OMIT card and no other section with the same name is encountered, an error message is written on the system output unit.

9. Control sections can be nested (i.e., control sections can be placed within the boundaries of other control sections), but each inner nest must be entirely within the next outer level of nesting. If an outer level of nesting is deleted, all control sections within the boundaries of this nest are deleted. If an inner level of nesting is deleted or inserted, any references to locations between the end of the inner section and the end of the outer section are not adjusted. In Figure 16A, for example, if control section A is deleted from within control section B, all references to the locations in area C will be incorrect. They will, in fact, refer in the final absolute program to instructions in control section D. This problem does not arise with an EVEN control section which coincides with the beginning of another control section, since the EVEN is not treated as a nested control section in this case.

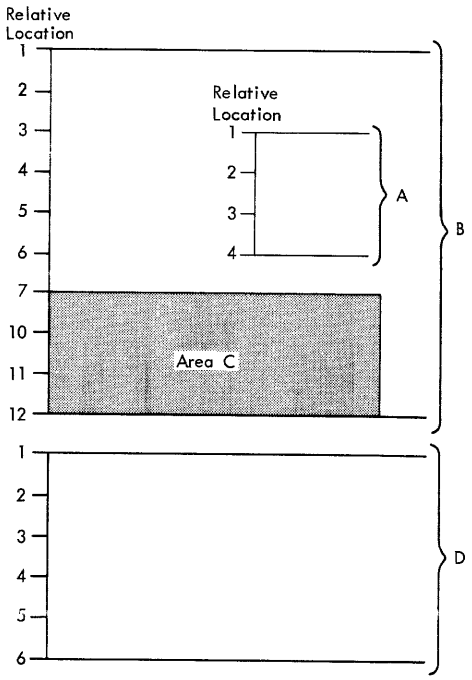


Figure 16A. Nested Control Sections Where Inner Level is Deleted

10. Explicit inclusion of a control section through specification of a $USE card does not necessarily force the inclusion of all embedded control sections.

11. A subroutine in the Subroutine Library is called automatically if a control section name in a Library subroutine is identical to that of a virtual control section and no real control section with that name is contained in any input program deck.

12. Control sections of routines in the Subroutine Library may not be renamed.

13. A control section name specified by an ENTRY, CONTRL, or SAVE pseudo-operation should not be the same as the name of the deck that contains it.

14. If an EVEN appears within a control section, it must be at the same relative location as the beginning of the control section.

15. Text that is placed in a control section by means of an ORG pseudo-operation (as in the case of data entered in a COMMON block by a FORTRAN IV BLOCK DATA subprogram) is never deleted. Instead, it is loaded into the retained control section. (See rule 6 above.)

File Name Rules

The following list defines the rules for the formation and usage of file names:

1. A file name is composed of up to 18 alphameric characters, excluding parentheses and quotation marks.

2. Whenever a file name appears on a Loader control card, it must be enclosed in quotation marks (4-8 punch). If the file name is qualified by a deck name,

the entire expression is enclosed in quotation marks and the file name is enclosed in parentheses.

3. A file may be renamed through specification on a $NAME card. If the new name that the programmer species on the $NAME card does not already exist, the programmer must insert a $FILE card containing this name.

4. File names cannot be specified on $USE or $OMIT cards.

5. If a file is renamed, any control card that refers to the old name is ignored.

6. If the 18-character names of two or more files are identical within a program, the Loader retains the information contained in the first $FILE card encountered and ignores any subsequent $FILE cards with the same file name.

Component Control Card for the Loader

The following card instructs the IBJOB Monitor that only the Loader is needed to process the deck headed by the card.

$IBLDR Card

The first card in a relocatable binary deck is the $IBLDR card in BCD format. The $IBLDR card is generated by the Assembler.

The format of the $IBLDR card is:

1	8	16
$IBLDR	deckname	[, options]

where deckname identifies the deck to be processed. The deck name must be six or fewer alphameric characters. Characters that cannot be used in the deck name are parentheses, commas, slashes, quotation marks, equal signs, and embedded blanks.

The variable field begins in column 16. The options of the variable field are described in the following text. These variable field options can be changed by the programmer.

Input Options

$$\left[,\left\{\begin{matrix}\underline{\text{NOLIBE}}\\ \text{LIBE}\end{matrix}\right\}\right]$$

NOLIBE—The object program is in the current input file.

LIBE—The object program is in the Subroutine Library.

When LIBE is specified, the application must consist entirely of object programs from the Subroutine Library. LIBE and NOLIBE specifications cannot be used within the same application. If neither LIBE nor NOLIBE is specified, it is assumed that the object program is in the current input file (NOLIBE).

Text Options

$$\left[,\left\{\begin{matrix}\underline{\text{TEXT}}\\ \text{NOTEXT}\end{matrix}\right\}\right]$$

TEXT—When this option is specified, the Loader loads the text section of the deck that follows.

NOTEXT—The sections of the deck that contain control information are loaded, but the text section is not loaded.

If neither TEXT nor NOTEXT is specified, the Loader loads the text section of the deck that follows (TEXT).

Loader Control Cards

This section provides the format specifications for the control cards that the Loader processes. These control cards describe file and program loading modifications for an entire object program and, therefore, are not required for most Processor applications. The Loader control cards may be used to:

1. Override file or label descriptions that appear in the object program.
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.
4. Modify control section names or file names.
5. Delete control sections.

$FILE Card

The Assembler normally generates a $FILE card in processing a FILE pseudo-operation. But, since the Loader retains the first $FILE card it reads for any given file as its proper definition, the programmer may override certain options in an Assembler-generated $FILE card by placing a $FILE card of his own at the beginning of the deck involved. These rules, however, must be followed:

1. Any file specifications that are not changed must be repeated on this card.
2. The file usage option may not be changed.
3. The mode option may not be changed.
4. The block size option may be changed only within the limits of the MIN and MULTI specifications in the IBMAP FILE pseudo-operation.
5. The unit assignment option may not be changed to a non-Hypertape unit if HYPER was specified in the IBMAP FILE pseudo-operation.
6. The unit assignment option may not be changed to specify card equipment if the no-Hollerith-conversion option (NOHCVN) was specified on the IBMAP FILE pseudo-operation. The unit may not be changed to specify tape, disk, or drum equipment if the required Hollerith-conversion option (REQHCV) was specified.

A $ETC card can be used to extend the variable field of a $FILE card.

The format of the $FILE card is:

1	16
$FILE	'filename' [options]

where 'filename' is an alphameric name of 18 or fewer characters that identifies the file. The 'filename' must be enclosed by quotation marks (4-8 punch), and it must begin in column 16. The specifications for the options may be entered in any order thereafter. Specifications are separated from the file name and from each other by commas.

Unit Assignment Options

[, unit 1, unit 2]

The unit 1 specification is the primary unit; the unit 2 specification is the secondary unit used for reel switching. Unit specifications are described in the section "Unit Assignment."

Mounting Options

[, MOUNT / DEFER / READY / or / MOUNTi / DEFERi / READYi]

MOUNT — The message is printed before execution, and a stop occurs for the required operator action.

DEFER — The operator message and stop are deferred until the file is opened.

READY — The message is printed before execution, but a stop does not occur. System units are normally given the READY option if a mounting option is not specified.

The operator is notified by an on-line message of the impending use of an input/output unit. These options refer to both unit 1 and unit 2. They govern the type of message to be printed and the operator action required when an input/output unit is to be put into use.

The operation for the alternate options is the same as the MOUNT, DEFER, or READY options except that i refers to a particular unit, as follows:

i=1	unit 1
i=2	unit 2

If one of these units is specified, it supersedes any general mounting option specified for unit i. The following option, for example, causes the MOUNT operation for unit 1 and the DEFER operation for unit 2:

```
MOUNT,DEFER2
```

If none of the options is specified, the message is printed before execution, and a stop occurs for the required operator action (MOUNT).

File List Options

[, LIST / NOLIST]

LIST — The file appears in the operator's mounting instructions.

NOLIST — The file does not appear in the operator's mounting instructions.

If neither LIST nor NOLIST is specified, the file appears in the operator's mounting instruction (LIST).

File Usage Options

[, INPUT / OUTPUT / INOUT / CHECKPOINT / or / CKPT]

INPUT — The file is an input file.

OUTPUT — The file is an output file.

INOUT — The file may be either an input file or an output file. The object program sets the appropriate bits in the file block.

CHECKPOINT or CKPT — The file is a checkpoint file.

If neither INPUT, OUTPUT, INOUT, CHECKPOINT nor CKPT is specified, the file is an input file (INPUT).

Block Size Option

[, BLOCK / or / BLK = xxxx]

The symbol xxxx represents a number (0000-9999) that specifies the block size for this file. The field may be omitted provided the file is included in a pool or group where the block size can be determined.

If SEQ, SEQUENCE, or CKSUM is specified in the $FILE card, the block size number must include the count for the one-word checksum and the block sequence word.

Activity Option

[,ACT=xx]

The symbol xx represents a number (00-99) that specifies the relative activity of this file with respect to other files. The higher the number, the more active the file. If this field is omitted, the activity is assumed to be 01. This value is used in determining the number of output buffers to assign to each buffer pool in the object program.

Reel Switching Options for Unlabeled Files

[, ONEREEL / MULTIREEL / or / REELS]

ONEREEL — Reel switching should not occur.

MULTIREEL or REELS — Reel switching should occur. The publication *IBM 7090/7094 IBSYS Operating System: Input/Output Control System,* Form C28-6345, contains a discussion of reel switching facilities. Every output file switches reels if an end-of-tape condition occurs.

If no option is specified, the Loader assumes that reel switching should not occur (ONEREEL).

Reel Searching Options for Labeled Files

[, NOSEARCH / SEARCH]

NOSEARCH — If an incorrect label is detected when opening an input file, IOCS stops for operator action.

SEARCH — IOCS initiates a multireel searching procedure for the file with the desired label.

If neither NOSEARCH nor SEARCH is specified, it is assumed that IOCS will stop for operator action (NOSEARCH).

File Density Options

[,{HIGH / LOW / 200 / 556 / 800}]

Trailer label operations are always performed in the same density as that of the file.

HIGH — The tape density switch is assumed to be set so that the execution of an SDH instruction will result in the correct density being used.

LOW — The tape density switch is assumed to be set so that execution of an SDL instruction will result in the correct density being used.

200 — The tape density switch is assumed to be set so that the execution of an SDL instruction will result in a file recording density of 200 cpi.

556 — The tape density switch is assumed to be set so that the execution of an SDL instruction will result in a file recording density of 556 cpi.

800 — The tape density switch is assumed to be set so that the execution of an SDH instruction will result in a file recording density of 800 cpi.

This field specifies the density at which the file is to be read or written. If a system unit is assigned to this file, the specification for this unit supersedes any of the preceding specifications.

If neither HIGH, LOW, 200, 556, nor 800 is specified, the tape density switch is assumed to be set so that execution of an SDH instruction will result in the correct density being used (HIGH).

Mode Options

[,{BCD / BIN / MXBCD / MXBIN}]

BCD — The file is in BCD mode.

BIN — The file is in binary mode.

MXBCD — The file is in mixed mode, and the first record is BCD.

MXBIN — The file is in mixed mode, and the first record is binary.

If neither BCD, BIN, MXBCD, nor MXBIN is specified, it is assumed that the file is in BCD mode (BCD).

The MXBCD and MXBIN options may not be specified for a file unless a look-ahead word is attached to each physical record of the file. Since this word is not written by IOCS and cannot be programmed using FORTRAN IV or COBOL languages, the MXBCD and MXBIN options can be used with compiled programs only when output it handled by programs coded in the MAP language.

Label Density Options

[,{SLABEL / HILABEL / LOLABEL / FLABEL}]

SLABEL — All header label operations are performed in the installation standard label specified density.

HILABEL — All header label operations are performed in high density.

LOLABEL — All header label operations are performed in low density.

FLABEL — All header label operations are performed in the same density as that of the file.

If neither SLABEL, HILABEL, LOLABEL, nor FLABEL is specified, the standard label density specification, defined by the assembly parameter SLABEL, is used to process labels. As distributed, the standard specification is high density. An installation can change the standard specification by changing the assembly parameter SLABEL, which is in the Loader.

Only the use of the MAP pseudo-operation LABEL or of a $LABEL card denotes a labeled file, whether one of the label density options is specified or not.

Block Sequence Options

[,{NOSEQ / SEQ / or / SEQUENCE}]

NOSEQ — This specifies that the block sequence number is not written or checked.

SEQ or SEQUENCE — If reading, check the block sequence number. If writing, form and write a block sequence number.

If neither NOSEQ, SEQ, nor SEQUENCE is specified, the block sequence number is not written or checked (NOSEQ).

Check Sum Options

[,{NOCKSUM / CKSUM}]

NOCKSUM — This specifies that the checksum is not written or checked.

CKSUM — If reading, check the checksum. If writing, form and write the checksum. CKSUM can only be specified when SEQ or SEQUENCE has been specified in the $FILE card.

If neither NOCKSUM nor CKSUM is specified, the checksum is not written or checked (NOCKSUM).

Checkpoint Options

[,{NOCKPTS / CKPTS}]

NOCKPTS — This specifies that no checkpoints are initiated by this file.

CKPTS – Checkpoints are initiated by this file.

If neither NOCKPTS nor CKPTS is specified, no checkpoints are initiated (NOCKPTS).

Checkpoint Location Option

[, AFTERLABEL]

AFTERLABEL – Checkpoints are written following the label when a reel switch occurs. This option can only be used when the file is labeled.

If the CKPTS option is specified and this field is blank, checkpoints are written on the checkpoint files when a reel switch occurs.

File Close Options

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

SCRATCH — The file is rewound upon termination of the application.

PRINT — The file is to be printed. It is rewound and unloaded upon termination of the application. PRINT appears in the removal message that is printed on-line at the end of execution.

PUNCH — The file is to be punched. It is rewound and unloaded upon termination of the application. PUNCH appears in the removal message that is printed on-line at the end of execution.

HOLD — The file is to be saved. It is rewound and unloaded upon termination of the application. HOLD appears in the removal message that is printed on-line at the end of execution.

If the unit assigned is the system input unit, the system output unit, or the system peripheral punch unit, there is no rewind and no message is printed.

If neither SCRATCH, PRINT, PUNCH, nor HOLD is specified, the file is rewound upon termination of the application (SCRATCH).

Starting Cylinder Number Option

[,{CYL / or / CYLINDER} = {x / xxx}]

The symbol x is the number (0-9) of the starting cylinder number of this file if it is on drum storage. If the file is on disk storage, the symbol xxx is the number (000-249) of the starting cylinder number. The equal sign is required. The programmer must specify the starting cylinder number when disk-storage or drum storage is specified for the file. This field does not apply if a system unit function that is assigned to disk storage or drum storage is specified for the file.

Cylinder Count Option

[,{CYLCOUNT / or / CYLCT} = {xx / xxx}]

xx is the number (00-10) of consecutive cylinders used by this file if it is on drum storage. If the file is on disk storage, xxx is the number (000-250) of consecutive cylinders used by this file. The equal sign is required. The programmer must specify the cylinder count when disk storage or drum storage is specified for the file. This field does not apply if a system unit function that is assigned to disk storage or drum storage is specified for the file.

Disk and Drum Write Checking Option

[, WRITECK]

Write checking is performed after each disk and drum write sequence for this file.

Hypertape Reel Switching Options

[,{HNRNFP / HRFP / HRNFP / HNRFP}]

For reel switching to occur, the programmer should specify whether the Hypertape is to be rewound and/or protected.

HNRNFP — Designates that the Hypertape is not to be rewound or file protected.

HRFP — Designates Hypertape rewind and file protection.

HRNFP — Designates Hypertape rewind, but no file protection.

HNRFP — Designates that the Hypertape is not to be rewound, but is to be file protected.

If no option is specified, HNRNFP is assumed by the Loader.

$LABEL Card

The $LABEL card provides label information for the file. Omission of this control card indicates that the file is unlabeled. 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 by using adjacent commas (,,). The last field is considered to be 18 characters long, with embedded blanks allowed. Files which are assigned units SYSOU1, SYSOU2, SYSPP1, or SYSPP2 may not be labeled.

All pertinent information must be on this control card. $ETC cards are not allowed.

The format of the $LABEL card is:

1	16
$LABEL	'filename', [{serial / or / home address}], [reel]
	, [{date / or / days}], [name]

where 'filename' is an alphameric literal of 18 or fewer characters that identifies the file. It must be enclosed by quotation marks (4-8 punch). The 'filename' must start in column 16.

Serial Number Option

[,{serial / or / home address}]

serial — If a tape label is desired, serial represents an alphameric field of five or fewer characters. Standard input labels are checked against this serial number if it is present. Standard output labels for this file will contain this serial number only if a reel number greater than 1 is specified in the reel sequence number field. Output serial numbers are normally taken from the label already present on the tape on which the first reel of the file is written.

home address – If a labeled disk file or drum file is desired, this field must contain two BCD characters that specify the home address −2.

The IBJOB Processor itself requires a home address of 00 when using disk or drum as a file during processing.

NOTE: The serial number option does not change disk or drum format, but only tells the object program the content of the format.

Reel Sequence Number Option

The reel sequence number option is:

[, reel]

This is a numerical field of four or fewer characters. It specifies the reel sequence number of the first reel of a file. When the field is omitted, the sequence number is assumed to be 1 for an output file or 0 for an input file. The reel sequence number is adjusted at object time to reflect reel switching, and it is checked in standard input labels. If the field is omitted for an input file, the test is bypassed. If a disk label or drum label is desired, this field must be omitted by using two adjacent commas.

Retention Cycle Options

$$\left[,\left\{\begin{matrix}\text{date}\\ \text{or}\\ \text{days}\end{matrix}\right\}\right]$$

date – A date can be entered in this field. The format is y/d where y is a one-digit or two-digit number indicating the year, and d is a number of three or fewer digits indicating the day of the year. The slash is used to separate the two numbers. This entry is not checked and is used merely to provide additional information in the label.

days – This is a numeric field of four or fewer characters. It specifies the number of days a tape is to be retained from the date it is written. An attempt to write a labeled file on this tape before the end of the retention period results in an error message. If the field is omitted, a value of zero is assumed.

File Identification Option

[, name]

This is an alphameric literal of 18 or fewer characters that specifies the file identification name in the label. This name is checked with the name in the input label. This field must follow the last comma on the card. If this field is omitted for an input file, the file identification is not checked. For an output file, this name is placed in the output label. If this field is omitted for an output file, the file identification on the existing output label is set to zeros.

$POOL Card

The $POOL card designates the files that are to share common buffer areas. A $ETC card can be used to extend the variable field for a $POOL card. The format of the $POOL card is:

1	16
$POOL	'filename$_1$' . . . 'filename$_n$'
	$\left[,\left\{\begin{matrix}\text{BLOCK}\\ \text{BLK}\end{matrix}\right\}=\text{xxxx}\right]$ [, BUFCT=xxx]

where each 'filename' is the name of a file to be included in the pool. Each 'filename' is an alphameric literal of 18 or fewer characters and is enclosed in quotation marks (4-8 punch). Deck name qualification is meaningless, since the Loader assigns only one file block for each unique file name in the application. An error message results if a file specified as NOPOOL in the IBMAP FILE pseudo-operation is also specified on a $POOL or $GROUP card, or if any other file dictionary entries for this named file require a pool. In this event, the NOPOOL option is ignored.

Block Size Option

$$\left[,\left\{\begin{matrix}\text{BLOCK}\\ \text{BLK}\end{matrix}\right\}=\text{xxxx}\right]$$

The symbol xxxx represents a number (0000-9999) that specifies the block size for this pool. If this field is omitted, the pool block size used is the same as the largest block size of a file in the pool.

Buffer Count Option

[, BUFCT=xxx]

xxx is a number (001-999) that specifies the number of buffers to be assigned to the pool. It must be equal to or larger than the open count of the pool specified on the $GROUP card. If this field is omitted, the Loader attempts to assign at least two buffers to each file.

For further information concerning buffer pools, see "Input/Output Buffer Allocation."

$GROUP Card

The $GROUP card is used to allocate buffer areas and to specify how the buffers are to be shared by a group of files. A $ETC card can be used to extend the variable field of the $GROUP card. If the $GROUP card is not used, the Loader attempts to assign at least two buffers to each file.

The format of the $GROUP card is:

1	16
$GROUP	'filename$_1$' . . . 'filename$_n$'
	[, OPNCT=xx] [, BUFCT=xxx]

where each 'filename' is the name of a file to be included in the group. Each 'filename' is an alphameric literal of 18 or fewer characters and is enclosed in quotation marks. Deck name qualification is meaningless, since the Loader assigns only one file block for each unique file name in the application. An error message results if a file specified as NOPOOL in the IBMAP FILE pseudo-operation is also specified on a $POOL or $GROUP card, or if any other file dictionary entries for this named file require a pool. In this event, the NOPOOL option is ignored.

Open Count Option

[, OPNCT=xx]

The symbol xx represents a number (01-99) that specifies the number of files within the group that are open concurrently during the execution of the program. This count determines the minimum buffer count necessary for processing the group of files. If this field is omitted, the count is assumed to be equal to the number of files in the group.

Buffer Count Option

[, BUFCT=xxx]

The symbol xxx represents a number (001-999) that specifies the number of buffers to be assigned to this group. It must be equal to or larger than the open count of the group. If this field is omitted, the Loader attempts to assign at least two buffers to each file.

For further information concerning buffer pools, see "Input/Output Buffer Allocation."

$USE Card

The $USE card provides a method of specifying a particular control section that is to be used with the object program at execution time. Normally, the first occurrence of a control section is retained, and sections with the same name in other decks are deleted. The $USE card causes the control section in a specified deck to be retained and all control sections with the same name in other decks to be deleted. A $ETC card can be used to extend the variable field of the $USE card.

The format of the $USE card is:

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

where the fields in the variable field consist of alphameric literals. The first six or fewer characters of the field are the deck name. Following the deck name is the external name of the control section consisting of six or fewer characters enclosed in parentheses.

$OMIT Card

The $OMIT card provides a method of deleting a control section from a specific deck or from all decks in which it appears. To delete the control section from a single deck, the external name for the control section must be preceded by the name of the deck. To delete the control section from all decks in which it occurs, it is only necessary to specify the external name for the control section. A $ETC card can be used to extend the variable field of the $OMIT card.

The format of the $OMIT card is:

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

where the fields in the variable field consist of alphameric literals. A literal may contain just the external name (six or fewer characters) of a control section, or it may contain a deck name of six or fewer characters followed by the external name of a control section enclosed in parentheses.

$NAME Card

The $NAME 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 unique name. This control 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. A $ETC card can be used to extend the variable field of the $NAME card.

The format of the $NAME card is:

1	16
$NAME	{deckname(exname)=exname, . . . {exname=exname, . . . {'deckname(filename)' {='filename', . . . {'filename'='filename', . . .

where the 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, the name and the qualifier 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 is replaced by the name on the right only in the deck named.

$SIZE Card

The $SIZE card allows the programmer to specify the size of blank COMMON.

The format of the $SIZE card is:

1	16
$SIZE	//=n

where n is a decimal number that specifies the size of blank COMMON. The double slash and the equal sign are required. If n is less than the largest blank COMMON required by an object program and its subroutines, this specification is ignored by the Loader.

Blank COMMON is located in such a way that the last location occupied is the highest available core storage location as defined by the IBJOB communication location IBJCOR. The starting location of blank COMMON is always an even core storage location.

$ETC Card

The $ETC card may be used to extend the variable field of a $FILE, $POOL, $GROUP, $USE, $OMIT, $NAME, or another $ETC card. It may not be used to extend the variable field of a $LABEL card. The presence of a $ETC card in-

dicates that more information is associated with the control card immediately preceding the $ETC card. Therefore, the order in which $ETC cards occur is very important, and all $ETC cards for a particular control card must immediately follow that control card. Information appearing in the variable field may be any information allowed on the control card that the $ETC card is extending, but an individual field cannot be split between two cards.

The format of the $ETC card is:

1	16
$ETC	variable field information

Input/Output Buffer Allocation

The rules of storage allocation pertaining to input/output buffer pools and the effect of $POOL and $GROUP cards on such allocation procedures are described in this section.

The Loader normally assigns core storage not used by an object program as input/output buffers. Storage is not allocated by the Loader when the programmer is programming at the Input/Output Executor (IOEX) level or when the programmer generates his own file control blocks. Since each object program is entirely relocatable and the amount of usable core storage can only be determined by the Loader, the Loader does the following:

1. The storage not assigned to the system or to the object program is apportioned as input/output buffers for the files of the object program.

2. The IOCS initialization sequences of DEFINE and ATTACH are generated and loaded in front of the object program. At the end of these calling sequences, an instruction is generated to transfer control to the first instruction to be executed.

3. IOCS is coded so that any IOCS-detected error that would have resulted in a machine stop causes the calling of an error routine that closes all files in use and proceeds to the next job segment or Processor application.

4. If the Loader encounters a file dictionary entry specifying NOPOOL, the named file is processed for unit assignment only. It is not attached to a buffer pool, and the file does not appear in the generated file list.

General Buffer Assignment

In the absence of $POOL and $GROUP cards, the rules of input/output buffer storage allocation are:

1. Each file is a separate reserve group.

2. A different buffer pool is created for each distinct blocking size encountered. All files that have the same blocking size are assigned to the same pool, regardless of whether they are input or output files.

3. Storage is assigned to each pool in three steps:
   a. The pool is given one buffer for each file. If available storage is not sufficient for this allocation, execution is terminated, and an appropriate error message is printed.
   b. An additional buffer for each file is allocated to the pool. If available storage does not permit this allocation, a weighing factor is formed from the number of additional buffers desired, multiplied by the total activity of the files in that pool. The pool with the largest weighing factor is then assigned one buffer, if possible. If this assignment is not possible, the weighing factor of the pool is set to zero. If this assignment is possible, it is made and the weighing factor of the pool is reduced. The pool that now has the largest weighing factor is given a buffer. This continues until all the weighing factors are reduced to zero.
   c. The remainder of storage, if any, is apportioned by the ratio of the output activity of each pool, i.e., the sum of the activity of each output file in that pool compared with the sum of the total output file activity.

Storage is allocated first to the pool with the greatest buffer size, so that the remainder may be assigned to a smaller pool.

4. The amount of storage used by each buffer pool is:

```
BUFCT*(BUFSIZ+2)+2
```

Buffer Assignment with $POOL and $GROUP Cards

$POOL and $GROUP cards may be used to direct the assignment of input/output buffers to certain files, pools, or groups. Normally, because each program requires a relatively small amount of the total core storage, sufficient storage is available to assign many buffers to each pool. However, if the program is large or the number of files is great, the programmer may, by using $POOL and $GROUP cards, specify a more efficient assignment of buffers. The use of $POOL and $GROUP cards is not considered the normal case.

1. $POOL cards cause all files mentioned on the control card to be assigned to the same buffer pool. If any of the files are also specified on $GROUP cards, all files specified in that group are automatically associated with the 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 is specified on a $POOL card, that pool is given exactly that number of buffers. Checks are made to ensure that the specified count is sufficient. A sufficient count is defined as the sum of the number of nongrouped files in the pool plus the buffer counts

of all groups of that pool. If a buffer count is not specified, the pool is allocated buffers as described in the section "General Buffer Assignment."

3. If block size is not specified on a $POOL card, the size is the maximum block size of the files assigned to that pool.

4. $GROUP cards cause the specified files to be grouped under a single reserve group control word. If a buffer count is specified, this count is used and must be at least equal to the open count of those files that are open concurrently. If the open count is not specified, it is equal to the number of files in the group. If the buffer count is not specified, extra buffers are allocated to the pool to which that group is assigned. This is described in item 3c of the section "General Buffer Assignment."

5. Groups can be created within specified pools by naming at least one of the files in the group on a $POOL card, as well as on the $GROUP card.

Unit Assignment

This section describes the unit assignment specifications for the $FILE card, the use of intersystem unit assignment, and the order in which files are assigned to units.

Unit Assignment Notation

The following notation is used to explain unit assignment specifications:

NOTATION	EXPLANATION
X	a real channel (A through H)
P	a symbolic channel (S through Z)
I	an intersystem channel (J through Q)
k	a unit number (0 through 9)
a	the access mechanism number (0 or 1)
m	the module number (0 through 9)
s	the Data Channel Switch (also called interface) (0 or 1)
M	the model number of a 729 Magnetic Tape Unit (II, IV, V, or VI)
D	a 1301/2302 Disk Storage
H	a 7340 Hypertape Drive
N	a 7320 Drum Storage

Unit Assignment Specifications

NOTATION	EXPLANATION
blank	Use any available 729 Magnetic Tape Unit.
M	Use any available 729 Magnetic Tape Unit, model M, where M is either II, IV, V, or VI.
X	Use any available 729 Magnetic Tape Unit on channel X.
P	Use any available 729 Magnetic Tape Unit on channel P.
X(k)	Use the kth available 729 Magnetic Tape Unit on channel X. The parentheses are required.
PM	Use any available 729 Magnetic Tape Unit, model M, on channel P.
P(k)M	Use the kth available 729 Magnetic Tape Unit, model M, on channel P. The parentheses are required.
I(k)	Use the kth available 729 Magnetic Tape Unit on channel I. The parentheses are required. This specification can be used for input and output units.
I(k)M	Use the kth available 729 Magnetic Tape Unit, model M, on channel I. The parentheses are required. This specification can be used only for output units.
I(k)R	Use the kth available 729 Magnetic Tape Unit on channel I, and release the unit from reserve status after the application has been completed. The parentheses are required.
XDam/s	Use 1301/2302 Disk Storage on channel X, access mechanism number a, module number m, and data channel switch s.
XNam/s	Use 7320 Drum Storage on channel X, access mechanism number a (0), module number m (0, 2, 4, 6, or 8), and data channel switch s.
XHk/s	Use a 7340 Hypertape Drive on channel X, unit number k, and data channel switch s.
IN, IN1, IN2	Use the system input unit (SYSIN1) and the alternate system input unit (SYSIN2) as the primary and secondary units, respectively.
OU, OU1, OU2	Use the system output unit (SYSOU1) and the alternate system output unit (SYSOU2) as the primary and secondary units, respectively.
PP, PP1, PP2	Use the system peripheral punch unit (SYSPP1) and the alternate system peripheral punch unit (SYSPP2) as the primary and secondary units, respectively.
UTk	Use the system utility unit, number k.
RDX	Use the card reader on channel X. If a card reader is specified, OPTHCV or REQHCV must be specified in the FILE pseudo-operation.
PRX	Use the printer on channel X.
PUX	Use the card punch on channel X. An asterisk in the unit 2 field indicates that the secondary unit for a file is to be a unit on the same channel and of the same model type as the primary unit. This unit, if available, is assigned after all other unit assignments have been made.
INT	The file is an internal file.
NONE	No units are assigned. A file control block is generated but does not refer to a unit control block.

Intersystem Unit Assignment

To provide for the passage of data through a series of related applications, intersystem unit assignments are made. The use of this specification allows an object program to write an intermediate output file on a unit and to reserve that unit for later use as input or output for an object program in a different application. This is done by using symbolic channels J through Q.

When a $FILE card for an input file is encountered by the Loader, the primary intersystem channel and

relative unit, if present, are used for scanning the unit control blocks to find a matching intersystem unit, i.e., a unit in reserve status that has the same intersystem channel designation and relative unit. If a match is found, the input file from the $FILE card is assigned to that physical unit. If a match is not found, a tape assignment error is noted and execution is not allowed.

For an intersystem unit designated as output, the same scanning of the unit control blocks is performed. However, unlike the input file processing, if a match is not found, the intersystem channel is treated as a symbolic channel (S through Z) and a reserve flag and indicative data are placed in the unit control block.

If an error occurs, either prior to or during execution of a program that uses intersystem output units, these units are removed from reserve status and are returned to the availability chain. Subsequent references to these intersystem units as input causes a tape assignment error, and an error message is written on the system output unit.

Order of Assignment

Files are assigned to units in the following order:

1. All files on system or card units are assigned first.
2. Input files on intersystem channels are assigned. If the designated unit does not already exist in reserve status, a tape assignment error is noted and execution is not allowed.
3. Files on specified (real) channels are assigned.

If during steps 1, 2, and 3 there are insufficient units on the requested channels, a message is printed indicating that the object-time tape assignment can not be completed because of insufficient units on the specified channels.

4. Output files on intersystem channels are assigned. Those intersystem channels with the largest requirement are processed first. Starting with the highest real channel, the channels are processed to determine whether the intersystem channel requirements can be met. When a real channel is found that contains sufficient available units for the intersystem channel, that real channel is chosen. If there is no real channel that can meet the intersystem channel requirements, a real channel is chosen to assign as many of the intersystem units to the channel as possible. The remaining intersystem units are now assigned by again finding the intersystem channel with the greatest requirements and matching it with a real channel.
5. Files on symbolic channels are assigned. The procedure is the same as that described under intersystem assignment.
6. Only files with model specifications are assigned. Units are chosen, starting at the highest unit number of the first available channel.
7. Files with omitted specifications are assigned. If, during steps 4, 5, 6, or 7, there are insufficient units available for assignment, a message is printed indicating that object time tape assignment can not be completed because of insufficient units on the system.
8. Secondary units are assigned with the primary units and on the same channel as the primary units. When the unit 2 specification is omitted, the secondary unit is the same as the primary unit. If the primary unit is a Hypertape drive, the secondary unit must also be a Hypertape drive.

Overlay Feature of the Loader

The overlay feature provides a method for processing programs that exceed the capacity of core storage. The programmer divides the program to be executed into links. A link can contain one or more program decks. One of the links (called the main link) is loaded into core storage with the overlay subroutine and the tables required for program execution, and it remains in core storage throughout the execution of the program. The Loader writes the other links (called dependent links) on an external storage unit. The external storage unit can be disk storage, drum storage, or magnetic tape. The dependent links are written in a scatter-load format and have block sizes of 464 words for 729 tape and 1301 disk storage, 524 words for 7320 drum storage, or 969 words for 2302 disk storage, depending on which device the Loader is assembled for.

The Overlay Structure

Figure 17 is an example of the structure of an overlay application. In Figure 17, the vertical lines represent links into which the program is divided. The horizontal lines from which the vertical lines proceed indicate the logical origins of the links.

Note that, as Figure 17 implies, a continuous chain of links is always in core storage, from the deepest link required, back through whatever links precede it, to the main link. For example, in Figure 17 the possible configurations of links are:

1. link 0 (main link only)
2. links 0 and 1
3. links 0 and 2
4. links 0 and 3
5. links 0 and 4
6. links 0, 4, and 5
7. links 0, 4, and 6

The loading of a link is caused by the execution of a call from a link that is presently in core storage to a link that is not in core storage. When a link is loaded,

it overlays the link in core storage with the same logical origin, and it overlays any links in core storage with deeper logical origins extending downward from the original link overlaid.

As noted above, a continuous chain of links is always in core storage. For example, if link 6 in Figure 17

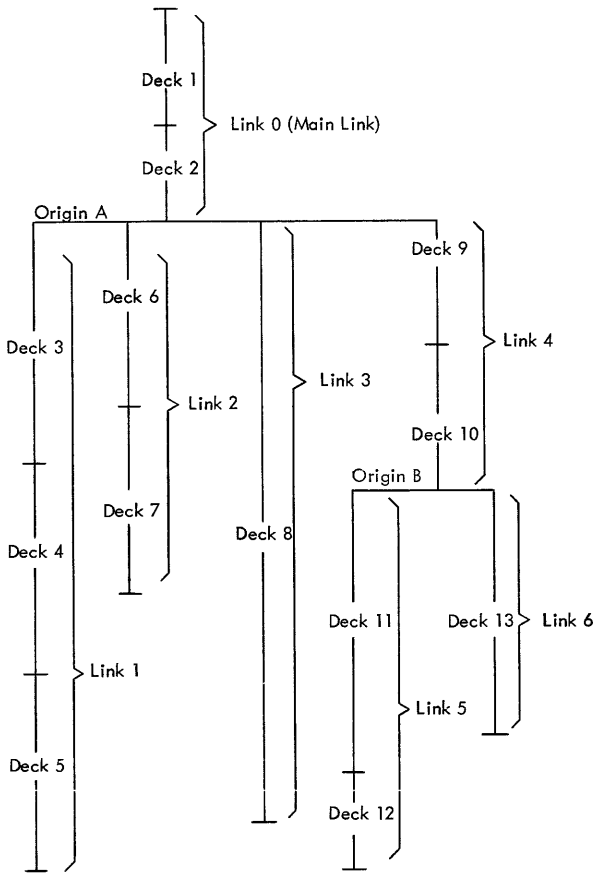


Figure 17. Example of Overlay Structure

were to be called from link 0, link 4 would also be loaded, even though specific reference was not made to link 4.

If a link already in core storage is called, it is not reloaded.

Each deck within a link is called, or referred to, by the MAP pseudo-operation CALL. The CALL pseudo-operation is the only operation that causes overlay. FORTRAN and COBOL statements that are translated into a CALL pseudo-operation that refers to another deck can be used to cause overlay. A CALL statement is not needed after every link.

The CALL Statement

The primary rule regarding CALL statements that cause links to be loaded is that, normally, no deck may either directly or indirectly call for itself to be overlaid.

The following types of CALL statements are valid:

1. A call toward a deeper link in the same chain is permissible. This type of call may or may not cause a link to be loaded, depending on whether or not the link is already in core storage.

2. A call within a link is always permissible. This type of call does not cause a link to be loaded.

3. A call from a deeper link to decks within the same chain of links toward the main link is permissible, provided the deck that it calls does not cause the originating deck to be overlaid.

If an invalid CALL statement is used, the program is not executed unless NOFLOW is specified in the variable field of the $IBJOB card.

Since the overlay structure is defined at load time, all CALL statements that cause overlay must be defined at load time. A CALL statement of the form:

```
CALL     **
```

where the address is supplied at execution time, cannot initiate overlay.

Referring to the overlay structure in Figure 17, the following examples may be given:

1. A call from deck 3 to deck 4 is permitted. A call within the same link never causes a link to be loaded.

2. A call from deck 2 to deck 12 is permitted. This call from link 0 to link 5 initiates the loading of links 4 and 5.

3. A call from deck 12 to deck 9 is permitted. This call is in the chain of links toward the main link.

4. A call from deck 13 to deck 3 is not permitted. This call causes the loading of link 1, thereby overlaying link 6, which contains the deck in which the CALL statement originated.

5. A call from deck 11 to deck 9, followed by a call from deck 9 to deck 13, is not permitted. This call from deck 11 to deck 9 is valid, but, since deck 9 contains a call to deck 13, link 6 would overlay link 5, which contains the deck in which the CALL statement originated.

Virtual Control Sections

During the analysis of virtual control sections, virtual control sections other than the CALL type are also checked for validity. This type of virtual reference cannot cause a link to be loaded, but may cause an error if reference is made to a section that is not in core storage. The following rules should be considered:

1. A reference to a control section in a deeper link in the same chain is permissible, but it may be in error if the deeper link has not been loaded into core storage. If LOGIC or DLOGIC was specified on the $IBJOB card, a warning message is printed.

2. A reference within a link is always permissible.

3. A reference from a deeper link to a control section within the same chain of links toward the main link is always permissible.

4. A reference to a section that is not in the allowable chain of links is not permitted, since, by the definition of overlay structure, the section referred to would not be in core storage. If NOFLOW is specified in the variable field of the $IBJOB card, this type of reference is permitted.

Storage Allocation During Execution

Figure 18 illustrates how the overlay structure in Figure 17 would be assigned to core storage. Links having the same logical origin are loaded starting at the same absolute location, unless the programmer has specified an absolute loading address for one or more links.

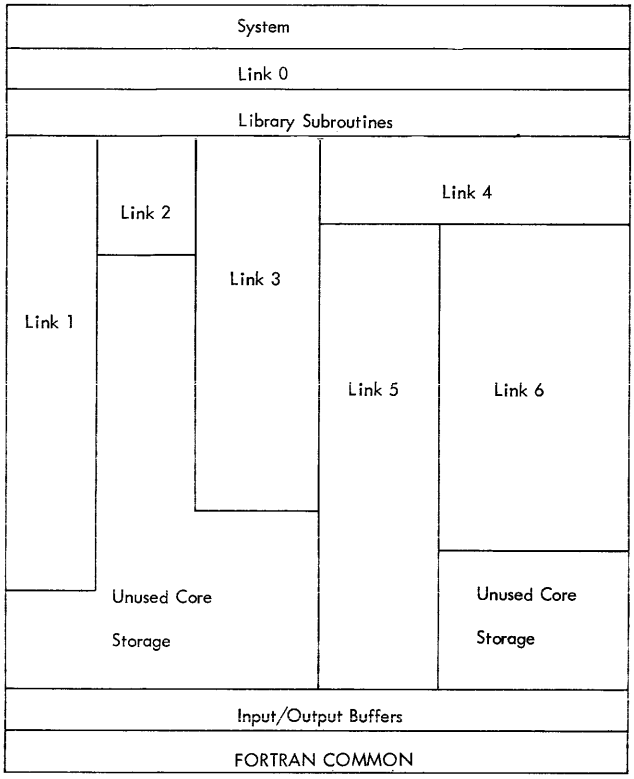


Figure 18. Overlay Core Storage Allocation

Library subroutines in the main link are loaded following the input decks that constitute the main link. Nearly all of the Library subroutines can be included in deeper links by using a $INCLUDE card. The exceptions are noted under "$INCLUDE Card." The input/output buffers occupy the unused core storage area between the longest possible link configuration and the highest available core storage location. The FORTRAN COMMON area, if used, has priority and is assigned the highest available core storage location and, therefore, is assigned following the input/output buffers.

Overlay Control Cards

Two control cards are used with the overlay feature of the Loader. The $ORIGIN card is used to specify the logical origin of links. The $INCLUDE card is used to specify a deck (or any control section) be loaded with a link other than the one with which it would normally be loaded.

The order in which options are specified on the control cards is not significant unless otherwise specified.

$ORIGIN Card

The logical origins that are specified on $ORIGIN cards govern the structure of an overlay deck. The decks appearing first in the program are assigned to the main link and are usually not preceded by a $ORIGIN card. A $ORIGIN card must precede the main link if the overlay link-loading subroutine .LOVRY is included in the input program rather than being read from the Subroutine Library. When a $ORIGIN card is used to designate the main link, the logical origin specified by this card cannot be used on succeeding $ORIGIN cards or an error condition occurs, since more than one main link would then be specified.

The $ORIGIN card initiates an overlay link for the decks that follow. Decks following the $ORIGIN card are assigned to the same link until the occurrence of another $ORIGIN card, a $ENTRY card, or an end-of-file condition.

All pertinent information must be on this control card. The $ETC card may not be used to extend the variable field information.

The following text indicates the options that may be specified on the $ORIGIN card.

The format of the $ORIGIN card is:

1	16
$ORIGIN	logical origin [, options]

where the logical field must be specified and must be the first information contained in the variable field. The field contains an alphameric literal of six or fewer characters, one of which must be nonnumeric. Characters that cannot be used are parentheses, equal signs, commas, slashes, quotation marks, and embedded blanks.

Absolute Origin Option

[,absolute origin]

This field contains five or fewer numeric characters specifying an absolute location at which the link is to be loaded. If the number is expressed in octal form, an alphabetic O must precede the number. The field is used only if a program requires that a link be loaded at a specific location. It has no effect on the overlay structure. It merely determines the loading point for this particular link and all those links following that are without $ORIGIN cards to specify different absolute origins.

Unit Specification Options

[, {SYSUT2 / SYSxxx}]

This field specifies the input/output unit on which the dependent links are written. Any of the following seven system units may be specified:

SYSUT2 (or UT2)	SYSLB4 (or LB4)
SYSUT3 (or UT3)	SYSCK1 (or CK1)
SYSLB2 (or LB2)	SYSCK2 (or CK2)
SYSLB3 (or LB3)	

NOTE: SYSCK2 is not available if debugging has been requested.

If the system library unit SYSLB2, SYSLB3, or SYSLB4 is specified and is also used to store the system components, an error message is written on the system output unit and execution is not allowed.

If the field is omitted, the system utility unit (SYSUT2) is assigned. It is assumed that the unit chosen is in ready status and that it is not used for any purpose other than loading links during execution.

Rewind Options

$$\left[,\left\{\begin{matrix}\underline{\text{NOREW}}\\ \text{REW}\end{matrix}\right\}\right]$$

NOREW — The input/output unit containing the link is not to be rewound after the link is loaded.

REW — The unit is to be rewound.

If neither NOREW nor REW is specified, the unit is not rewound (NOREW).

$INCLUDE Card

The $INCLUDE card specifies that the decks and/or the control sections named in the variable field be included in the link in which this control card appears, rather than in the link to which they would normally be assigned.

The format of the $INCLUDE card is:

1	16
$INCLUDE	$\left\{\begin{matrix}\text{deckname}\\ \text{exname}\end{matrix}\right\},\ldots$

The subfields of the variable field contain alphameric literals that specify either a deck name (usually a library subroutine) or a real control section name of nonzero length (usually a block of data or coding) to be included in this link.

If a library subroutine is specified, the deck name of the subroutine (and not one of its entry points) must be given. Library subroutines are placed automatically in the main link, so that they are available to all subsequent links. A library subroutine may, however, be assigned to a dependent link by means of a $INCLUDE card. A subroutine or control section cannot be loaded in more than one link. If it is called from more than one link, it must be loaded in a link that is available to all calling links.

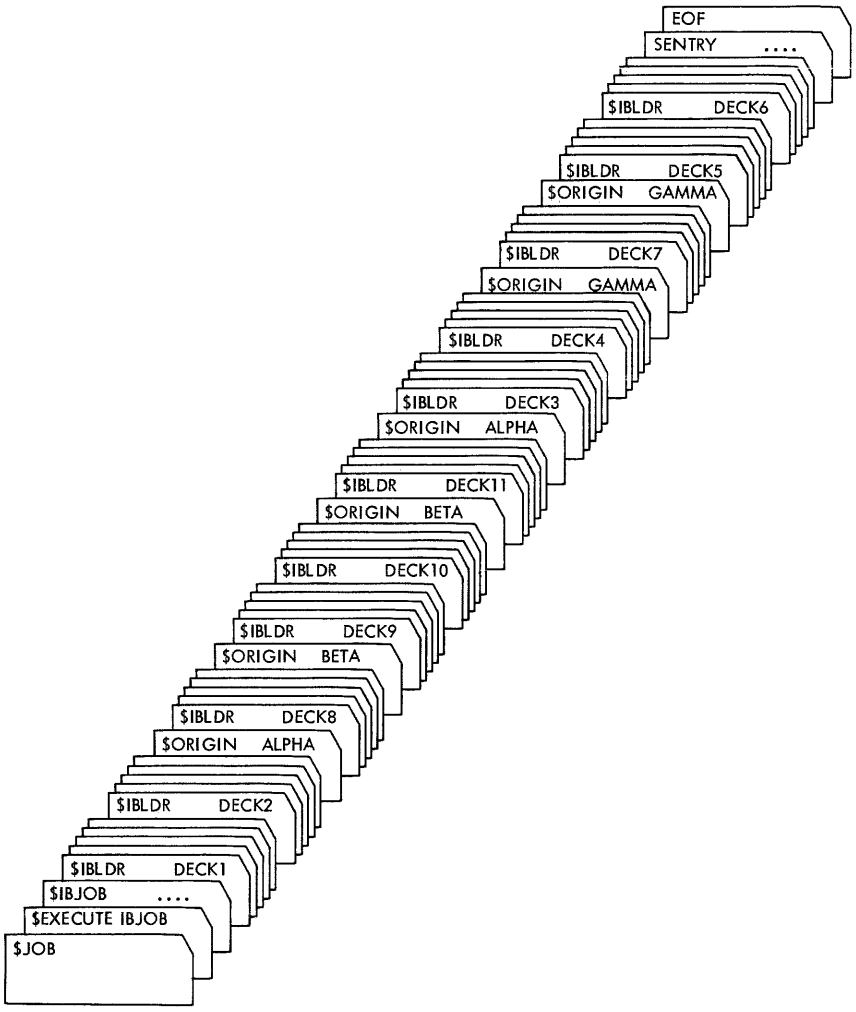


Figure 19. Sample Overlay Control Card Deck

The following subroutines must always be in the main link and, therefore, may not be specified on a $INCLUDE card:

1. .FPTRP – Floating-Point Trap Subroutine
2. .LXCON – Execution Control Subroutine
3. .LOVRY – Overlay Link Loading Subroutine
4. The subroutine(s) designating the level of IOCS used by the object program. The object-time debugging routines, if used, may not be specified on a $INCLUDE card. The Library IOCS routines .IODEF, .IOCSS, .IOCS, .IOCSM, .IOCSD, .IOCSL may not be specified on a $INCLUDE card.

The variable field of a $INCLUDE card may be extended over more than one card, using either the $ETC card or another $INCLUDE card. The $INCLUDE card may appear immediately following the $ORIGIN card specifying the link, between the decks within the link, or immediately following the last deck of the link.

Control Card Usage

Figure 19 illustrates how a deck would be set up to produce the program structure given in Figure 20.

In Figure 19 the $ORIGIN card, which first uses logical origin ALPHA, immediately follows the main link (decks 1 and 2). All links using logical origin ALPHA, therefore, proceed from the main link. Every new logical origin encountered on a $ORIGIN card specifies that all links using this logical origin will proceed from the previous link. The $ORIGIN cards containing the logical origins BETA and GAMMA are placed after the links from which they proceed. In this manner, the $ORIGIN cards are used to form the overlay structure.

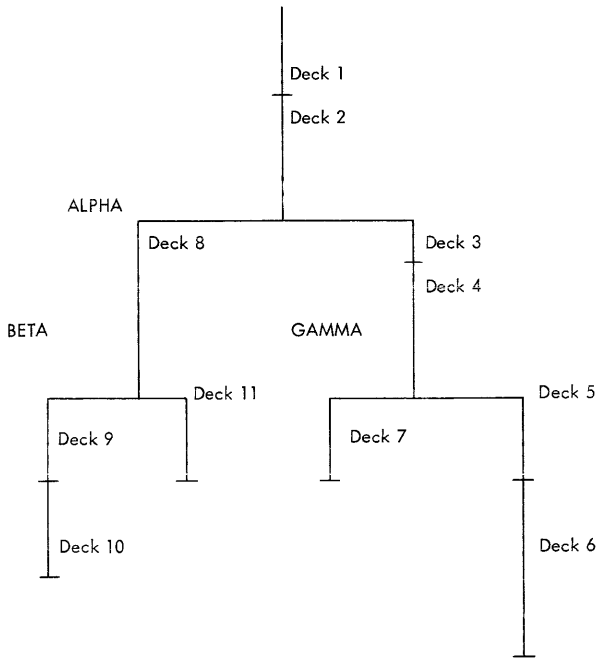


Figure 20. Overlay Structure for Sample Control Card Deck

The following examples are given to aid the programmer in the use of overlay control cards. To include the subroutine FLOG and the control section XYZ in the link that contains deck 1, the sequence shown in Figure 21 could be used.

When a deck or section is assigned to a link by means of a $INCLUDE card, care must be taken that the link incorporating the deck or control section be available to all other links that refer to or call the deck or section.

If a $INCLUDE card is used to move a block of instructions or data from a deck to some other link, it is possible to cause the external link file to be written in a format that cannot be used efficiently during execution. For example, in Figure 22 the instructions or data in section XYZ that are to become part of link A are not encountered by the Loader for processing until after link A and a portion of link B have been written onto the system utility unit (SYSUT3). Therefore, on SYSUT3, the information in section XYZ is isolated from the main portion of link A. If SYSUT3 is a tape file, some tape has to be spaced over when loading link A in order to load the XYZ portion. This situation can usually be avoided by specifying a unique unit for the storing of the link that contains the $INCLUDE card.

It should be noted that the previously mentioned condition occurs only when the section specified on the $INCLUDE card is internal to some deck and contains text. This condition does not occur when assigning library subroutines or control sections that do not contain text to other links by means of the $INCLUDE card.

```
1       8         16
        .
        .
        .
$ORIGIN           ALPHA
$INCLUDE          FLOG,XYZ
        .
        .
        .
      (DECK 1)
        .
        .
        .
```

Figure 21. Overlay Control Cards to Include Subroutine FLOG and Control Section XYZ in Link Containing Deck 1

```
1       8         16
        .
        .
        .
$ORIGIN           A,SYSUT3  (LINK A)
        .
        .
        .
$INCLUDE          XYZ
$ORIGIN           B,SYSUT3  (LINK B)
$IBLDR DECK1
        .
        .
        .
$IBLDR DECK2                (CONTAINS XYZ)
        .
        .
        .
```

Figure 22. Inefficient Use of $INCLUDE Card to Include Control Section XYZ in Link A

CALL Transfer Vector

During loading, an analysis is made of all CALL statements in the program. If a CALL statement causes overlay, the transfer address of the CALL statement is modified to refer to a transfer vector of the form:

```
pfx     entry point, , link number
TXI     .LOVRY
```

For example, the statement:

```
CALL    .SUBPR
```

may be modified by the Loader to:

```
CALL    .TV001
```

and starting at location .TV001, the Loader would generate the following two words:

```
pfx     .SUBPR, link number
TXI     .LOVRY
```

This transfer vector is constructed by the Loader and is stored with the object program in a generated control section called .LVEC. During execution, if the called deck was loaded into core storage, pfx is set to TXL and a transfer is made to the entry point. If the called deck was not loaded into core storage, pfx is set to TXH and a transfer is made to the link loading subroutine .LOVRY. This subroutine loads the required links and resets all the transfer vector words involved to indicate properly the load status of the links.

If LOGIC is specified on the $IBJOB card, the absolute location of .LVEC is indicated in the logic listing.

$ENTRY Card With Overlay

When a $ENTRY card is used to name the first instruction to be executed in a program using overlay, the instruction named must be contained in the main link.

The Subroutine Library contains relocatable subroutines for use by both the system itself and the object program. These subroutines are incorporated into the object program at load time by the Loader, either at the programmer's request or automatically in response to program requirements. Subroutines may be added to and deleted from the Subroutine Library by using the Librarian, as described in the section "The Librarian."

The Subroutine Library consists of system, COBOL, and FORTRAN IV subroutines. The operating system uses certain Library subroutines for maintaining control and communication among the system programs. The COBOL subroutines include subroutines needed for movement, conversion, input, and output of data. The FORTRAN IV section of the Library includes the FORTRAN mathematics library, the FORTRAN input/output library, and the FORTRAN utility library. The FORTRAN mathematics subroutines and certain FORTRAN utility subroutines available for use by the applications programmer are described in this section. The remainder of the subroutines are used only by the system and the compilers, and are described in the section "Subroutine Library Information."

FORTRAN IV Mathematics Library

The FORTRAN IV mathematics library contains three types of subroutines: single-precision, double-precision, and complex. FORTRAN IV, MAP, and COBOL programmers can use these subroutines to perform mathematical computations. Some subroutines are capable of performing more than one kind of computation. These capabilities are called "functions." Each function has its own entry point in the subroutine.

The library exists in two versions: a 7090 version and a 7094 version. The main difference between them is the way double-precision functions are evaluated. In the 7090 version the closed subroutine FDAS has been added to simulate double-precision instructions. In the 7094 version double-precision instructions perform the computations. These instructions save time and core storage space. They are also more accurate for very small double-precision numbers when used in conjunction with the double-precision floating-point trap routine. This routine is described in the section "Floating-Point Trap Subroutine."

Calling Sequences to FORTRAN IV Mathematics Subroutines

The calling sequence to a subroutine function depends on the programming language used. In each case, however, the programmer specifies an entry point and the names of one or more core storage locations containing arguments. Each function has its own entry point. The name of this entry point is distinct from the name of the subroutine containing it. These names are considered as control sections and as such can be changed through the use of the $NAME card. (See "$NAME card.")

Input to a subroutine normally consists of arguments. In a COBOL program there must also be a core storage location in which the subroutine places the result of its computation. The general form of a calling sequence in each programming language is shown in Figure 23. The specific forms for calling the various functions are shown in Figures 25A, 25B, 25C, and 26.

Source Language	Calling Sequences For Functions Having One Argument	Calling Sequences For Functions Having Two Arguments
FORTRAN IV	y=entry point (argument) The answer is stored in y.	y=entry point (arg1, arg2) The answer is stored in y.
MAP	CALL entry point (argument) The answer is left in the AC or AC-MQ.	CALL entry point (arg1, arg2). The answer is left in the AC or AC-MQ.
COBOL	CALL 'cobol entry point' USING result, argument. The answer is stored in result.	CALL 'cobol entry point' USING result, arg1, arg2). The answer is stored in result.

Figure 23. General Form of Calling Sequences to FORTRAN IV Mathematics Subroutines

Arguments

Most arguments used in the mathematics subroutines must be normalized floating-point numbers. (Exceptions are in the FXP1, FXP2, FDX1, and FMTN subroutines.)

A double-precision argument is contained in two adjacent core storage words. The first word contains the high-order portion of the argument and is referred to as its location. The second word contains the low-order portion of the argument.

A complex argument is also contained in two adjacent words. The first word contains the real portion of the argument and is referred to as its location. The second word contains the imaginary part of the argument.

MAP programmers coding for the 7094 must use an EVEN pseudo-operation to store each double-precision or complex argument, starting at an even core storage location. Otherwise, as soon as a double-precision instruction in a subroutine refers to the argument, a floating-point trap occurs. On the 7090 or 7094 II such core storage alignment is not necessary.

COBOL programmers using data-types 4 and 5 on a 7094 cannot use the double-precision or complex subroutines. (For a description of data-types, see *IBM 7090/7094 IBSYS Operating System, Version 13; COBOL Language,* Form C28-6390.)

Results

Each function produces a single result. Double-precision and complex functions produce two-word results. For FORTRAN IV programs, the single-precision result is stored in the leftmost variable in the calling sequence. For COBOL programs, the result is stored in the result word specified. For MAP programs, results are left in the AC. The high-order portion of double-precision results is left in the AC, and the low-order portion in the MQ. The real portions of complex answers are left in the AC, and the imaginary portions in the MQ.

Error Handling for FORTRAN IV Mathematics Subroutines

There are certain limits on the range of numbers over which an input argument yields meaningful results. These ranges are shown in Figures 25A, 25B, 25C, and 26 for each subroutine.

When the valid argument range for a subroutine is exceeded in a program, the execution-error-monitor subroutine FXEM prints an error message. It then either terminates execution or supplies a conventional answer and returns control to the program. The FXEM subroutine is described under "FORTRAN IV Utility Library" in the section "Subroutine Library Information" in Part 2.

Among the conventional answers that FXEM may give is the largest possible positive floating-point number. This is $2^{127} - 2^{100}$ in single-precision numbers, or $2^{127} - 2^{73}$ for double-precision numbers. For brevity, the number is written as Ω in Figures 25A, 25B, and 25C.

The distributed version of FXEM always supplies a conventional answer when entered from a mathematics subroutine. Except in exceptional circumstances, however, it is safer to terminate execution. The user can provide for termination by presetting the FXEM control bits as described under "FORTRAN IV Utility Library."

Error Codes

In each of Figures 25A, 25B, and 25C there is a column headed "Error Code." The numbers in this column correspond to the respective option control bit position in the first and third words of the FXEM control section, i.e., the OPTWD1 and OPTWD3 described under "FORTRAN IV Utility Library."

As an illustration of how this code is used, consider what happens when the single-precision square root subroutine encounters a negative (i.e., invalid) argument. In this case, the square root subroutine, upon detecting the negative argument, makes an entry into the FXEM subroutine, specifying error code 13. Upon receiving the error code, FXEM examines bit 13 of OPTWD1. If bit 13 is zero, FXEM prints the appropriate error message and terminates execution. If bit 13 is one, this means that the option for computing the square root of the absolute value of the argument will be taken; accordingly, after printing a warning message, FXEM returns control to the square root subroutine, which changes the sign of the argument and proceeds upon its normal course.

Floating-Point Trap Subroutines

The computer must be in the floating-point trap mode for the proper operation of the FORTRAN IV mathematics subroutines. It is in this mode at the start of object program execution and continues so until it executes an LFTM instruction. Floating-point traps are handled by the standard floating-point subroutine on a 7090 and by the standard or its alternate on the 7094. The difference between these subroutines is discussed under "Alternate 7094 Floating-Point Trap Subroutine."

Floating-Point Overflow

Except for the subroutines dealing with complex numbers, floating-point overflow will never occur because the arguments are screened upon entry. If an argument is such that it could cause an overflow, it is immediately treated as out of range and control passes to the FXEM subroutine.

For some of the complex subroutines the occurrence of floating-point overflow is possible even though both the real and imaginary parts of a given argument themselves lie within the valid argument range. This is because either the real or the imaginary part of certain complex answers is a function of both the real and the imaginary part of the argument. If this happens, it is too costly in both core storage space and execution time to screen out invalid arguments prior to the particular arithmetic operation that could cause the overflow. An occurrence of such an overflow causes a floating-point trap and an overflow error message. Since the floating-point trap subroutine sets an overflowed

register to Ω before proceeding, the answer produced by a complex subroutine is unreliable.

Floating-Point Underflow

Several of the mathematics subroutines can cause floating-point underflow. An underflow causes a floating-point trap. The floating-point trap subroutine then sets the register involved to a signed zero and returns control to the subroutine from which the trap originated. Often an underflow is irrelevant to the computation of the function involved. This is true with low-order or remainder underflow for the single-precision or complex function routines. Setting zero into the involved register does not in any way influence the accuracy of answers. Both a double-precision low-order underflow and a high-order underflow are relevant to the accuracy of answers.

A relevant underflow occurs only when the answer iteslf is very close to the underflow threshold. Since numbers below this threshold cannot be represented in the machine registers, setting underflowed registers to zeros yields the greatest possible accuracy.

Alternate 7094 Floating-Point Trap Subroutine

As distributed, the system tape for the 7094 Subroutine Library has both a standard and an alternate floating-point trap subroutine. Each has the entry point name of .FPTRP. The user must select the subroutine he wants.

The two subroutines differ only in the treatment of MQ underflow resulting from a double-precision instruction. The alternate subroutine returns control to the trapped subroutine, with the MQ unchanged (i.e., not set to zero). Full 54-bit significance is thus retained. No errors will result from this action except when a MAP program attempts independent manipulation of high-order and low-order parts of double-precision numbers.

The alternate subroutine is considerably more accurate when the result of the arithmetic operation is less than $10^{-30.7}$ in magnitude. It is also more accurate when used with double-precision subroutines. The accuracy figures for double-precision subroutines in Appendix H are based on the use of the alternate subroutine. Figure 24 shows the difference in accuracy between the two versions.

7090 Double-Precision Simulation

Most of the 7090 double-precision mathematics subroutines use FDAS, a common utility subroutine for the rapid simulation of double-precision instructions. FDAS may also be called by a 7090 MAP program. The FDAS calling sequence and specifications are shown in Appendix H.

Mathematical Subroutine	Argument Range	Maximum Accuracy Using Standard Subroutine	Maximum Accuracy Using Alternate Subroutine
FDXP	$-89.415986 < x < -70.701013$	8 decimal digits	16 decimal digits
FDSC	$0 < \|x\| < 2^{-104}\pi$		
In addition, if $0 <$ Argument $< 2^{-102}$ and if the low-order part of the argument is inexact because the alternate subroutine is absent, then this inaccuracy is reflected in the answer produced by FDSQ and FDAT. The answer is then accurate to a maximum of only 8 decimal digits.			

Figure 24. Accuracy of Standard Versus Alternate 7094 Floating-Point Trap Subroutines

Evaluating Accuracy

To evaluate the accuracy of the subroutines, see Appendix H, "FORTRAN IV Mathematics Subroutines – Algorithms, Accuracy, and Speed." This appendix also contains the algorithms that are the mathematical basis for each subroutine.

Subroutine Reference Tables

Figures 25A, 25B, 25C, and 26 contain information on the FORTRAN IV mathematics subroutines for quick reference.

Information such as algorithms, accuracy statistics, and average speeds is contained in Appendix H. Core storage requirements are listed in Appendix I. These tables are self-explanatory, except for the symbol Ω, which has been defined previously under "Error Handling for FORTRAN IV Mathematics Subroutines."

NOTE: Some of the ranges in the following figures are represented in powers of 2. The corresponding approximate decimal values are as follows:

$2^{-127} \cong 5.878 \times 10^{-39}$
$2^{20} \cong 1.049 \times 10^{6}$
$2^{25} \cong 3.355 \times 10^{7}$
$2^{50} \cong 1.126 \times 10^{15}$

Subroutine Name	Definition	Calling Sequences F = FORTRAN IV M = MAP C = COBOL	Valid Argument Range	Error Code	Options	
					If Argument Range Is	Then The Answer Is
square root FSQR	$y=x^{\frac{1}{2}}$	F: y=SQRT(x) M: CALL SQRT(x) C: CALL '.CSQRT' USING y, x.	$0 \leqslant x$	13	$x<0$	$\lvert x\rvert^{\frac{1}{2}}$
exponential FXPF	$y=e^{x}$	F: y=EXP(x) M: CALL EXP(x) C: CALL '.CEXP' USING x, y.	$x \leqslant 88.029692$	8	$x>88.029692$	Ω
exponentiation fixed-point base and fixed-point exponent FXP1	$y=i^{j}$	F: y=i**j M: CALL .XP1.(i, j) C: CALL '.CXP1' USING y, i, j.	$i \neq 0,\ j \geqslant 0$	1	$i=0$ $j=0$	0
				2	$i=0$ $j<0$	0
exponentiation floating-point base and fixed-point exponent FXP2	$y=x^{i}$	F: y=x**i M: CALL .XP2.(x, i) C: CALL '.CXP2' USING y, x, i.	$x \neq 0,\ i \geqslant 0$	3	$x=0$ $i=0$	0
				4	$x=0$ $i<0$	0
exponentiation floating-point base and floating-point exponent FXP3	$y=x^{a}$	F: y=x**a M: CALL .XP3. (x, a) C: CALL '.CXP3' USING y, x, a.	$x>0$ and $a \geqslant 0$	5	$x<0$ $a \neq 0$	$\lvert x\rvert^{a}$
				6	$x=0$ $a=0$	0
				7	$x=0$ $a<0$	0
logarithm FLOG	$y=\log_e(x)$	F: y=ALOG(x) M: CALL ALOG(x) C: CALL '.CALOG' USING y, x.	$0<x$	9	$x=0$	$-\Omega$
				10	$x<0$	$\log_e \lvert x\rvert$
	$y=\log_{10}(x)$	F: y=ALOG10(x) M: CALL ALOG10(x) C: CALL '.CAL10' USING y, x.	$0<x$	9	$x=0$	$-\Omega$
				10	$x<0$	$\log_{10} \lvert x\rvert$
sine/cosine FSCN	y=sine(x) (x expressed in radians)	F: y=SIN(x) M: CALL SIN(x) C: CALL '.CSIN' USING y, x.	$\lvert x\rvert<2^{25}$	12	$\lvert x\rvert \geqslant 2^{25}$	0
	y=cosine(x) (x expressed in radians)	F: y=COS(x) M: CALL COS(x) C: CALL '.CCOS' USING y, x.	$\lvert x\rvert<2^{25}$	12	$\lvert x\rvert \geqslant 2^{25}$	0
tangent/cotangent FTNC	y=tan(x) (x expressed in radians)	F: y=TAN(x) M: CALL TAN(x) C: CALL '.CTAN' USING y, x.	$\lvert x\rvert<2^{20}$ and x may not be an odd integral multiple of $\frac{\pi}{2}$ (see note)	73	$\lvert x\rvert \geqslant 2^{20}$	0
				74	$x \cong k\frac{\pi}{2}$ where k is an odd integer	Ω

Figure 25A. Single-Precision Subroutines in the FORTRAN IV Mathematics Library

Subroutine Name	Definition	Calling Sequences F = FORTRAN IV M = MAP C = COBOL	Valid Argument Range	Error Code	Options: If Argument Range Is	Options: Then The Answer Is
tangent/cotangent FTNC (cont.)	$y=\cot(x)$ (x expressed in radians)	F: y=COTAN(x) M: CALL COTAN(x) C: CALL '.CCOTN' USING y, x.	$\|x\|<2^{20}$ and x may not be a multiple of π (see note 1)	73	$\|x\|\geq 2^{20}$	0
				74	$x\cong k\pi$ where k is an integer	Ω
arctangent FATN	$y=\arctan(x)$ (y expressed in radians)	F: y=ATAN(x) M: CALL ATAN(x) C: CALL '.CATAN' USING y, x.	any argument	inapplicable	inapplicable	inapplicable
	$y=\arctan(x_1/x_2)$ (y expressed in radians)	F: y=ATAN2(x_1,x_2) M: CALL ATAN2(x_1,x_2) C: CALL '.CATN2' USING y, x_1, x_2.	$(x_1, x_2)\neq(0, 0)$	11	$(x_1, x_2)=(0, 0)$	0
arcsine/arccosine FASC	$y=\arcsin(x)$ (y expressed in radians)	F: y=ARSIN(x) M: CALL ARSIN(x) C: CALL '.CARSN' USING y, x.	$\|x\|\leq 1$	72	$\|x\|>1$	0
	$y=\arccos(x)$ (y expressed in radians)	F: y=ARCOS(x) M: CALL ARCOS(x) C: CALL '.CARCS' USING y, x.	$\|x\|\leq 1$	72	$\|x\|>1$	0
hyperbolic sine/cosine FSCH	$y=\frac{1}{2}(e^x-e^{-x})$	F: y=SINH(x) M: CALL SINH(x) C: CALL '.CSINH' USING y, x.	$\|x\|\leq 88.029692$	75	$\|x\|>88.029692$	Ω
	$y=\frac{1}{2}(e^x+e^{-x})$	F: y=COSH(x) M: CALL COSH(x) C: CALL '.CCOSH' USING y, x.	$\|x\|\leq 88.029692$	75	$\|x\|>88.029692$	Ω
hyperbolic tangent FTNH	$y=\frac{e^x-e^{-x}}{e^x+e^{-x}}$	F: y=TANH(x) M: CALL TANH(x) C: CALL '.CTANH' USING y, x.	any argument	inapplicable	inapplicable	inapplicable
error function FERF	$y=\frac{2}{\sqrt{\pi}}\int_0^x e^{-u^2}du$	F: y=ERF(x) M: CALL ERF(x) C: CALL '.CERF' USING y, x.	any argument	inapplicable	inapplicable	inapplicable
gamma/loggamma FGAM	$y=\int_0^\infty u^{x-1}e^{-u}du$	F: y=GAMMA(x) M: CALL GAMMA(x) C: CALL '.CGAMA' USING y, x.	$2^{-127}<x<34.843$	76	$x\leq 2^{-127}$ or $x\geq 34.843$	Ω
	log base e of above	F: y=ALGAMMA(x) M: CALL ALGAMMA(x) C: CALL '.CALGM' USING y, x.	$0<x<2.0593\times10^{38}$	77	$x\leq 0$ or $x\geq 2.0593\times10^{38}$	Ω

Note: For more details on the valid argument ranges for the tangent/cotangent subroutine and how these ranges can be expanded or reduced through the FMTN subroutine, see "Error Control Modification for the FTNC Subroutine—FMTN."

Figure 25A. Single-Precision Subroutines in the FORTRAN IV Mathematics Library (cont.)

Subroutine Name	Definition	Calling Sequences F = FORTRAN IV M = MAP C = COBOL	Valid Argument Range	Error Code	Options	
					If Argument Range Is	Then The Answer Is
square root FDSQ	$y=x^{\frac{1}{2}}$	F: y=DSQRT(x) M: CALL DSQRT(x) C: CALL '.CDSQR' USING y, x.	$0 \leqslant x$	22	$x<0$	$\lvert x\rvert^{\frac{1}{2}}$
exponential FDXP	$y=e^{x}$	F: y=DEXP(x) M: CALL DEXP(x) C: CALL '.CDEXP' USING y, x.	$x \leqslant 88.029692$	19	$x>88.029692$	Ω
exponentiation floating-point base and fixed-point exponent FDX1	$y=x^{i}$	F: y=x**i M: CALL .DXP1. (x, i) C: CALL '.CDXP1' USING y, x, i.	$x \neq 0$ $i \geqslant 0$	14	$x=0$ $i=0$	0
				15	$x=0$ $i<0$	0
exponentiation floating-point base and floating-point exponent FDX2	$y=x^{a}$	F: y=x**a M: CALL .DXP2. (x, a) C: CALL '.CDXP2' USING y, x, a.	$x>0$ $d \geqslant 0$	16	$x<0$ $a \neq 0$	$\lvert x\rvert^{a}$
				17	$x=0$ $a=0$	0
				18	$x=0$ $a<0$	0
logarithm FDLG	$y=\log_e(x)$	F: y=DLOG(x) M: CALL DLOG(x) C: CALL '.CDLOG' USING y, x.	$0<x$	20	$x=0$	$-\Omega$
				21	$x<0$	$\log_e \lvert x\rvert$
	$y=\log_{10}(x)$	F: y=DLOG10(x) M: CALL DLOG10(x) C: CALL '.CDL10' USING y, x.	$0<x$	20	$x=0$	$-\Omega$
				21	$x<0$	$\log_{10} \lvert x\rvert$
sine/cosine FDSC	y=sine(x) (x expressed in radians)	F: y=DSIN(x) M: CALL DSIN(x) C: CALL '.CDSIN' USING y, x.	$\lvert x\rvert<2^{50}\pi$	23	$\lvert x\rvert \geqslant 2^{50}\pi$	0
	y=cosine(x) (x expressed in radians)	F: y=DCOS(x) M: CALL DCOS(x) C: CALL '.CDCOS' USING y, x.	$\lvert x\rvert<2^{50}\pi$	23	$\lvert x\rvert \geqslant 2^{50}\pi$	0
arctangent FDAT	y=Arctan(x) (y expressed in radians)	F: y=DATAN(x) M: CALL DATAN(x) C: CALL '.CDATN' USING y, x.	any argument	inapplicable	inapplicable	inapplicable
	$y=\arctan(x_1/x_2)$ (y expressed in radians)	F: y=DATAN2(x_1,x_2) M: CALL DATAN2 (x_1,x_2) C: CALL '.CDAT2' USING y,x_1,x_2.	$(x_1,x_2) \neq (0,0)$	24	$(x_1,x_2)=(0,0)$	0
Modular arithmetic—FDMD	see note	F: y=DMOD (b,d) M: CALL DMOD (b,d) C: CALL '.CDMOD' USING y,b,d.	any argument	inapplicable	inapplicable	inapplicable

Note: B modulo D is defined as B−[B/D] *D. The brackets indicate that only the integer portion of the expression within them is used in evaluating the equation.

Figure 25B. Double-Precision Subroutines in the FORTRAN IV Mathematics Library

Subroutine Name	Definition Notation: $z=x_1+ix_2$	Calling Sequences F = FORTRAN IV M = MAP C = COBOL	Valid Argument Range	Error Code	Options	
					If Argument Range Is	Then The Answer Is
square root FCSQ	$y=z^{\frac{1}{2}}$ real $y\geqslant 0$	F: y=CSQRT(z) M: CALL CSQRT(z) C: CALL '.CCSQR' USING y,z.	any argument (see note 1)	inappli-cable	inapplicable	inapplicable
exponential FCXP	$y=e^z$	F: y=CEXP(z) M: CALL CEXP(z) C: CALL '.CCEXP' USING y,z.	$x_1\leqslant 88.029692$	26	$x_1>88.029692$	$\Omega[\cos(x_2)+i\sin(x_2)]$
			$\lvert x_2\rvert<2^{25}$	27	$\lvert x_2\rvert\geqslant 2^{25}$	$0+0i$
exponentiation complex base and fixed-point exponent FDX1	$y=z^i$	F: y=z**i M: CALL .CXP1, (z,i) C: CALL '.CCXP1' USING y,z,i.	$z\neq 0+0i$ $i\geqslant 0$	14	$z=0+0i$ $i=0$	0
				15	$z=0+0i$ $i<0$	0
logarithm FCLG	$y=PV\log_e(z)$ (see note 2)	F: y=CLOG(z) M: CALL CLOG(z) C: CALL '.CCLOG' USING y,z.	$z\neq 0+0i$ (see note 1)	28	$z=0+0i$	$-\Omega+0i$
sine/cosine FCSC	$y=\sin(z)$	F: y=CSIN(z) M: CALL CSIN(z) C: CALL '.CCSIN' USING y,z.	$\lvert x_1\rvert<2^{25}$	29	$\lvert x_1\rvert\geqslant 2^{25}$	$0+0i$
			$\lvert x_2\rvert\leqslant 88.029692$	30	$\lvert x_2\rvert>88.029692$	(see note 3)
	$y=\cos(z)$	F: y=CCOS(z) M: CALL CCOS(z) C: CALL '.CCCOS' USING y,z.	$\lvert x_1\rvert<2^{25}$	29	$\lvert x_1\rvert\geqslant 2^{25}$	$0+0i$
			$\lvert x_2\rvert\leqslant 88.029692$	30	$\lvert x_2\rvert>88.029692$	(see note 3)
absolute value FCAB	$y=\lvert z\rvert$	F: y=CABS(z) M: CALL CABS(z) C: CALL '.CCABS' USING y,z.	any argument (see note 1)	inappli-cable	inapplicable	inapplicable
arithmetic FCAS	$y=z_1\times z_2$	F: y=z1* z2 M: CALL .CFMP.(z1,z2) C: CALL '.CCFMP' USING y,x1,x2.	any argument (see note 1)	inappli-cable	inapplicable	inapplicable
	$y=z_1\div z_2$	F: y=z1/z2 M: CALL .CFPD.(z1,z2) C: CALL '.CCFDP' USING y,x1,x2.	any argument (see note 1)	inappli-cable	inapplicable	inapplicable

Note 1: Floating-point overflow can occur.

Note 2: PV = principal value. The answer given will be from that branch where the imaginary part lies between $-\pi$ and π. More specifically, $-\pi<y_2\leqslant\pi$ unless $x_1<0$ and $x_2=-0$, in which case $y_2=-\pi$.

Note 3: The optional answers for complex sine/cosine are as follows:

x_2	sin (z)	cos (z)
>88.029692	$\frac{\Omega}{2}[\sin(x_1)+i\cos(x_1)]$	$\frac{\Omega}{2}[\cos(x_1)-i\sin(x_1)]$
<-88.029692	$\frac{\Omega}{2}[\sin(x_1)-i\cos(x_1)]$	$\frac{\Omega}{2}[\cos(x_1)+i\sin(x_1)]$

Figure 25C. Complex Subroutines in the FORTRAN IV Mathematics Library

Subroutine Name	Definition	Calling Sequences F = FORTRAN IV M = MAP C = COBOL	Valid Argument Range	Error Code	Options	
					If Argument Range Is	Then The Answer Is
double precision floating-point trap routine (7094 library only) .FPTRP	floating-point trap subroutine	entered only via trap	inapplicable	inapplicable	inapplicable	inapplicable
FMTN (see note 1)	resets accuracy guarantee for single-precision tangent subroutine	F: CALL MTAN(*k*) M: CALL MTAN(*k*) C: (not available)	$0 \leqslant k$ where *k* is an integer	inapplicable	inapplicable	inapplicable
double-precision arithmetic simulator FDAS	performs double-precision addition, multiplication, and division on a 7090.	F: not available M: (see note 2) C: not available	any argument	inapplicable	inapplicable	inapplicable

Note 1: A detailed description of the FMTN subroutine is in "Error Control Modification for the FTNC Subroutine — FMTN."
Note 2: The MAP language calling sequences are described under "7090 Double-Precision Arithmetic Simulator — FDAS."

Figure 26. Miscellaneous Subroutines in the FORTRAN IV Mathematics Library

FORTRAN Utility Library

This section describes the FORTRAN utility subroutines for testing machine indicators and recording the status of the console and selected portions of core storage. Other FORTRAN utility subroutines are described in the section "Subroutine Library Information."

Machine Indicator Test Subroutines

The following subroutines are referred to by CALL statements in the FORTRAN IV language. In the descriptions of the subroutines an I is used to specify any integer expression, and a J is used to specify any integer variable.

FSLITE Subroutine

The FSLITE subroutine is used to test sense lights. The source program statements are:

```
CALL    SLITE(I)
```

If I = 0, all sense lights are set OFF. If I = 1, 2, 3, or 4, the corresponding sense light is set ON.

```
CALL    SLITET(I,J)
```

Sense light I (1, 2, 3, or 4) is tested and set OFF. If the sense light is ON, the variable J is set to 1; if it was OFF, J is set to 2.

FSSWTH Subroutine

The FSSWTH subroutine is used to test sense switches. The source program statement is:

```
CALL    SSWTCH(I,J)
```

Sense switch I (1, 2, 3, 4, 5, or 6) is tested. If the sense switch was OFF, J is set to 1; if it was ON, J is set to 2.

FOVERF Subroutine

The FOVERF subroutine is used to test the Overflow Indicator. The source program statement is:

```
CALL    OVERFL(J)
```

If an overflow condition exists, the variable J is set to 1; if a nonoverflow condition exists, J is set to 2. The machine is always left in a nonoverflow condition after execution.

FDVCHK Subroutine

The FDVCHK subroutine is used to test the Divide Check Indicator. The source program statement is:

```
CALL    DVCHK(J)
```

If the divide check indicator was ON, the variable J is set to 1; if it was OFF, J is set to 2. The divide check indicator is always left in OFF status after execution.

Dump Subroutine

The FDMP subroutine records select portions of core storage on the system output unit. Either of two entry points, DUMP or PDUMP, is specified, followed by the limits of the dump and the dump format.

DUMP is the entry point for a dump of the object program. Upon completion of the dump, the contents of core storage are restored and control is returned to .LXCON, the postexecution subroutine.

PDUMP is the entry point for a snapshot dump of selected portions of core storage. After the dump has been taken, the contents of core storage are restored, control is returned to the object program, and execution of the program continues.

FORTRAN Logical Unit	System File	Mode		Function
		Standard Package	Alternate Package	
01	SYSUT1	Binary	Binary or BCD	Input or output
02	SYSUT2	Binary	Binary or BCD	Input or output
03	SYSUT3	Binary	Binary or BCD	Input or output
04	SYSUT4	Binary	Binary or BCD	Input or output
05	SYSIN1	BCD	BCD	Input
06	SYSOU1	BCD	BCD	Output
07	SYSPP1	Binary	Binary	Output
08	System Availability Chain	BCD	BCD	Input or output

Figure 27. Correspondence Between FORTRAN Logical Units and System Files

The dump subroutine can be used in a FORTRAN IV program, a MAP program, or a COBOL program. The following CALL statements are used in both FORTRAN IV and MAP programs:

```
CALL    DUMP(A1,B1,F1, . . . . . ,Ai,Bi,Fi)
CALL    PDUMP(A1,B1,F1, . . . . ,Ai,Bi,Fi)
```

where A and B, which are symbolic addresses, specify the limits of the area to be dumped.

F indicates the dump format and can be one of the following integers:

0 octal dump
1 floating-point dump
2 integer dump
3 octal dump with mnemonics

After entering linkage mode, the following statements can be used in a COBOL program to call the dump subroutine:

```
CALL 'DUMP' USING data-name-1, data-name-2,
                  numeric-literal-1 [, data-name-3,
                  data-name-4, numeric-literal-2, . . . . .]
CALL 'PDUMP' USING data-name-1, data-name-2,
                   numeric-literal-1 [, data-name-3,
                   data-name-4, numeric-literal-2, . . . ]
```

The data-names represent any working storage areas or constants. If the data-name is a file-name, the file control block for that file is written on the system output unit.

The numerical literals are integers that specify the format of the dump. The integers and the corresponding dump formats are the same as those for a FORTRAN IV or MAP program.

If an integer for the format is not given when this subroutine is used, the dump format is octal. If no limits are specified following the entry point and if the format is not specified, all of core storage is dumped in octal.

FORTRAN Files

The input/output devices used in data transmission are always referred to symbolically in FORTRAN IV programs. Symbolic references may be made to a constant FORTRAN logical unit or to a variable unit in the standard FORTRAN input/output package as well as the alternate package.

Constant Units

Any FORTRAN IV source program input/output statement that refers to a constant unit (for example, READ(1, 10)A, where the reference is to the constant FORTRAN logical unit 01) causes the library file routine corresponding to that unit to be loaded with the object program. A file routine contains a MAP language FILE pseudo-operation that determines various file specifications, such as unit assignment, block size, and file type. The unit assignment specification establishes correspondence between FORTRAN logical units and symbolic units. See Figure 27.

If additional logical units are desired, a file routine, in the following format, must be inserted into the user's program:

```
            ENTRY    .UNxx.
.UNxx.      PZE      UNITxx
UNITxx      FILE     Specifications
```

where xx is a two-digit FORTRAN logical unit number.

If the additional logical units are to be permanent, a file routine must be inserted in the Subroutine Library and an entry must be made in the table that describes these routines in the FVIO subroutines.

Variable Units

Any FORTRAN IV source program input/output statement that refers to a variable unit causes the FVIO subroutine (.FVIO in the alternate input/output package) 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 FVIO subroutine (or .FVIO) takes the value of the variable unit at the time the variable input/output statement is to be executed, and refers to a table to determine which file routine is required.

Under the IBJOB Processor system a programmer can submit jobs divided into sections. Each individual section can be in MAP, FORTRAN, COBOL, or relocatable binary. IBJOB can thus process sections in different languages in the same job. This flexibility has useful consequences:

1. Programmers can use the best features of each language. For example, the flexibility in the control and manipulation of data under the MAP language can be combined with the simplicity of FORTRAN mathematical capabilities.
2. Sections of a large job can be coded and tested by different programmers at different times, and then combined.
3. Jobs that have already been tested can be saved in relocatable binary card form for later combination with other sections, thus reducing compile and assembly time.

Examples of Programming in Sections

Configurations that can be used in multilanguage programming are shown in Figures 28, 29, and 30. Note that any one of the decks shown in these examples can be replaced by its relocatable binary equivalent (see "Relocatable Binary Decks").

NOTE: If the entry point to a called COBOL program section is defined – as in Figure 28 – by the deckname of the COBOL section, the call to the COBOL section cannot include parameters and return must be made through the STOP RUN verb (see a description of the STOP RUN verb in the COBOL manual). But if the entry point is defined – as in Figure 29 – by the ENTRY POINT clause of the ENTER verb, return to the calling deck is by the RETURN clause of the ENTER verb. The latter method of definition is preferable.

Grouping FORTRAN Source Decks

To make the best use of the new FORTRAN IV Compiler speed, the programmer should group all FORTRAN source decks together in a multilanguage program.

COBOL-FORTRAN Program Adjustments

Programmers who wish to use COBOL and FORTRAN decks in the same program should be aware that they may encounter some problems. For example, FORTRAN and COBOL references to multidimensional arrays are made on a different basis.

```
1       8        16
$JOB             MAP MAIN PROGRAM CALLING FORTRAN SUBROUTINE
$EXECUTE         IBJOB
$IBJOB           MAP
$IBMAP  MAP1     NODECK,M94
MAP1A   SAVE
        .
        .
        .
        CALL     FTC1A(P1,P2)
        .
        .
        .
        RETURN   MAP1A
        END
$*FOLLOWING IS A FORTRAN SUBROUTINE CALLING A COBOL SUBROUTINE
$IBFTC  FTC1     NODECK,M94,XR7
       SUBROUTINE FTC1A(A,B)
        .
        .
        .
       CALL CBC1
        .
        .
        .
       RETURN
       END
$*FOLLOWING IS A COBOL SUBROUTINE WITH AN
$*ENTRY POINT THROUGH THE DECKNAME
$IBCBC  CBC1     NODECK,XR7
        .
        .
        .
                 STOP RUN.
$CBEND
(END-OF-FILE CARD)
$STOP
```

Figure 28. MAP Main Program Calling FORTRAN Subroutine

```
1      8        16
$JOB            FORTRAN MAIN PROGRAM CALLING MAP, COBOL
$EXECUTE        IBJOB
$IBJOB          MAP
$IBFTC MAIN     NODECK,M94,XR7
       .
       .
       .
      CALL MAP1A(A,B)
       .
       .
       .
      CALL CBC1A(C,D)
       .
       .
       .
      STOP
      END
$*FOLLOWING IS A MAP SUBROUTINE CALLING A COBOL SUBROUTINE
$IBMAP MAP1     NODECK,M94,XR7
MAP1A  SAVE
       CLA*     3,4        (PICK UP MAIN PROGRAM ARGUMENT A)
       .
       .
       .
       SXA      SVXR4,4
       CALL     CBC1A(E,F)
SVXR4  AXT      **,4       (RESTORE XR4)
       .
       .
       .
       STO*     4,4        (STORE IN MAIN PROGRAM ARGUMENT B)
       RETURN   MAP1A
       END
$*FOLLOWING IS A COBOL SUBROUTINE WITH ENTRY POINT DEFINED BY
$*ENTRY POINT CLAUSE
$IBCBC CBC1     NODECK,M94,XR4
       .
       .
       .
       P1. ENTER LINKAGE-MODE.
           ENTRY POINT IS 'CBC1A'
           RECEIVE FACTOR
           PROVIDE RESULT.
       P2. ENTER COBOL.
       .
       .
       .
       P3. ENTER LINKAGE-MODE.
           RETURN VIA 'CBC1A'.
       P4. ENTER COBOL.
       .
       .
       .
           STOP RUN.
$CBEND
(END-OF-FILE CARD)
$STOP
```

Figure 29. FORTRAN Main Program Calling MAP and COBOL Subroutines, and MAP Subroutine Calling COBOL Subroutine

Another problem occurs when a COBOL deck and a FORTRAN deck refer to the same physical input unit. In such a case, the IOCS look-ahead buffering feature could disrupt the sequence of information read in. The programmer can avoid the problem by placing the data referred to by each deck on a separate physical unit.

Where COBOL and FORTRAN decks in a program refer to the same physical output unit and the order of the data written on the file is not important, the programmer can ignore the problem. If the COBOL file is closed first, however, it should be closed with no rewind.

```
1       8       16
$JOB            COBOL MAIN PROGRAM CALLING FORTRAN, MAP
$EXECUTE        IBJOB
$IBJOB          MAP
$IBCBC COBOL1   LIST
        .
        .
        .
        P1. ENTER LINKAGE-MODE.
            CALL 'FTC1A' USING ADDEND, SUM.
            CALL 'MAP1A' USING TAX, DISCOUNT,
            RETURNING ERROR-RETURN.
        P2. ENTER COBOL.
        .
        .
        .
        ERROR-RETURN
        .
        .
        .
            STOP RUN.
$IBFTC FTC1     LIST
      SUBROUTINE FTC1A(A,B)
        .
        .
        .
      RETURN
      END
$IBMAP MAP1     LIST
MAP1A  SAVE     E
        CLA*    3,4         (PICK UP FIRST ARGUMENT)
        .
        .
        .
        STO*    4,4         (STORE IN SECOND ARGUMENT)
        RETURN  MAP1A,1     (ERROR RETURN)
        .
        .
        .
        RETURN  MAP1A       (NORMAL RETURN)
        END
$STOP
(END-OF-FILE CARD)
```

Figure 30. COBOL Main Program Calling FORTRAN and MAP Subroutines

Processor Monitor Information

The Processor Monitor provides a common operating environment for the IBJOB language translators. It supervises loading of the IBJOB components and regulates the compiling, assembling, loading, and executing of a job. The Monitor consists of three parts: job control, process control, and the input/output editor. Figure 31 shows the relationships among job control, process control, and the other components in the IBJOB Processor system.

Job control is called into storage by the IBSYS Monitor when a $EXECUTE card specifying IBJOB is encountered in a source program. It directs the over-all processing of a job; it calls process control into storage to supervise the assembly of a source program into binary form for the Loader; it calls the Loader to load the assembled program and to start its execution. Process control returns to job control at the end of the assembly process. The Loader returns when a job cannot be executed. When assembly errors are serious enough to halt processing or a program cannot be executed, job control returns to the IBSYS Monitor.

After execution the object program returns directly to the IBSYS Monitor, because the Loader has almost entirely overlaid the IBJOB Monitor with the object program. Usually, the IBSYS Monitor then reads the input file to determine the requirements of the next job. But if load-time debugging is requested for the program just executed, the IBSYS Monitor calls job control back into storage. Job control calls in process control to initiate processing of the debugging information obtained during execution of the program.

Process control supervises the assembly of a source program by calling into storage the Load-Time Debugging compiler, the Assembler, the COBOL Compiler, or the FORTRAN Compiler, as they are required to translate the different types of decks. It also calls in the debugging postprocessor routines after a program has been executed for which load-time debugging is requested. After the debugging listings are written on the output tape, control returns to process control, which looks for the next $EXECUTE card on the output file. If this card specifies IBJOB, process control proceeds

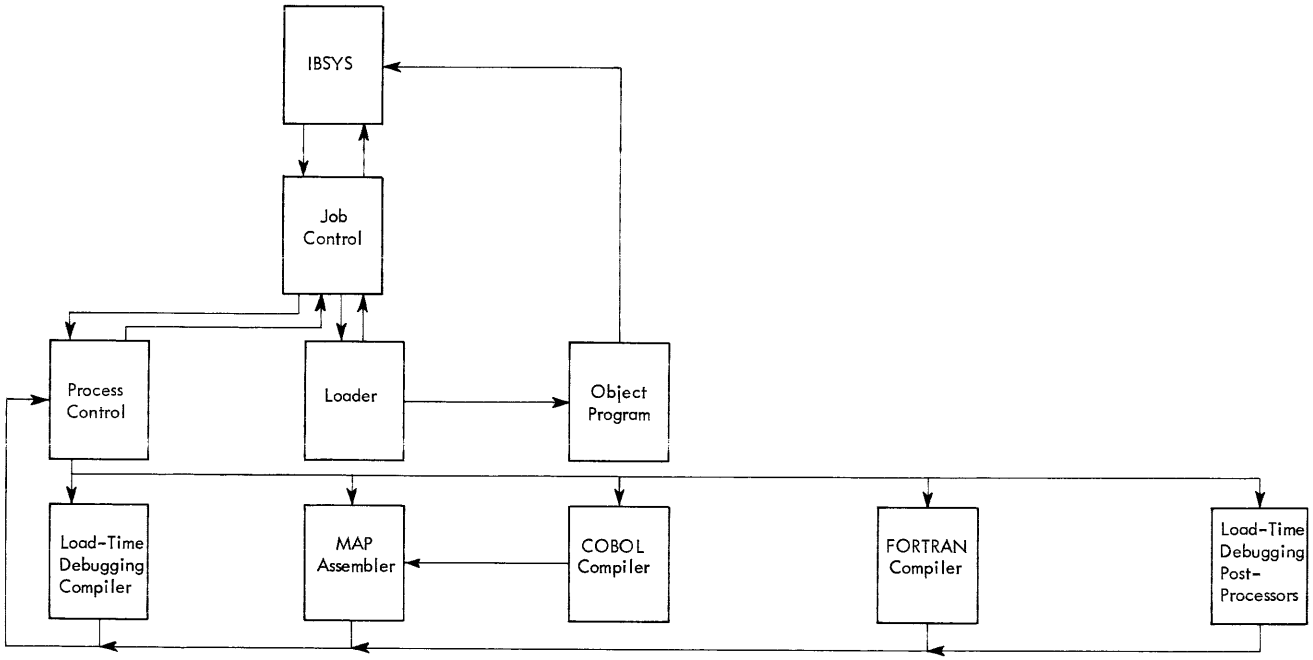


Figure 31. Relationship among Job Control, Process Control, and other IBJOB Components

as usual to supervise the assembly of the next program. If IBJOB is not requested, process control returns to the IBSYS Monitor.

The input/output editor is called into storage with process control, which uses it to regulate the actions of the Input/Output Executor whenever any Processor input/output functions are to be performed.

The following is an example of how the three sections of the Monitor control the flow of processing for a typical program:

PROGRAM		PROCESSOR ACTIONS
$IBSYS $JOB $EXECUTE	IBJOB	The IBSYS Monitor calls job control into storage. Job control calls process control into storage.
$IBJOB	MAP, LOGIC	Process control initializes the input/output editor and uses it to read the $IBJOB card. It scans options, opens the load file for the Assembler and FORTRAN Compiler output to the Loader, and uses the input/output editor to read the next card on the input file.
$IBMAP	NODECK, M94	Process control calls the Assembler into storage.
(MAP deck with CALL DUMP as last step)		The Assembler uses the input/output editor to read the MAP deck from the input file, then translates the deck into binary form for Loader processing. During assembly the input/output editor writes the Assembler output on the load file for Loader processing and on the output file for printed listings. The Assembler returns control to process control, which uses the input/output editor to read the next card on the Process control input file.
$IBFTC	NODECK, M94	Process control calls the FORTRAN Compiler into storage.
(FORTRAN deck called by MAP deck)		The FORTRAN Compiler uses the input/output editor to read the FORTRAN deck from the input file. The Compiler assembles the deck into binary form for Loader processing. During assembly the input/output editor writes the Compiler output on the load and output files. The Compiler returns control to process control. Process Control uses the input/output editor to read the next card on the input file.
(end-of-file-card)		Process control returns control to job control, indicating the program can be loaded. Job control transfers control to the Loader, which loads the program. The Loader uses the input/output editor to read the load file and to write the core storage map and cross-reference table on the output file. The Loader overlays all of the IBJOB Monitor with the object program except for a few saved communication words.
		It transfers control to the object program, which executes. The dump routine called by the program writes the dumped information on the output file. The program post-execution routine returns control to the IBSYS Monitor.
$IBSYS $STOP		

Job Control Operations

Job control contains routines to determine whether it should call process control or the Loader into storage. It also contains an ACTION routine that can call the IBJOB system components into storage.

When job control receives control from the IBSYS Monitor, it reads the system units position table into the IBJOB communication area (See Appendix C, "IBJOB Communication Area") and then uses the table to call process control into storage. The system units position table lists the position of each record on the symbolic input/output unit (usually SYSLB1) on which the IBJOB system programs are stored.

When a source program has been assembled, process control transmits a word to job control. By testing this word, job control determines whether processing should be continued by calling the Loader into storage, or whether it should be halted by transferring control to the IBSYS Monitor through location SYSRET in the IBSYS nucleus. If the program cannot be executed after it is loaded, the Loader returns control to job control, which returns to SYSRET.

ACTION Routine for Calling IBJOB Components

The ACTION routine is used by both job control and process control to locate IBJOB components on the system library file and to read them into storage when they are needed. The routine can be used to position the system library to a particular system record or to position to and read one or more system records. It is not necessary for the calling program to know the order or format of the records on the system library. All that the calling program need supply to the ACTION routine is the label of the required action. The ACTION routine consults three tables and performs the desired action. These tables are the action table, the action list, and the system unit position table.

The action table is used to identify the action label supplied by the calling program. Each entry consists of two words, the first of which is:

```
BCI  1, label
```

where label is a six-character alphameric name of the required action. The second word is either

```
PZE  a
```

where a is the location of the action list, or

```
pfx    n
```

where the second word is the action list itself, pfx is either MZE or MON, and n is an entry in the system unit position table. In this case, the action list is limited to one word.

The action list may consist of any number of one-word entries, the last of which must have a negative prefix code. Each word is of the form:

```
pfx    n
```

where n is again an entry in the unit position table and pfx is one of the following:

1. PZE or MZE, which causes the system unit to be positioned as indicated.
2. PON or MON, which causes the system unit to be positioned as indicated and the record at that position to be read.

The unit position table consists of a one-word entry for each standard record on the input/output unit(s) where the IBJOB system is stored. An entry is designated by the n portion of an action list entry. The unit position table can have either of two formats, depending on whether the system is on one or more than one magnetic tape unit, or on disk storage.

When tape is being used, the format for an entry is:

```
pfx    rccnt, , flcnt
```

where pfx is the library tape number on which the record exists and where rccnt and flcnt indicate the position of the record on that unit. Flcnt is the number representing the file position of a particular component. Rccnt is the number representing the record position within the component file.

If pfx is PON, indicating system library unit 1 (SYSLB1), the file position is relative to the position of the first IBJOB Processor file. When job control reads this relative position table into core storage for process control use, it also converts the flcnt part of each entry to an actual file position. The actual file position is the number of files between the load point and the system component involved. If pfx is other than PON, flcnt is already the actual file position.

When the system is stored on disk or drum, the unit position table is automatically converted to a track position table. The decrement portion of each entry contains the starting track address of the address of the record. These track addresses are obtained by scanning the table of record names and track addresses contained in the IBSYS Supervisor.

System Record Format

Each standard system record is preceded by the following sequence:

```
IOCP SYSFAZ, , 1
BCI  1, recnm
```

This is followed by a command to read a transfer to an entry point into the appropriate transfer location in job control or the component involved.

The rest of the record is scatter-loaded, each section being preceded by the proper IOCX command, the last of which must be IOCT.

Prepositioning Feature

The ACTION routines in the Processor Monitor are also used to preposition a system library unit when possible. If the system is assembled for disk, drum, or Hypertape, the prepositioning feature is inoperative.

Using One Library Unit: If the IBJOB Processor is stored on an IBM 729 Magnetic Tape Unit, the prepositioning feature performs the following actions:

IBJOB Processor components are stored on a system library tape in the following order: Monitor, Load-Time Debugging compiler, COBOL Compiler, Assembler, Loader, and Library. The Assembler is read into core storage in two stages. After the last record of the Assembler is read in, the next control card is read. If this control card is a $IBFTC, $IBCBC, or $IBMAP card, a backspace file operation is performed. If the next control card is not one of these three cards, no action is performed.

If the IBJOB Processor is stored on either an IBM 729 Magnetic Tape Unit or an IBM 7340 Hypertape Drive, the system tape is rewound after the last record of the Loader has been read into core storage.

Using Multiple Library Units: Two different configurations are possible when multiple library units are used to store the IBJOB Processor system programs. Either two IBM 729 Magnetic Tape Units can be used; or one tape unit can be used for the Subroutine Library, with the rest of the Processor system on disk or drum storage.

The configuration of two tape units makes full use of the prepositioning feature. Both units must, however, be on the same channel and all records of a component of the IBJOB Processor must be on the same tape. If more than two units are used, the prepositioning feature applies only to the unit that contains the Processor Monitor. Any other units are not rewound but only positioned when a record is required.

After the last record of the Assembler is read into core storage, the next control card is read. If this control card is a $IBFTC, $IBCBC, or $IBMAP card, a backspace file operation is performed for the tape that contains the Processor Monitor. The other system tape is rewound. The backspace file operation is not begun if neither of the compilers nor the Assembler is on the tape that contains the Processor Monitor.

After the last record of the Loader has been read into core storage, both system tapes are rewound.

The configuration using tape with drum or disk storage requires that only the files for the Library be on tape. The tape for the Library must be on an IBM 729 Magnetic Tape Unit. After the last record of the Loader has been read into core storage, the Library tape is rewound.

Process Control Operations

Process control contains routines to initialize the input/output editor and direct its operations. It also contains routines for controlling the assembly of a source program and for calling the load-time debugging post-processors. It uses a control card search routine mainly to determine which component must be called into storage during the assembly process and to obtain information for the input/output editor. It uses an option scan routine to record job specifications for use by the other IBJOB components, and it uses an error procedure routine after each component has assembled a deck, to evaluate the effect of program or machine errors on future processing. Process control uses the ACTION routine in job control to read the components into storage.

Initialization of the Input/Output Editor

After being called by job control, process control scans the system unit function table in the IBSYS nucleus to make sure that the necessary input/output units have been assigned to Processor functions. It then attaches and opens the files it will use and initializes the input/output editor.

Process control initializes the input/output editor by supplying to it through the entry point IOEDIT the locations of the file control blocks for the input/output units that will be used during processing and the location of the Monitor communication word COMCEL. Certain bits in COMCEL are used to store information for the input/output editor. Process control opens and closes all input/output editor files and positions them when necessary. All file control blocks used by the input/output editor are located in process control. Process control and the input/output editor normally use system units for the following purposes:

System Library Unit (SYSLB1): This is the storage unit for the IBJOB Processor. The system may also be stored on several units. In this case, alternate units, such as SYSLB2, SYSLB3, and SYSLB4, would also be used.

System Input Unit (SYSIN1): This is used for the IBJOB Processor input. Alternate units may also be used if required.

System Output Unit (SYSOU1): This is used for the IBJOB Processor listing output. Alternate units may also be used if required.

System Peripheral Punch (SYSPP1): This is used for the IBJOB Processor punched output.

System Utility Unit 1 (SYSUT1): This is not used by the process control or by the input/output editor.

System Utility Unit 2 (SYSUT2): This is used for the IBJOB Processor load file.

System Utility Unit 3 (SYSUT3): This is not used by process control or by the input/output editor.

System Utility Unit 4 (SYSUT4): This is used for COBOL compiler output to the Assembler.

System Alternate Checkpoint Unit (SYSCK2): This is used for the debugging file, which contains output from the Load-Time Debugging compiler, the Loader, and the Load-Time Debugging interpreter routines. It is used by the Load-Time Debugging editor and translator as the input file. If the system is stored on more than one unit, SYSLB2, SYSLB3, and SYSLB4 may also be used. Alternate units for input and output can also be designated if required.

Control Card Search

After initialization of the input/output editor, process control starts to search for control cards. The control card search routine calls the input/output editor to get the next line of input. If the current job is to be loaded, any card that contains a dollar sign in column 1 and is not recognized as an IBJOB Processor control card is added to the load file, which is the basic input to the Loader. An unrecognized card is ignored if the program is not to be loaded. Each recognized card is listed and may be printed on-line, and any necessary action, which may include a scan for options, is taken. The option scan routine is described under "Process Control Option Scan."

Cards without a dollar sign in column 1 are not added to the load file. Binary cards that are not within an object deck cause an error message to be written. Programs that are assembled using the ABSMOD option on the $IBMAP card are not added to the load file.

$IBJOB Card Action

The $IBJOB card must be the first card of every IBJOB Processor application. Unless either the NOSOURCE option or the NOGO option without a MAP, LOGIC, or DLOGIC option is specified, process control starts a load file and places the $IBJOB card in it as the first card. Any cards that contain a dollar sign in column 1 and that are not recognized by process control, as well as any object decks and Assembler output, are placed in the load file. If the NOSOURCE option is present, signifying that there is no compilation or assembly in this application, or a $IBREL card is encountered, signifying that there are only object decks from that card on, process control returns control to job control. Job control, in turn, calls the Loader to load from the system input unit.

Component Control Card Action

Cards that signal process control to call other IBJOB components into storage are listed below.

$IBDBL Card: When the $IBDBL card is encountered, process control sets the input/output editor controls for a debugging compilation and calls the compiler section of the Load-Time Debugging Processor. The Load-Time Debugging compiler calls the input/output editor for this control card, which must immediately precede every debugging request deck.

$IBFTC Card: Process control sets the input/output editor controls for FORTRAN IV compilation and calls the FORTRAN IV Compiler when the $IBFTC card is encountered. The FORTRAN IV Compiler calls the input/output editor for this control card, which must immediately precede every FORTRAN IV source deck, into storage.

$IBCBC Card: Process control sets the input/output editor controls for a COBOL compilation and calls the COBOL Compiler when the $IBCBC card is encountered. The COBOL Compiler calls the input/output editor to read this card. It must immediately precede every COBOL source deck into storage. Process control automatically calls the Assembler to assemble COBOL Compiler output if the error level permits assembly.

$IBMAP Card: Process control sets the input/output editor controls for a MAP assembly and calls the Assembler when a $IBMAP card is encountered. The Assembler calls the input/output editor to read this control card. It must immediately precede every symbolic deck into storage.

$IBLDR Card: Process control adds the $IBLDR control card and either the object deck following this control card, or its complement object deck if an alternate input unit specified, to the load file if the program is to be loaded. If the LIBE specification is found on the $IBLDR control card, only this card is added to the load file, and the alternate input is not sought. A $IBLDR card is the first card in the output deck of the Assembler.

Optional Control Card Action

Process control recognizes some control cards that can be considered independent of components. Typically, these cards control accounting functions or denote input/output variations from normal procedure. These cards are listed in the following text.

$ID Card: The $ID control card causes process control to transfer to the installation accounting routine.

$IEDIT Card: Process control uses the variable field of the $IEDIT control card to set input specifications for the application. It may appear at any place in a program, and the specifications remain in effect for the remainder of the application unless changed by another $IEDIT card.

$OEDIT Card: Process control uses the variable field of the $OEDIT card to set output specifications for the application. It may appear at any place in a program, and the specifications remain in effect for the remainder of the application unless changed by another $OEDIT card.

$* *Card:* The $* control card is a comments card, and its contents are printed on-line. No other action is taken. It may appear in any group of control cards.

$PAUSE Card: Processing halts when a $PAUSE card is encountered. The START button must be pressed in order to proceed.

$ENDREEL Card: A reel switch is performed between system input units SYSIN1 and SYSIN2 when a $ENDREEL card is encountered. This control card must be preceded by a file mark. This control card is recognized only by the input/output editor and the minimum, basic, and label levels of IOCS.

$IBREL Card: The $IBREL card indicates to the Processor Monitor that no more compilations or assemblies follow on the system input unit (SYSIN1).

$TITLE Card: The information contained in columns 16-72 is placed in the appropriate words of the heading buffer. The current date is also placed in the buffer unless the NODAT option has been specified. The title is printed on the text page, and a switch is set to indicate that the next Processor or Assembler control card should not cause the heading buffer to be reinitialized.

$DATA or End-of-File Card: Control is transferred to the Loader, if loading can be performed, when a $DATA or end-of-file card is encountered.

IBSYS Control Card Action

Control is transferred to the IBSYS Monitor when a $IBSYS, $JOB, $EXECUTE, or $STOP card is read or when execution of an object program has been completed.

Process Control Option Scan

An option scan may take place if a recognized control card has options that can be specified. Process control looks for all the options on $IBJOB, $IBLDR, $IEDIT, and $OEDIT cards and for the ABSMOD option only on $IBFTC, $IBCBC, and $IBMAP cards. Options on $IBDBL, $IBFTC, $IBCBC, and $IBMAP cards are scanned only by the component program except for the ABSMOD option on $IBFTC, $IBCBC, or $IBMAP cards. Scanning begins in column 16 and ends when a blank character is encountered. Each set of characters terminated with a comma, or a blank in the case of the last set, is treated as an option. The option is compared to a dictionary of options and the proper action is taken if a matching entry is found. Unrecognized specifications are ignored in all cards but the $IEDIT and $OEDIT cards. If an unrecognized specification is found on either of these cards, an error message is given. If specifications are not found on a control card, the standard option of the installation is assumed.

Process Control Error Procedure Routine

When the FORTRAN IV Compiler, the COBOL Compiler, or the Assembler returns control to process control, an error word is left in the accumulator. If no error was detected by the subsystem, the accumulator address and decrement portions contain zero. A suspected machine error is indicated by a nonzero decrement, and a source program error is indicated by an error level number in the address. This error level number determines the error procedure used by process control:

LEVEL	PROCEDURE
1	Allow loading if requested.
2	Do not allow loading.
3 or greater	Do not allow loading. If the return is from the COBOL Compiler, do not allow assembly.

Although the compiler section of the Load-Time Debugging Processor also prints error messages, the errors do not prevent loading or execution of the source program. If no source program errors are indicated, a machine error indication causes process control to print a message on-line specifying the possible operator options and then to pause. The options are to retry the application, to go on to the next application, or to go on to the next job. The operator must specify one of the options and press START. If the retry option is chosen, the system input, system output, and system peripheral punch units are returned to their original positions at the beginning of the application and control is returned to the control card search routine. No alternate input or output units are repositioned by process control.

If a source program error of level 2 or greater is indicated, process control does not allow execution of the program or retry options, but goes on to the next control card, regardless of whether a machine error accompanied the source program error.

Input/Output Editor Operations

The input/output editor performs all Processor input/output functions. It provides a line of input or accepts punch or listing output upon request. The input/output editor writes COBOL Compiler output and reads it for the Assembler. It also writes the load file and reads it for the Loader, but it does not perform intermediate input/output operations or on-line printing.

Process control transfers to the input/output editor through the entry point IOEDIT. All information taken from $IEDIT, $OEDIT, or $IBJOB cards that affect the input/output editor is transmitted through this entry point.

The input/output editor consist of four sections:

1. The IOEDIT utility routine, which initializes and controls the other sections.
2. The input editor, which regulates input to the Processor.
3. The output editor, which regulates all listing output from the Processor.
4. The punch editor, which regulates all punched output from the Processor.

IOEDIT Routine

The IOEDIT routine uses the information transmitted to it by process control to select the proper editor section for the job requested. It initializes this section by setting flag bits in the required file control blocks, truncating buffers and releasing them, and transferring control to the section.

Input Editor

The input editor contains two reading routines that use the Input/Output Control System (IOCS). The primary routine reads only the input file on the system input unit (SYSIN1). The secondary one reads an input file on an alternate unit. This file may be the Assembler input file (output from the COBOL Compiler), the load file input to the Loader (Load-Time Debugging compiler, Assembler, or FORTRAN Compiler output, or previously assembled binary decks), an input file on an optional unit specified on a $IEDIT card, or an Alter deck that the Monitor has moved to the system utility unit (SYSUT2). Only one secondary file can be open at a time. Primary files and secondary files may consist of BCD card images, binary card images, or Prest input. BCD card images must be recorded in the BCD mode, 84 characters per card image. Binary card images must be recorded in binary mode, of 168 characters per card image. The last four characters of BCD records and the last eight characters of binary records must be standard look-ahead bits. These bits are described in the publication *IBM 7090/7094 IBSYS Operating System, Version 13, Operators Guide,* Form C28-6355. The input editor accepts blocked input of up to ten BCD cards or up to five binary cards in each physical record. $-control cards must be unblocked.

After being called by a request for a line of input, the input editor determines, from a control location set by process control, whether input is to be read from the primary file or from the secondary file. It then locates the next line and returns control to the calling program, leaving the location of this line and its word count in the accumulator. To ensure that a line is saved, the calling program must move it before requesting another line.

If an error condition is sensed, the input editor returns control to the calling program, sets the sign of the accumulator to minus and puts a 1 in its address portion. If an end-of-file condition is sensed, the input editor returns control to the calling program, sets the

accumulator to minus and puts a 0 in the address portion.

Files used by the input editor are opened, positioned if necessary, and closed by process control. Process control transmits the locations of the file control blocks to the input editor to allow their initialization. This transmission is performed through IOEDIT, the entry point to the input/output editor.

Output Editor

The third section of the input/output editor is the output editor. It can list on more than one output unit.

Normally, the listing file is on the system output unit (SYSOU1). If an alternate output unit has been specified on a $OEDIT card, the output editor places the listing output of the Processor Monitor on the system output unit, but it uses the alternate output unit for all IBJOB Processor component listing output until the end of the application or until a $OEDIT card specifying the system output unit is encountered.

The output editor keeps page counts and line counts, and it ejects pages and inserts page headings when necessary. The page headings are given to the output editor by process control through IOEDIT.

The output editor generates two types of output. The contents of the word at location TYPOU determines the type of output generated. When location TYPOU is zero, the output is in BCD mode, blocked up to five lines per block. This output can be printed on the IBM 720 Printer. When the contents of TYPOU are nonzero, the output is in binary mode, blocked up to five lines per block. This output can be printed off-line, using the IBM 1401 Peripheral Input/Output Program. The first word of each output block is a block control word. The word contains $(7600000000xx)_8$, where xx is the number of records (in BCD) contained in the block. The first word of each record within the block is a record control word. This word contains $(5xxxxx200460)_8$, where xxxxx is the number of characters (in BCD) contained in the record.

A call to the output editor initiates a new line when it is requested by the calling sequence and when the last line has already been filled.

Process control opens and closes files used by the output editor and transmits the locations of the file control blocks to the output editor by means of IOEDIT.

Punch Editor

The punch editor accepts 80-column card images for three types of output. Card images may be either column binary or BCD, and the three types of output are:

Punch file
Load file
Compiler output file

All punch editor output files are in binary form. BCD card images for the punch file are recorded in column binary form, without column binary bits, when they are placed in the punch file. BCD card images for the other files are inserted in BCD form and are written in binary form. Duplicates of the $IBJOB control cards are punched when they are read.

If the punch editor determines that output is not from the COBOL Compiler, the card image is placed in the punch file. If it is in BCD form, the punch editor records it in column binary form, without column binary bits. The card is also placed in the load file if the proper bit in the control word COMCEL has been set.

File control blocks for the punch editor files are kept in process control. Their locations, along with the location of control flags, are transmitted to the punch editor through the entry point IOEDIT.

IBJOB Processor Maintenance Cards

$DUMP Card

The $DUMP card causes the specified portion of system records to be dumped. The $DUMP card must be inserted in a source program after the $IBJOB card and before the control card calling the component from which the dumped information is to be taken. It cannot, however, be placed within a particular deck. As an example, consider the following sequence of cards:

```
                $IBJOB
[insertion point a]
                $IBMAP
                [MAP deck]
[insertion point b]
                $IBFTC
                [FORTRAN deck]
```

A $DUMP card referring to the FORTRAN Compiler could be inserted at either point a or point b. A card for the Assembler, however, can be inserted only at point a.

The format of a $DUMP card is:

```
1          6  8          16
$DUMP      n  cxxxxx     loc1/loc2, loc3/loc4, . .
```

where n is a digit that designates whether the output is to be single-spaced or double-spaced. A 1 in column 6 designates single-spaced output. Any digit greater than 1 designates double-spaced output. If this field is omitted, the output is single-spaced.

The field starting in column 8 contains an alphameric character (c) and a five-digit octal number (xxxxx) that specifies an absolute location. The alphameric character is the sixth character of the name of the system record whose loading causes a dump request to be inserted. The dump occurs immediately before the instruction at location xxxxx is executed. Location xxxxx may, if needed, be outside the system designated by character c. (See "Restrictions on Dump Requests.")

The field starting in column 16 contains the limits of those portions of core storage to be dumped. Each portion of core storage is specified by two five-digit octal numbers. The lower limit is specified first and is separated from the upper limit by a slash. Consecutive sets of limits must be separated by commas. The first blank encountered in the variable field designates the end of the control card. Another $DUMP card that specifies the same system record character and location may be used to extend the variable field. The portions of core storage

are dumped in the reverse order of their appearance on the card. The $DUMP card specifications are effective only during the processing of the source program in which they are inserted.

The first $DUMP card that is read causes the dump routine to be loaded into core storage starting at location 76237_8. If an accounting routine is in core storage, it is overlaid by the dump routine, and locations SYSIDR and SYSPID in the communications region of the IBSYS nucleus are reset. The upper core storage limit in location IBJCOR in the Processor Monitor communication area is set to 76236_8.

Restrictions on Dump Requests

Dump requests have the following restrictions:

1. Certain system records (IBJOB, JDUMP, IBJOBB, and IBJOBC) are already in core storage when dump requests are made. For this reason a $DUMP card specifying any of these records has no effect. A dump request may, however, be inserted into any location in core storage, including the areas occupied by these records, by using the sixth character of a more accessible record name. For example, when the system record IBMAPJ is called into core storage, a dump request specifying J11526 may be inserted, starting in column 8 on the $DUMP card. Location $(11526)_8$ is actually in the IBJOB system record, which is in core storage at the same time as IBMAPJ. In this way information can be dumped during the operation of IBJOB without actually specifying it.

If system record IBMAPJ were called into core storage more than once after the $DUMP card had been read, a loop would occur. This is due to the method used for inserting dump requests. This method is described in item 4.

2. Dump requests may not be inserted in IOCS coding or in output editor coding, since the dump routine uses both IOCS and the output editor for processing output. If dump requests are inserted in these areas, a loop occurs.

3. The system records IBLDRP and IBLDRQ cannot be dumped, since these records occupy the same area of core storage as the dump routine.

4. $DUMP requests may not replace TSX instructions that are followed by parameters; they may not replace instructions that are modified in any way; and they may not be inserted where they will appear within CALL psuedo-operation expansions. These restrictions are due to the method used for inserting dump requests. The method is:

   a. The instruction at the location $xxxxx_8$ specified on the $DUMP card is saved in a table contained in the dump routine.
   b. An instruction to transfer control to the dump routine is placed in the specified location.
   c. When the transfer instruction has been executed, the specified portions of core storage are dumped.
   d. The instruction that was saved is placed in the following sequence:

Loc	instruction
Loc+1	TRA xxxxx+1
Loc+2	TRA xxxxx+2
Loc+3	TRA xxxxx+3

   e. When the dump is completed, control is transferred to location Loc.

$PATCH Card

The $PATCH control card is used to insert temporary patches in system records, thereby eliminating an unnecessary system edit run. The $PATCH card is inserted in a source program in the same position as a $DUMP card. The format of a $PATCH card is:

1	8	16
$PATCH	cxxxxx	instr1, instr2, . . .

where the field starting in column 8 contains an alphameric character (c) and a five-digit octal number (xxxxx) that specifies an absolute location. The alphameric character is the sixth character of the name of the system record whose loading causes a patch to be inserted. The patch is inserted starting at location $xxxxx_8$. Location xxxxx may, if necessary, be outside the system designated by cc. (See "Restrictions on Patch Requests.")

The field starting in column 16 contains 12-digit octal instructions that are to be loaded into core storage starting at location $xxxxx_8$. Consecutive instructions must be separated by commas. The first blank encountered in the variable field designates the end of the control card. Only four octal instructions may be placed on one $PATCH card.

$PATCH cards can be used to change or delete instructions in a system record. For example, the card:

```
$PATCH   M23000   076100000000
```

changes the instruction at $(23000)_8$ in record IBLDRM to a NOP. $PATCH cards can also be used to add instructions. For example, to insert the instruction ADD 16000 between the following instructions in IBMAPJ:

LOCATION	OPERATION CODE	OPERAND
12000	CLA	15000
12001	STO	14000

use the following $PATCH card sequence:

CONTROL CODE	LOCATION IDENTIFIER	INSTRUCTION IN OCTAL
$PATCH	JI2000	TRA 17000
$PATCH	JI7000	CLA 15000
$PATCH	JI7001	ADD 16000
$PATCH	JI7002	TRA 12001

where the locations 17000 through 17002 are available for patching purposes.

Restrictions on Patch Requests

Patch requests have the following restrictions:

1. System records IBJOB, JDUMP, IOCSB, IBJOBB, and IBJOBC cannot be patched directly. They can be patched, however, by using the sixth character of more accessible record names. For example, a $PATCH card for record IBLDRM specifying a location that is actually within IOCSB will effectively patch IOCSB.

2. System records IBLDRP and IBLDRQ cannot be patched, since they occupy the same area of core storage as the dump routine, which contains the patch routines.

The 7090/7094 FORTRAN IV Compiler (IBFTC) translates programs written in the FORTRAN IV language into machine language. The Compiler operates within the IBSYS/IBJOB environment and produces text for the Loader (IBLDR) compatible with that produced by the Assembler (IBMAP).

The following important features of the FORTRAN IV Compiler should be noted:

1. It has its own assembly processor, which eliminates the need for using the IBMAP assembly program.
2. It is a two-pass system, composed of an instruction generation pass (phase A) and an assembly pass (phase B).
3. It has a phasing technique for performing multiple compilations when a processor application contains a program set. (A program set is any group of consecutive FORTRAN IV source programs as shown in Figure 32. Note that a control card other than a $IBFTC card terminates a program set.) Phasing permits phase A processing for all programs of a program set, followed by phase B processing for all programs of the set. When necessary, error message processing for all programs of the set is then performed. The Compiler is thus read into core storage only once for an entire program set.

Structure of the FORTRAN IV Compiler

The Compiler is composed of the two processing phases A and B and an error message processor. Phases A and B are required for each compilation. The message processor, which contains all BCD error messages, is read into core storage only if an error has been found in one or more programs of a program set. If the message processor is required, it is called after phase B processing of the last program in a program set.

As a result of this arrangement, the FORTRAN IV Compiler printout contains the following:

1. The source programs of a program set.
2. The storage maps (and assembled program listing if requested by the $IBFTC control card) in the same order as the source programs.
3. Any diagnostic messages (again, in source program order) that may be generated.

Phase A contains the statement scanners for all source statements except the DATA and FORMAT statements. Phase B contains the DATA and FORMAT statement scanners, the storage allocator routine, and the assembly processor routine. The latter two routines

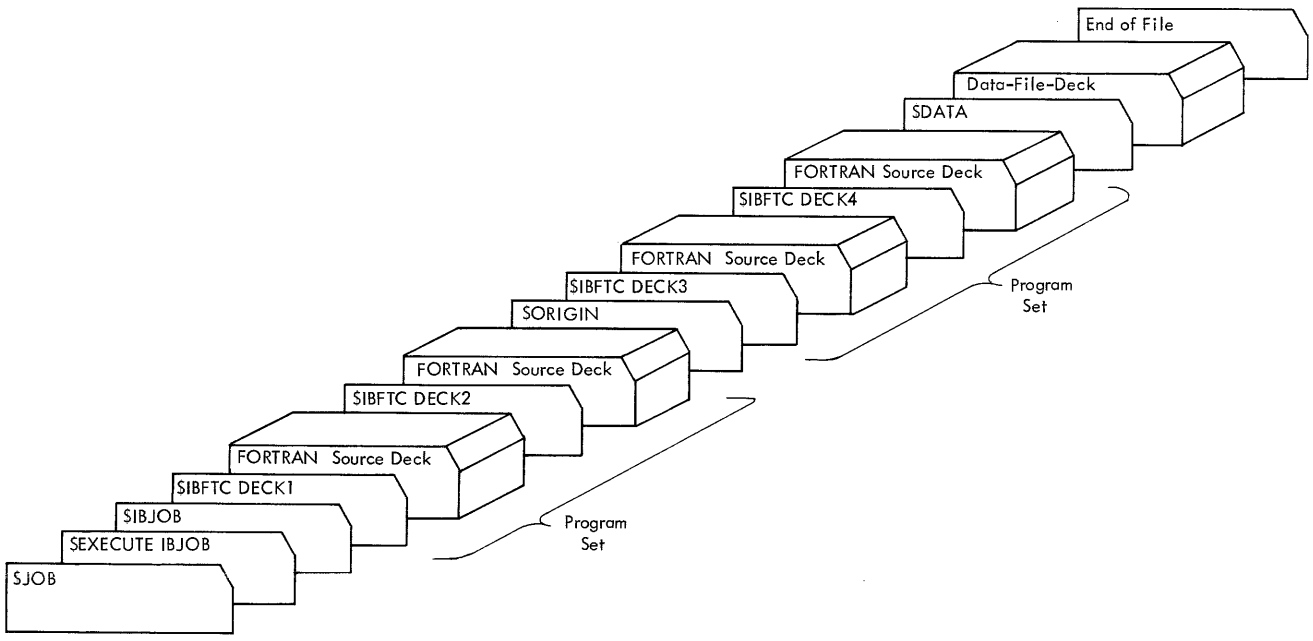


Figure 32. FORTRAN IV Program Sets in a Processor Application

provide the output for the storage maps and the listings, respectively.

The FORTRAN IV Compiler may reside on tape, disk, or drum storage. Regarded as a tape system, the Compiler consists of three records, containing phase A, phase B, and the message processor, respectively. Core storage is laid out in accordance with an overlay principle. Routines and communication areas that are common to all phases remain in the numerically lowest part of core storage. Throughout compilation, control passes from general routines to particular routines and then returns to the general routines at the completion of particular-routine processing. Figure 33 shows the overall flow of Compiler processing.

Control among parts of the Compiler is directed by the FORTRAN IV Compiler control routine (FCC). This routine passes control first to phase A control, then to phase B control, and finally, where relevant, to the message processor. Phase A control and phase B control direct the activities of their respective phases and call the required subprocessors during the time that each has control. The subprocessors most frequently use the following utility subroutines: the table routine (TR), which handles all compiler table activity; the BCI collector routine, which is used by all statement scanners for preliminary preparation of source statements for translation; the subscript processor, which handles subscript combinations; the conversion routine, which converts constants to binary representation; and the name table routine, which tabulates information about all names used in the program.

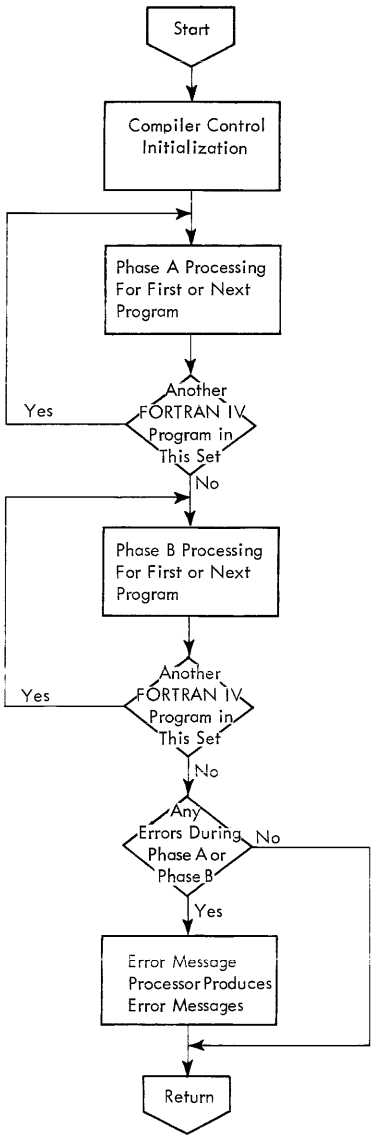


Figure 33. FORTRAN IV Compiler Overall Flow of Processing

Phase A

Figure 34 shows the flow of phase A processing. The phase A source statement scanners perform complete processing of all program statements except the DATA and FORMAT statements. For nonexecutable statements, processing results in the tabulation of information contained therein. For executable statements, processing consists of both tabulation and compilation. Compilation is the process whose final result is the transformation of a source statement into a sequence of machine instructions. Phase A compilation generates sequences of one-word and sometimes two-word internal instruction formats (IIF's). If a sequence of IIF's is associated with the same internal formula number (IFN), a word is added to the sequence to contain this number (see "Internal Formula Number Generation"). Internal instruction formats generated by the statement processors are called main file IIF's.

Main file IIF's are subsequently transformed by the phase B assembly processor into machine instructions. Certain main file IIF's (such as those related to subscript variables) are not complete when produced by the statement processors. For these cases, supplementary processing, such as is described later in Relcon analysis and Nestag processing, provides the additional information in time for the assembly processing.

Phase A processing proceeds in a statement-by-statement manner. However, when the end of a DO nest is reached, that is, the final statement of an outermost DO has been processed, control is passed to the Nestag processor. This routine performs an analysis of the entire nest, resulting in the generation of indexing instructions associated with the nest. The DO indexing IIF's are called Dotag file IIF's. They are inserted at various locations in the main file containing the DO nest, though they usually are at the beginnings and ends of the separate DOs of the nest. The fact that they must

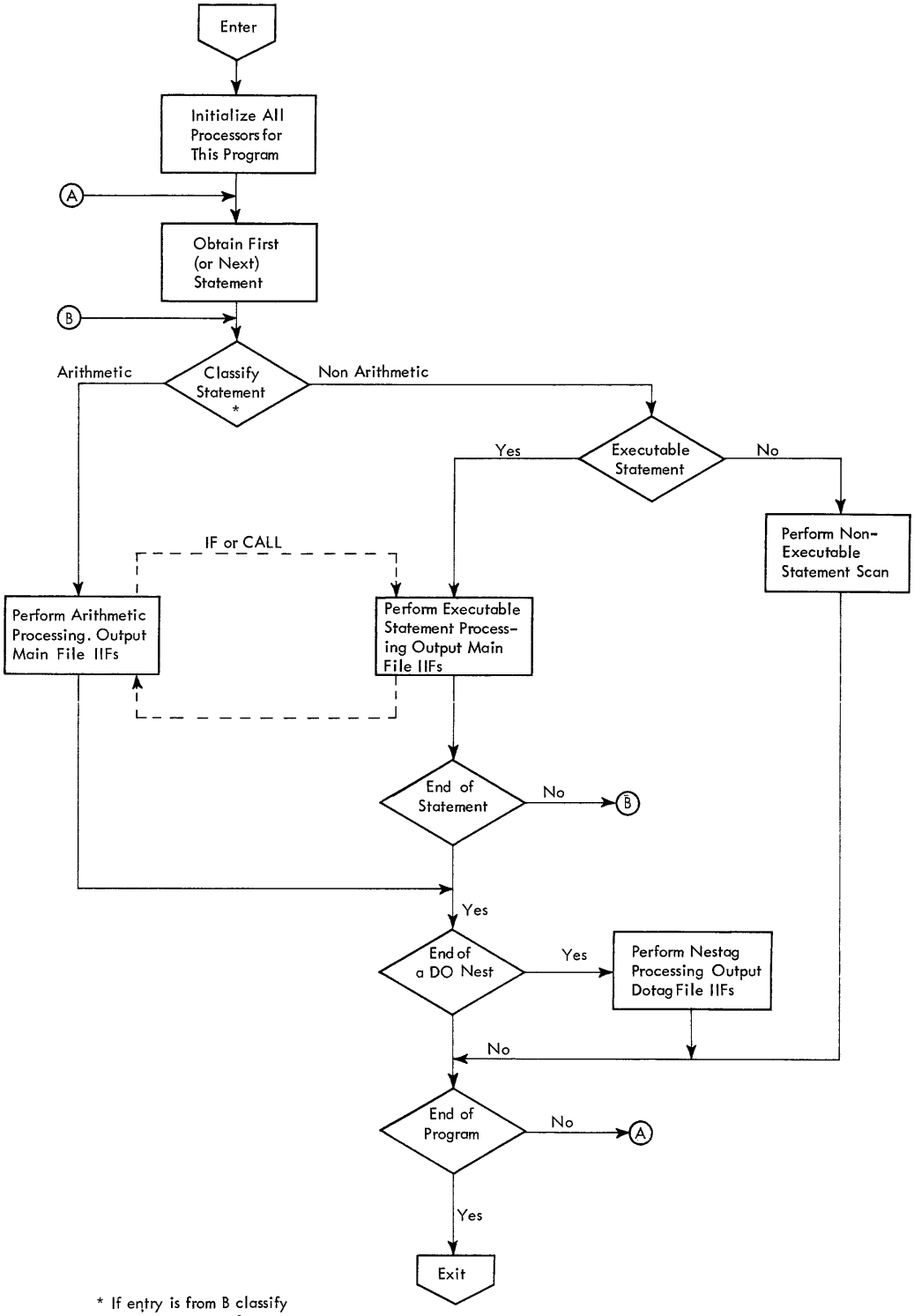


Figure 34. Phase A Flow of Control

be inserted implies that a merge of the main file and Dotag file IIFs must subsequently occur in phase B.

The final output from phase A is the interphase tables file, which consists of tables required for phase B processing.

The main file, the Dotag file, and the interphase tables file are placed on the system utility tapes, SYSUT3, SYSUT1, and SYSUT4, respectively. The main file and the Dotag file are created and written through double buffering during the course of phase A processing. The interphase tables file is written directly from working storage at the end of phase A processing. If a compilation is not being phased, SYSUT4 is not used.

Phase B

Figure 35 shows the flow of phase B processing. Input to phase B is the main file, the Dotag file, and the interphase tables file. Relcon IIF's (indexing instructions referring to statements that occur outside DO nests) are generated at the beginning of phase B. These instructions apply to subscripted variables with subscripts of one or more integer variables.

After Relcon IIF's are generated, the DATA and FORMAT statement processing is performed. Next, storage is allocated, and the storage map is generated and printed. Finally, IIF's from the main file and the Dotag file, and Relcon IIF's from storage tables, are merged during the main assembly pass, resulting in the generation of Loader text. In addition, prologue (initialization) instructions are created where applicable.

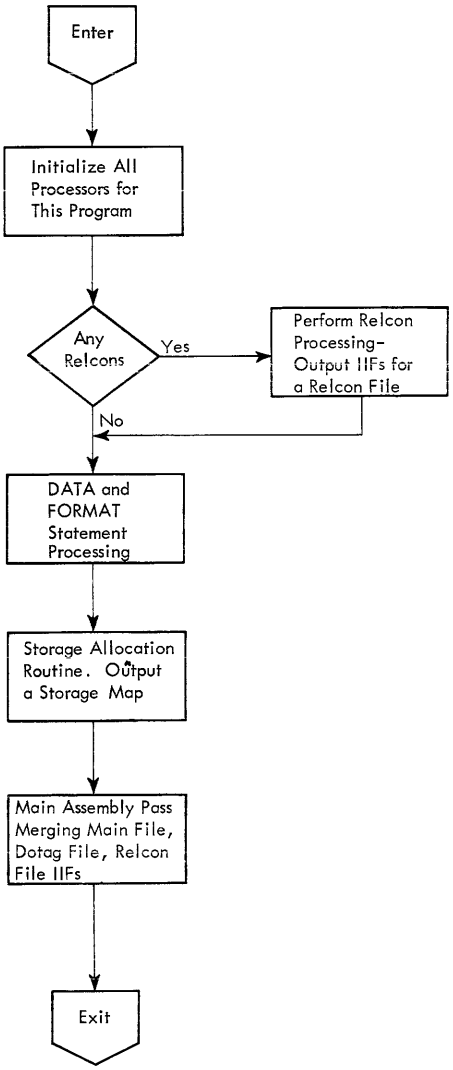


Figure 35. Phase B Flow of Control

Assembly Processing

In order that phase B of the compiler may perform the assembly, all symbols of the program must be assigned relative core storage locations prior to the start of the assembly. For Internal Formula Numbers (IFN's) this is accomplished by treating each IFN as the beginning of a specially tabulated block of coding. (See the section "Internal Formula Number Generation.") For source program symbols, other than statement numbers (External Formula Numbers or EFN's), this is accommodated by the core storage layout of the program. (The treatment of EFN's is explained in the section "Internal Formula Number Generation.")

The core storage layout of the object program is such that program data, consisting of program variables and arrays, are assigned core storage locations ahead of the executable program instructions. Program constants, parameters, and intermediate storage are assigned locations immediately following the program data. Since information as to their total size is accessible before the start of the second pass, this leaves the indeterminately-sized part of the program for the last stages of assembly. This part of the program may be regarded as composed of proper instructions (consisting of instructions from the main file, the Dotag file, and Relcon processing) and prologue (initialization) instructions. Prologue instructions are the most indeterminate with respect to size since they are not known until after the proper instructions have been assembled. Therefore, the prologue instructions are assembled last. They are appended to the end of the assembled program, but are executed first.

Internal Formula Number (IFN) Generation

An IFN is created when it appears possible that the associated instruction may be referred to by another

instruction in the program, or when it appears possible that, at a subsequent stage of processing, additional compiled instructions will be inserted at that point. More generally, an IFN is created when a reference may be made to the associated instruction either at object time or during process time. In the latter case, it is needed for the assembly merge.

IFN's go into a sequentially increasing ordered table that contains a count of the total number of IIF's generated for each IFN of the table. More than one instruction generator may contribute to the IIF count for any one IFN. Each IFN table entry is created at the point in the compilation where the IFN is encountered.

For IFN's associated with external formula numbers (EFN's), a different technique is used since the EFN is often encountered before its proper sequential location in a program. As an example, consider the statement

GO TO 50

where statement number (EFN) 50 occurs at a later point in the program than the GO TO statement. In this case, the reference to the IFN that later will be associated with the EFN of 50 is made indirectly through the EFN table entry created by "hashing" for value 50. ("Hashing" is explained in the section "Table Construction.") When statement 50 is actually reached during statement processing, the EFN entry is given a pointer showing the true IFN table entry.

The executable statement processors and the Nestag routine in phase A, and the Relcon routine in phase B, each contribute to the count recorded in the IFN table entries. Thus, when the phase B assembly routine assumes control, it can define the location of each IFN by using a simple cumulative totaling method. The relative location of each IFN is then defined as the total count appearing before the IFN in the IFN table.

Internal Instruction Formats (IIF's) for Main File

The primary contributors to main file instructions, quantitatively, are usually the arithmetic translator, which processes arithmetic and logical statements; the input/output list processor; and the IF and the GO TO statement processors. Main file IIF's may be incomplete with respect to address fields and index register (or tag) fields. The address field of an IIF generated by either an IF or a GO TO statement processor contains a pointer to an EFN entry. This pointer is subsequently replaced by a pointer to an IFN table entry. The address fields of IIF's generated by the arithmetic translator and the input/output list processor may be pointers to either the NAME or the SUBAK table.

If the field refers to a simple variable name, a pointer to a NAME table entry is used. If the reference is to a subscripted variable with constant subscripts, a pointer is provided to a SUBAK table entry that consists of an array name plus a constant addend. IIF's referring to other subscripted variables may require an address field that either refers to a SUBAK table entry or must be initialized at object time. In addition, an index register may be required for the IIF.

The FORTAG table, which contains information concerning subscripted variables appearing in the source program, is used in supplementary processing (e.g., Nestag and Relcon routines) to provide the assembler with this additional information.

Internal Instruction Formats (IIF's) for Dotag File

The Nestag routine produces Dotag instructions. These are indexing instructions that arise from the configuration of the following factors within a DO nest: the DO statements comprising the nest, parameters of the DO statements, definition points for these DO parameters, subscripted array variables, definition points for integer variables appearing in subscripts, and program transfers. The preceding elements generate sets of instructions consisting mainly of the following major categories:

1. Index register load value computation instructions — occur at DO beginnings.
2. Actual index register loading instructions — occur at DO beginning.
3. Index register increment value computation instructions — occur at DO beginning.
4. Actual index register incrementing instructions — occur at DO endings.
5. End of DO test value computation instructions — occur at DO beginnings.
6. Actual end of DO testing instructions — occur at DO endings.
7. Computation instructions for IIF address initialization — usually occur at DO beginnings but may also occur within the DO.
8. DO transition instructions
   a. Bridge instructions for a normal exit from an inner DO to an outer DO — occur at both inner DO beginning and ending.
   b. Trasto instructions for a transfer exit from an inner DO to an outer DO — occur completely outside the nest.

Internal Instruction Formats (IIF's) from Relcon Analysis Routine

The Relcon instructions are indexing instructions that arise from the appearance of subscripted array variables outside of a DO nest. The sets of IIF's generated from Relcon instructions fall into the following two categories:

1. Index register load value computation instructions. These may be compiled either in-line or as

closed subroutines (called relcontines). In the latter case, a TSX instruction linking to the relcontine is also generated.

2. Actual index register loading instructions.

Both the Nestag routine and the Relcon routine perform index register assignment by relating FORTAG table entries (corresponding to subscript combinations) to specific IIFs which are otherwise complete. Absolute index register assignments are passed to the assembly routine by means of SYMAB table entries. These relate symbolic registers to absolute registers.

Table Handling

Table handling is accomplished by the Table Routine, which uses a pool of chained buffer areas. These areas are called TR (table routine) buffers.

Temporary areas for the various Compiler processors are included in these buffers. When a table or temporary area is no longer needed, the relevant buffer space is freed for subsequent use.

Tables in which duplicate entries must be found are treated by a hashing technique to avoid sequential searches for duplicates. Hashing provides the location of a special, compact pointer table that, in turn, points to the location of the entry in the TR buffer area. The purpose of this indirect referencing technique is to avoid the scattering of table entries — with the consequent loss of core storage space — that occurs with most similar techniques.

The layout of the tables in the TR buffer area shared by all processors is such that one set of tables (the interphase tables) is allocated core storage moving from numerically high to numerically lower locations while the remaining tables occupy storage moving in the opposite direction. See Figure 36. A table-overflow stop occurs during compilation only when these two areas overlap. A diagnostic message is generated if this happens.

Within the lower set of TR tables, many of the tables go into subsets applicable only to a current DO nest. This enables these particular tables to be erased when Nestag processing for the DO nest is complete. The space can then be made available for further use.

The Table Routine is common to both phases A and B. Processors in these phases accomplish their handling of (storing of, searching for, retrieving, etc.) table entries through the use of special macro-instructions, each designed to perform a special table-handling function. The property of these macro-instructions that makes them especially valuable is that the calling processor need not know the location of sequential table entries, which may be in separate buffers with widely differing core storage locations.

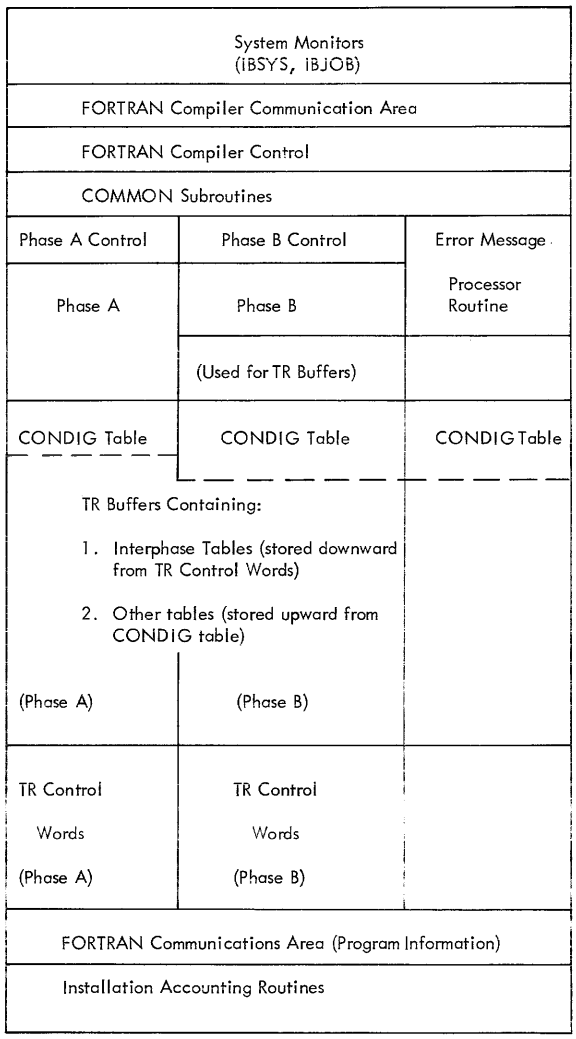


Figure 36. Core Storage Layout for FORTRAN IV Compiler

Diagnostic Handling

The error message processor is the last main component of the FORTRAN IV Compiler. Its function is to print diagnostic information about errors discovered by the various processors in phase A and phase B.

Preliminary Error Handling

The DIAG Routine is read into core storage along with FORTRAN Compiler control, and it remains in storage during compilation and assembly (phases A and B). The DIAG routine constructs the control and diagnostic (CONDIG) table, which contains the following information to be used by the error message processor:

1. The sequential number of the error; i.e., first error, second, etc.
2. The message number that identifies the message.
3. The error level.
4. The BCD message inserts.

Error Message Processor Action

The error message processor is called at the end of phase B processing (that is, at the end of an assembly or assemblies) if any error has been detected during phase A or B processing. This routine causes error output to be printed as follows:

1. The error message processor places the following identification in the subheading of each page:

   COMPILATION yyyy

where yyyy is the program name for which the related error messages are being printed.

2. The error message processor begins the diagnostic messages from phase A with the message:

   DIAGNOSTIC MESSAGES

3. The error message processor heads the diagnostic from phase B with the message:

   PHASE B DIAGNOSTIC MESSAGES

4. For each error occurring during the operations of either phase A or phase B, the routine causes printing of one of the following lines as the first line of a message:

   i SOURCE ERROR j LEVEL k-xxxx
   or
   i LOGIC ERROR j LEVEL k-xxxx

where i is a sequential number that the DIAG routine assigns to this message for this source program, j is the actual error message number, k is a number from one to five representing the severity level of the error and xxxx is the error level explanation.

The significance of the value k and the explanation xxxx is:

LEVEL (k)	EXPLANATION (xxxx)
1	Warning Only
2	Loading Suppressed
3	Assembly Deleted
4	Compilation terminated, error scan continues
5	Compilation abandoned

5. The error message processor then causes the second line of the message to be printed. Insert information comes from the control and diagnostic table. For example, the second line of the message may be:

   THE VARIABLE xx IS NOT DIMENSIONED

where xx is the BCD message insert.

The error message processor uses the control and diagnostic table to find the proper error message and its corresponding BCD message inserts, if any. This table is unique in three respects:

1. It always begins in core storage location OCPHAF.
2. Its length for the diagnostic messages of each compilation is always less than or equal to the length specified in storage location ZCDIN.
3. It remains in core storage for the duration of the compilation of a program set.

The 7090/7094 COBOL Compiler (IBCBC) is the component of the IBJOB Processor that translates a COBOL source program into the MAP language. The COBOL language was developed for business applications by a committee of the Conference on Data Systems Language (CODASYL), as a cooperative effort of computer users in industry, the Department of Defense, and other Federal Government agencies and computer manufacturers.

The 7090/7094 COBOL Compiler (IBCBC) operates under the control of the Processor Monitor, which is under the control of the System Monitor (IBSYS).

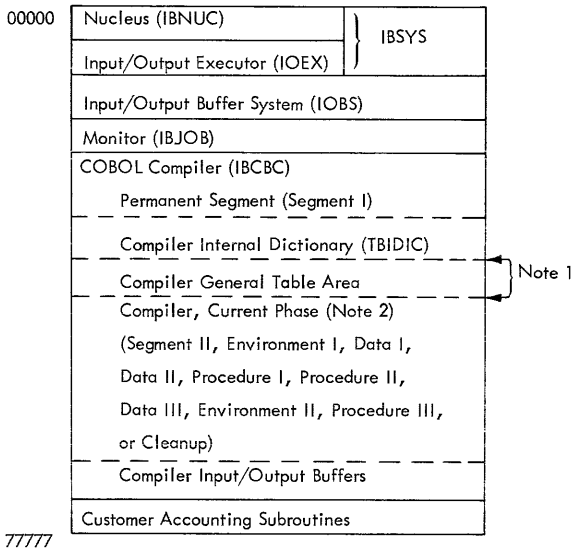
Input to the COBOL Compiler is a COBOL source program which has been put onto the system input unit (SYSIN1). Output from the COBOL Compiler consists of the following:

1. An augmented replica of the source program on the system output unit (SYSOU1)
2. A list of messages describing errors detected during compilation (also on the system output unit)
3. A tape of generated symbolic instructions

At the conclusion of a compilation, control is returned to the Processor Monitor, which calls upon the Assembler to assemble the generated symbolic instructions in a form acceptable to the Loader.

The COBOL Compiler consists of 11 program segments, which are shown in Figure 37. The first of these remains in core storage throughout the compilation process. The other 10 segments are loaded into core storage successively, with each new segment replacing all or most of the preceding segment. Loading of the 11 segments occurs once for each compilation of a source program.

These 11 segments are discussed in the order in which they are brought into core storage. The discussion is quite general, and highly important subroutines are mentioned briefly.



Notes:
1. Boundaries indicated by arrows fluctuate dynamically during compilation, according to need.
2. The various segments listed are placed in this portion of core storage consecutively, no segment being replaced until it has completely finished its assignment.

Figure 37. Core Storage Map for the 7090/7094 COBOL Compiler

Segment I

Segment I of the COBOL Compiler is loaded first and remains in core storage throughout the compilation process. The program portions of this segment are described in the following text.

COBOL Supervisor

This portion of the COBOL Compiler has three primary functions:

1. It prepares all lines of communication with the IBJOB Processor and initializes all COBOL Compiler input/output operations.
2. It controls the processing flow for all major phases of the COBOL Compiler.
3. It returns program control to the IBJOB Processor and indicates whether or not a call is to be made to the Assembler.

General Purpose Subroutines

COLAG (Collection Agency): This subroutine receives generated coding from various portions of the COBOL Compiler and converts it to symbolic form for the Assembler.

CITRUS (Coalesced Indirect Table Reference Unification Scheme): Most of the tables used within the COBOL Compiler are under control of the general table-handling CITRUS subroutine.

ERPR (Error Print): As source program errors are detected by various portions of the COBOL Compiler, requests for generation of the related error messages are directed to this subroutine.

GETBUF and PUTBUF (Internal File Handler): Certain files, notably those for intermediate forms of error messages and of procedure text, are handled so that automatic overflow onto tape occurs when the capacity of an assigned core storage area is exceeded.

DICT1 (Dictionary Lookup Subroutine 1): This subroutine is used during the first scan pass (for each source program division) to determine whether a name has occurred before. It is also used to enter newly defined names in the External Dictionary. (Corresponding to each External Dictionary entry there is an Internal Dictionary entry containing information about the item.)

DICT2 (Dictionary Lookup Subroutine 2): This subroutine is used during the second scan pass (for each source program division) to determine whether a previously undefined name can now be defined. The DICT2 subroutine is closely related to the DICT1 subroutine, and the two share many instructions.

ULSC (Unit Level Scan): This subroutine is used extensively during the first scan pass (for each source program division) to isolate and classify the word units of the source program text.

GLSC (Group Level Scan): This subroutine calls upon the ULSC subroutine and classifies groups of units.

GRIN (Group Interpreter): This interpretive subroutine compares encoded main program questions against classified answers from the GLSC subroutine and chooses main program instructions to be executed based on the test results. The ULSC, GLSC, and GRIN subroutines are used only for first pass scanning.

PUTCP and PUTCPM (Constant Pool Handler): This subroutine saves generated numeric constants (avoiding redundant generation) for actual generation by the Cleanup segment of the COBOL Compiler.

PUTSPM (Symbolic Constant Pool Handler): This subroutine performs the same kind of function as the PUTCPM subroutine, except that the constants are symbolic (represented by an encoded word-logic form called T2 text).

GETBL (Get a Base Locator), GETGN (Get a Generated Name), and GETTS (Get a Temporary Storage Word): These three subroutines perform the function of assisting the various portions of the COBOL Compiler in the generation of instructions by supplying unique words of the desired type and by keeping count of the total number supplied.

File and Table Control Blocks

Each file using IOCS and each table using the CITRUS subroutine require a control block. The control blocks for all of the files and the CITRUS tables used by the COBOL Compiler are defined in Segment I.

Transfer Table

A table of simple transfers to the various general purpose subroutines appears in Segment I. Its purpose is to permit rearrangement of the subroutines without having to reassemble each segment of the COBOL Compiler to correct the subroutine references.

Communication Words

A region of communication words is maintained in Segment I to maintain communication between two segments of the COBOL Compiler not simultaneously in core storage.

Segment II

The second segment of the COBOL Compiler generates initialization instructions on the Assembler tape. It then proceeds through the Identification Division of the source program, recognizing the various entries of this division. The contents of each entry are moved unaltered to the output area. During a scan of the Identification Division, Segment II looks for the A-margin appearance of any other division header. When a valid division header is detected, or when a $CBEND card or an end-of-file is found, Segment II returns control to the COBOL Supervisor. If the source program has no Identification Division, control passes immediately from Segment II to the COBOL Supervisor.

Environment I

The Environment I segment is called by the COBOL Supervisor to perform the scan of the source program's Environment Division.

The primary functions of the Environment I scan are:

1. During the scan of the SPECIAL-NAMES paragraph, dictionary entries are made to relate the COBOL hardware names to the mnemonic-names and switch-status-names in the source program.

2. During the scan of the FILE-CONTROL paragraph, a four-word entry is made into the dictionary for each file named in a SELECT clause. The type of equipment (card or tape) to which the file is assigned is noted by setting particular bits in the dictionary entry. Other information concerning each file is obtained during the scan of the Data Division File Description paragraph.

3. For each file named in a SELECT clause text is created. This text is used by Environment II to create FILE and LABEL cards which are sent to the Assembler. The information pertaining to file-names, the unit or units to which the files are assigned, and the RERUN and APPLY options specified are considered to be text and are placed into an internal file for subsequent processing by Environment II.

4. Upon encountering the source program DATA DIVISION card, control is returned to the COBOL Supervisor.

Data I

The Data I segment of the COBOL Compiler is loaded as soon as the key words DATA DIVISION are encountered in the source program. The principal functions of Data I are:

1. To scan the Data Division of the source program.
2. To build the Data Dictionary.
3. To prepare text (in core storage) for Data II that reflects the postponed problems of OCCURS, REDEFINES, and VALUE. The constants associated with VALUE are also preserved in text.
4. To prepare text for Environment II. This text consists of the File Description information concerning RECORDING MODE, BLOCK CONTAINS, RECORD CONTAINS, LABEL RECORD clauses, and the list of all data record names in each file.

Upon encountering the Procedure Division, Data I returns control to the COBOL Supervisor.

Data II

Using text from Data I and the partially formed internal Dictionary as core-located input, the Data II segment does the following:

1. Assigns object-time base locaters and builds a table to record the assignment. A base locator is a location which normally points to the beginning of a logical record within an input or output buffer. A base locator is also a location assigned to serve as a pointer to the beginning of data after a variable length array.
2. Generates an out-of-line subroutine which calculates the length and byte of each data item. If the length of a data item is not known at compile time, because a suborganization contains an OCCURS . . . DEPENDING ON . . . , clause Data II generates a length calculation subroutine which also performs the function of updating the dependent base locator. Data II builds tables to record which of the subroutines needs to be executed when the value of a given quantity is altered. A quantity item is a data item whose name appears after DEPENDING ON in an OCCURS clause.
3. Generates the core storage reservation for all fixed-location data items. This task is complex in that allowance is made for loading of the proper initial values of data items.
4. Builds a table giving the object-time displacement of each data item.

If a data item's location is dependent upon the contents of a base locator, as is the case with input or output logical records, the displacement of the item is defined to be its distance, in words, from the first data word whose location is determined by the same base locator. For other data items, displacement is the distance from the first data word in the area for working Storage items.

When there are no more dictionary entries to be processed, the Data II segment returns control to the COBOL Supervisor.

Procedure I

Procedure I is called upon by the COBOL Supervisor to perform the first scan of the Procedure Division text.

The major functions of the Procedure I segment are:

1. Each procedure-name at point of definition in the Procedure Division is entered into the dictionary, and a text word is created which refers to that dictionary entry.
2. Text words are created in an encoded form for each of the COBOL words found in the source program.
3. All references to source language names are looked up in the dictionary. If the name being processed is a data-name, a text word is created which refers to the dictionary entry of that data-name. If the reference cannot be found, an error message is given. This case arises when the data-name is insufficiently or incorrectly qualified, or cannot be found. If the name being processed is a procedure-name, the text created for the procedure-name is in BCD form. Before this phase is completed, all source-language procedure-names are defined and entered into the dictionary. During the Procedure II phase, BCD procedure-name references are looked up and encoded text is generated for them.
4. Structural analyses are made of all source language statements to ensure that all source language verb structures conform to the prescribed COBOL rules of composition.
5. When the $CBEND card is encountered, Procedure I returns control to the COBOL Supervisor.

The output of the Procedure I phase is a preliminary form of word-logic text called T1 text.

Procedure II

Procedure II is called upon by the COBOL Supervisor to perform the second pass evaluation of the Procedure Division text. The input to Procedure II is the partially developed T1 text generated by Procedure I. The major functions of Procedure II are:

1. To supply final dictionary references for the name definitions deferred by Procedure I. If the reference cannot be made, an appropriate error message

is printed. This happens when a name is either insufficiently or incorrectly qualified, or is not defined at all.

2. To arrange, augment, or convert certain verb structures, MOVE CORRESPONDING is converted to several individual MOVE sentences. The PERFORM structure is converted to MOVE and COMPUTE statements having embedded instructions which are to be sent to the COLAG subroutine during Procedure III. The AT END clause of the READ verb is changed to an IF structure. STOP verbs are changed to DISPLAY verbs, followed by instructions which perform the object-time stop.

3. To convert arithmetic and logical phrases to a form of Polish notation which is easily processed by the Procedure III code generators.

The output of Procedure II is the completed form of T1 text with all dictionary references in the correct form. The text string is a series of T1 text words representing procedure-names at point of definition and verb clauses.

Environment II

The Environment II segment is called by the COBOL Supervisor to perform the following functions:

1. For each file named in a SELECT entry, the file characteristics are summarized. This summation consists of the following:
   a. A comparison of each block size specification to the calculated lengths of the records in the associated file.
   b. A determination of whether or not all records in the file are the same fixed length. The information is used by the READ and WRITE instruction generators in Procedure III.
   c. A check to determine that at least one OPEN and at least one CLOSE have been issued for each file.
   d. A check to determine that a file has not been specified as being both input and output.

2. Certain initialization instructions are sent to the instruction collection agency (COLAG). These instructions pertain to the opening of the checkpoint file and the determination at object-time of the type of unit (card or tape) assigned to a particular file.

3. An IBMAP FILE card is generated onto the Assembler tape for each file named in a SELECT clause. For each labeled file, a LABEL card, which contains the information from the VALUE OF clauses in the associated File Dictionary description, is generated.

Upon completion of the processing of all files in the source program, Environment II returns control to the COBOL Supervisor.

Data III

The Data III segment is entered by the COBOL Supervisor to perform the following functions:

1. The Data III phase generates object-time subroutines which set pointer words, known as positional indicators, so that they address given array elements.

Each subscripted data item in the source program is located at object time through a positional indicator (PI). A different position indicator is used for each unique subscripted reference in the Procedure Division of the source program. For example, the coding

```
MOVE    A (I, J, K)    TO X.
MOVE    B (I, J, K)    TO X.
MOVE    A (I, J, M)    TO X.
MOVE    A (I, J)       TO X.
MOVE    A (I, J, K)    TO X.
```

causes four position indicators to be generated. The last subscript reference does not create a new position indicator because it is identical to the first reference.

For each position indicator generated in the object program, there is also a subroutine generated to set the contents of that position indicator. It is the function of Data III to extract from the Internal Dictionary the information concerning the base of the array and the distance between the elements within the array. Data III also generates the object-time subroutines that set the position indicators. Calls upon these subroutines are generated by the Procedure III Supervisor when they are needed.

2. The Data III phase forms a table of all data items that contain a quantity item (including with each of the data items the subroutines called upon when the quantity changes). This table is used by the quantity item analyzer in Procedure III.

3. The Data III phase places the base locator number in the Internal Dictionary in the place of the level-number for those data items which are located by a base locator.

4. A list is compiled of all variable-length records and also of all fixed-length records that contain a data-item described by an OCCURS . . . DEPENDING ON clause. This list is used by the READ instruction generator in Procedure III to call upon the appropriate length calculation subroutines to adjust the lengths and base locations of data items affected by a READ statement.

Upon completion of these functions, Data III returns control to the COBOL Supervisor.

Procedure III

At the time the procedure text becomes input to Procedure III, sentences have been reduced to statement segments. Each such statement consists of a verb, or its equivalent, and its related operands. The supervisory program for Procedure III gives these statements, one at a time, to specific instruction generator programs, the program chosen depending upon the verb under consideration. These instruction-generating pro-

grams produce the appropriate object-time instructions in an encoded form called T2 Text.

Subscript Calculations

Prior to routing each statement, the supervisory program examines the statement operands to find all occurrences of T1 Text words that indicate subscripted data operands. For the first occurrence of each word within a statement, the supervisory program generates a call to the appropriate position indicator (array pointer) calculation subroutine. These subroutines were generated during Data III, and the supervisory program is able to determine which one to use by consulting the Position Indicator Table.

Treatment of Incoming Procedure-Names at Point of Definition

The supervisory program gives incoming procedure-names directly to the instruction collection agency (COLAG) rather than allowing them to reach the instruction generators.

If the procedure-name is a paragraph name or a section name, i.e., not a generated name, and if the scope of a PERFORM verb terminates with the preceding paragraph, the supervisory program generates an instruction immediately preceding the procedure-name. This generated instruction has the following form:

```
TRA *+1
```

It is modified to provide the PERFORM return linkage.

Computation of Variable Lengths

A data item that appears as a data-name after the DEPENDING ON portion of an OCCURS clause is known as a quantity item. If any of the instructions generated for a statement alter the object-time contents of any quantity item, each generator potentially involved (MOVE and READ, at present) adds these data items to a list which it builds throughout its functioning. When such a generator's functions are concluded, but before it returns control to the COBOL Supervisor, it gives its list to the quantity item analyzer. This, in turn, issues the proper calls to the base locator and length calculation subroutines generated in Data II.

Instruction Generators

The source language statements OPEN, CLOSE, READ, WRITE, MOVE, DISPLAY, ALTER, GO TO . . . DEPENDING ON and IF and IF NOT are converted to machine language by means of a set of subroutines known as instruction generators. Certain other verbs are also handled by the same generators, having been changed to one of the above verbs before Procedure III is entered.

After the procedure text has been completely processed, program control is returned to the COBOL Supervisor.

Cleanup

The Cleanup phase is called by the COBOL Supervisor as the last major segment of the compilation process.

The primary functions of the Cleanup phase are:

1. The symbolic constants that were placed into the Symbolic Constant Pool are sent to the instruction collection agency (COLAG) in regular instruction format.
2. All error message references that were sent to the ERPR subroutine are arranged in ascending order, according to the associated source language card numbers. Each error message reference is expanded to a full error message form that is sent to the Input/Output Editor to be placed on the system output unit (SYSOU1).
3. BSS instructions are sent to the instruction collection agency (COLAG) to reserve storage for base locators, position indicators, and temporary storage and result storage locations.
4. The constants placed into the Numeric Constant Pool are sent to the instruction collection agency (COLAG) and placed on the Assembler tape in the form of octal constants.

Upon completion of these cleanup functions, control returns to the COBOL Supervisor.

The 7090/7094 Macro Assembly Program, IBMAP, operates under the 7090/7094 IBJOB Processor Monitor. Input for the Assembler comes either from the COBOL Compiler or from programmer-coded MAP language instructions. IBMAP provides loader input that is indistinguishable from that supplied by the FORTRAN IV Compiler. Thus, to the Loader, a source language program written in the COBOL, FORTRAN IV, or MAP language appears in the same format. As shown in Figure 38, assembler output is in the form of relocatable binary text.

Assembler Design

Assembler activity is divided between two main phases. Phase 1 consists of initialization and a pass (pass 1) over the source program to form program dictionaries. Phase 2 consists of interlude procedures and the second pass (pass 2) by the assembly program.

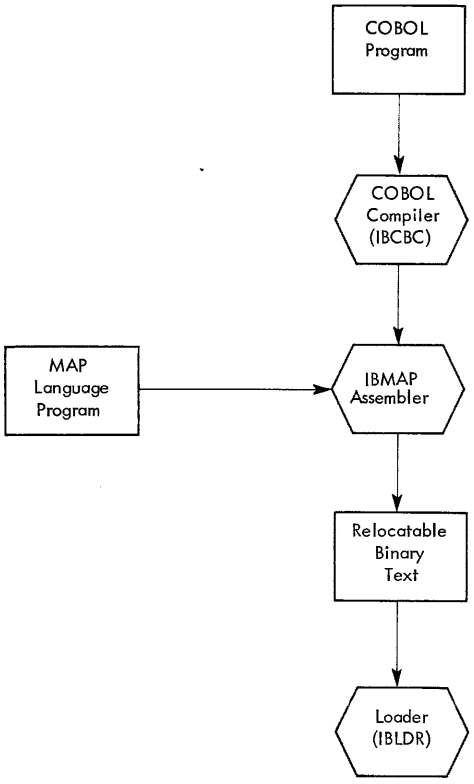


Figure 38. Assembler Input and Output

The interlude, which is the time between pass 1 and pass 2, is used to determine the values assigned to the various symbols. Pass 2 is made over an internal form of binary information, rather than over the BCD information of the original source program.

The core storage layout diagram in Figure 39 shows the relative locations of routines, tables, and dictionaries as they occur during phases 1 and 2.

Phase 1

The main functions of phase 1 are to initialize the Macro Assembly Program and to create tables necessary for definition by interlude procedures. A general flow chart for phase 1 is given in Figure 40.

Initialization

The initialization routine, which receives control directly from the IBJOB Processor Monitor, performs the following four functions:

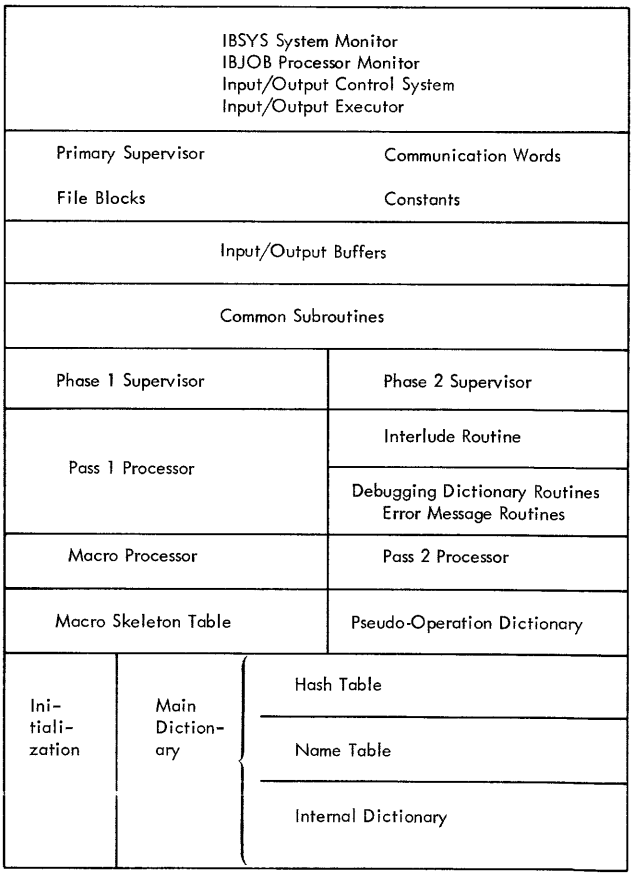


Figure 39. Core Storage Layout

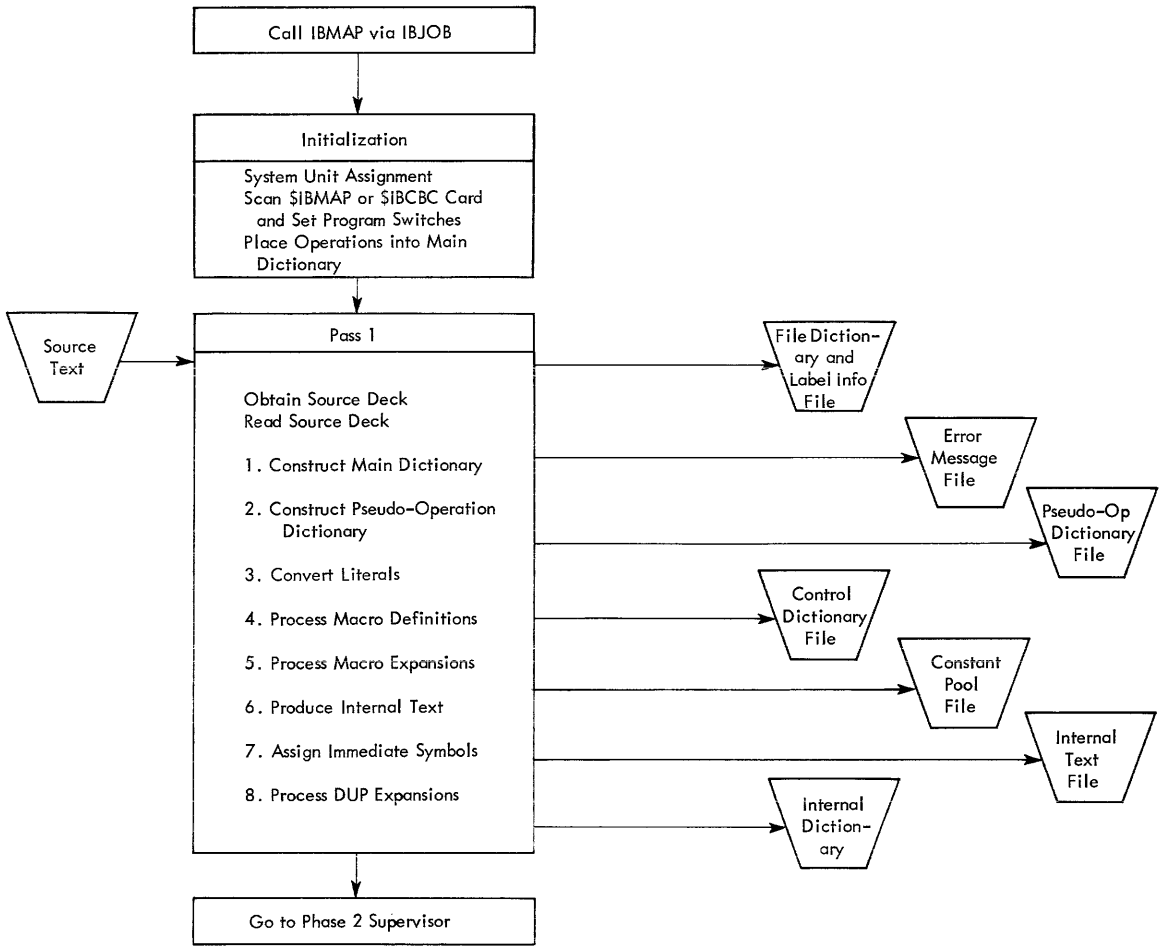


Figure 40. Phase 1 General Flow Chart

1. Examines the system unit assignment table in the IBSYS System Monitor to locate the physical units to be used by the IBJOB Processor in performing input/output functions for the Assembler.

2. Scans the $IBMAP or $IBCBC control card and sets program switches (governing conditions pertaining to the entire assembly) according to the information in the variable field of that card.

3. Places the operation codes of the instructions into the main dictionary.

4. Places the system symbols requested on the $IBMAP card into the main dictionary.

Upon completion of these functions, control is passed to the primary supervisor, which, in turn, passes control to the phase 1 supervisor. The initialization routine is loaded with the pass 1 processor but is subsequently overlaid by the main dictionary.

Pass 1

Input to the pass 1 processor consists of the source text card images. During this pass, the entire source program deck is read. As each card is read, the following processing occurs:

1. The main dictionary is constructed. Items entered into this dictionary are:
   a. Symbols in the name field
   b. Qualified symbols
   c. Location-counter symbols
   d. New operations that may be defined by operation-defining pseudo-instructions.

The main dictionary is the main information table of the assembler and consists of the following parts:

a. The name table, containing the external BCD representation of each program symbol to which the internal text refers. There is one entry for each symbol, and the table is formed nonsequentially by a scattering principle. This procedure minimizes the time for pass 1 text production, which requires the immediate replacement of each BCD symbol by an internal identifier.

b. The internal dictionary, containing one entry in binary form for each program symbol. The

dictionary is formed linearly (that is, entries are in the same order as in the source program).

c. The reference (or hash) table, providing the necessary relationship between the scattered information of the name table and the linear information of the internal dictionary. This table is half the length of the name table.

During pass 1, two types of entries are made into the main dictionary. The first is called a real entry. This is an entry for which information is placed in both the name table and the internal dictionary. Reference table correspondence is also generated at this time.

The second entry type is called a nominal entry. This is one for which only the symbol is entered into the name table. There is no internal dictionary entry; thus there is no reference table entry. Nominal dictionary entries are made when encountering variable field symbols for which no entries have yet been made in the dictionary. A subsequent real entry for a given symbol replaces the nominal entry.

2. The pseudo-operation dictionary is constructed but is not placed into its final location since that area is occupied during pass 1 by the macro-skeleton table, which is a table of macro-expansions expressed in a compressed form. Items in this dictionary consist essentially of information in the variable fields of any pseudo-operations that may affect a location counter.

3. Literals are converted to binary form and stored in a literal pool, which is a table of all unique literals appearing in the source program.

4. Macro-definitions are coded and placed in the macro-skeleton table.

5. Card images required by the expansion of a macro-instruction are produced from the macro-skeleton table.

6. Card images required by DUP pseudo-instructions are produced.

7. The truth value of IFF or IFT pseudo-instructions is evaluated to determine whether or not to scan the following card.

8. Internal text corresponding to each source program card is produced. Original card images are retained only for listing purposes. If listing is not required (by the $IBMAP card option, NOLIST), the card images are deleted.

9. Immediate symbols are assigned S-values at the point at which they are used. An immediate symbol is a symbol used in a SET, IFT, IFF, or DUP pseudo-operation, each of which must be evaluated during phase 1. The S-value of an immediate symbol is determined as follows: if the symbol is defined by one or more SET pseudo-operations, its S-value is the value of the variable field of the last such SET encountered. If the symbol is not defined by a SET, its S-value is either 1 (if it has previously appeared in any name field in the program) or 0 (if it has not appeared in a name field).

Output from pass 1 (and, therefore, phase 1) consists of the internal text file (which is on tape), the internal dictionary (which is kept in core storage), and the following internal files:

File dictionary and label information
Pseudo-operation dictionary
Control dictionary
Error message
Constant pool

When phase 1 is completed, the interlude and pass 2 routines of phase 2 are brought into core storage and assembler control passes to the phase 2 supervisor.

Phase 2

The major work of the assembler occurs during phase 2. The flow of processing for this phase is shown in Figure 41.

Interlude

Input to the interlude routine consists of the FILE dictionary and label information, the control dictionary, and the pseudo-operation dictionary. During the interlude, processing occurs in the following order:

1. The FILE pseudo-instruction file is examined. From this, information is used in construction of the file dictionary. Any necessary $FILE and $LABEL card images are written at this time, and the file dictionary is printed.

2. The pseudo-operation dictionary is brought into core storage. It overlays the area that was occupied by the macro-skeleton table during phase 1. The definitions of all location symbols are then determined. This consists of assigning either absolute or relative locations to all pseudo-instructions.

3. The control dictionary is examined and defined. This dictionary is complete except for the inclusion of virtual symbols. The control dictionary enables the Loader, IBLDR, to make cross-references among programs. Each control dictionary entry consists of a binary coded decimal name for external identification, the entry length (that is, the number of words), and its position in the source deck relative to the beginning of the program.

There are four types of control dictionary entries. The first type of entry, called a control-section entry, is a reference to a combination of instructions or data that is to occupy a contiguous block of core storage. This type entry is produced by the CONTRL and COMMON pseudo-operations.

The second entry type is called an external reference entry. It is produced either by a CALL operation or from a virtual symbol. In the latter instance, the symbol is its own external name.

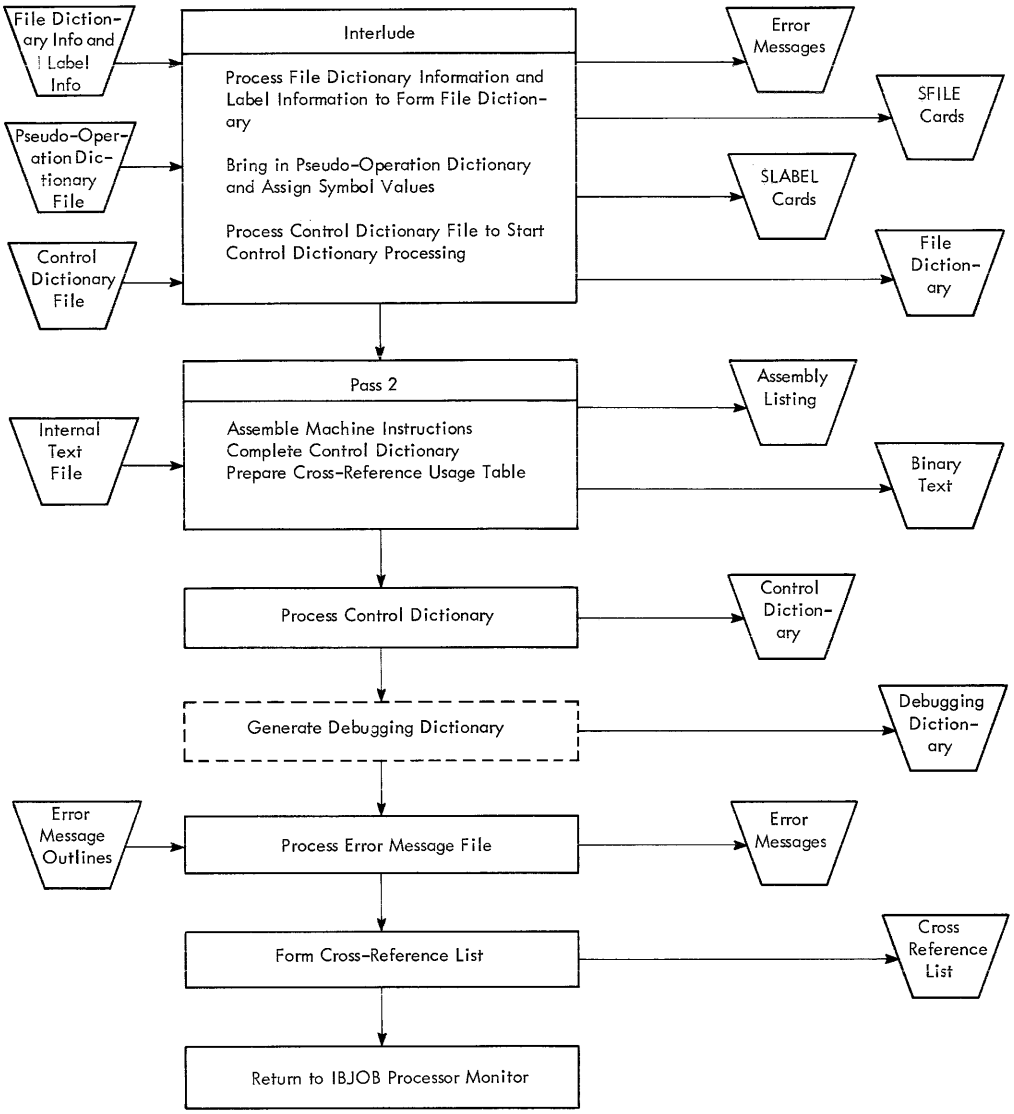


Figure 41. Phase 2 General Flow Chart

Entry-point entries constitute the third type of control dictionary entry. They are produced by ENTRY pseudo-operations and permit a point within a program segment to be referred to from outside the segment.

The fourth type of entry is the EVEN entry. Its function is to force the current location counter to an even value to ensure an even address for the next instruction. Even entries are produced by EVEN pseudo-operations.

If there is any output from the interlude routine, it is in the form of error messages, $FILE cards, and $LABEL cards.

Pass 2

Pass 2 performs the final assembly pass over the internal text. Phase 2 actually begins at this point.

Input to pass 2 consists of the internal text file and the error message file (if one exists). During pass 2, the following functions are performed:

1. The machine instruction corresponding to each text item is assembled.
2. The assembly listing is prepared for the input/output editor.
3. The binary deck image is prepared for the input/output editor.
4. A determination is made as to which symbols are virtual, and the control dictionary is completed (see "Interlude").
5. The cross-reference usage table is prepared for the input/output editor.

Upon completion of this pass, the control dictionary is processed to produce binary cards and list informa-

tion, if required by the options in the $IBMAP control card.

In pass 2, any debugging information encountered during processing the internal text file is modified slightly. After processing the control dictionary, the debugging dictionary is formed, if specified by an option on the $IBMAP or $IBCBC card.

Next, the error message file is examined. Outlines of any messages issued during assembly are interpreted, and the appropriate text is sent to the input/output editor for listing. Lastly, if requested by the REF option on the $IBMAP card, the cross-reference usage of symbols in the object program is prepared for listing.

Phase 2 output consists of the assembly listing and the binary deck if specified on the $IBMAP card.

Upon completion of phase 2, program control is returned to the IBJOB Processor Monitor.

The Load-Time Debugging Processor consists of routines to compile the debugging request package submitted by the programmer, execution time routines to recognize and interpret debugging situations, and postprocessor editor and translator routines to list debugging data in the form requested. The actions of these routines are supplemented by actions of the Assembler and the FORTRAN Compiler in providing information from the source decks for the Loader, and by actions of the Loader in loading the debugging instructions for execution with the program.

Load-Time Debugging Operations

Figure 42 shows the flow of load-time debugging operations. The source program is read in — normally from the system input unit. It must contain a debug request deck preceding the actual program instructions. Segments of the program may be in the FORTRAN or MAP language, in the form of relocatable binary text, or in any combination of these. The COBOL programmer can use load-time debugging by specifying debugging locations from an Assembly listing of the COBOL deck.

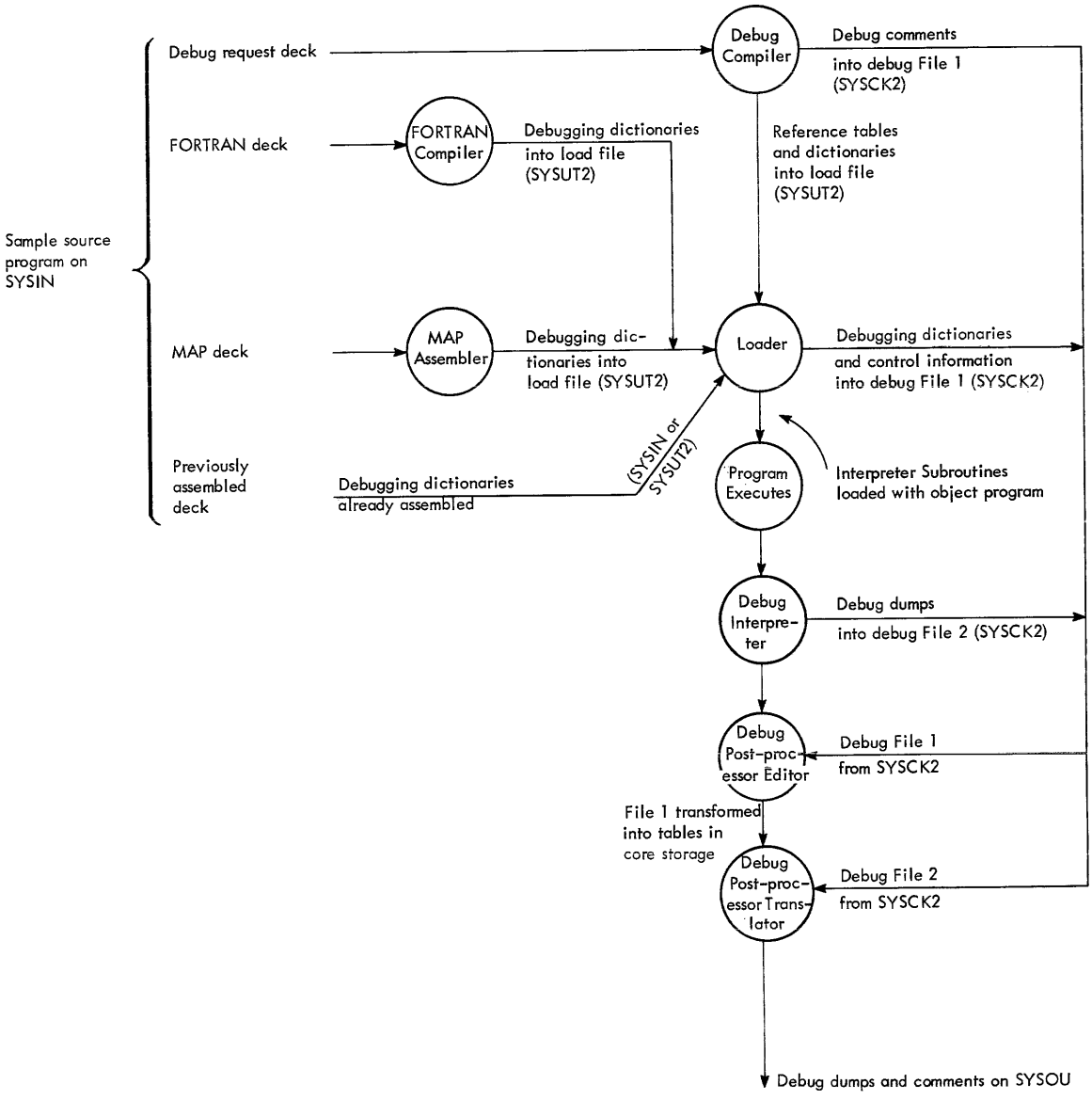


Figure 42. Load-Time Debugging Program Flow

This type of debugging should not be confused, however, with compile-time debugging for COBOL decks.

Load-time debugging is accomplished in the following steps:

1. The debugging compiler routines store the comments that are to be listed with the dumps and use the debugging request packet to prepare reference tables.
2. The Assembler and FORTRAN IV Compiler prepare debugging dictionaries from the actual program.
3. The Loader sets up the debugging mechanism for execution with the actual program.
4. The execution time routines perform the requested debugging actions during execution of the object program.
5. The debugging postprocessor editor and translator routines translate the dumps, establish their format, and list them.

As shown in Figure 42, a deck assembled before this job already contains a debugging dictionary. For this reason, no work by the Assembler or by the FORTRAN IV Compiler is needed, and the deck passes immediately to the Loader.

Debugging Compiler Routines

The $IBDBL card that immediately follows the $IBJOB card in any load-time debugging job causes the IBJOB Monitor to call the debugging compiler into storage. The compiler reads in the entire debugging request deck and produces two different sets of data:

1. A series of reference tables are written out as the first part of the load file. These tables are headed by a BCD card with the code $RDICT starting in column 1. The information in the tables provides the execution time routines with the points in the program where debugging has been requested, symbols used in debugging statements, and instructions for dumping that were coded from the debugging requests themselves.

Since one of the execution time routines, the interpreter, is constructed on a modular basis, only those segments needed by the debugging requests are called into storage. For this reason, the reference tables also include a dictionary of interpreter control sections.

2. The comments that the programmer wants listed in connection with the debugging dumps are written into the first file on SYSCK2.

Upon reading a *DEND card, the debugging compiler returns control to the IBJOB Monitor.

The next step is to prepare a dictionary of the symbols defined in the source program that can be compared with the symbols that the debugging compiler has already written on the load file from the request deck. This dictionary is prepared by the Assembler or the FORTRAN IV Compiler, depending on the program deck involved.

Debugging Actions by the Assembler

Modal and dimensional information has been supplied by the MAP programmer in his source program. The Assembler sets aside this data until its normal operations have been concluded. It then creates a debugging dictionary for each deck it has read. If a full debugging dictionary has been specified on the appropriate control card, the dictionary refers to all symbols contained in the assembled program. If the short dictionary has been requested, only those symbols specified by the KEEP pseudo-operations in a MAP source deck are included.

The debugging dictionaries are written out on the load file as the last part of the relocatable binary deck produced by the Assembler. They specify for future debugging action the name of each symbol, its mode and dimensions, and its relative or absolute location.

Debugging Actions by the FORTRAN Compiler

The FORTRAN IV Compiler performs the same debugging actions on a FORTRAN deck as the Assembler performs on a MAP deck. The input is a program in problem-oriented language, and the output, as far as debugging is concerned, is a debugging dictionary in binary form.

Debugging Actions by the Loader

The Loader receives the series of reference tables and dictionaries in the load file as its debugging input. It sets up the debugging mechanism as follows:

1. It loads a debugging program block into core storage directly after the job program. This block consists of debugging reference tables and the necessary execution time routines.
2. It enters the prefix of each program instruction, before which debugging action has been requested, into an STR insertion table, referred to by location. The Loader replaces each such prefix in the program with the prefix for an STR instruction. The execution time debugging routines include an STR supervisor to monitor each occurrence of an STR in the program, and routines to interpret the debugging requests associated with the STR.
3. It writes the debugging dictionaries created by the Assembler and the FORTRAN IV Compiler onto the first file on SYSCK2, along with other reference material needed by the postprocessor routines.

Execution Time Routines

The STR supervisor monitors the action of all STR instructions in the job program. Each debugging STR transfers control to the interpreter routines, which in turn perform the debugging actions requested in the instructions coded by the debugging compiler. The in-

formation requested in each dump is written onto the second file on SYSCK2. The STR supervisor then performs the instruction which was replaced by the STR and returns control to the program for further execution.

Postprocessing: The Editor and Translator Routines

When the job program has been executed, the IBJOB Monitor transfers control to the debugging postprocessor editor. The editor reads the first file from SYSCK2 into core storage. This file now contains the dump comments saved by the debugging compiler, and the reference tables and dictionaries produced by the Loader. The editor rearranges this data into a form more readily usable by the postprocessor translator, reads in the translator (overlaying most of itself), and transfers control to it.

The translator reads in the dumped information previously written into the second file on SYSCK2 by the interpreter routines. The format and limits of each dump are established according to the request for it, and any associated comments are added. Each listing is then written on SYSOU for printing. At the end of its work the translator returns control to the editor, which in turn relinquishes control to the IBJOB Monitor.

The Loader is a component program of the IBJOB Processor system. It accepts all the assembled decks for a job application produced by the Assembler and/or the FORTRAN IV Compiler, transforms them into one executable object program, and then transfers control to this program. Figure 43 shows the relationship of the Loader to adjoining components in the system.

Input to the Loader consists of a load file and those subroutines that the Loader selects from the IBJOB Library for inclusion in the object program. The load file is a repository of data for the Loader from the Assembler, the FORTRAN IV Compiler, or the debugging compiler. It is normally kept on symbolic input/output unit SYSUT2. But if the entire program consists of binary decks from previous assemblies or compilations, the Loader can be instructed to use input file SYSIN1 as the load file through the NOSOURCE option punched on the $IBJOB card. The load file contains program decks assembled by the Assembler and the FORTRAN IV Compiler on the current machine run, or decks inserted by the programmer from previous assemblies. If the programmer has made special requests in regard to files or control sections, the load file will also contain the appropriate control cards. And if he has requested load-time debugging, there will be a deck of reference tables from the compiler section of the Load-Time Debugging Processor.

While creating a single executable program from its input, the Loader deals with several problems.

Absolute Address Assignment

The Loader determines the absolute address that each instruction or data word in a program must occupy at execution time. Where instructions refer to addresses, the Loader assigns an absolute value to each reference.

First, the Loader solves another problem. When parts of the same program are coded separately and in different source languages, they are assembled as

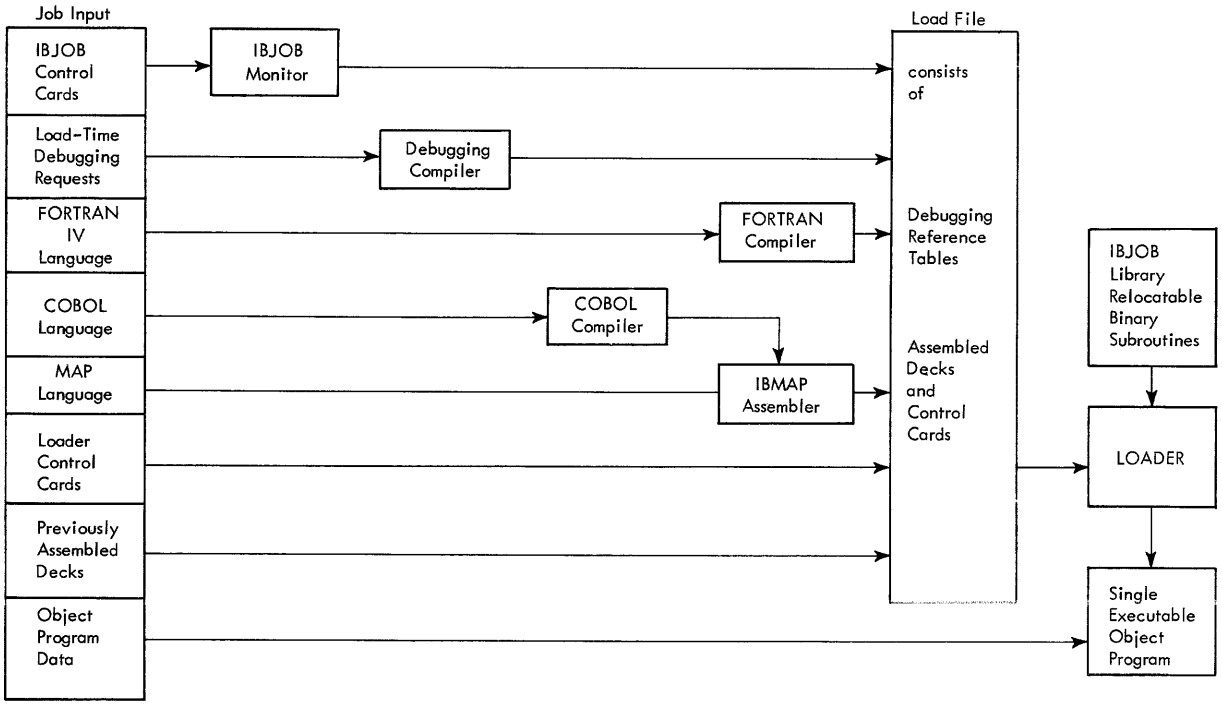


Figure 43. The Loader in Relation to Adjoining Components in the IBJOB Processor System

separate decks. Even though each deck received by the Loader has been assembled into the same binary format, the address locations are relative only to other locations in the same deck. Some address references may have no value at all, since they refer to symbols within another deck. The Loader precedes absolute address assignment by resolving the relationships among the decks themselves.

Program Loading

Even when all addresses and address references have been made absolute, the program must be loaded into its proper storage position for execution. At this stage in Loader operations, the parts of the program may be scattered over several areas of storage. The Loader uses a technique called "scatter-loading" to bring the object program into its proper position.

Library Subroutines

The Loader can add standard subroutines from the Processor Library (IBLIB) to the object program. Some of the subroutines, such as .LXCON, which controls post-execution operations, are added automatically to every program. The Loader selects others, such as the .LOVRY subroutine for loading overlay links, when the object program requires them. However, most Library subroutines are loaded because the input decks or other Library subroutines require them.

Input/Output Environment

The Loader provides the input/output environment necessary for execution of a program. It assigns input/output units to files, allocates buffer storage to files, chooses the appropriate input/output control routines for the program's needs, and generates the instructions necessary for initializing these routines.

Overlay

When the object program and its work area requirements are too large for core storage, the programmer can request the Loader to set up an overlay pattern by using $ORIGIN and $INCLUDE control cards. Overlay mode alters Loader operations considerably. Absolute addresses within different links, for example, may be the same, and a different series of debugging routines must be called in for load-time debugging. The Loader also performs an extensive checking operation to determine the validity of calls that would cause links to be loaded.

Load-Time Debugging

When the programmer has requested load-time debugging, the load file contains information from the debugging compiler based on his requests. The assembled decks generated by the Assembler and FORTRAN IV Compiler also contain dictionaries that will help the Loader define the symbols used in these requests. Using this material, the Loader constructs a mechanism that dumps the information during program execution, as requested by the programmer.

Communications from the Loader

The Loader reports program or machine malfunctions through error messages. (See part 3 of this manual for a list of Loader messages and their explanation.) It reports input/output configurations with machine operator messages. On request it also lists a load map of core storage allocation at execution time, a cross-reference table of control sections and file names, and a buffer pool assignment list.

Configuration of the Loader

The Loader is divided into several parts. The first part is the load suprevisor, which is called into storage by the Processor Monitor (see Figure 44). It remains there throughout the Loader operations to call in the other parts of the Loader. These parts are an initialization section and the main program sections 1 through 5. (Another part, section 6, does not operate during load-time and is described in "The Librarian section.")

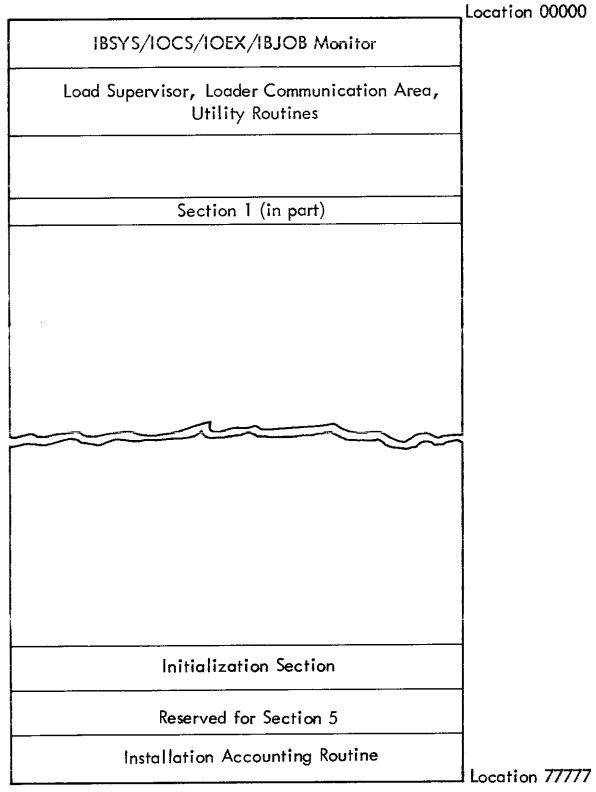


Figure 44. Storage Allocation for Load Supervisor and Initialization Section

Throughout its operations the Loader maintains two utility areas within the load supervisor section. The first is a communication area having the addresses of the tables, dictionaries, and lists with which the Loader works. The second area contains subroutines that are used in common by all Loader sections.

Loader Operations

Loader working storage requirements are such that part of a program having more than 6,000 instructions may have to be siphoned off into external input/output devices for temporary storage. The process may be called overflow or spill, depending on the conditions requiring it. The Loader performs this operation automatically by keeping a running estimate of its storage needs. A detailed description of overflow and spill, not essential to an overall understanding of Loader operations, is given later in this section. (See "External Storage For Text.")

Since the Loader scans the program instructions part of the load file twice, it is called a two-pass loader. The first pass yields information used by the initialization section and sections 1 through 3; second pass information is used by sections 4 and 5.

The following is a description of how these sections operate.

Initialization

The load supervisor is called into storage by the Processor Monitor. It in turn calls the initialization section into upper core storage and part of section 1 into lower core storage. This part of section 1 consists of routines that are not used by section 2; it is to be overlaid when section 2 is brought into storage. The remaining section 1 routines are used by both section 1 and section 2. They are read into upper core storage over the initialization section after it has completed its work.

The initialization section sets up an input/output environment for Loader use. It assigns input/output units, defines buffer pools, attaches files to the pools, and opens the files. One of the input files thus opened contains the input to the Loader. If no compilation or assembling of source language cards has been performed during this machine run, the input is read from SYSIN1. If there has been compilation or assembling, the input is normally in a load file on SYSUT2. If there has been compilation or assembling for only part of the program and the programmer has used a $IBREL card, the Loader reads its input first from SYSUT2 and then from SYSIN1.

The first part of the load file is a deck of reference tables from the debugging compiler, if debugging has been requested. This deck is headed by a $RDICT control card. It provides information on the points in the program where debugging has been requested and on the symbols referred to in the debugging statements. It also contains instructions for dumping that were compiled from the debugging statements themselves.

The second part of the load file is a collection of program decks in binary format mixed with control cards generated by the Assembler, the FORTRAN IV Compiler, or both. These decks either have been assembled on the current machine run or added by the programmer from a previous run. Each deck contains a file dictionary (if input/output files have been requested), a set of relocatable binary instructions called "text," which are the binary translation of the source program instructions, a control dictionary section, a debugging dictionary (if debugging has been requested), and overlay control cards (if overlay has been requested).

The load file also contains Loader control cards transmitted directly from the source deck by the Processor Monitor. These cards may appear in front of, between, or following the binary decks.

In the section "Loader Input" there is a more detailed description of a load file.

The initialization section now reads the first card image in the load file. If this card is not a $IBJOB card, loading terminates.

The initialization section next transfers control to the installation accounting routine (if any) to record that the Loader is now in control.

After receiving control again, the initialization section scans the $IBJOB card and encodes its parameters in the Loader communication area.

If a $RDICT card appears directly after the $IBJOB card, the initialization section reads the debugging reference tables into core storage. The debugging tables are read in at this time, because section 1 uses them while reading the rest of the load file into storage.

The initialization section then returns control to the load supervisor. If the next card on the load file is a $EDIT card, the load supervisor calls section 6 into core storage to edit the Subroutine Library (see "The Librarian"). Otherwise, the load supervisor calls in those parts of section 1 that are not already in core storage, overlaying the initialization section.

Section 1

Section 1 defines and attaches an IOCS internal file buffer area for storing relocatable binary text. This file has all the properties of any other IOCS file. All or part of its contents may also be transferred to any appropriate device, such as disk or tape.

Section 1 now begins to read the rest of the data from the load or input files into four storage areas. Load-time debugging information is stored in a buffer attached to SYSCK2. The remaining areas are in core

storage and contain the internal text file for text, the control dictionaries, and the control information storage block, where additional control information is stored.

This control information is stored in the form of chains of entries. In each entry is a word containing the addresses of the preceding and succeeding entries. Each chain is located by the Loader through a communications word that contains the addresses of the first and last entries.

Chaining permits section 1 to store data in the control information storage block without regard to the length of the chains and in such a way that the information can be found easily.

Figure 45 shows section 1 and the working areas in storage.

While transferring the load file into storage, section 1 processes the data as follows:

The options on the $IBLDR card, the first card in each binary deck, are examined. If LIBE appears, the deck

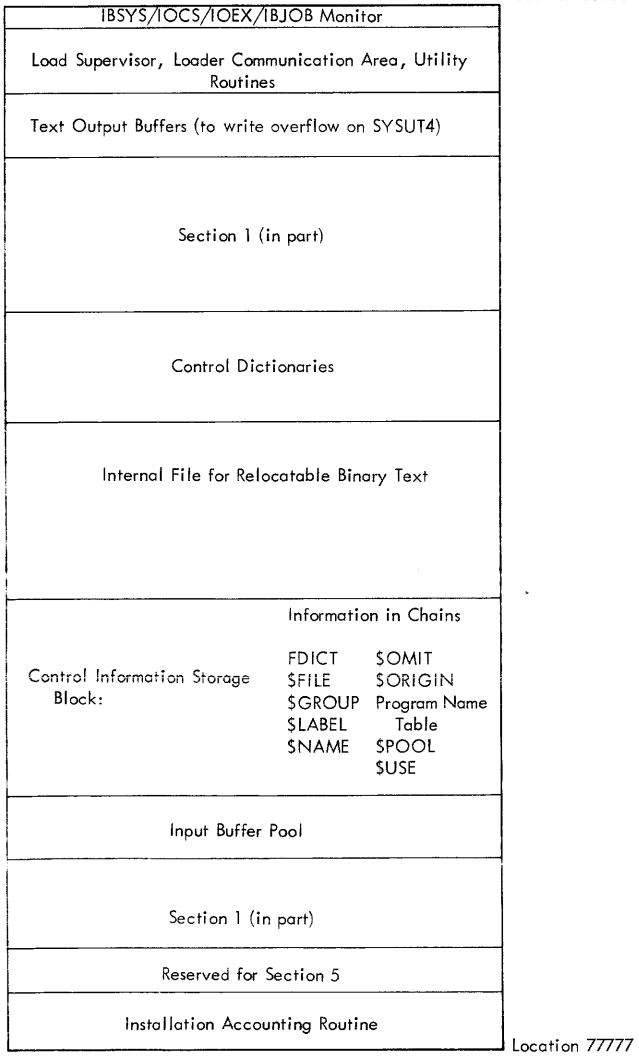


Figure 45. Storage Allocation During Section 1 Operations

name punched in columns 8-13 is placed in a virtual section name list which is stored temporarily in the area reserved for section 5. This list is for section 2 use in determining the subroutines to be included in the program from the Subroutine Library. Otherwise, the deck name is stored in the program name table chain in the control information storage block. This chain is to contain the name of every deck, including those in subroutines, that will appear in the program to be executed. If NOTEXT is punched, the Loader bypasses all relocatable binary text cards.

If debugging has been requested, a search is made in the debugging reference tables for each deck name as it is encountered. If a match is found, corresponding chains within the debug tables are realigned accordingly. If there are no decks encountered for any names cited in the debugging reference tables, error messages are printed out.

$FDICT, $TEXT, $CDICT, and $DDICT cards delimit sections of the binary deck. $DKEND cards prepare the Loader for either a new deck or an end of file.

Each file dictionary encountered generates an entry in the external file dictionary chain in the control information storage block. This chain is used by section 2 to generate file blocks.

Each relocatable binary text card image is transferred directly into the internal file area in order of its occurrence.

Each two-word control dictionary entry is stored in the control dictionary area, and a third word is added for chaining control sections with the same name. These chains are called "like name" section chains. "Like name" section chains link together control sections with the same name in all of the control dictionaries for a program. They facilitate absolute address assignment and the selection of subroutines to be included in the program. Control dictionary entries are ordered by their occurrence in the binary card input and are thus in ascending order within program decks. Control dictionaries from succeeding decks are stored in adjacent blocks.

If the ALTIO option has been specified on the $IBJOB card, the names of the normal FORTRAN input/output subroutines are changed to those of the alternate input/output routines.

The debugging dictionary in each deck is written out on the debug file. At the same time section 1 uses the entries to expand the debug reference tables.

The first $ORIGIN card sets the Loader to overlay mode. An entry is made for each card in the $ORIGIN card chain in the control information storage block, and special entries are made in the program name table chain and appropriate control dictionaries.

A $INCLUDE card generates similar entries in the program name table chain and control dictionaries.

The contents of $NAME cards are stored in a $NAME card chain in the control information storage block, with added data from any $ETC cards that follow. Each field on a card represents a separate entry in the chain, and entry lengths vary between three and eight words.

The contents of the $USE and $OMIT cards are entered in a $USE and $OMIT card chain in the control information storage block, with added data from related $ETC cards. The entries are similar in format to those of the $NAME card chain, but with a maximum length of three words.

One entry for each $POOL or $GROUP card and its associated $ETC card is made in a $POOL and $GROUP card chain in the control information storage block. The standard values for parameters that have been omitted on the card are also placed in the entry. The information stored in the $POOL and $GROUP card chain is used by section 3 to allocate storage to buffer areas.

One ten-word entry for each $LABEL card is made in a $LABEL card chain. Standard values for omitted options are entered automatically.

One seven-word entry for each $FILE card is made in a $FILE card chain, with standard values for omitted options entered automatically.

The size of Blank COMMON is entered from a $SIZE card in the location CCSIZE.

The standard entry point to the object program at execution time is entered from a $ENTRY card into location ENTCC+1.

Section 1 has now accepted all the data in the job application except subroutines from the Subroutine Library. It now uses the $NAME, $USE, and $OMIT card chains to modify information in the control dictionaries.

The new control section names replace the old in the $NAME card chain. The same adjustments are made in any like name section chains associated with these names in the control dictionaries. All section entries are then removed from the $NAME card chain, leaving only file entries for section 2 use.

A scan of the $USE and $OMIT card chain causes bits to be placed in the appropriate control dictionary entries.

Section 1 now returns control to the load supervisor. The load supervisor calls section 2 into storage (see Figure 46).

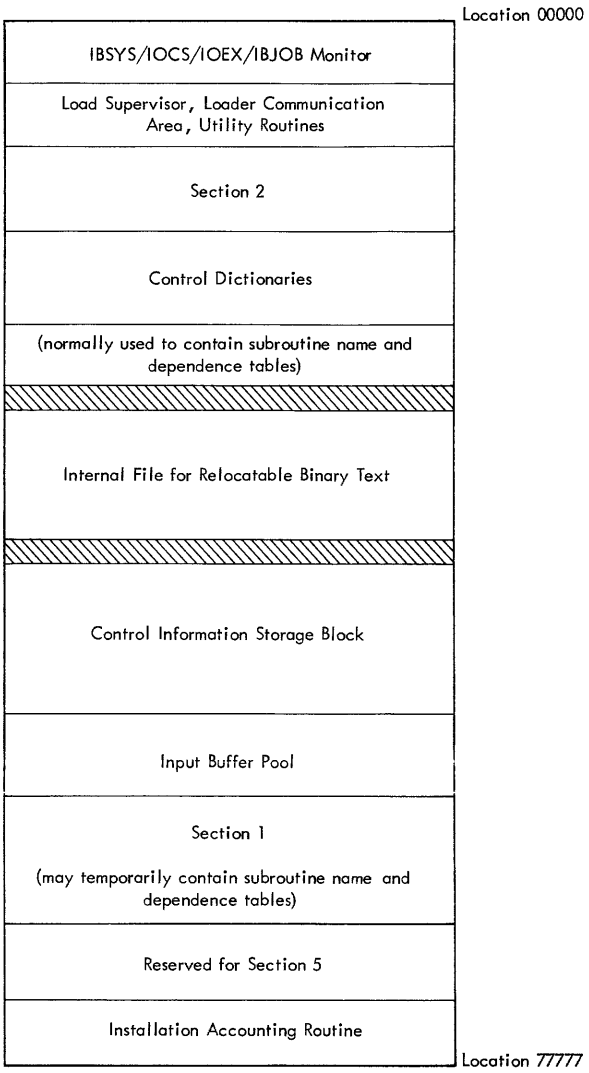


Figure 46. Storage Allocation During Section 2 Operations

Section 2

Section 2 prepares the way for the Loader to assign input/output units and file buffers, generate IOCS calling sequences, and give absolute locations to program parts. It determines the names of the subroutines that must be included as part of the program, and it builds up a body of cross-reference data for Loader sections that follow.

Section 2 first reads control information from the Subroutine Library into core storage. The Library is in three parts. The first part contains two lists of control sections: the subroutine name table and the subroutine dependence table. The second part contains the same control information as a load file: control dictionaries, file dictionaries, and Loader control cards referring to the subroutine text. The third part contains the relocatable binary text of all the subroutines in the Subroutine Library.

The subroutine name table contains the names of all real control sections appearing in the Library each with its file record or block number. The subroutine dependence table contains, for each entry in the name table, a table showing the other control sections that must be loaded with that control section. Section 2 reads these tables into core storage next to the control dictionaries. However, if the control information storage block and the control dictionaries section are so large that there

is not enough space between them, the tables are read in over the section 1 routines in upper core storage.

Section 2 now constructs a virtual section name list to tabulate all possible control section names that must be inserted in the program from the Library. First the names of the subroutines, such as .LXCON, that will be used automatically are added. Then the names of all control sections that are not defined in the program itself (i.e., virtual sections) are added. These virtual section names are added to the list through the Equality Reduction routine. This routine:

1. Examines each control dictionary entry for its type and determines whether it has been marked for deletion.
2. Deletes all entries for real control sections with the same name except either the first that occurs in the dictionaries or the one designated through $USE and $OMIT cards.
3. Examines all like name section chains for virtual control section names, i.e., names that have not been chained to real control sections that have been retained. These virtual control section names are added to the virtual section name list.

The Loader next attempts to match up all names in the subroutine name table and the virtual section name table. When a match is found, the name of the subroutine is entered into a required subroutine number list. This list is stored temporarily in the area that will be used later for section 5. If there are any names left in the virtual section name list after the comparison, a diagnostic is written on SYSOU1 and a flag is set to suppress execution. Otherwise, as soon as the list is completed, it is sorted into numerical order based on the record or block numbers.

The second part of the Library file is now processed to obtain the control information for routines entered in the required subroutine package number list. The routines used are the same section 1 routines in upper core storage used to process control information from the load file. If this part of section 1 has been overlaid by the subroutine name and dependence tables, it is now read in again over the subroutine name and dependence tables for section 2 use, since the tables are no longer needed for processing.

When all subroutine control information has been processed, section 2 uses the Equality Reduction routine to chain subroutine and load file data in the control dictionaries.

Where overlay is involved, the names on $INCLUDE cards are now assigned to the proper links. In the program name table, the $INCLUDE entries replace the other entries with the same name. In the control dictionaries, entries for control sections that are to appear in another link are flagged for section 4.

CALL statements and references between links are analyzed for validity. Downward CALL statements are marked in the control dictionaries as requiring transfer vectors. Unless NOFLOW has been specified on the $IBJOB card, a CALL statement that would cause the calling link to be overlaid results in an error message. Virtual sections that do not contain CALL statements and deleted real control sections are also checked for violation of overlay rules. Link numbers for subroutines that will be loaded into links on the overlay file are stored in the required subroutine package number list.

The control dictionaries are now almost complete. All that remains is to account for Blank COMMON, if required. The Blank COMMON name // is entered in the program name table chain; a Blank COMMON control dictionary is added to the control dictionaries area; and a like name section chain is set up within it. The length of Blank COMMON is set to the largest storage area requested by any deck in the program, and the control sections for all other Blank COMMON requests are deleted.

Section 2 next transforms those chains of information in the control information storage block that are needed by the remaining Loader sections into a more compact table form. These tables are stored in core storage directly above the control dictionaries.

The program name table chain becomes a separate table stored immediately after the required subroutine package number list. The original chain was composed of five-word entries. The first word is a chain to the other entries in the chain. The second contains the name of the Deck or subroutine. The third word is a chain to the appropriate external file dictionary chain entry. The fourth is a chain to the appropriate control dictionary entry. The fifth contains information for the overlay mechanism. In the process, the first word is dropped from each entry.

File dictionary entries in the external file dictionary chain are renamed, using what remains of the $NAME card chain.

A new table, called working file block 1, is made up from the $FILE card chain entries. Each unique file name generates a 12-word file block within the new table.

Working file block 1 is searched to find file names equivalent to the file name entries in the external file dictionary chain (i.e., comparing the $FILE card entries with the references in the program file dictionaries). If a match cannot be made, an error message results, and the first word in each external file dictionary chain entry is set to MZE 0, signaling that references to the file are to be ignored. If a match is made, the file check portion of the external file dictionary chain entry being considered is transferred to a position in the appropriate 12-word file block. The first word of the external file dictionary chain entry is changed to PZE k, where k is the index position of the file block.

A file synonym list is made up from the external file dictionary chain and the file blocks for each deck that contains references to files. Each reference now has its corresponding entry in this list. The entry contains the constant to be added to the location of the first working file block to determine the location of the file block referred to.

The second word in each program name table entry is now changed to chain the entry to the appropriate file synonym list entry. Section 4 uses file synonym lists with the program name table to assign absolute locations to file references in the program.

A $POOL and $GROUP list with buffer count specifications is built up from a comparison of file names in the working file block to those in the $POOL and $GROUP card chain. The $POOL and $GROUP list entries are chained to the appropriate working file block entries.

A working file block 2 is generated from each working file block 1. This new block, which is later placed in the object program, is similar in format to working file block 1.

Entries from the $LABEL card chain, if any, are stripped from the control information storage block and stored in the appropriate file blocks within working file block 2.

Having created the cross-reference tables necessary to set up an input/output environment for the program, section 2 now completes its work by establishing the standard entry point to the program at execution time as follows:

1. Location ENTCC is tested to determine whether an entry point has been named by a $ENTRY card.

2. If there is an entry point name, an equivalent name is sought in the control dictionaries. If none is found, execution is suppressed and an error message is written. If the name is found, its location in the program name table is put into location ENTCC and the name of the deck where the entry appears is put into location ENTCC+1.

3. If no entry point has been named, the Loader makes a search through the control dictionaries for the standard entry point name ".....". If this is found, it is inserted in ENTCC, with its deck name in ENTCC+1. But if there is no entry point ".....", the location of the first program name table entry is used, with its deck name.

Having prepared the way for section 3, section 2 now returns control to the load supervisor.

Section 3

The Loader has the data it needs to create an input/output environment for the program and to begin assigning absolute address values to program sections. Section 3 is brought in to perform these tasks. Since all the essential data from the control information storage block has been stored in other reference tables, the load supervisor reads section 3 into this storage area. It also calls in section 4, since both sections use the same routine for storing the absolute binary text. Section 4 overlays the areas formerly occupied by section 2. (See Figure 47 for the storage configuration.) Since section 5 is included in the same system record with section 4, it is read into storage at the same time.

Section 3 proceeds as follows:

Using data from working file block 1, routine IOUASG assigns input/output units to files and makes up a reserve unit table listing its assignments. It then replaces the unit encodings in both working file blocks with the addresses of the appropriate unit control blocks in IBSYS.

Unless NOLIST has been punched on the appropriate $FILE card, routine OPLIST uses data from working file block 1 to print out file mounting messages to the machine operator.

At this point the Loader can start setting aside those parts of the program that are in their final form for

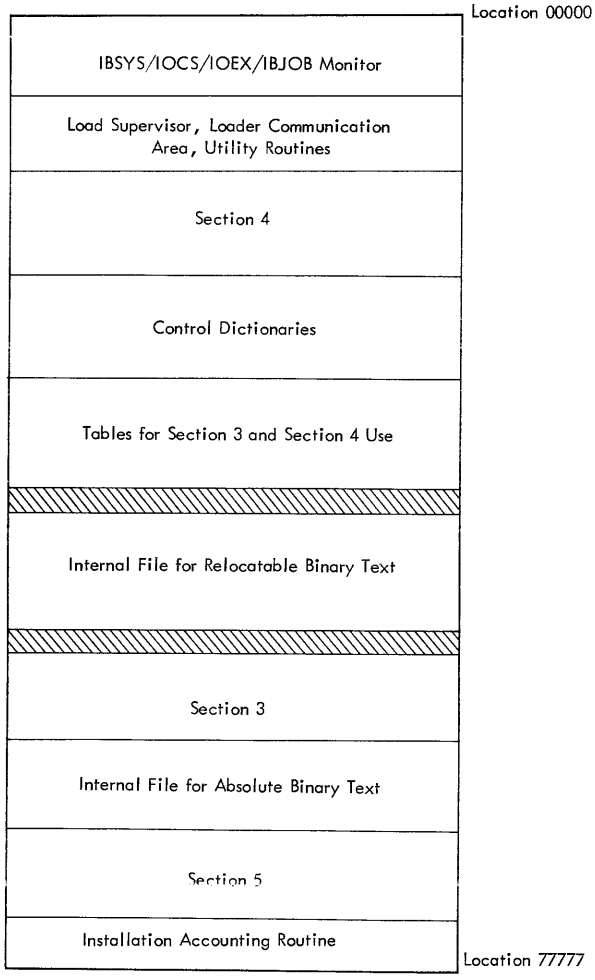


Figure 47. Storage Allocation During Section 3 Operations

program execution. The Loader defines and attaches an internal file to contain these program parts until the entire program can be loaded.

The most important parts of the program that will occupy this file are the program instructions when all absolute address values have been assigned. For this reason, the file is called the "absolute text" file. As indicated in Figure 47, section 3 uses the storage area formerly occupied by section 1 and the input buffer file for the new internal file. This file is a standard IOCS buffer pool set up in scatter-load format for temporary storage of absolute text before the program is loaded into position for execution. As explained in the section "External Storage For Text," this file may at the same time exist as an external file attached to SYSUT1.

Routine FPBLK places working file block 2 and the reserve units table in the absolute text file.

Routine RLIOB uses working file block number 1 and the $POOL and $GROUP list to estimate the storage needed for program initialization. This area must include the following if required by the object program: routines to define buffer pools and to attach files to pools, instructions to transfer control to an installation accounting routine at the start of execution and to transfer control to the object program, a file list, and an intersystem reserve unit list. RLIOB uses this estimate to determine the origin of the program.

Routine CDABS assigns an absolute address to each control dictionary entry. The preface entry of the block for each deck is marked to indicate the occurrence of deletions or insertions in that deck (i.e., whether there is a discontinuity in the location counter for that deck). The program name table is used to find the location of each control dictionary block. The link reference table is used in overlay job applications to assign the same absolute locations to links with the same logical origins.

If the DLOGIC or LOGIC option has been punched on the $IBJOB card, the CDLOG routine prints out the logic cross-reference table from the control dictionaries.

Since the amount of core storage that the object program and its subroutines will need at load time is now known, routine STASS can assign the remaining storage as buffers for various files. Using the $POOL and $GROUP list, working file block 1, and a series of its own tables, STASS makes up the exact pool and group arrangement for each file, and the final input/output initialization block. This block includes the actual DEFINE and ATTACH instructions for object program buffer pools and the final file list. STASS also sets the absolute address of any nonstandard labeling routines and places them into the file list. STASS checks the data in each file dictionary entry against actual file assignments.

The Loader next generates IOCS calling sequences and stores them in the absolute text file. It also generates part of the call on the object program and all the linkage mechanisms between the IBJOB Monitor job control section, the input/output instructions, and the object program.

If the LOGIC option has been selected, CDLOG prints out the buffer pool assignment list.

Communication words, defining the absolute location of object program file information that may be necessary for the object program or its postexecution control routine, are moved into the final absolute text file.

If the MAP option has been punched on the $IBJOB card, routine MAPROG prints a load map of storage allocation and overlay link numbers for the object program.

Section 4

At this point in Loader operations the various source decks, subroutines, and control sections within the program that is to be loaded have absolute locations; the individual instructions do not have such locations. The Loader must now make a second pass over the relocatable binary text file and control is passed to section 4 to perform this task.

Section 4 completes all Loader functions except the actual loading of the program into position for execution. It performs the following tasks:

1. Assigns absolute addresses to all text words, including the text words of subroutines that will be used by the program.
2. Sets up an overlay mechanism, if needed.
3. Prints messages regarding text errors for each deck.
4. Creates – and assigns absolute addresses to – the instructions that transfer control to the object program.

If debugging has been requested, section 4 changes into absolute form the remaining relative addresses and address references in the debugging tables that will be used during program execution. These tables are now:

1. A request dictionary containing the symbols used in the debugging request statements, with their absolute locations.
2. An STR insertion table for each deck in which debugging is to be performed, containing one entry for each point in the deck where dumps are to be made.
3. A table of constants to be used by the debugging interpreter routines.
4. The original debugging statements coded into a form more easily used by the interpreter routines.
5. A control dictionary through which the Loader can load the proper debugging interpreter routines that are needed for this job application.

Ordinarily, the combined subroutines and coded text is now stored in the absolute text file and builds downward toward the internal text file, as shown in

Figure 48. In the case of an overlay job application, however, only the main program (link 0) goes into the absolute text file. For this reason, the Loader must first set up the overlay pattern. Upon determining that an overlay job is involved, section 4 creates three reference tables to be used during execution of the object program.

The link director table records the storage status of each link and points to the other reference tables. It consists of one-word entries, one entry for each overlay link.

The link record chain lists the input/output units on which overlay links are written. It consists of two-word entries: one entry for each $ORIGIN card and one entry for each control section containing text that was relocated in another link by means of a $INCLUDE card.

The transfer vector table consists of two-word entries, one entry for each CALL to a unique entry point that could cause the loading of overlay links.

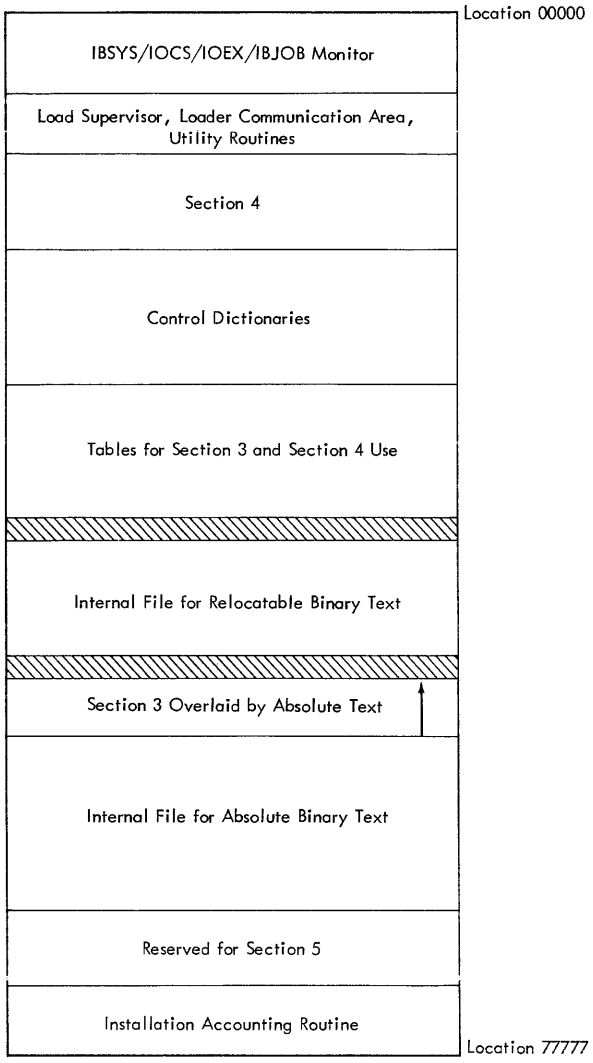


Figure 48. Text Storage Allocation During Section 4 Operations

A fourth table, the link reference table, is retained by the Loader to control the writing of text into the link buffers.

Section 4 next stores in the absolute text file the link director table, the link record chain, and the transfer vector table. These tables are all in an "overlay communication area" that is to be loaded with the main link.

Section 4 now turns to the final assignment of absolute address values to the binary text of the program instructions. The Loader first creates control break tables, using the control dictionaries. There is one control break table for each deck, reflecting all insertions and deletions that have occurred in previous processing. If the deck has no insertions or deletions, a control break table is not formed.

The Loader next processes the text, word by word. When a dictionary reference is found, the control dictionary is used to assign it an absolute value. File references are evaluated in terms of the absolute addresses of the file blocks, using the file synonym list. When subroutine text is to be inserted into the program, the Loader consults the required subroutine package number list, opens the subroutine file, and processes the subroutine text in exactly the same manner. In overlay applications, the process is the same, except that special bits in the control dictionary entries cause the Loader to assign addresses in relation to link origin, and the text is transferred into the link file buffers, instead of into the internal text files.

If overlay is involved, each word that is not in the main program is stored on the proper storage unit specified for its link.

All Loader messages pertaining to text within a deck are printed as soon as the deck has been processed. The CALL to the object program is created and stored in the absolute text file. Next the Loader stores the debugging requests dictionary and STR insertion table in the main link in absolute scatter-load form. The debugging control section dictionary and the STR insertion table are coded to be written on the debug file.

If debugging has been requested, the absolute location of each text word as it is processed is searched for in the STR insertion table. If the location is found, the prefix of the text word following is placed in the appropriate entry in the STR insertion table, and an STR prefix replaces the prefix of the text word.

Section 4 now saves certain words from the IBJOB communication area that are needed by the executing program. One of these words, for example, is a transfer location for returning control to the IBSYS Monitor. Section 4 stores the word in the postexecution routine .LXCON.

Section 5

All parts of the program for execution have now been created. The Loader must still store them in the proper locations. Control is transferred to section 5 for this final task.

As shown in Figure 49, section 5 works with two storage blocks: the absolute text file containing the scattered program parts, and an area between this file and IOEX into which the program is to be loaded in the proper order for execution. This program loading area contains data that, with a few exceptions, are of no use to the executing program: the IBJOB Monitor, the load supervisor area, section 4, and the reference tables and control dictionaries used during section 3 and 4 operations.

Section 5 first determines whether loading should be performed. If NOGO has been punched on the $IBJOB card or has been set due to a high level error during Loader operations, an error message is printed and control returns to the IBJOB Monitor.

If loading has not been suppressed, section 5 now prepares the program loading area to receive the program parts. It clears each storage location in the program loading area by setting it to:

```
STR    0,,0
```

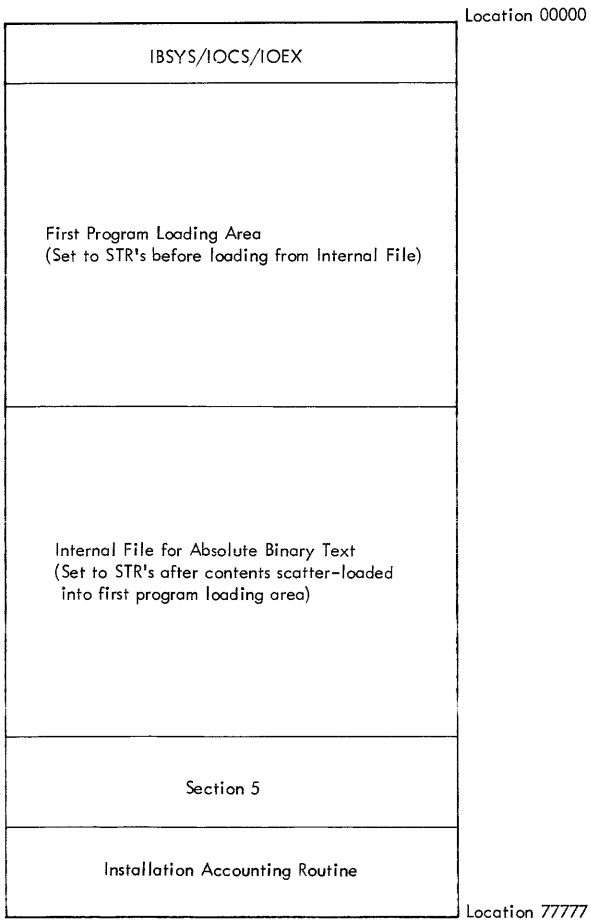


Figure 49. Storage Allocation During Section 5 Operations

The contents of the absolute text file are now scatter-loaded into the program loading area. Control words interspersed in the text direct segments of words into their proper program sequential positions. When the internal absolute text file has been read, it too is set to STR's.

The routine UNISUB removes the names of the input/output units from the availability chains in the IBSYS nucleus.

If an operator stop for tape mounting is necessary at this point, the Loader prints the appropriate message and pauses.

The remainder of core storage up through the last few instructions of section 5 are now set to STR's, and section 5 transfers control to the pre-execution package section of the object program.

Control of Program Execution

Among the program parts loaded by section 5 into position for execution, several work together as a package to handle whatever may happen during execution. The package consists of:

1. The object program file blocks
2. The reserve units table
3. The object program file list
4. A group of pre-execution initialization instructions
5. The post-execution routine .LXCON

The first three items in the package are omitted if there are no $FILE cards in the object program or in any of the Subroutine Library subroutines being used in the program.

Figure 50 shows how the package controls execution of the program. After Loader operations have been completed, section 5 transfers control to the pre-execution initialization instructions section.

The pre-execution instructions transfer control to the installation accounting routine, if any, to record that program execution has begun. Buffer pools to be used by the program are defined, and files are attached to the pools. The last instruction is a CALL to the object program to begin execution.

One of three events can occur as a result of program execution. (1) The program can execute successfully. (2) IOCS may be unable to continue, resulting in a system stop. (3) The program can contain an STR instruction requesting a system dump. In all three cases, the program transfers control to .LXCON. After a successful execution the object program transfers control to .LXCON through the entry point .LXRTN. The entry point .LXSTP is used for system stops; .LXSTR is used for system dumps.

.LXCON closes all open files used by the object program. If any one of the file blocks contains the options PRINT, PUNCH, or HOLD, the input/output unit involved

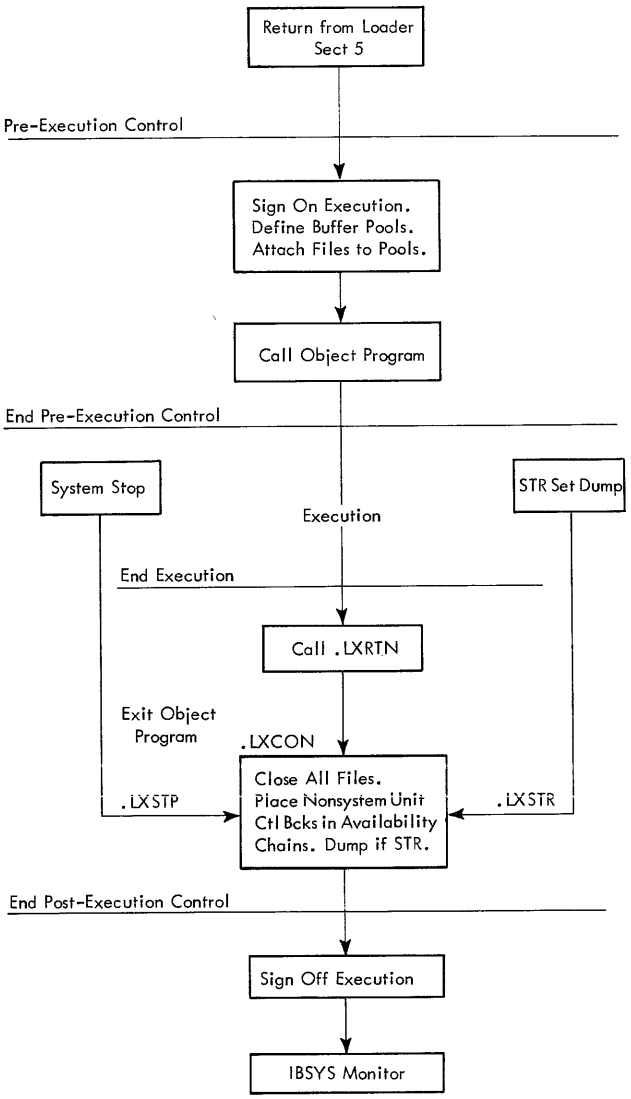


Figure 50. How the Pre-Execution Package Controls Program Execution

is rewound and unloaded, and a message for the operator is printed on-line. Nonsystem units are returned to the unit availability chain, and the intersystem reserve characters, if any, are placed in the appropriate unit control block.

If a system dump is requested by a transfer to .LXSTR, .LXCON prints an on-line message indicating that a dump is being taken, closes all required files, and transfers to the system dump routine. If the transfer from the object program is through .LXSTP, .LXCON prints an on-line system stop message and closes the files. In either case, program execution is terminated by a transfer to the installation accounting routine, if any, followed by a transfer to IBSYS for the next processor application.

NOTE: When load-time debugging has been requested, .LXSTR is set to transfer to the debugging interpreter routine. If the STR was inserted in the program for debugging purposes, the interpreter performs the action requested for that program location and eventually returns control to the object program. If the STR has no connection with debugging and the routine for taking debug dumps is not in core storage, the interpreter transfers to .LXCON for the normal action on an STR. If, however, the coding for debug dumps is in core storage, the interpreter takes a full core storage dump and transfers to .LXCON through .LXRTN. .LXCON closes the required files, transfers to the installation accounting routine, and then to IBSYS.

External Storage for Text

External storage is used for two purposes during loading: to "overflow" and "spill" excess relocatable binary text during section 1 and 2 operations, and to "overflow" absolute text during section 3, 4, and 5 operations. Figure 51 shows how the external storage is used.

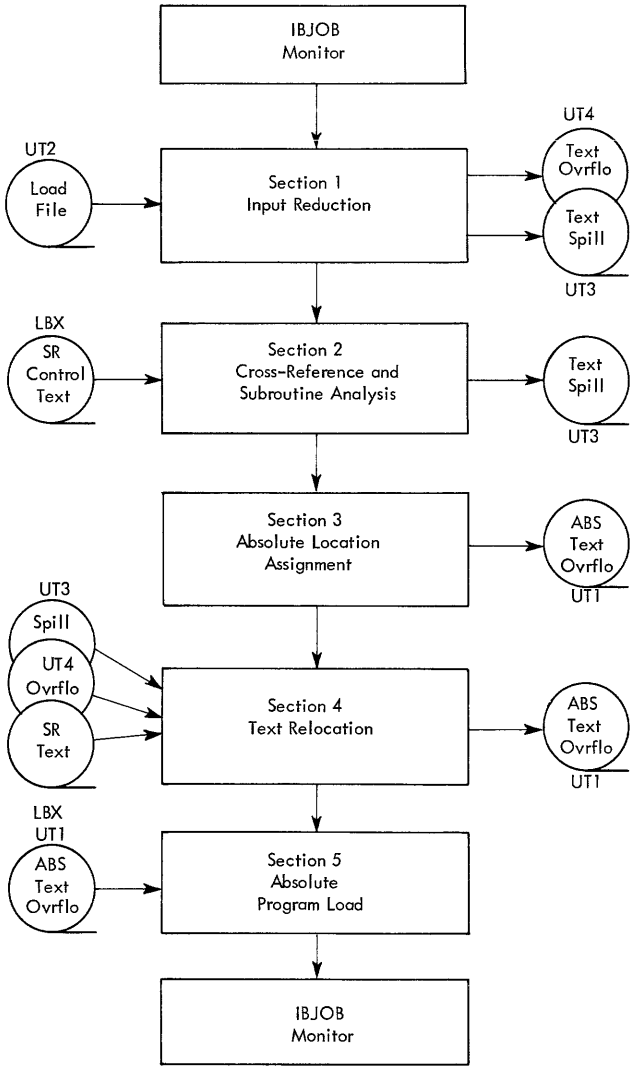


Figure 51. Overflow Pattern During Loader Operations

Overflow and Spill of Relocatable Text

The relocatable text portion of each binary deck is read from the load file or input file and moved into an internal file in core storage between the control dictionaries section and the control information storage block (see Figure 45). As the Loader continues to read in the binary decks, the control dictionaries section builds upward toward the internal text file and the control information storage block builds downward toward it.

Since neither section 1 nor section 2 process text in any way, the text can be transferred to any convenient place when it is in danger of being overlaid by data in the other two areas, or of exceeding its allotted internal file space.

This transfer can occur when the amount of text exceeds the capacity of the internal file, in which case the excess thereafter overflows onto SYSUT4, or one or both of the other storage sections could threaten to overlay the internal file, in which case the entire internal file spills onto SYSUT3.

Absolute Text Overflow

External storage is also used during the absolute address assignment process. If the amount of absolute binary text produced by section 3 exceeds the absolute internal text file, the excess overflows onto SYSUT1. During section 4 operations relocatable text is read back into storage from SYSUT3 and SYSUT4. If the absolute text produced by section 4 exceeds the absolute internal file, SYSUT1 is used for the excess. SYSUT1 is also used for words that, when scatter-loaded by section 5, would overlay a part of the internal file. Such words are held on SYSUT1 along with the excess text produced by sections 3 and 4 until section 5 time. When section 5 scatter-loads the absolute text of the program into position for execution, it loads the text from the internal file first, and then the text in the external file on SYSUT1.

Loader Input

Input to the Loader consists of the load file and subroutines taken from the IBJOB Library. The "Subroutine Library Information" section describes the Subroutine Library format.

The load file is made up of output from the Assembler, the FORTRAN IV Compiler, the load-time debugging compiler, or any combination of these three. It is written on symbolic input/output unit SYSUT2 for Loader use by the Processor Monitor.

Figure 52 shows the possible contents of a load file and their sources. The elements in the figure from the $IBLDR card through the $DKEND card make up a binary deck. There may be several such decks in the load file, depending on the number of source decks in the program.

Element	Card Format	Source
Load-time debugging reference tables	BCD and Column Binary	Debugging compiler
$IBLDR card	BCD	Generated by Assembler or FORTRAN IV Compiler
$FILE, $POOL, $GROUP, $LABEL cards	BCD	Generated by Processor Monitor or transmitted directly from source deck
$FDICT card	BCD	Generated by Assembler or FORTRAN IV Compiler
File dictionary text	Column Binary	
$TEXT card	BCD	
Relocatable binary text of source program instructions	Column Binary	
$CDICT card	BCD	
Control dictionary text	Column Binary	
$DDICT card	BCD	
Debugging dictionary text	Column Binary	
$DKEND	BCD	
$ENTRY, $SIZE, $USE, $OMIT, $NAME, $ORIGIN, $INCLUDE cards	BCD	Transmitted directly from source deck by Processor Monitor. May appear anywhere between binary decks.

Figure 52. Elements of Loader Input

In addition to the binary decks written on the load file during the current assembly process, the programmer may also add decks punched from SYSPP1 during previous assemblies. These previously assembled decks will be written on the load file in the same order as that in which the programmer has placed them in the original source program. If all the decks in the source program are previously assembled binary decks the programmer may have the NOSOURCE option punched on the $IBJOB card for the job. The entire program in this case is read by the Loader directly from symbolic unit SYSIN1. If the programmer adds a packet of load-time debugging requests to a program composed entirely of previously assembled decks, the NOSOURCE option is ignored, and the load file is built up on SYSUT2.

Each section in a binary deck is prefaced by a BCD card. In each source card the code indicating the section of the deck that follows ($FDICT, $TEXT, $CDICT, or $DDICT) begins in column 1. The six-character name of the deck starts in column 8. The $IBLDR and $DKEND cards that bound the deck as a whole are in the same format.

Each section in a binary deck is sequenced independently, starting with sequence number 0. The cards within any section must be in proper sequence and the sections themselves, if present, must be in the order shown in Figure 52.

Load File Binary Cards

The first two words on all load file binary cards are in the following form:

WORDS	BITS	MEANING
1	S, 1	If these bits are 11 and bit 3 is 0, this is a standard IBJOB Processor deck.
	2	If this bit is 0, the Loader must verify the check sum in word 2. If the bit is 1, no verification is required.
	3	This bit is usually a 0. If it is 1, an error message is printed indicating that the card is not part of an IBJOB deck.
	4	If this bit is 0, this is a load file deck. If it is 1, this deck is Prest or Cprest.
	5-7	These bits define the section of the deck as follows: 001 means this card is in the file dictionary. 010 means this card is in the relocatable text of the program instructions. 011 means this card is in the control dictionary. 100 means this card is in the debugging dictionary. 101 means this card contains relocatable binary text produced by the Load-Time Debugging Compiler.
	8-12	01010. This is the standard 7-9 punch, signifying a column binary card.
	13-17	These bits contain the number of words in the card, starting with word 3.
	21-35	These bits contain the card sequence number.
2	S, 1-35	This word contains the logical check sum of word 1 and all the data words on the card.

The contents of words 3-24 vary according to whether the words are part of the file dictionary, relocatable binary text, control dictionary, or debugging dictionary.

File Dictionary Binary Text

Words 3-22 of a binary text card in the file dictionary contain up to four 5-word entries, one entry for each file referred to in the deck. Words 23 and 24 are not used.

The file dictionary, if present, is used to validate the contents of the Loader-generated file block and the assignment of each file to a buffer pool; to transmit the name of a nonstandard label section to IOCS; and to pass the 18-character file name through to the Loader, so that correspondence can be determined between the file index numbers used in text and the file names used by the programmer. File type, mode, allowable input/output units, and blocking requirements are normally unchanged by the generated object program; hence, they are recorded in the file dictionary as a check against modification by $FILE cards.

File Dictionary Card Format: The file check entries in each card are as follows:

Words 3-7	file check entry i
Words 8-12	file check entry i+1
Words 13-17	file check entry i+2
Words 18-22	file check entry i+3
Words 23-24	not used

The format of each five-word entry is:

WORDS	BITS	MEANING
1	S	If=1, mixed mode file
	1	If=1, last FDICT entry
	2	If=1, file is NOPOOL
	3-5	not used
	6	If=1, binary mode; if=0, BCD mode
	7-8	If=00, input If=01, output If=10, input or output If=11, checkpoint
	9-14	not used
	15-17	If=001, card equipment only If=010, 729 Magnetic Tape, disk or Hypertape If=011, any input/output device If=100, Hypertape only
	18-20	If=000, n=block size (see bits 21-35) If=001, n is the minimum allowable block size. If=010, block size is a multiple of n
	21-35	n
2		If=0, standard labels (if any) If≠0, external reference name of nonstandard label routine
3-5		18-character file name

Relocatable Binary Text

Relocatable binary text cards contain the binary translation of the original source program instructions. Words 3, 4, and 5 are control words that describe the contents of words 6-24, which are the actual instructions.

The sign bit of each control word is ignored, and the remaining 35 bits are divided into seven 5-bit control groups. For example, in word 3:

Bits 1-5	describe word 6
6-10	word 7
.	.
.	.
.	.
31-35	word 12

In the same way, the control groups in word 4 describe the contents of text words 13 through 19. The first five control groups in word 5 describe the remaining text words 20 through 24.

The Loader uses the control groups when it assigns absolute values to addresses and address references. For this reason, the control groups are also called relocation bits.

The first bit in each control group describes the type of text word. The number 1 means a standard data word; the number 0 means a special entry word.

Standard Data Word: The five-bit control group for a standard data word is of the form 1 ab cd. The 1 in-

dicates a standard data word. The letters ab and cd describe the contents of the decrement and address data fields, respectively, as follows:

CODE	MEANING
00	This data field has a constant value.
01	This data field has a simple relocatable value.
10	This data field contains a file or control dictionary reference. The field itself describes which type.
11	The Loader must evaluate a complex expression to determine the value of this data field.

The prefix and tag of a standard data word are constant.

For example, consider this instruction and its control group:

```
1 00 01          TIX  NAME,1,4
```

Reading the control group from left to right, this is a standard data word with a decrement of constant value and a relocatable address.

Constant form — The relocation code 00 indicates to the Loader that the value expressed in the decrement or address referred to is constant and need not be altered.

Simple relocatable form — The relocation code 01 indicates to the Loader that the value expressed in the decrement or address referred to is a relocatable address, relative to the origin of the deck. As an example, consider the following instructions, where A has a relative address value of 100 and B of 163:

SYMBOLIC CODE	CONTROL GROUP IN BINARY	BINARY TEXT IN OCTAL
CLA A	1 00 01	0500 00 0 00100
ADD A+1	1 00 01	0400 00 0 00101
TXH B,4,A	1 01 01	3 00100 4 00163

Dictionary reference form — Dictionary references are denoted by a relocation code of 10. Each data field referred to by bits in this form contains a 15-bit code identifying the dictionary reference. The four high-order bits in the field denote the type of dictionary, and the remaining bits give the reference number within the dictionary. The high-order bit codes are:

000 0	Control dictionary reference, followed by eleven bits giving the relative location of an entry in the deck control dictionary.
000 1	File dictionary references followed by an eleven-bit file index number.

As an example, consider the following instructions:

SYMBOLIC CODE	CONTROL GROUP IN BINARY	BINARY TEXT IN OCTAL
CLA A,4	1 00 10	0500 00 4 00006
PZE AFILE	1 00 10	0 000000 0 04013

In both cases, the code 10 in the cd bits of the control group indicates that the address portions of the words refer to dictionary entries. In the address portion of the first instruction the first two octal numbers, 00, appear in binary form as 000 000. The four high-order bits of these numbers, 000 0, indicate that A is a symbol in the control dictionary for this deck, and the low-order bits indicate that it is the sixth entry. In the address portion of the second instruction the first two octal numbers, 04, appear in binary form as 000 100. The four high-order bits 000 1 refer the Loader to the file dictionary for this deck. The low-order bits specify that AFILE is the eleventh (13_8) file referred to in the deck.

Complex expression form — The relocation code 11 indicates to the Loader that it must evaluate a complex expression to determine the value of the decrement or address referred to. For example, the Loader would have to evaluate the address portion of this word as a complex expression:

```
CLA  6*TABLE+2*TABLE+3
```

The steps in the evaluation are expressed in a series of words directly following the data word that contains the complex data field. The string of words can be in one of two forms — long-form complex or short-form complex — depending on what the data field itself contains.

If the decrement or address field contains zero, the expression is long-form complex. A string of one or more words, each with its own control group, follows the data word. Each word represents a simple arithmetic step in the computation. If both the decrement and the address of the data word are complex, the string of words referring to the decrement occurs first.

Each word in a complex expression string has the following form:

```
pfx   a,b,c
```

Pfx defines the type of simple arithmetic operation to be performed, as follows:

PFX	ARITHMETIC OPERATION
PZE	addition
PON	subtraction
PTW	multiplication
PTH	division

The address portion a and the decrement c are the first and second quantities, respectively, upon which the arithmetic operation is to be performed.

The tag portion b refers to the location where the result of the arithmetic operation can be stored temporarily until needed later on in the computation. There are seven such locations, called result words. The Loader recognizes them by number, 0 through 6. A tag of 7 indicates to the Loader that the current data word completes the computation, and the result should be stored in the data field being evaluated.

For example, an evaluation word in this form:

```
PTH  a,1,c
```

means to the Loader that the result of a ÷ c is stored temporarily in storage word number 1.

When the a or c portion of a word represents the result from previous arithmetic operations, the Loader finds the word where that result is stored as follows:

1. In the control group for the word, relocation bits 11 indicate that the corresponding decrement or address in the word represents a result.

2. The decrement or address itself contains the number of the word in which the result has been stored.

For example:

RELOCATION BITS	DATA WORD
1 11 11	0 00001 1 00002

indicates that both the c and a portions of this arithmetic operation are results of previous arithmetic operations in the string. The result in the c portion can be found in the first result word, and the result in the a portion is in the second result word.

As an example of the words generated for a long-form complex expression, consider the instruction cited previously:

CLA 6*TABLE+2*TABLE+3

where TABLE is a control section whose location is to be assigned by the Loader.

Assume that the control section named TABLE is the fifth entry in the control dictionary. The data words instruction would appear as:

RELOCATION BITS	DATA WORD	MEANING
1 00 11	0500 00 0 00000	The address portion of this CLA instruction must be evaluated according to the following string of data words.
1 00 10	2 00006 1 00005	The result of the operation TABLE*6 is stored in result word 1.
1 00 10	2 00002 2 00005	The result of TABLE*2 is stored in result word 2.
1 11 11	0 00001 1 00002	The sum of the contents of result words 1 and 2 is stored in result word 1.
1 11 00	0 00001 7 00003	The contents of result word 1 added to 3 are stored in the address portion of the first data word above. The computation is now complete, and the word is stored in the final text file, the remaining data words being disregarded.

If the decrement or address field in the data word does not contain zero, the expression is short-form complex. This form may be used to express complex fields of the following form:

NAME+C

where NAME is the external name of the control section and C is some constant. The 15-bit field is formed as:

Bit 1 — 0 means C is added.
1 means C is subtracted.

Bits 2 to n — This is the entry number of NAME in the control dictionary for this deck. As many bits are used as are required to express the total length of the control dictionary; e.g., if the dictionary contains 16 entries, 5 bits would be used—bits 2 through 6.

Bits n to 15 — A constant of (15−n+1) bits to be added to, or subtracted from, the location assigned to the name referred to. Note that the long-form complex process may have to be used if the length of the control dictionary plus the length of the addend exceeds 14 bits.

The minimum value of n is 6.

As an example of evaluating a short-form complex expression, consider the instruction

CLA BLOCK+50,1

BLOCK is a control section 50 words long. Its control dictionary name, BLOCK, is the eighth entry in a 26-entry dictionary. The data word for the instruction is 0500 00 1 10062, and the bits in the address field 10062 are interpreted as:

```
1 2   6 7        15
0 01000 000110010
+ 8       50
```

Special Entry Word: Special entry data words refer to program origins, and to BSS and VFD storage areas and other instructions that affect location counters. Control groups for all special entry words are of the following form:

0 SSSS

The 0 in the first bit denotes a special entry word The four bits SSSS that follow specify the type of special entry as follows:

0000
Indicates to the Loader that there are no more data words to process on this card.

0001
Indicates that the location counter is affected; the data word is of the following form:

pfx a, ,relative location

where:
pfx = PZE if a is an absolute origin.
pfx = PON if a is the relative origin.
pfx = PTW if BSS is of length a.
pfx = PTH if the instruction is an EVEN pseudo-operation.

The address portion a is the number of the entry in the control dictionary for this deck that determines whether the Loader should generate an AXT 0,,0 instruction to force the next text word into an even location.

pfx = MZE if a is a dictionary origin in complex format. The relative location of this instruction is as it appears in the listing.

NOTE: Origins are an integral part of text; each card does not carry its relative load address.

0010
A CALL expansion follows; the data word is:

PZE 0

This control code and the word in this form are required for the overlay mechanism of the Loader.

0011
0100
0101 } Unused codes
0110
0111

10VV

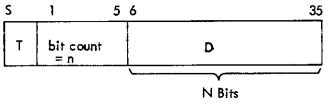
Relocation bits in this form indicate that the data word results from a VFD instruction. VV represents bit configurations indicating the kind of action the Loader must take to evaluate the data contained in the word.

For each field in a VFD instruction, the Assembler generates a word. Up to 30 bits of the data from that field are contained in bits 6-35 of the generated word. When there are more than 30 bits of data in the VFD field or there is more than one field listed, the data is contained in words that follow. As many words are generated as are needed to contain the data.

For example, consider these VFD instructions and bits 6-35 of the words generated by the Assembler:

INSTRUCTION	DATA WORD, BITS 6-35
VFD H36/ABCDEF	2122232425
	0000000026
VFD 12/3,06/47	0000000003
	0000000047

The Loader must evaluate the data contained in the generated words and load it as a table according to specifications by the Assembler. The specifications for loading are contained in bits S, 1-5 of the generated words. The code used is:



T = 0 if the current series of generated words continues beyond this word.

T = 1 if the current series terminates with this word, which is filled with zeros in the unused lefthand positions.

N < 30 if the rightmost n bits of this word are to be loaded into the table.

The specifications for evaluating the data are contained in the relocation bits VV associated with each generated word. The code used is as follows:

VV = 01 if D contains the relative address of a data word. The address is to be relocated just as in a standard data word and is substituted for a 15-bit address. The rightmost n bits are then inserted into the VFD string to be loaded.

VV = 10 if D contains a dictionary reference. It is to be evaluated just as in a standard data word dictionary reference and is substituted for a 15-bit address. The rightmost n bits are then inserted into the VFD string.

VV = 11 if D is the address of a data word that is represented by a complex expression. The Loader must evaluate the string of data words that follow in the same way that it evaluates complex expressions in standard data words. The final result is substituted for a 15-bit address, and the rightmost n bits are then inserted into the VFD string.

As examples of VFD data words and the control groups associated with them, consider the following, where the relative location of A is 103, the dictionary location B is 3, and the size of the control dictionary is 25:

VFD EXAMPLES	RELOCATION BITS	DATA WORD
VFD H30/ABCDE	0 10 00	76 2122232425
VFD H36/ABCDEF	0 10 00	36 2122232425
	0 10 00	46 0000000026
VFD 15/A, O6/47, 15/B+2	0 10 01	17 0000000103
	0 10 00	06 0000000047
	0 10 11	57 0000003002

1100
1110 } unused codes

1111

Indicates to the Loader the end of text. The address portion of the corresponding special entry word contains the relative location of the first instruction to be executed if this deck is named on a $ENTRY card.

Control Dictionary Binary Text

Words 3-24 of a control dictionary text card contain up to eleven 2-word entries. A control dictionary defines procedure and/or data areas for a deck that may be deleted, replaced, or referred to by other program segments that have been separately assembled or compiled. Each entry in the control dictionary supplies an external reference name of a control section, its location relative to the origin of the deck, and its length. If the control section is virtual, its length is entered as 0.

The first entry is the preface entry. This is not a true control section entry, but gives information about the deck as a whole. The remaining entries refer to particular control sections. The length of the first control section following the preface entry encompasses the entire deck. All other real control sections can be considered as nested within this first section.

Preface Entry Format: The preface entry contains the location of the first executable instruction, the deck length, the machine for which the deck was assembled, and the size of the control dictionary. The contents of the two words are:

```
pfx   x, , n
pze   p, , mach
```

where:

x = the relative location of the first executable instruction.

n = the length of the deck.

p = the power of 2 that includes the number of entries in this control dictionary.

mach = 0 if this deck was assembled for a 7090.
= 4 if this deck was assembled for a 7094 (may not run on a 7090).

pfx = PZE if this is a relocatable deck.
= MON if x is the absolute location at which this deck is to be loaded.

Format for All Other Entries: Control dictionary entries other than preface entries are of the following format:

```
word 1 = BCI      1, exname
word 2 = pfx      t, , n
```

where exname is the six-character alphameric external name of the control section.

pfx = PZE when this section is real (it exists in this deck).
= PON when this section is an EVEN pseudo-operation; the length always equals 0; exname is zeros.

= PTW when this section is a virtual section (only references to the section appear); exname must appear as a real control section in at least one other deck in the program (including decks from the Subroutine Library).

t = the relative location of the beginning of a control section. It is equal to zero if pfx is equal to PTW.

n = the length of a control section. It may be equal to zero.

All binary text cards for a control dictionary except the last are full.

Control dictionary entries are stored in order of increasing relative location. Nested control sections having the same relative origin are stored in order of decreasing section length.

Debugging Dictionary Binary Text

Words 3-24 in the binary text card in a debugging dictionary contain entries of from one to three words. Entries may be split between cards. Therefore, every card except the last must be full.

The debugging dictionary is processed by the Loader. It is modified and written on SYSCK2 following any information previously written by the debugging compiler. All this information makes up the first file of SYSCK2. The debugging postprocessor uses this information together with the dumps (the second file of SYSCK2) to produce the final debugging output.

Debugging Dictionary Text Card Format: Each entry in the debugging dictionary is from one to three words long. Mode changes that occur at a location for which there is no associated symbol cause a one-word entry in the debugging dictionary as:

S	1	2	3-17	18-20	21-35
0	A	M	0	M	value

The M bits in positions 2, 18, 19, and 20 make up a four-bit mode designator. A is 0 if the value in bit positions 21-35 is relative, 1 if it is absolute.

A symbol having no dimensions causes the following entry:

S	1	2	3-17	18-20	21-35
1	A	M	0	M	value
SYMBOL					

where A, M, and the value in bit positions 21-35 are as above, and SYMBOL is the BCD representation of the symbol.

A relative symbol having one dimension causes the following entry to be generated:

S	1	2	3-17	18-20	21-35
1	0	M	1st dim.	M	value
SYMBOL					

where 1st dim. is the dimension.

A relative symbol having two or three dimensions, or an absolute symbol having one, two, or three dimensions, results in the following entry:

S	1	2	3-17	18-20	21-35
1	1	M	1st dim.	M	value
SYMBOL					
0			2nd dim.	B	3rd dim.

where B is 0 if the value in bit positions 21-35 is relative, 1 if absolute. The 2nd (and 3rd) dimensions will be zero if not applicable.

Even Storage Feature

The specification of even storage for data or instructions is accomplished by the use of the MAP language EVEN pseudo-operation. This EVEN pseudo-operation is ignored by IBMAP when assembling for the 7090 or 7094 II.

The EVEN pseudo-operation causes the generation of an entry in the control dictionary, specifying a control section of zero length that has the special name of all zeros.

Since each EVEN control dictionary entry now represents a point where an even absolute location must be assigned, the Loader may now expand that control section to length 1 if it is necessary to force an even location. In addition to the generation of a zero-length control section, the EVEN pseudo-operation causes the placement of an EVEN entry in the binary text. An EVEN appearing within a control section must have the same relative location as the start of the control section.

Upon encountering this text entry, the Loader generates an AXT 0,0 instruction if the current absolute location is odd. Since no reference in text can ever appear in the EVEN control section, generation of the AXT does not affect execution of the program. Since the AXT instruction, if generated, always occupies odd storage, its execution is free.

Errors may occur when the programmer forgets that relative locations or the length of a block of data may change due to the insertion of AXT instructions.

Format for an EVEN Control Dictionary Entry

The control dictionary entry for an EVEN pseudo-operation appears as:

```
word 1    BCI    1,000000
     2    PON    r, , 0
```

The field r is the relative location which must be assigned an even absolute location.

Format for an EVEN Relocatable Binary Text Entry

In relocatable binary text an EVEN pseudo-operation appears as:

RELOCATION BITS	DATA WORD
0 0001	PTH b

The field b conforms with the 15-bit code for control dictionary references (such as 10) and refers to the correct EVEN control section.

EVEN Program Example

An example of how the Loader deals with EVEN pseudo-operations is:

REL LOC CTR	RELOCATION BITS	OCTAL	SYMBOLIC	
100	0 00 01	3 00000 0 00004	EVEN	
100	1 00 01	0 60000 0 00026	STZ	A, 4
101	1 00 01	2 00001 4 00100	TIX	*-1, 4, 1
102	1 00 00	0 00000 0 00000	HTR	0
103	0 00 01	3 00000 0 00005	EVEN	
103	0 00 01	2 00000 0 00002	BSS	2

If the program origin assigned by the Loader is even, the second EVEN text entry at 103 causes the generation of an AXT instruction that moves the BSS to relative location 104. If the origin assigned is odd, the first EVEN text entry generates an AXT instruction, moving the STZ instruction to relative location 101.

ORG = 10000		ORG = 20001	
10100	0 60000 0 10026	20101	0 77400 0 00000
10101	2 00001 4 10100	20102	0 60000 0 20027
10102	0 00000 0 00000	20103	2 00001 4 20102
10103	0 77400 0 00000	20104	0 00000 0 00000
10104	5 00000 0 00000	20105	0 77400 0 00000
10105	5 00000 0 00000	20106	5 00000 0 00000
		20107	5 00000 0 00000

Subroutine Library Information

The Subroutine Library consists of system subroutines, FORTRAN IV subroutines, and COBOL subroutines. The system subroutines are used by the IBJOB Processor to maintain control and communication among the system programs. The FORTRAN IV section of the Subroutine Library includes the FORTRAN mathematics library, the FORTRAN input/output library and the FORTRAN utility library. The COBOL subroutines include subroutines needed for the movement, conversion, input, and output of data.

The subroutines in the FORTRAN mathematics library and the subroutines in the FORTRAN utility library that are available to the applications programmer are described in the section "Subroutine Library (IBLIB)." The system subroutines, the FORTRAN input/output subroutines, the FORTRAN utility routines used at execution time, and the COBOL subroutines are described in this section.

The Subroutine Library is stored in two files on a specified system library unit (it may be the same unit as the Loader). The first file contains two lists and the control part of each subroutine.

List 1: This is a list of all real control section names appearing in the Subroutine Library. Each control section name appearing in the list is unique. Associated with each entry in this list is a position in the second list (dependence list) and the name of the subroutine in which this control section appears. Each entry also contains the record number giving the position of the subroutine in the control information and text files.

List 2: For each entry in the control section name list, the dependence list contains a table showing the control sections which must be loaded for execution of this control section. Therefore, a given control section is said to be dependent upon those control sections whose names appear in the corresponding dependence list portion. Multilevel dependency is allowed (that is, the dependent sections of each item in the dependence list are called with the original dependent subroutine).

Because equal names of control sections cause deletion of all but one of the control sections in the object program, it is possible for an object program to include a control section which will be used by a library subroutine. This is done in the object program by specifying a control section name that is the same as the name which appears in the dependence list. Conversely, care must be taken in specifying control section names, both in object programs and in library subroutines, to avoid inadvertently causing this replacement. As a means of expressing dependency, this string of control section names is written using the following conventions:

1. One half-word (18 bits) is used for each entry.
2. The three high-order bits in each entry make up a code indicating the position of the entry in the list, as follows:

000	First entry in the list
100	Second entry in the list
010	All other entries

3. Each dependency list contains 15-bit index references to the real section name list (List 1) for each required name. Each 15-bit index reference appears in either the decrement portion or the address portion of a word, with an operation code (3 bits) preceding it in the prefix or tag, respectively.

The first Library subroutine file also contains the control dictionaries for all the subroutines. It may also contain Loader control cards and the file dictionaries. A $IBLDR card must precede each control dictionary. Any of the following control cards may appear after the $IBLDR card if a subroutine requires their use: $FILE, $LABEL, $POOL, or $GROUP cards.

If a file dictionary is included, it must appear after these control cards and must be immediately preceded by a $FDICT card.

List 1, containing the subroutine name and its associated record numbers, is used to locate a particular subroutine on tape, and is used to compute a track address when the Library is on disk.

The second Subroutine Library file contains the relative binary text for all library subroutines. Each relative binary text deck is preceded by a $TEXT card and followed by a $DKEND card.

The use of the above format for the Subroutine Library permits rapid acquistion of needed subroutines without multipass searching of the library unit.

System Subroutines

The operating system uses the following Library subroutines to initialize communications regions used by the operating system components to transfer control to the object program and other system components. System subroutines also provide input/output support, overlay features, debugging facilities, and error routines. Those subroutines marked with an asterisk are

loaded with each object program. The remainder of the subroutines are loaded when needed, as determined by control cards, specifications on control cards, and the requirements of the object program.

SUBROUTINE	DESCRIPTION
*.IBSYS	Defines the indexes of the system units and the location of the System Monitor (IBSYS) communication region.
*.IOEX	Defines the location of the Input/Output Executor (IOEX) entry points.
*.JBCON	Relocates Processor Monitor communication words, used during object program execution, to an area immediately adjacent to IOEX.
*.LXCON	Closes files used by the object program, thereby stopping all input/output activity. The post-execution subroutine is required for all object program executions. It is normally entered at the end of object program execution, but it is also entered if execution is terminated by a system stop or an STR instruction. Control is returned to the System Monitor.
.IBDBI	Monitors STR instructions during debug executions, and executes debug requests. It is required by all debug executions. This subroutine is loaded at a location preceding all input decks and subroutines not explicitly associated with the Debugging Processor.
.DSTRN	Searches a table of debug request points in a program not using overlay for .IBDBI and is required by all nonoverlay debug executions.
.DSTRO	Searches a table of debug request points in a program using overlay for .IBDBI and is required by all overlay debug executions.
.IODEF	Contains the primary Input/Output Control System (IOCS) communication region. General entry and exit routines used by all IOCS packages are contained in this subroutine. The actual communication region required for a given level of IOCS is initialized by .IOCSF for standard FORTRAN IOCS; .IOCSM for Minimum IOCS; .IOCSM and .IOCSB for Basic IOCS; and .IOCSM, .IOCSB, and .IOCSL for Label IOCS. Loading of this subroutine is suppressed if the alternate FORTRAN input/output package is requested.
.IOCSF	Contains the text for the special IOCS used by FORTRAN IV object programs if the standard FORTRAN input/output package is requested. This subroutine also initializes the communication region required for standard FORTRAN IOCS.
.IOCS	Contains the text for all levels of relocatable IOCS.
.IOCSM	Initializes the communication region required for Minimum IOCS.
.IOCSB	Initializes the communication region (in addition to that initialized by .IOCSM) required for Basic IOCS.
.IOCSL	Initializes the communication region (in addition to those initialized by .IOCSM and .IOCSB) required for LABEL IOCS.
.LOVRY	Loads overlay links. It is required for all object programs using the overlay feature.
.LXSL	Initializes read and write selects, and provides linkage to IOEX when the alternate FORTRAN input/output package is requested for a FORTRAN IV object program.
*.FPTRP	Determines the condition that caused the floating-point trap, sets appropriate registers accordingly, and writes a message on the system output unit, giving the cause of the trap and the octal location at which it occurred. Location COUNT contains the maximum number plus one, of times that messages will be written for each execution. This number is set at five plus one, but it can be changed by the programmer. The number in location COUNT, which is the control section .COUNT, may be changed by addressing this control section in a MAP language subroutine of the following form: ENTRY .COUNT COUNT DEC n END where n is a decimal number. Messages are then written n-1 times. The value set by this procedure applies only for one program.
.RAND	Provides for processing of random records on 1301/2302 Disk Storage.

NOTE: The Debugging Processor subroutine .IBDBI must follow subroutine .LXCON in the Subroutine Library. The Input/Output Control System subroutines must be in the following order in the Library:

.IODEF
.IOCSF
.IOCS
.IOCSM
.IOCSB
.IOCSL

FORTRAN IV Input/Output Library

The FORTRAN IV input/output library contains a group of subroutines used to implement the source language input/output statements. The input/output library contains two input/output packages. The standard package and the alternate package coordinate all binary and BCD input/output operations for a FORTRAN IV program. Both packages are specified by parameters on the SIBJOB card. The available input/output support for a FORTRAN IV program is summarized as:

REQUEST	PACKAGE USED	INPUT/OUTPUT SUPPORT	
ALTIO	Alternate	IOEX	Trap Supervisor
FIOCS	Standard	FIOCS	Modified version of the minimum level of IOCS
blank	Standard	IOCS	Minimum level or Basic level or Labels level (The level of IOCS to be used is determined by the Loader.)

NOTE: With the exception of ALTIO, if the programmer requires a higher level of IOCS than what is specified, the Loader automatically loads the higher level with the program.

Standard FORTRAN IV Input/Output Package

The standard FORTRAN IV input/output package is one of two such packages in the Subroutine Library to coordinate input/output operations for FORTRAN programs. The subroutines in this package handle interface functions, conversions, and buffer-building. Data is transmitted using one of the several levels of the Input/Output Control System (IOCS).

The interface subroutines in the standard package control the input/output operations that are requested by actual FORTRAN IV source program statements. A unique interface subroutine exists for each type of input/output statement. Because the interface subroutines are unique, only those subroutines needed for the specific operation requested must be in core storage at execution time. These subroutines are:

SUBROUTINE	DESCRIPTION	ENTRY POINT	DESCRIPTION
FRDD	Controls reading of BCD records.	.FRDD.	Entry point for BCD read; called for source program statement READ (unit, format) list.
FRDU	Controls reading of BCD records.	.FRDU.	Entry point for BCD read; called for source program statement READ (unit, NAMELIST name).
FWRD	Controls writing of BCD records.	.FWRD.	Entry point for BCD write; called for source program statement WRITE (unit, format) list.
FWRU	Controls writing of BCD records.	.FWRU.	Entry point for BCD write; called for source program statement WRITE (unit, NAMELIST name).
FRDB	Controls reading of binary records.	.FRDB.	Entry point for binary read; called for source program statement READ (unit) list.
FWRB	Controls writing of binary records.	.FWRB.	Entry point for binary write; called for source program statement WRITE (unit) list.
FRCD	Controls on-line reading of cards and conversion of alphameric card to BCD.	.FRCD.	Entry point for card read; called for source program statement READ format, list.
FPRN	Controls printing by the on-line printer.	.FPRN.	Entry point for source program statement PRINT format, list.
FPUN	Controls punching by the on-line punch.	.FPUN.	Entry point for source program statement PUNCH format, list.
FRWT	Rewinds designated unit.	.FRWT.	Entry point for the source program statement REWIND unit.
FEFT	Writes a file mark on the designated unit.	.FEFT.	Entry point for the source program statement END FILE unit.
FBST	Backspaces the designated unit one record.	.FBST.	Entry point for the source program statement BACKSPACE unit.

The buffer-building subroutines (FCNV, FIOB, FIOH, and FIOS) control the conversion and movement of data into and out of the buffers. These subroutines are identical for both FORTRAN IV input/output packages and are described in the following list of miscellaneous subroutines. The remainder of the subroutines described are needed to implement the input/output statements.

SUBROUTINE	DESCRIPTION	ENTRY POINT	DESCRIPTION
FCNV	Performs all necessary conversions for input and output list items.[1]	.FCNV.	Entry point for input and output list.
FIOB	Processes list items for binary transmission.	.FIOB.	Entry point for binary transmision routines.
		.FBLT.	Entry point for single-precision binary input/output list.
		.FBDT.	Entry point for double-precision binary input/output list.
		.FRLR.	Entry point for end of list for binary input.
		.FWLR.	Entry point for end of list for binary output.
FIOH	Scans FORMAT statements and links to the object program to begin conversion of data.	.FIOH.	Entry point for BCD transmission routines.
		.FRTN.	Entry point for end of list for BCD input.
		.FFIL.	Entry point for end of list for BCD output.
FIOS	Initializes all input/output library IOCS calling sequences for binary and BCD transmission.	.FIOS.	Entry point for all input/output interface routines.
		.FSEL.	Entry point for BCD or binary read.
		.FRTD.	Entry point for BCD write.

[1]Conversions performed on source program input data and object program input data give the same result.

SUBROUTINE	DESCRIPTION	ENTRY POINT	DESCRIPTION
		.FRTB.	Entry point for binary write.
		.FILR.	Entry point to initialize input/output command for reading.
		.FILL.	Entry point to initialize input/output command for writing.
		.FCLS.	Entry point to close a file.
		.FOPN.	Entry point to open a file.
		.TOUT.	Entry point at which the call to entry point .FOUT. is loaded if subroutine UN06 is used by object program.
FIOU	Controls processing of lists of variables and arrays associated with a NAMELIST name for BCD input.	.FIOU.	Entry point for input of BCD variables and arrays referred to by a NAMELIST name; called by FRDU.
FOUT	Writes blocked records on the system output unit.	.FOUT.	Entry point for subroutine FOUT. If subroutine FOUT is loaded with an object program, the calls to entry point .LXSEL in subroutines .LXCON, .FPTRP, and FXEM are overlaid by a call to subroutine FOUT.
FSLBO	Controls processing of lists containing nonsubscripted binary array names for output.	.FBLO.	Entry point for output of nonsubscripted binary arrays consisting of single-precision or complex data.
		.FBDO.	Entry point for output of nonsubscripted double-precision binary arrays.
FSLBI	Controls processing of lists containing nonsubscripted binary array names for input.	.FBLI.	Entry point for input of nonsubscripted binary arrays consisting of single-precision or complex data.
		.FBDI.	Entry point for input of nonsubscripted double-precision binary arrays.
FSLDI	Controls processing of lists containing nonsubscripted BCD array names for input.	.FSLI.	Entry point for input of nonsubscripted BCD arrays consisting of single-precision or complex data.
		.FSDI.	Entry point for input of nonsubscripted double-precision BCD arrays.
FSLDO	Controls processing of lists containing nonsubscripted BCD array names for output.	.FSLO.	Entry point for output of nonsubscripted BCD arrays consisting of single-precision or complex data.
		.FSDO.	Entry point for output of nonsubscripted double-precision BCD arrays.
FSLI	Sets up indexing for input to nonsubscripted arrays.	.SLI.	Entry point for input of single-precision arrays.
		.SLI1.	Set by subroutine FSLDI or FSLBI to contain appropriate entry point to subroutine FRWD or FRWB, depending upon whether a single-precision array is BCD or binary.
		.SDI.	Entry point for input of double-precision arrays.
		.SDI1.	Set by subroutine FSLDI or FSLBI to contain appropriate entry point to subroutine FRWD or FRWB, depending upon whether a double-precision array is BCD or binary.
FSLO	Sets up indexing for output of nonsubscripted arrays.	.SLO.	Entry point for output of single-precision arrays.
		.SLO1.	Set by subroutine FSLDO or FSLBO to contain appropriate entry point to subroutine FRWD or FRWB, depending upon whether a single-precision array is BCD or binary.
		.SDO.	Entry for output of double-precision array.
		.SDO1.	Set by subroutine FSLDO or FSLBO to contain appropriate entry point to subroutine FRWD or FRWB, depending upon whether a double-precision array is BCD or binary.
FVIO	Establishes identification between a variable logical unit and the cor-	.FVIO.	Entry point called for any FORTRAN input/output source statement

SUBROUTINE	DESCRIPTION	ENTRY POINT	DESCRIPTION
	responding FORTRAN file.		that specifies a variable unit.
FWRO	Controls processing of lists of variables and arrays associated with BCD output.	.FWRO.	Entry point for output of BCD variables and arrays referred to by a NAMELIST name; called by FWRU.

NOTE: The 7094 user can choose between two FCNV routines: the standard routine and the optional routine. Appendix E describes how to specify the desired version. The differences between the two versions are:

1. The optional routine is generally faster.
2. The handling of output data for the optional routine differs from the standard as follows:
   a. If an output number that has been converted by E-, D-, F-, or I- conversion requires more space than is allowed by the field width w, the number is disregarded and the field is filled with asterisks. If the number requires fewer than w spaces, the leftmost spaces are filled with blanks.
   b. For output with E-conversion, if the specification nPEW.d requires n+d decimal digits (where n+d is greater than 8) n+d−8 zeros are appended as the low-order digits.
   c. For output with D-conversion, if the format specification nPDW.d requires n+d decimal digits (where n+d is greater than 16), n+d −16 zeros are appended as the low-order digits.

Alternate FORTRAN IV Input/Output Package

The alternate package consists of a group of Library subroutines that coordinate binary and BCD input/output operations for FORTRAN IV programs. These subroutines use the Input/Output Executor (IOEX) to supervise trapping operations. Because the IOCS routines are not used, the locations previously used by the standard package for the required level of IOCS are available to the object program. In addition, the locations used by IOCS for buffers are available because the alternate package supplies the necessary buffers.

When the alternate package is requested, the entry points in the calling sequences generated during compilation are the entry points for the subroutines in the standard package. The Loader automatically substitutes the entry points for the necessary subroutines in the alternate package when the program is loaded. The standard package entry points and the corresponding alternate package entry points are:

STANDARD PACKAGE	ALTERNATE PACKAGE
.FRDD.	..FRDD
.FRDU.	..FRDU
.FWRD.	..FWRD
.FWRU.	..FWRU
.FRDB.	..FRDB
.FWRB.	..FWRB
.FRWT.	..FRWT
.FEFT.	..FEFT
.FBST.	..FBST
.FIOS.	..FIOS
.UN06.	..UN06
.FVIO.	..FVIO

Unit Definition and Input Files

The alternate package is compatible with the standard package in respect to data language specifications and unit definition by FILE pseudo-operations. However, if disk or Hypertape is specified in the FILE pseudo-operations when the alternate package is requested, an irrecoverable error results and the job is terminated.

The input data files supplied for an object program using the alternate package may be in binary mode, BCD mode, or a combination of both modes. Output from a FORTRAN IV program using the alternate package may be in binary mode, unblocked BCD mode, or a combination of both modes. However, the alternate package does not provide a look-ahead feature to facilitate reading of files with mixed mode. Therefore, the standard package cannot process mixed mode files that were written by the alternate package.

Overlay Compatibility

Since the alternate package is compatible with the overlay feature, overlay may be used in a FORTRAN IV program to load the necessary subroutines from the alternate package. These input/output support routines may reside in the lower level links of an overlay job, thereby increasing the storage space available for the object program. When running a FORTRAN IV overlay job, the necessary subroutines are loaded by means of a generated CALL pseudo-operation, which forces overlay of subroutines already in core storage. More information concerning the use of the overlay feature can be found in the section "The Loader (IBLDR)."

Subroutines

The subroutines in the alternate package handle interface functions and data transmission. The alternate package uses the buffer-building subroutines FCNV, FIOB, and FIOH in the standard package. The buffer-building subroutines handle conversion and movement of data into and out of the buffers. These subroutines are described under "Standard FORTRAN IV Input/Output Package."

The interface subroutines control the input/output operations requested by actual FORTRAN IV source program statements. A unique interface subroutine exists

for each type of input/output statement. Because the subroutines are individual, only those subroutines needed for the requested operation must be in core storage at execution time. The interface subroutines in the alternate package are:

SUBROUTINE DECKNAME	DESCRIPTION	ENTRY POINT	DESCRIPTION
FRDD.	Controls reading of BCD records.	..FRDD	Entry point for BCD read; substituted at load time for source program statement READ (unit, format) list.
FRDU.	Controls reading of BCD records.	..FRDU	Entry point for BCD read; substituted at load time for source program statement READ (unit, NAMELIST name).
FWRD.	Controls writing of BCD records.	..FWRD	Entry point for BCD write; substituted at load time for source program statement WRITE (unit, format) list.
FWRU.	Controls writing of BCD records.	..FWRU	Entry point for BCD write; substituted at load time for source program statement WRITE (unit, NAMELIST name).
FRDB.	Controls reading of binary records.	..FRDB	Entry point for binary read; substituted at load time for source program statement READ (unit) list.
FWRB.	Controls writing of binary records.	..FWRB	Entry point for binary write; substituted at load time for source program statement WRITE (unit) list.
FEFT.	End-of-file routine using IOEX.	..FEFT	Entry point substituted at load time for source program statement END FILE unit.
FRWT.	Rewinds tape using IOEX.	..FRWT	Entry point substituted at load time for source program statement REWIND unit.
FBST.	Backspaces tape using IOEX.	..FBST	Entry point substituted at load time for source program statement BACKSPACE unit.

The alternate package also contains subroutines to perform input/output initialization, read/write selects, definition of output unit, and identification of logical units. The subroutines used by the alternate package to perform these functions are:

SUBROUTINE DECKNAME	DESCRIPTION	ENTRY POINT	DESCRIPTION
FIOS.	Initializes all input/output IOEX calling sequences.	..FIOS	Entry point for all alternate interface subroutines.
FBCD.	Contains BCD buffer and performs all BCD initialization.	..FBCD	Entry point for a BCD read request; used with ..FIOS.
		..FBCW	Entry point for a BCD write request; used with ..FIOS.
		..FBCD	Entry point for location of BCD buffer.
FBIN.	Contains binary buffers and performs all binary initialization.	..FBID	Entry point for binary read request; used with ..FIOS.
		..FBIN	Entry point for binary write request; used with ..FIOS.
		..FBIB	Entry point for location of primary binary buffer.
UNIT06	Defines the tape output unit and prevents loading of FOUT.	..UN06	Entry point for the subroutine to write on the system output unit.
FVIO.	Establishes identification between variable logical unit and the correct FORTRAN file; also prevents loading of FOUT.	..FVIO	Entry point for any input/output source statement that specifies a variable unit.

NOTE: In addition to performing specific input/output operations, two of the subroutines also inhibit the loading of a special subroutine (FOUT) used by the standard package. Subroutine FOUT is loaded by the standard package whenever the system output unit is to be used for output. When used, FOUT forces the use of the IOCS routines. For this reason, the loading of FOUT is prevented whenever the alternate package is used.

The alternate package uses the Library subroutine .LXSL for transmission and linkage to IOEX. This subroutine is not contained in the alternate package, but is loaded by both the standard and alternate packages.

Correspondence Between FORTRAN Symbolic Units and System Files

The input/output devices used in data transmission are always referred to symbolically in FORTRAN IV input/output statements. Object program input/output operations are handled by the standard package or the alternate package buffering routines and one of the levels of IOCS. The relationship between the symbolic unit and the system file is shown in Figure 53. The normal input/output configuration contains eight symbolic units.

FORTRAN Symbolic Unit	System File	Mode		Function
		Standard Package	Alternate Package	
01	SYSUT1	Binary	Binary or BCD	Input or output
02	SYSUT2	Binary	Binary or BCD	Input or output
03	SYSUT3	Binary	Binary or BCD	Input or output
04	SYSUT4	Binary	Binary or BCD	Input or output
05	SYSIN1	BCD	BCD	Input
06	SYSOU1	BCD	BCD	Output
07	SYSPP1	Binary	Binary	Output
08	System Availability Chain	BCD	BCD	Input or output

Figure 53. Correspondence between FORTRAN Symbolic Units and System Files

The standard package contains eight file subroutines named UN01, UN02, UN03, UN04, UN05, UN06, UN07, and UN08. These subroutines correspond to the eight FORTRAN symbolic units and produce the $FILE cards used by the Loader to set up file specifications for each file. The file subroutines are in the form:

```
                ENTRY  .UNxx.
.UNxx.          PZE    UNITxx
UNITxx          FILE   specifications
```

where xx is the two-digit FORTRAN symbolic unit number, and the file specifications are as follows:

```
UNIT01  FILE  , UT1, READY, INOUT, BLK = 256, BIN, NOLIST
UNIT02  FILE  , UT2, READY, INOUT, BLK = 256, BIN, NOLIST
UNIT03  FILE  , UT3, READY, INOUT, BLK = 256, BIN, NOLIST
UNIT04  FILE  , UT4, READY, INOUT, BLK = 256, BIN, NOLIST
UNIT05  FILE  , IN, READY, INPUT, BLK = 14, MULTIREEL, BCD, NOLIST
UNIT06  FILE  , OU, READY, OUTPUT, BLK = 110, MULTIREEL, BCD,
              NOLIST
UNIT07  FILE  , PP, READY, OUTPUT, BLK = 28, MULTIREEL, BIN,
              NOLIST
UNIT08  FILE  ,, MOUNT, INOUT, BLK = 22, BCD
```

Subroutine UN06 contains the additional entry point .BUFSZ in the form

```
.BUFSZ          PZE  BUFSIZ
BUFSIZ          EQU  22
```

where BUFSIZ contains the maximum BCD logical record size.

The use of a library subroutine of this form produces the $FILE cards used by the Loader to establish correspondence between FORTRAN logical unit xx and the associated system file. The file specifications listed previously are the standard FORTRAN file specifications for all system files. Since the density is not specified, high density is assumed. If another system unit is later assigned to a file, the file specifications for the system unit function override the density and file-closing specifications set by the generated $FILE card.

Although all specifications for the input/output units are standard for FORTRAN IV, the file specifications for any unit may be altered. The Library file subroutine is reassembled, substituting the desired specifications in the variable field of the FILE pseudo-operation. (The publication *IBM 7090/7094 IBSYS Operating System: Macro Assembly Program (MAP) Language*, Form C28-6392, describes the use of the FILE pseudo-operation.)

Reassembly of the library subroutine produces a $FILE card with the new file specifications, enabling the Loader to establish a corresponding file with these specifications. Symbolic location .UNXX. is entered into the control dictionary as an external symbol by the ENTRY pseudo-operation. File UNITXX is entered into the file dictionary for this Library subroutine. Because of the FILE pseudo-operation, whenever UNITXX appears in the variable field of any instruction, the relocatable reference is to the file dictionary entry for file UNITXX. The generated $FILE card establishes the correspondence between file UNITXX and the related system unit. The file specifications are those in the variable field of the FILE pseudo-operation. At execution time, the address field of symbolic location .UNXX. is set by the Loader as the absolute address of the file control block of the corresponding unit.

Subroutine Library Listing Output

The loading of file subroutine UN06 forces subroutine FWRD to use subroutine FOUT when the system output unit is written on. Subroutine FWRD must precede subroutine UN06 in the Subroutine Library, enabling FWRD to call subroutine FOUT.

Subroutine FOUT generates two types of output. The status of bit 2 in the word at location .FDPOS determines the type of output to be generated. When bit 2 contains a 1, the output is in BCD mode, blocked up to five lines per block. When bit 2 contains a 0, the output is in binary mode. The blocking factor, i.e., the number of logical records per block, is a function of the buffer size and maximum record size specified in subroutine UN06. The first word of each binary output block is a block control word. This word contains $(76xxxxxxxxxx)_8$, where x . . . x is the number of records contained in the block. The first word of each record within the block is a record control word. This word contains $(5xxxxx200460)_8$, where xxxxx is the number of characters in the logical record.

If output is to be printed on the IBM 720 Printer or listed off-line by a 1401 utility program that simulates this type of output, the file specification for file UNIT06 must be

```
UNIT06  FILE  ,OU, READY, OUTPUT, BLK = 110, MULTIREEL, BCD, NOLIST
```

When the IBM 720 Printer is used for output, the following change must be made in the FCNV subroutine

```
LIMIT           EQU  20
```

NOTE: When using the optional 7094 conversion routine, the change is

```
LIMIT    EQU 21
```

The maximum size of BCD logical records specified in .BUFSZ must be changed to 20.

If the contents of the system output unit are to be printed using the IBM 1401 Peripheral Input/Output Program, the file subroutine UN06 must contain the following file specifications:

```
UNIT06 FILE  ,OU,  READY, OUTPUT, BLK = 116, MULTIREEL, BIN, NOLIST
```

FORTRAN IV Utility Library

The utility library contains subroutines used by the FORTRAN IV Compiler to restore information destroyed by the object program, perform error diagnostics, aid the translation of FORTRAN IV into FORTRAN II, and return control to the operating system. These subroutines are described as follows:

SUBROUTINE	DESCRIPTION	ENTRY POINT	DESCRIPTION
.ERAS.	Provides four erasable locations used by object program.	E.1,E.2, E.3,E.4	Erasable words used by object program.
.XCC.	Provides four constants for FORTRAN IV Compiler use.	C.1 C.2 C.3 C.4	Erasable words used by FORTRAN IV Compiler.
FPARST	Used to determine for FORTRAN IV, address of desired part of double-precision or complex pair, as specified in FORTRAN II program.	PART	Entry point to determine address or quantity to be obtained.
		STORE	Entry point for obtaining address into which a quantity is to be stored.
FXEM	Controls object program error procedure.	.FXEM.	Entry point for execution error diagnostics.
		.FXOUT	Entry point at which the call to entry point .LXSEL is overlaid by a call to subroutine FOUT if FOUT is loaded with an object program.
		.FXARG	Parameter for the call to subroutine FOUT.
XIT	Returns control to subroutine .LXCON.	EXIT	Entry point for ending program execution.

The utility library also contains some subroutines available for use in a FORTRAN IV program. These subroutines are described in the section "Subroutine Library (IBLIB)."

The utility subroutine FXEM is called when an object program error is found by the Subroutine Library, and an error message is written on the system output unit. The error messages are described in the section "Subroutine Library Error Messages." Normally, when an error occurs, execution is terminated and control is given to subroutine .LXCON to return control to the operating system. However, some subroutines have optional exits that allow execution to continue.

These optional exits are controlled by bits in locations OPTWD1, OPTWD2, and OPTWD3. These bits correspond to error codes listed with the error messages and the optional exits. Error codes 1-35 are controlled by bits 1-35 of OPTWD1; codes 36-70 are controlled by bits 1-35 of OPTWD2; and codes 71-77 are controlled by bits 1-7 of OPTWD3. To use an optional exit, the bit associated with the relevant error code must be set to 1. A control section in subroutine FXEM sets to 1 each bit relevant to an optional exit enabling use of all permissible exits. The control section used is:

```
                ENTRY   .OPTW.
.OPTW.          OCT     377777777740
                DEC     0
                OCT     376000000000
                END
```

In the distributed version of the Subroutine Library, the bits for error codes 1-30 and 71-77 are set to 1 for the use of optional exits. If the optional exits are not to be used, the appropriate bits may be set to zero by changing locations OPTWD1, OPTWD2, or OPTWD3. These locations are the three octal words in the .OPTW. control section. The exits set by changing these locations apply only for one application. The use of optional exits may be made standard by reassembling subroutine FXEM with the bits set as desired.

In addition to the error conditions found by the Subroutine Library, an object program calls subroutine FXEM when an invalid value for a computed GO TO statement is found. This error condition has the error code 55 and has no optional exit. If subroutine FXEM is called by a programmer-designed routine and a nonstandard error code argument is used, the error code is written on the system output unit, execution is terminated, and control is given to subroutine .LXCON.

An error-flow trace is given each time subroutine FXEM is called. The trace lists the sequence of calls, in reverse order, through any number of levels of subprograms out of the main program.

Three pieces of information are given for every CALL statement in the sequence: the name of the routine in which the CALL statement occurs; the absolute location of the CALL in core storage; and the line or identification number of the CALL statement as it appears in a listing of the given routine.

A complete error-flow trace is not possible in a MAP routine if a call is made to an entry point within the same routine. This cannot occur in a routine written in the FORTRAN language or in a subroutine in the Subroutine Library.

COBOL Subroutines

The Subroutine Library contains a group of subroutines used by COBOL object programs. The COBOL Compiler generates the instructions needed to call the appropriate subroutine. The Loader completes the object program by loading the necessary subroutines and inserting the addresses assigned to the subroutines into any instructions referring to the subroutines.

COBOL object programs most frequently use the MOVPAK subroutines, which move, convert, and edit data. Another group of COBOL subroutines in the Subroutine Library coordinate input/output operations. A third group of subroutines perform arithmetic operations, additional conversion, and comparison of alphabetic fields. A fourth set of subroutines provide access to FORTRAN mathematical subroutines from a COBOL program.

MOVPAK Subroutines

The MOVPAK Subroutines move data from a source field to a receiving field, performing necessary conversions and editing operations. The subroutine used to perform the necessary operation depends on the data in the source field and the type of data desired in the receiving field.

These subroutines use four special locations to store information. The MOVPAK subroutines are usually called by one of four major entry points and then by another call to one of the special subroutines to perform the move.

MOVPAK Subroutine Special Locations

The MOVPAK subroutines use four special locations to retain information while the subroutines are executed. Two of these locations indicate the position in storage of the data involved in the move. The other two locations indicate certain conditions that may result when the subroutines are executed.

Location .CAREF indicates the source field of the data to be moved. .CAREF contains

```
PZE  location,,byte
```

where location is the address of the first word of the source field involved in the move. However, since a source field may begin with part of a word, the first byte position of the source field must also be indicated. The value for byte may be 0, 1, 2, 3, 4, or 5, depending on the byte that begins the source field. Since a byte is defined as six consecutive bits, byte position 0 is bits S through 5 of a word in storage, byte position 1 is bits 6 through 11, etc.

Location .CBREF indicates the receiving field of the data to be moved. .CBREF contains

```
PZE  location,,byte
```

where location is the address of the first word of the receiving field and byte the first byte position involved in the move. The byte position for the receiving field is not required to coincide with the byte position for the source field.

Location .COFLO is set to nonzero whenever any one of the numeric move or convert MOVPAK subroutines detects the truncation of significant high-order digits. .COFLO contains

```
PZE  **
```

The location is tested by generated instructions to determine if a SIZE ERROR has occurred. If a SIZE ERROR has occurred, a message is written indicating that significant digits were truncated, and execution of the program continues.

Location .CUFLO is set to nonzero whenever a floating-point underflow results from a move operation. .CUFLO contains

```
PZE  **
```

At present, no generated instructions test the status of this location.

NOTE: The floating-point trap routine also indicates occurrences of floating underflow and overflow. (See the description of .FPTRP in "System Subroutines.")

MOVPAK Major Entry Points

The MOVPAK subroutines are called by using one of four major entry points. The entry point used depends on the number of address reference words that must be set up. These words, if used, follow the call to the entry point and indicate the position of data in storage. The information contained in the address reference words is used to set locations .CAREF and .CBREF.

If both a source address reference word and a receiving address reference word must be set up, entry point .CMPAK is used. The call to this subroutine is generated as:

```
TSX   .CMPAK,4
  source address reference word
  receiving address reference word
  (begin specific move call)
```

This entry uses the address reference words to set the contents of locations .CAREF and .CBREF. Control is then transferred by a specific move call to the subroutine that performs the move. The move call used depends on the type of data used. These move calls are described in the section "MOVPAK Subroutine Calls."

If the source field is an arithmetic register or if no source field is needed, entry point .CMPK1 is used. The following coding is generated to call this entry point:

```
TSX   .CMPK1,4
  receiving address reference word
  (begin specific move call)
```

If the information that results from a conversion or edit operation is to remain in an arithmetic register,

entry point .CMPK2 is used. The following coding is used:

```
TSX  .CMPK2,4
  source address reference word
  (begin specific move call)
```

If no address reference words are needed, the following call to entry point .CMPK3 is used:

```
TSX  .CMPK3
(begin specific move call)
```

This entry is used when the source field is in an arithmetic register and the result is to be left in an arithmetic register. Entry point .CMPK3 is also used when any necessary address references have been previously stored in locations .CAREF or .CBREF.

Address Reference Words

A call to any MOVPAK entry point except .CMPK3 is followed by one or more address reference words. The address reference words are in one of three forms, depending on the location of the data.

If the data is in working storage, the following coding is used:

```
PZE  location,,byte
```

where location is the address of the first word of the field involved in the move, and byte is the first byte position of the field involved.

If the data is in an IOCS buffer, the following coding is used for the address reference word:

```
MZE  BL+nnn,,SP+nnn
```

where BL+nnn is the base locator reference and SP+nnn is the displacement from the base. A base locator locates the first word of data in an IOCS buffer. The displacement of the data from the base is a constant. The word-displacement from the base is in the address of the constant, and the byte-displacement, if any, is in the decrement.

If the data is subscripted items in either working storage or an IOCS buffer, the following coding for the address reference is used:

```
MON  PI+nnn
```

where PI+nnn indicates the position of the data within the storage area. The positional indicator is calculated at execution time.

MOVPAK Subroutine Calls

After the call has been made to one of the four entry points of the MOVPAK subroutines, another call transfers control to a specific subroutine within MOVPAK. Usually, a TSX instruction transfers control to the entry point and a TXI instruction transfers control to the specific subroutine; specific exceptions will be noted in the following discussion. Although most MOVPAK subroutines may be called by any one of the four major entry points, subroutine .CEXAM is only called by CMPK2. This subroutine processes text strings created by the EXAMINE and IF (CLASS) analyzers. The calling sequence for this subroutine is:

```
TSX  .CMPK2,4
address reference word
TXI  entpt,1,n
```

where the address reference word corresponds to the description in the section "Address Reference Words" and entpt is one of the following entry points:

.CEXAM for true EXAMINE statements
.CXAMA for IF....ALPHABETIC statements
.CXAMN for IF....NUMERIC statements

and n is the length of the address reference word.

The following abbreviations are used in the discussions of the calls to the subroutines:

AA	Alphabetic field (The PICTURE clause contains only A's.)
AN	Alphanumeric field (The group item or PICTURE clause contains X's.)
CH	Characters (This category includes ALL, QUOTE and HIGH-VALUE figurative constants.)
FP	Floating point (floating-point binary)
ID	Internal decimal (fixed-point binary, synchronized right)
IN	Internal decimal not SYNCHRONIZED RIGHT
RP	Report field (The PICTURE clause contains editing characters.)
SD	Scientific decimal (floating-point BCD)
SP	SPACE (figurative constant)
XD	External decimal (fixed-point BCD)
ZE	ZERO (figurative constant)

The first two letters in the description of each type of move represent the type of data in the source field; the last two letters represent the type of data in the receiving field. For example, the move from internal decimal to external decimal is designated by the letters IDXD.

Literals are classified as alphanumeric, numeric, or floating point, as appropriate.

Those combinations of moves shown on the following pages that are legal are listed in the description of the MOVE verb in the COBOL manual.

AAAA, AAAN, ANAA, ANAN

Most moves of simple BCD alphanumeric or alphabetic fields are handled by generated in-line instructions if the fields are short. Other cases are handled by calls to one of several subroutines depending on the data. If an alphabetic or alphanumeric field is to be moved to another field that is also alphabetic or alphanumeric, the following call is used:

```
TXI  .CANA1,1,number
```

where number is the number of characters to be moved.

If the move involves the same type of data as described above, but also requires additional blank characters, the following sequence of calls is used:

```
TXI   .CANA2,1,number1
TXI   .CANA3,1,number2
```

where number1 is the number of characters to be moved, and number2 is the number of additional blank spaces to be inserted.

If the move involves alphabetic or alphanumeric information, and the initial byte position or word length of the source and/or target field is not known at compilation time, the following call is used:

```
TXI   .CANA4,1,decrement
PZE   control1,control2
```

where decrement defines the nature of control1 and control2 as follows:

DECREMENT VALUE	EXPLANATION
1	Control2 contains the length of the receiving field in words.
2	Control1 contains the length of the source field in words.
4	Control2 contains the address of the location containing the word length of the receiving field.
8	Control1 contains the address of the location containing the word length of the source field.

These conditions may exist in combination, giving the decrement a maximum value of 15.

CHAN

If characters are moved to an alphanumeric field, the following calling sequence is used:

```
TXI   .CHCAN,1,number
OCT   characters
```

where number is the number of characters to be inserted and characters is six characters of the type to be inserted.

FPAN

A move of a floating point item to an alphanumeric field is handled in the same manner as an alphanumeric or alphabetic move. The calling sequence used depends on the conditions described in the description of alphabetic moves (AAAA).

FPFP

A move involving floating point information is handled by generated in-line instructions. The MOVPAK subroutines are not used.

FPID

If a single-precision floating point item is to be converted to internal decimal, the following calling sequence is used:

```
TSX   .CF1ID,4
      receiving field control word
```

where the receiving field control word contains the following information:

prefix	PZE if the scaling factor to be used is positive; MZE if the scaling factor to be used is negative.
address	contains the scaling factor applied in the PICTURE clause of the internal decimal item.
decrement	contains the number of nines in the PICTURE clause of the internal decimal item. If the number of nines exceeds 10, the data is treated as double-precision data.

If double-precision data is used, the following calling sequence is used:

```
TSX   .CF2ID,4
      receiving field control word
```

where the receiving field control word contains the same information as described for single-precision data.

Subroutine .CF1ID (or .CF2ID) converts the floating-point number in the AC to an internal decimal number and leaves the result in the AC (or the AC-MQ). The calling sequence for these subroutines is a direct entry to MOVPAK and is not preceded by a TSX instruction to one of the four MOVPAK entry points.

FPIN

A conversion from floating point to internal decimal not synchronized right involves several intermediate steps. The floating point item is converted to internal decimal (FPID) and the resulting internal decimal item is converted to an internal decimal item with the same scaling factor as the desired result (IDID). The following calling sequence is then used to complete the conversion to an internal decimal item not synchronized right:

```
TXI   .CIDIN,1,number
```

where number is the character length of the receiving field, i.e., the least multiple of six bits needed to contain the desired internal decimal field and its sign.

FPRP

If a floating point item is to be moved to a report field, the data is first converted to internal decimal (FPID) and then moved to the report field. The calling sequence used is given in the description of a move from internal decimal to a report field (IDRP).

FPSD

If a single-precision floating point item is to be converted to scientific decimal, the following calling sequence is used:

```
TXI   .CF1SD,1,0
      receiving field control word
```

where the receiving field control word contains the following information about the PICTURE clause of the scientific decimal item:

prefix	PZE if no decimal point appears in the PICTURE clause; MZE if a decimal point appears in the PICTURE clause.
address	contains the scaling factor applied to the mantissa in the PICTURE clause.

tag	0 if the mantissa is negative and the exponent is negative; 1 if the mantissa is negative and the exponent is positive; 2 if the mantissa is positive and the exponent is negative; 3 if the mantissa is positive and the exponent is positive.
decrement	contains the total length of the receiving field in characters.

If the data is double-precision, the following calling sequence is used:

```
TXI   .CF2SD,1,0
      receiving field control word
```

where the receiving field control word contains the same information as described for single-precision data.

Subroutine .CF1SD (or .CF2SD) converts the floating-point number in the AC (or AC-MQ) to scientific decimal as specified in the PICTURE clause.

FPXD

A conversion from floating point to external decimal involves an intermediate conversion of floating point to internal decimal (FPID). The conversion is then completed as described for conversion from internal decimal to external decimal (IDXD).

IDAN

A move of an internal decimal item to an alphanumeric field is handled as described for a conversion from internal decimal to external decimal (IDXD).

IDFP

If a single-precision floating point item is to be converted to floating point, the following calling sequence is used:

```
TSX   .CIDF1,4
      source field control word
```

where the source field control word contains the following information:

prefix	PZE if the scaling factor is positive; MZE if the scaling factor is negative.
address	contains the scaling factor applied in the PICTURE clause of the internal decimal item.
decrement	contains the number of nines in the PICTURE clause of the internal decimal item. If the number of nines exceeds 10, the data is treated as double-precision data.

If double-precision data is involved in the move, the following calling sequence is used:

```
TSX   .CIDF2,4
      source field control word
```

where the source field control word contains the same information as described for single-precision data.

Subroutine .CIDF1 (or CIDF2) converts the internal decimal number in the AC (or AC-MQ) to floating point and leaves the result in the AC. The calling sequence to these subroutines is a direct entry to MOVPAK and is not preceded by a TSX instruction to one of the four MOVPAK entry points.

IDID

A move involving internal decimal data is usually handled by generated in-line scaling instructions. However, if scaling of an internal decimal item is used as an intermediate stage of a multistage move, the scaling function is performed by a MOVPAK subroutine, e.g., see IDSD.

IDIN

If an internal decimal item is to be converted to another internal decimal item not synchronized right, the following calling sequence is used:

```
TXI   .CIDIN,1,number
```

where number is the character length of the receiving field, i.e., the least multiple of six bits needed to contain the desired internal decimal item and its sign.

IDRP

If an internal decimal item is to be moved to a report field for editing, the following calling sequence is used:

```
TXI   .CIDRP,1,number
```

where number is the number of digits to be edited. This instruction is followed by one or more instructions from the report field instruction set. The instructions used depend on the characters that constitute the PICTURE clause for the report field. The members of the instruction set have the following general form:

```
TXI   entpt,1,number
```

where entpt is one of several entry points, and number is the number of consecutive occurrences of the character in the PICTURE clause. The entry point may be one of the following, depending on the character used:

.CR999	if the character in the PICTURE is a 9
.CRZZZ	if the character in the PICTURE is a Z
.CRAAA	if the character in the PICTURE is an *
.CR000	if the character in the PICTURE is a 0
.CRBBB	if the character in the PICTURE is a B

Another series of instructions inserts a character into the report field. In the following instruction, the particular character inserted depends on the sign of the report field:

```
TXI   .CRSIN,1,character1+64*character2
```

where character1 is inserted if the sign of the report field is plus, and character2 is inserted if the sign of the report field is minus.

NOTE: Symbols such as "character1" refer to the decimal representation of the desired BCD character. The BCD character is first converted to octal, and then to decimal before insertion. (See Example 1 which follows.)

In the following instruction, the particular character inserted depends on the insertion of a significant digit in the report field:

```
TXI   .CRSIG,1,character3+64*character4
```

where character3 is inserted if a preceding significant digit has been inserted, and character4 is inserted if a preceding significant digit has not been inserted. If the following instruction has been executed, and the floating-sign character has not been inserted, the character inserted in the report field may be character4, character5, or character6:

```
TXI   .CRFLS,1,character5+64*character6
```

where character5 is the floating-sign character ultimately inserted in the report field if the field is plus, and character6 is inserted if the field is minus. If the first digit value in the source field is zero, a blank or the appropriate choice of character5 or character6 is inserted as a result of this instruction. The first floating-sign position is passed over and not counted by this instruction.

The possible combination of character pairs is as follows:

CHARACTER NUMBER	CHARACTER PAIRS							
1	+	space	space	space	space	space	.	$
2	−	−	C	R	D	B	.	$
3	,	,	,					
4	,	space	*					
5	+	space	$					
6	−	−	$					

The following instruction is used when other floating-sign positions follow the position just filled:

```
TXI   .CRFFF,1,number
```

where number is the total number of consecutive floating-sign positions.

The following instruction is used when no other floating-sign positions follow the position just filled:

```
TXI   .CRFFQ,1,number
```

where number is the total number of consecutive floating-sign positions.

If no other floating-sign positions follow, but a comma precedes the next digit, the following instruction is used:

```
TXI   .CRFFC,1,number
```

where number is the total number of consecutive floating-sign positions.

The report field string is terminated by the following instruction:

```
TXI   .CRQIT,1,value
```

where value contains a zero if zero suppression is not desired, and contains the character length of the receiving field if zero suppression is desired.

The following examples show the series of instructions needed to handle PICTURE clauses for two report fields.

Example 1: PICTURE IS $$$, $$$, 99 is handled by the following instructions:

```
TXI   .CRFLS,1,43+64*43
          character5 and character6 are $
TXI   .CRFFF,1,2
TXI   .CRSIG,1,59+64*48
          character3 is a , and character4 is
          a space
TXI   .CRFFQ,1,3
TXI   .CRSIN,1,27+64*27
          character1 and character2 are .
TXI   .CR999,1,2
TXI   .CRQIT,1,0
```

Example 2: PICTURE IS ZZZ,ZZZ.ZZ+ is handled by the following:

```
TXI   .CRZZZ,1,3
TXI   .CRSIG,1,59+64*48
          character3 is a , and character4 is
          a space
TXI   .CRZZZ,1,3
TXI   .CRSIN,1,27+64*27
          character1 and character2 are .
TXI   .CR999,1,2
TXI   .CRSIN,1,16+64*32
          character1 is a + and character2
          is a −
TXI   .CRQIT,1,11
```

IDSD

If an internal decimal item in the accumulator or in the combined accumulator and multiplier-quotient is to be converted to a scientific decimal item, the following calling sequence is used:

```
TXI   .CIDSD,1,0
source field control word
receiving field control word
```

where the source field control word contains the following information:

prefix	PZE if the scaling factor is positive; MZE if the scaling factor is negative.
address	contains the scaling factor applied in the PICTURE clause of the internal decimal item.
decrement	contains the number of nines in the PICTURE clause of the internal decimal item. If the number of nines exceeds 10, the data is treated as double-precision data.

and the receiving field control word contains the following information about the PICTURE clause of the scientific decimal item:

prefix	PZE if no decimal point appears in the PICTURE clause; MZE if a decimal point appears in the PICTURE clause.
address	contains the scaling factor applied to the mantissa of the PICTURE clause.
tag	0 if the mantissa is negative and the exponent is negative; 1 if the mantissa is negative and the exponent is positive; 2 if the mantissa is positive and the exponent is negative; 3 if the mantissa is positive and the exponent is positive.
decrement	contains the total length of the receiving field in characters.

IDXD

The internal decimal item in the accumulator or in the combined accumulator and multiplier-quotient is converted to external decimal by one of the following three

calls, depending on the sign provision of the receiving field:

If the receiving field has no sign provision, the following calling sequence is used:

```
TXI  .CIDX1,1,number
```

where number is the number of characters to be converted.

If the receiving field always has a sign over the low-order bit, the following calling sequence is used:

```
TXI  .CIDX2,1,number
```

where number is the number of characters to be converted.

If the receiving field has a sign over the low-order bit when the sign is minus, the following calling sequence is used:

```
TXI  .CIDX4,1,number
```

where number is the number of characters to be converted.

INAN

An internal decimal item not synchronized right is converted to a synchronized internal decimal number (INID) before the conversion to alphanumeric is completed (IDAN).

INFP

An internal decimal item not synchronized right is first synchronized right (INID) before the item is converted to floating point (IDFP).

INID

If an internal decimal item not synchronized right is to be converted to a synchronized internal decimal item, the following calling sequence is used:

```
TXI  .CINID,1,number
```

where number is the character length of the source field, i.e., the least multiple of six bits needed to contain the defined internal decimal field and its sign. The results are left in the accumulator or in the combined accumulator and multiplier-quotient. The resulting internal decimal number is then scaled as desired (IDID).

ININ

If a move involves internal decimal items not synchronized right, the source field is first converted to a normal synchronized internal decimal item (INID) for decimal point alignment. The resulting item is converted to the desired internal decimal item not synchronized right (IDIN).

INRP

To move an internal decimal item that is not synchronized right to a report field, the source item is first converted to a synchronized internal decimal item (INID). The resulting internal decimal item is then moved to the report field (IDRP) and edited as desired.

INSD

If an internal decimal item that is not synchronized right is to be converted to a scientific decimal item, the source item is first converted to a synchronized internal decimal item (INID). The resulting internal decimal item is then converted to scientific decimal (IDSD).

INXD

If an internal decimal item that is not synchronized right is to be converted to an external decimal item, the source item is first converted to a synchronized internal decimal item (INID). The resulting internal decimal item is then converted to scientific decimal (IDXD).

RPAN

If an item in a report field is to be moved to an alphanumeric field, the data in the report field is treated as alphanumeric data (ANAN).

SDAN

If a scientific decimal item is to be converted to an alphanumeric item, the scientific decimal item is treated as an external decimal item (XDAN).

SPFP

If a single-precision scientific decimal item is to be converted to a floating point item, the following calling sequence is used:

```
TXI  .CSDF1,1,0
     source field control word
```

where the source field control word contains the following information about the PICTURE clause of the scientific decimal item:

prefix	PZE if no decimal point appears in the PICTURE clause; MZE if a decimal point appears in the PICTURE clause.
address	contains the scaling factor applied to the mantissa in the PICTURE clause.
tag	0 if the mantissa is negative and the exponent is negative; 1 if the mantissa is negative and the exponent is positive; 2 if the mantissa is positive and the exponent is negative; 3 if the mantissa is positive and the exponent is positive.
decrement	contains the total length of the receiving field in characters.

If the item is double-precision, the following calling sequence is used:

```
TXI  .CSDF2,1,0
     source field control word
```

where the source field control word contains the same information as described for a single-precision item.

Subroutine .CSDF1 (or .CSDF2) converts the free-form contents of the scientific decimal field to single-

precision floating point in the accumulator (or to double-precision floating point in the combined accumulator and multiplier-quotient).

SDID

If a scientific decimal item is to be converted to an internal decimal item, the following calling sequence is used:

```
TXI   .CSDID,1,0
source field control word
receiving field control word
```

where the source field control word contains the following information:

prefix	PZE if no decimal point appears in the PICTURE clause; MZE if a decimal point apears in the PICTURE clause.
address	contains the scaling factor applied to the mantissa in the PICTURE clause.
tag	0 if the mantissa is negative and the exponent is negative; 1 if the mantissa is negative and the exponent is positive; 2 if the mantissa is positive and the exponent is negative; 3 if the mantissa is positive and the exponent is positive.
decrement	contains the total length of the receiving field in characters.

and the receiving field control word contains the following information:

prefix	PZE if the scaling factor to be used is positive; MZE if the scaling factor to be used is negative.
address	contains the scaling factor applied in the PICTURE clause of the internal decimal item.
decrement	contains the number of nines in the PICTURE clause of the internal decimal item. If the number of nines exceeds 10, the data is treated as double-precision data.

Subroutine .CSDID converts the numeric contents of the scientific decimal field to internal decimal and leaves the result in the accumulator or in the combined accumulator and multiplier-quotient.

SDIN

If a scientific decimal item is to be converted to an internal decimal item not synchronized right, the source item is first converted to a normal internal decimal item (SDID). The resulting internal decimal item is then converted to an internal decimal item that is not synchronized right (IDIN).

SDRP

If a scientific decimal item is to be moved to a report field and edited, the source item is first converted to an internal decimal item (SDID). The resulting internal decimal item is then moved to the report field (IDRP) and edited as desired.

SDSD

If a scientific decimal item is to be converted to another scientific decimal item of a different form, the following calling sequence is used:

```
TXI   .CSDSD,1,0
source field control word
receiving field control word
```

where the source-field and receiving-field control words both contain the following information:

prefix	PZE if no decimal point appears in the PICTURE clause; MZE if a decimal point appears in the PICTURE clause.
address	contains the scaling factor applied to the mantissa in the PICTURE clause.
tag	0 if the mantissa is negative and the exponent is negative; 1 if the mantissa is negative and the exponent is positive; 2 if the mantissa is positive and the exponent is negative; 3 if the mantissa is positive and the exponent is positive.
decrement	contains the total length of the receiving field in characters.

SDXD

If a scientific decimal item is to be converted to an external decimal item, the source item is first converted to internal decimal (SDID). The resulting internal decimal item is then converted to external (IDXD).

SPAA, SPAN, SPRP, SPSD, SPXD

If spaces or blanks are to be moved to an alphabetic, alphanumeric, report, scientific decimal, or external decimal field, the following calling sequence is used:

```
TXI   .CSPAN,1,number
```

where number is the number of spaces to be inserted.

XDAN

If an external decimal item is to be moved to an alphanumeric field, the data is treated as alphanumeric data and the move is carried out accordingly (ANAN).

XDFP

If an external decimal item is to be converted to floating point, the source item is first converted to internal decimal (XDID). The resulting internal decimal item is then converted to floating point (IDFP).

XDID

If an external decimal item is to be converted to an internal decimal item, the source item is first converted to internal decimal without scaling and the result is left in the accumulator or in the combined accumulator and multiplier-quotient. The sign of the source field is assumed to be over the low-order digit. The absence of a sign is treated as a plus.

Leading spaces in the source field are treated as zeros. The following calling sequence is used to call the subroutine to perform the conversion:

```
TXI   .CXDID,1,number
```

where number is the number of characters needed to convert the data to internal decimal.

If scaling is needed, the internal decimal result is converted to another internal decimal item with the desired scaling (IDID).

XDIN

If an external decimal item is to be converted to internal decimal not right synchronized, the source item is first converted to internal decimal (XDID) right synchronized. The resulting internal decimal item is then scaled if necessary (IDID), and converted to an internal decimal item not synchronized (IDIN).

XDRP

If an external decimal item is to be moved to a report field and edited, the following calling sequence is used:

```
TXI   .CXDRP,1,0
```

followed by one or more instructions from the external decimal TXI instruction set. The particular instructions used represent a means of constructing the proper string of digits for the receiving field. The instructions in the external decimal set are explained in the section "XDXD."

After the series of instructions from the external decimal set, the following instruction is used:

```
TXI   .CXDRQ,1,number
```

where number is the number of digits in the string prepared for the receiving field by the external decimal subroutine set.

Once the source item has been prepared for the receiving field, another series of instructions from the report field instruction set is used. These instructions are explained in the section describing the move of an internal decimal item to a report field (IDRP).

The entire series of instructions for the move from external decimal to a report field is concluded by the following instruction:

```
TXI   .CRQIT,1,value
```

where value equals 0 if zero suppression is not wanted. If zero suppression is desired, value is the number of characters in the PICTURE clause for the report field.

XDSD

If an external decimal item is to be converted to a scientific decimal item, the source item is first converted to internal decimal (XDID). The resulting internal decimal item is then converted to scientific decimal (IDSD).

XDXD

If only external decimal items are involved in a move or conversion operation, the following calling sequence is used:

```
TXI   .CXDXD,1,sign
```

where sign is 0 if the receiving field has no sign provision; 1 if the receiving field has a sign over the low-order digit when the sign of the field is minus; and 2 if the receiving field always has a sign over the low-order digit.

This instruction is followed by one or more instructions from the external decimal instruction set to construct the proper string of digits for the receiving field. The instructions used depend on the operation to be performed in preparing the string. The instructions in this set have the following form:

```
TXI   entpt,1,number
```

where entpt is one of several entry points, and number is the number of digits involved in the desired operation. The entry point may be one of the following, depending on the operation to be performed:

.CXMOV	if the digits are to be moved
.CXNZT	if the digits are to be tested for nonzero
.CXBYP	if the digits are to be bypassed
.CXINZ	if the digits are to be inserted
.CXRND	if a rounding operation is to be performed at the current position (number for this entry point is zero)

If a nonzero condition is encountered while using subroutine .CXNZT, location .COFLO is set to nonzero. Three alternate entry points (.CXMVS, .CXNZS, and .CXBYS) corresponding to entry points .CXMOV, .CXNZT, and .CXBYP are used if a sign appears over the last digit involved in the operation.

The following instruction terminates the instruction set for this move:

```
TXI   .CXDXQ,1,number
```

where number is the number of digits in the string prepared for the receiving field. The following examples show coding necessary to handle two COBOL source program statements.

Example 1: The statement COMPUTE B ROUNDED=A, ON SIZE ERROR . . . (where A's PICTURE IS S999V999 and B's PICTURE IS S9V9) results in the following move-call coding:

```
TXI   .CXDXD,1,2
TXI   .CXNZT,1,2
TXI   .CXMOV,1,2
TXI   .CXRND,1,0
TXI   .CXBYS,1,2
TXI   .CXDXQ,1,2
```

Example 2: The statement MOVE A TO B (where A's PICTURE IS S9V99 and B's PICTURE IS V9999) results in the following move call-coding:

```
TXI   .CXDXD,1,0
TXI   .CXBYP,1,1
TXI   .CXMVS,1,2
TXI   .CXINZ,1,2
TXI   .CXDXQ,1,4
```

ZEAN

If zeros are to be moved to an alphanumeric field, the following calling sequence is used:

```
TXI   .CZEAN,1,number
```

where number is the number of zeros to insert in the field.

ZEFP, ZEID

If zeros are to be stored in a floating point or internal decimal item, MOVPAK subroutines are not used. The value zero is stored by one or more generated in-line instructions.

ZERP

If zeros are to be moved to a report field, the required number of zero digits is provided by generated in-line instructions. These zeros are placed in one or more temporary storage words. The move is then treated as a move from external decimal to a report field (XDRP).

ZESD

If zeros are to be inserted in a scientific item, the accumulator is cleared to zero by the following generated in-line instruction:

```
CLA  location
```

where location contains all zeros.

The move is then treated as a move from floating point to scientific decimal (FPSD).

ZEXD

If zeros are to be inserted in an external decimal item, the move is treated as a move of zeros to an alphameric field (ZEAN).

COBOL Input/Output Subroutines

A group of COBOL input/output subroutines are provided to service COBOL programs. Some of these subroutines use communication location .CIOHS to keep a history of input/output operations. Location .CIOHS is of the form:

```
PZE  0,0,**
```

where the decrement contains the source language card number of the most recent input/output operation. This location is set by every OPEN, CLOSE, READ, and WRITE statement in the source program.

All COBOL input/output subroutines except .CIOHS are described in the following text in alphabetical order according to the symbolic name of the subroutine. In the following descriptions, the notation CP+nnn is used to refer to a location in the constant pool. The information contained in this location is explained each time the notation is used.

.ACEPT

The .ACEPT subroutine accepts data from an on-line peripheral device. The calling sequence for this subroutine is:

```
TSX  .ACEPT,4
PZE  WS+nnn,,device
```

where WS+nnn is the location of the first word of a 12-word BCD input area in working storage and device is an actual bit configuration that may be either of the following:

Bit 16 = 1 if the peripheral device is the system input unit.
Bit 17 = 1 if the peripheral device is the card reader.

.CBLER

Subroutine .CBLER is called if a base locator that has not been set is referred to. An error message is written on the system output unit, a core storage dump is taken, and execution of the object program is terminated. The calling sequence for this subroutine varies.

.CCDTY

Subroutine .CCDTY determines if card equipment is to be used for an input/output operation. The calling sequence for this subroutine is:

```
TSX  .CCDTY,4
PZE  file-name
(return1)
(return2)
```

where file-name is the name of the file used in the required operation, return1 is used if card equipment is to be used, and return2 is used if card equipment is not to be used.

.CCLOS

Subroutine .CCLOS closes input/output files. The calling sequence for this subroutine is:

```
TSX  .CCLOS,4
rop  file-name,,opt
```

where file-name is the name of the file to be closed, rop is the rewind option and may be one of the following:

```
PZE  Rewind and unload
PTW  Rewind
MZE  No rewind
MON  No rewind, no end-of-file mark
```

and opt describes the type of file and indicates the CLOSE REEL option as follows:

opt = 0 if the file-name does not refer to an optional file and the CLOSE REEL option is not wanted.
= 1 if the file-name refers to an optional file.
= 2 if the CLOSE REEL option is desired.
= 3 if the file-name refers to an optional file and the CLOSE REEL option is desired.

.CDISP

Subroutine .CDISP is used to display any number of data items on either the printer or the system output unit. The calling sequence for this subroutine is:

```
CAL  SP+nnn      } Stores over CALL
SLW  GNxxx+m+2   } parameters where
                 } the starting byte
                 } does not=0.
```

```
GNxxx    CALL  .CDISP,4
         TXI   *+2+n,,n
         PZE   device,,linkage director
         PZE   location1,,byte1
         PZE   location2,,byte2
          .
          .
          .
         PZE   locationm,,bytem
          .
          .
          .
         PZE   locationn,,byten
         TRA   *+1+n
         PZE   length1,,format1
         PZE   length2,,format2
          .
          .
          .
         PZE   lengthn,,formatn
```

where:

n is the number of items to be displayed; n can vary from m to l.

device is one of the following:

0 if the items are to be displayed on the printer

1 if the items are to be displayed on the system output unit

$location_i$ is the first-word address of the items or area to be displayed.

$byte_i$ is the byte displacement for the first item of the data to be displayed.

$length_i$ is either the length of the item in characters or the address of the location containing the length, depending on the format code.

$format_i$ is one of the following:

0 for alphabetic or alphameric items where $length_i$ is the length.

1 for external decimal, scientific decimal, or report items where $length_i$ is the length.

2 for internal decimal items where $length_i$ is a maximum of 18 characters; if more than 18 characters are specified, the item is truncated to 18 characters.

3 for floating point items where $length_i$ is the length.

4 for BCD items where $length_i$ is the address of a location containing the length of the item in characters.

5 for BCD items where $length_i$ is the address of a location containing the length of the item in words.

The first-word address and byte displacement may have to be determined and placed in the calling sequence at execution time via the CLA and SLW instructions.

.CDPLY

Subroutine .CDPLY is used to display a 12-word BCD image. The calling sequence for this subroutine is:

```
TSX  .CDPLY,4
PZE  image,,device
(Return)
```

where image is the address of the area to be displayed, and device is an actual bit configuration that may be one of the following:

Bit 15=1 if the area is to be displayed on the printer.

Bit 14=1 if the area is to be displayed on the system output unit.

Bit 13=1 if the area is to be displayed on the card punch. This option is not currently implemented.

.CEOBP

Subroutine .CEOBP is associated with an IOCS .READ or .WRITE calling sequence. Control is transferred to subroutine .CEOBP when the IOCS end-of-buffer error condition is encountered. This subroutine writes an error message and causes execution of the object program to be terminated.

.CERRP

Subroutine .CERRP is associated with an IOCS .READ calling sequence. Control is normally transferred to subroutine .CERRP when the IOCS error condition is encountered during a .READ calling sequence. This subroutine writes an error message and causes execution of the object program to be terminated.

.CEXNG

Subroutine .CEXNG is an error routine, called only by .CAR01, .CAR02, .CAR13, or CAR14 when the factors in an exponentiation operation are out of range. This subroutine prints an off-line error message indicating the location of the error within the program. The calling sequence for this subroutine is:

```
TSX  .CEXNG,4
PZE  CP+nnn
```

where CP+nnn specifies the location containing the number of the source language card that specified the original computation.

.CEXPR

Subroutine .CEXPR is called only by .CAR01, .CAR02, .CAR13, or .CAR14 to determine if the factors in an exponentiation operation are out of range. If they are, the error routine .CXNG is used; if they are not out of range, the execution of the object program continues. The calling sequence for this subroutine is:

```
TSX  .CEXPR,4
PZE  CP+nnn
```

where CP+nnn specifies the location containing the number of the source language card that specified the original computation.

.CKEYS

Subroutine .CKEYS places the setting of the entry keys in the multiplier-quotient register. The calling sequence for this subroutine is:

```
TSX   .CKEYS,4
(Return)
```

.COPEN

Subroutine .COPEN opens input/output files. The calling sequence for this subroutine is:

```
TSX   .COPEN,4
rop   filename, , opt
```

where filename is the name of the file to be opened, rop is the rewind option and may be one of the following:

```
PZE   Rewind
MZE   No rewind
MON   No rewind, no label action
```

and opt equals 1 if the file name refers to an optional file.

Additional COBOL Subroutines

COBOL subroutines are also provided to perform additional conversion operations, mathematical computations, and alphabetic comparisons. These subroutines use two locations for temporary storage of data used by the subroutines. Location .CARS1 is used if the data is single-precision, and locations .CARS1 and .CARS2 are used if the data is double-precision.

These subroutines are listed in alphabetical order according to the symbolic name of the subroutine. In the following descriptions, the notation CP+nnn is used to refer to a location in the constant pool. The information contained in this location is explained each time the notation is used.

.CAR01

Subroutine .CAR01 raises the double-precision floating-point number in the combined accumulator and multiplier-quotient to the single-precision power in .CARS1. The double-precision floating-point result is in the combined accumulator and multiplier-quotient. The calling sequence for this subroutine is:

```
TSX   .CAR01
PZE   CP+nnn
```

where CP+nnn is a location in the constant pool. This location contains the source language card number at which the original computation was specified. This constant is used by error routine .CEXNG only if the factors in the exponentiation are out of range.

.CAR02

Subroutine .CAR02 raises the double-precision floating-point number in the combined accumulator and multiplier-quotient to the double-precision power in .CARS1 and .CARS2. The double-precision floating-point result is in the combined accumulator and multiplier-quotient. The calling sequence for this subroutine is:

```
TSX   .CAR02,4
PZE   CP+nnn
```

where CP+nnn specifies the location containing the number of the source language card that specified the original computation. This information is used only in case of error.

.CAR03

Subroutine .CAR03 adjusts the sign of a double-precision fixed-point number located in the combined accumulator and multiplier-quotient. The result is left in the combined accumulator and multiplier-quotient. The calling sequence for this subroutine is:

```
TSX   .CAR03,4
```

.CAR04

Subroutine .CAR04 scales up the single-precision number in the accumulator by $1x10^{10}$ and then scales up the result by a constant located in the constant pool. The calling sequence for this subroutine is:

```
TSX   .CAR04,4
PZE   CP+nnn
```

where CP+nnn specifies the location containing the second scaling factor.

.CAR05

Subroutine .CAR05 scales up the number in the multiplier-quotient by $1x10^{10}$ and then scales up the result by a constant located in the constant pool. The calling sequence for this subroutine is:

```
TSX   .CAR05,4
PZE   CP+nnn
```

where CP+nnn specifies the location containing the second scaling factor.

.CAR06

Subroutine .CAR06 scales up the double-precision number in the combined accumulator and multiplier-quotient by a constant located in the constant pool. Upon entry to this subroutine, the high-order part of the number is in the accumulator and the low-order part of the numbers is in the multiplier-quotient. The calling sequence for this subroutine is:

```
TSX   .CAR06,4
PZE   CP+nnn
```

where CP+nnn specifies the location containing the scaling factor.

.CAR07

Subroutine .CAR07 scales up the double-precision number in the combined accumulator and multiplier-quotient by a constant in the constant pool. Upon entry

to this subroutine, the high-order part of the number is in the multiplier-quotient and the low-order part of the number is in the accumulator. The calling sequence for this subroutine is:

```
TSX  .CAR07,4
PZE  CP+nnn
```

where CP+nnn specifies the location containing the scaling factor.

.CAR08

Subroutine .CAR08 scales down the double-precision number in the combined accumulator and multiplier-quotient by a constant. The result is in the combined accumulator and multiplier-quotient. The calling sequence for this subroutine is:

```
TSX  .CAR08,4
PZE  CP+nnn
```

where CP+nnn specifies the location containing the scaling factor.

.CAR09

Subroutine .CAR09 scales down the double-precision number in the combined accumulator and multiplier-quotient by $1x10^{10}$ and then scales down the result by a constant. The result is in the multiplier-quotient. The calling sequence for this subroutine is:

```
TSX  .CAR09,4
PZE  CP+nnn
```

where CP+nnn specifies the location containing the scaling factor.

.CAR10

Subroutine .CAR10 divides the double-precision fixed-point number in locations .CARS1 and .CARS2 by the double-precision number in the combined accumulator and multiplier-quotient. The result is in the combined accumulator and multiplier-quotient. The calling sequence for this subroutine is:

```
TSX  .CAR10,4
```

.CAR11

Subroutine .CAR11 multiplies the double-precision number in the combined accumulator and multiplier-quotient by the double-precision number in locations .CARS1 and .CARS2. This subroutine also scales down the result by a constant and leaves the final result in the combined accumulator and multiplier-quotient. The calling sequence for this subroutine is:

```
TSX  .CAR11,4
PZE  CP+nnn
```

where CP+nnn specifies the location containing the scaling factor used after the multiplication.

.CAR12

Subroutine .CAR12 multiplies the double-precision number in the combined accumulator and multiplier-quotient by the double-precision number in locations .CARS1 and .CARS2. The result is scaled down by $1x10^{10}$, and then by a constant. The final result is in the combined accumulator and multiplier-quotient. The calling sequence for this subroutine is:

```
TSX  .CAR12,4
PZE  CP+nnn
```

where CP+nnn specifies the location containing the scaling factor.

.CAR13

Subroutine .CAR13 is a floating-point exponential routine. The single-precision floating-point number in the accumulator is raised to the single-precision floating-point power in location .CARS1. The base number in the accumulator may be a positive real number or a negative integer. The exponent in location .CARS1 may have any value. The result is a single-precision floating-point number in the accumulator. However, the result is limited to an accuracy of seven significant digits. If an accuracy of eight or more significant digits is desired, the double-precision exponentiation subroutine .CAR14 must be used. Subroutine .CAR14 is called when a picture of nine or more decimal digits is specified for either the base number or the exponent. Subroutine .CAR13 calls subroutine .CEXPR to determine if the result exceeds seven significant digits, and if it does, calls subroutine .CEXNG to indicate the error. The calling sequence for subroutine .CAR13 is:

```
TSX  .CAR13,4
PZE  CP+nnn
```

where CP+nnn specifies the location containing the number of the source language card that specified the exponential operation. This information is used in case of an error.

.CAR14

Subroutine .CAR14 is an exponential routine that uses either single-precision numbers or double-precision floating-point numbers as the base and exponent. The number located in the accumulator or combined accumulator and multiplier-quotient is raised to the power in location .CARS1 (or location .CARS1 and .CARS2). This subroutine is called whenever either the base or the exponent is a double-precision number. The result is a double-precision number in the combined accumulator and multiplier-quotient. Subroutine .CAR14 calls subroutine .CEXPR, and in case of an error, calls subroutine .CEXNG. The calling sequence for subroutine .CAR14 is:

```
TSX  .CAR14,4
PZE  CP+nnn
```

where CP+nnn specifies the location containing the number of the source language card which specified the exponential operation. This information is used in case of an error.

.CAR15

Subroutine .CAR15 converts the single-precision number in the accumulator to a double-precision number in the combined accumulator and multiplier-quotient. The calling sequence for this subroutine is:

```
TSX  .CAR15,4
```

.CBCDH

Subroutine .CBCDH converts a 12-word BCD image to a card image. The calling sequence for the subroutine is:

```
TSX  .CBCDH,4
PZE  BL+nnn, , TS+nnn
```

where TS+nnn is the location of the first word of a 12-word temporary storage area containing the BCD image and BL+nnn is the base locator for the storage area that will contain the card image.

.CBDCV

Subroutine .CBDCV converts the BCD control word that precedes each variable-length record to binary form. This control word contains the length of the record in words. Upon entering this subroutine, the control word is in the multiplier-quotient. After conversion, the record length is in binary form in the decrement of the accumulator. The calling sequence for this subroutine is:

```
TSX  .CBDCV,4
(Return)
```

.CBNCV

Subroutine .CBNCV converts the length of a variable-length record from binary to BCD form. Upon entering this subroutine, the record length in number of words is in the multiplier-quotient in binary form. The length is converted to BCD form and placed in the first five characters of a control word, which is added to the beginning of the variable-length record. The sixth character of the control word is zero. After conversion, the control word is in the logical accumulator. The calling sequence for this subroutine is:

```
TSX  .CBNCV,4
(Return)
```

.CCOMP

Subroutine .CCOMP performs an alphabetic comparison of two fields. The calling sequence for this subroutine is:

```
TSX  .CCOMP,4
op   .CCTAB,,6
PZE  loc1,t1,locator
PZE  length1,,6*byte1
PZE  loc2,t2,locator2
PZE  length2,,6*byte2
     (high return from comparison)
     (equal return from comparison)
     (low return from comparison)
```

where op is CVR or NOP, depending on the need to adjust the collating sequence before the comparison.

loc_i is the displacement from the base, if any. If there is no base, loc_i is the location of the field.

$locator_i$ is the location of the base locator. If there is no base locator, $locator_i$ is zero.

t_i is nonzero if the base locator is complex. If there is no base locator, t_i is zero.

$length_i$ is the length of the field in characters.

$byte_i$ is the nominal byte position.

.CCTAB

.CCTAB is a conversion table used in converting from the 7090 scientific collating sequence to the COLLATE-COMMERCIAL collating sequence used in COBOL. This table is used by subroutine .CCOMP if it is necessary to adjust the collating sequence before performing an alphabetic comparison of two fields.

.CGOGO

Subroutine .CGOGO provides access to FORTRAN mathematical subroutines from a COBOL source program. It is called by one of 32 access subroutines, depending on which FORTRAN subroutine is desired. There is one access subroutine for each FORTRAN subroutine as shown in Figure 54. These 32 access subroutines are called by a COBOL source language statement of the form:

```
CALL 'COBOL-entry-point' USING R X Y.
```

where COBOL-entry-point is the entry point to one of the 32 access subroutines and R, X, and Y are the parameters of an equation of the form R = f(X, Y). (If the equation is of the form R = f(X), the third parameter is not needed.)

The access subroutines load the accumulator with an indicator word before transferring control to subroutine .CGOGO. The actual entry to and return from the FORTRAN subroutine is performed from subroutine .CGOGO. Return from .CGOGO is to the original program, not the access subroutine. These access subroutines are in the following form:

```
xxxxxx   CLA    *+2
         TRA    .CGOGO
         opcode yyyyyy
```

where:

xxxxxx is the entry point to the access subroutine.

opcode is TRA if single-precision parameters are expected as output from the FORTRAN subroutine, or NOP if double-precision parameters are expected:

yyyyyy is the desired FORTRAN entry point.

FORTRAN Entry Point	COBOL Entry Point
.XP1.	.CXP1
.XP2.	.CXP2
.XP3.	.CXP3
.DXP1.	.CDXP1
.DXP2.	.CDXP2
.CXP1.	.CCXP1
EXP	.CEXP
DEXP	.CDEXP
CEXP	.CCEXP
ALOG	.CALOG
DLOG	.CDLOG
CLOG	.CCLOG
ALOG10	.CAL10
DLOG10	.CDL10
ATAN	.CATAN
DATAN	.CDATN
ATAN2	.CATN2
DATAN2	.CDAT2
SIN	.CSIN
DSIN	.CDSIN
CSIN	.CCSIN
COS	.CCOS
DCOS	.CDCOS
CCOS	.CCCOS
TANH	.CTANH
SQRT	.CSQRT
DSQRT	.CDSQR
CSQRT	.CCSQR
DMOD	.CDMOD
.CABS.	.CCABS
.CFMP.	.CCFMP
.CFDP.	.CCFDP

Figure 54. Correspondence Between the Entry Point to the FORTRAN Subroutine and the Entry Point to the COBOL Access Subroutine

.CHBCD

Subroutine .CHBCD converts a card image to a 12-word BCD image. The calling sequence for this subroutine is:

```
TSX  .CHBCD,4
PZE  .BL+nnn,,TS+nnn
```

where BL+nnn specifies the area containing the card image and TS+nnn is the first address of a 12-word temporary storage area that will contain the BCD image.

The Librarian is a section of the Loader used to maintain the Subroutine Library. By using the Librarian, subroutines can be replaced in, added to, and deleted from the Library. Any subroutines added to the Library must first be assembled by the Macro Assembly Program.

Subroutine Library Maintenance

The Subroutine Library consists of the following two files of information:

1. The control information file, consisting of the subroutine section-name table; the subroutine dependence table, and the Loader control cards, control dictionaries, and file dictionaries, if they exist.
2. The relocatable binary text file, consisting of the text portions of those library subroutines that have text.

The Library maintenance operation is essentially a four-phase process. The name table and the dependence table are skipped, and not used by the Librarian. Librarian control cards and subroutine decks are obtained from the input file. Appropriate positioning, replacements, insertions, or deletions are made in the control information file and the new control information file is formed and written on a work file.

The relocatable binary text portions of the subroutine decks obtained from the input file are written on a second work file. The operation and subroutine name from each Librarian control card are preserved in the Librarian action table to be used in processing the relocatable binary text file. At the completion of phase 1, the complete control information file has been formed.

In phase 2, the relocatable binary text portions of the subroutines obtained from the input file are merged with the existing text file, and the new relocatable binary text file is written behind the new control information file.

In phase 3, the combined control dictionaries of the library subroutines are used to generate the new subroutine section name table and the subroutine section dependence table. If the programmer so specifies on the $EDIT card (see below), a listing is prepared showing the subroutine name, the origin, if fixed, the length, and the starting record number of each subroutine. A list showing all real control sections and their dependencies is also produced.

Phase 4 consists of writing the subroutine section name table and subroutine section dependence table on the system utility unit (SYSUT4), followed by the control information and text files.

Subroutine Library maintenance is now complete, and the Librarian returns control to the Loader. A system edit is now necessary to replace the existing library files by the two new Library files generated by the Librarian on the system utility unit (SYSUT4).

NOTE: Subroutine Library maintenance normally follows an update of the Library symbolic input tape (see *IBM 7090/7094 IBSYS Operating System: Symbolic Update Program*, Form C28-6386). It can also be accomplished without symbolic update by using the ALTER feature of IBJOB (see "Altering an Input Deck").

Librarian Control Cards

The following control cards are necessary for the use of the Librarian:

$EDIT Card

The $EDIT card must follow the Processor control card ($IBJOB card). The $EDIT card causes the Librarian to be called by the Load Supervisor.

The format of this control card is:

1	16
$EDIT	[LOGIC]

If the LOGIC option is specified, the Librarian provides information showing the cross-referencing of subroutines in the Library.

$REPLACE Card

The $REPLACE card causes a subroutine in the Subroutine Library to be replaced.

The format of this control card is:

1	16
$REPLACE	srname [, ORG=nnnnn]

The current Library is copied up to, but not including, the subroutine name symbolized by srname. The named subroutine is then skipped in the current Library and the subroutine deck following the $REPLACE card is inserted in its place in the output Library files. If the subroutine deck is in MAP source language and must be assembled, it must be headed by a $IBMAP card. If it has already been assembled and is in relocatable binary form, the deck must be headed by a $IBLDR card.

The optional field ORG = nnnnn is used to assign an absolute origin to the subroutine that is being inserted

or to change its assembled absolute origin. The five-digit field nnnnn is the absolute origin, in octal.

$ASSIGN Card

The $ASSIGN card causes a subroutine to be assigned an absolute origin before it is placed in the Subroutine Library.

The format of this control card is:

1	16
$ASSIGN	srname, ORG=nnnnn

The Librarian copies the current Library up to, but not including, the subroutine name symbolized by srname. The named subroutine is then assigned the absolute origin specified by the octal number nnnnn and the subroutine is placed in the output library files.

Both the subroutine named and the origin to be assigned are mandatory on this control card.

$INSERT Card

The $INSERT card causes the Librarian to write a subroutine deck onto the Library file. The card must precede the subroutine deck. The deck is written onto the file, starting with the current position of the file.

If the subroutine deck is in MAP source language and must be assembled, it must be headed by a $IBMAP card. If it has already been assembled and is in relocatable binary form, the deck must be headed by a $IBLDR card.

The format of this control card is:

1	16
$INSERT	[srname] [, ORG=nnnnn]

The field srname is optional on this control card and will not be used by the Librarian. The optional field ORG = nnnnn is used to assign an absolute origin to the subroutine being inserted.

It should be noted that positioning is not performed with the $INSERT card; the insertion is made at the current position of the output Library file.

$AFTER Card

The $AFTER card is used to position the library file.

The format of this control card is:

1	16
$AFTER	srname

The $AFTER card causes the Librarian to copy the Library from its current position through the subroutine name symbolized by srname. The subroutine name is mandatory on this control card.

The $AFTER card is used in conjunction with the $INSERT card to position the file before inserting.

$DELETE Card

The $DELETE card causes subroutines to be removed from the Subroutine Library.

The format of this control card is:

1	16
$DELETE	srname

The $DELETE card causes the Librarian to copy the Library from its current position up to, but not including, the subroutine symbolized by srname. The named subroutine is then skipped on the current Library. The subroutine name is mandatory on this control card.

Restrictions Using Disk

If the Subroutine Library is to reside on disk storage, the system utility unit (SYSUT4) must be attached to the disk to obtain the proper block size. If SYSUT4 is not attached to disk, SYSUT3 may not be attached to disk.

Restrictions Using Drum

The block size of the Subroutine Library is determined by the Loader assembly parameter DRUM. If DRUM = 0, the block size is 464 words; if DRUM = 1, the block size is 524 words. As released, DRUM = 0. The entire Loader must be reassembled to set DRUM = 1. (See "System Library Preparation and Maintenance" in the publication *IBM 7090/7094 IBSYS Operating System: System Monitor (IBSYS)*, Form C28-6248.)

PART 3: IBJOB PROCESSOR ERROR MESSAGES

The following section lists the messages generated by the IBJOB Processor. The messages are arranged by component as follows: Monitor, FORTRAN IV Compiler, COBOL Compiler, Assembler, Load-Time Debugging Processor, Loader, and Subroutine Library.

To find the explanation for a message, the programmer should consider the sequence of Processor operations as indicated by the printed assembly listing. For example, a message printed on a listing after a $IBFTC card and before the next component control card has been generated by the FORTRAN IV Compiler. A message printed after a $IBLDR card has been generated by the Loader.

The following section lists, in alphabetical order, the messages generated by the IBJOB Monitor. The symbol "******" indicates a location in the error message where the Compiler inserts a variable word. Where such a word is the first in the message, the message is listed alphabetically by the next nonvariable word.

ABSMOD ASSEMBLIES CANNOT BE LOADED.
This message occurs when the ABSMOD option is encountered on a $IBMAP card, and loading is requested. A systems error is indicated.

ACTION LABEL INCORRECT.
This message occurs if an argument to the Action routine, sent by some part of the system to cause positioning or reading of the system unit, does not match any action table entries. A systems error is indicated.

ALTER FIELD ERROR, CARD AND INSERTIONS IGNORED.
This message occurs when a comma or blank is encountered in column 16 of an *ALTER control card or when a field is written as follows: (A*ALTER B,,) or (A*ALTER B,).

ALTER FIELD ERROR, COMMA TREATED AS BLANK.
This message occurs when a comma is encountered in columns 1-6 of an *ALTER control card or when a field is written as follows: (A*ALTER B ,C,). In the latter case, the last comma is treated as a blank.

ASSEMBLY DELETED.
This message occurs if an error in compilation has occurred such that assembly cannot be attempted.

BINARY RECORD(S) ENCOUNTERED WHILE SEARCHING FOR CARDS.
This message occurs when binary records are encountered by the IBJOB Monitor outside the limits of an object deck.

****** CARD WITH CORRECT DECK NAME NOT FOUND.
This message occurs when alternate input is requested and the deck requested cannot be found.

DUMP TABLE HAS OVERFLOWED.
This message occurs if more than 29 table words have been generated because of $DUMP cards. Table words are generated as follows: one word for each $DUMP card and an additional word for each set of dump limits on the card.

EOB OR EOT CONDITION. DECK CANNOT BE PROCESSED.
This message occurs when an indication of end of buffer or end of tape occurs while transferring Alter cards to the system utility unit (SYSUT2).

EOF OR REDUNDANCY IN ALTER FILE.
This message occurs when an end of file or redundancy is encountered by the Alter routine while trying to read Alter cards from the system input unit or the system utility unit (SYSUT2).

EOT ON INTERMEDIATE UNIT OR EOB EXIT. ERROR CONDITION.
This message occurs if the system input/output editor, while trying to write on an input/output unit, receives a signal from IOCS that an end-of-buffer condition exists. If the unit is 1301 Disk Storage, the condition is caused by exceeding the cylinder limits specified for the system function.

ERROR IN ALTER DECK.
This message occurs when a deck has been altered and an error detected (unused Alter cards, an end of file or redundancy while reading Alter cards, or a scan error in an Alter card).

ERROR READING OBJECT DECK.
This message occurs when a redundancy on the system input unit occurs while transferring an object deck to the load file.

****** HAS NO UNIT ASSIGNED. CANNOT PROCEED.
This message occurs if the system output unit or the system input unit has no unit assigned when the IBJOB Processor gains control.

****** HAS NO UNIT ASSIGNED. RESTRICTED USAGE OF IBJOB IS POSSIBLE.
This message occurs if the system peripheral punch or one of the system utility units SYSUT1, SYSUT2, SYSUT3, or SYSUT4 has no unit assigned when the IBJOB Processor gains control.

IBJOB SYSTEM SPLIT BETWEEN TWO CHANNELS IS ILLEGAL PROCEED TO NEXT JOB.
This message occurs when the IBJOB Processor is split into two units, and they are mounted on two different channels.

IBJOB VERSION ***** HAS CONTROL.
This message occurs each time the IBJOB Processor gains control from the System Monitor.

ILLEGAL BCD DATE IN BASIC MONITOR DATE CELL. ENTER CURRENT DATE IN KEYS (MMDDYY) AND HIT START.
This message occurs in IBJOB initialization if location SYSDAT in the System Monitor does not contain a valid date. The operator should enter the current date, in BCD, into the keys and press START.

INCORRECT DECK SET-UP
The IBJOB Monitor has encountered a component control card or an unrecognized card with a $ in column 1 either (1) without having processed a $IBJOB card for the current Processor application or (2) while still in control after the completion of a Processor application. This message will not occur when the IBJOB Monitor encounters unrecognized control cards after a $IBJOB card for the current Processor application has been processed.

INCORRECT DECK SET-UP, EOF ENCOUNTERED BEFORE *ENDAL.
This message occurs when Alter cards on the system input unit are being transferred to SYSUT2, and an end of file is encountered before an *ENDAL card.

MACHINE OR SYSTEM FAILURE HAS OCCURRED. RETRY IS IMPOSSIBLE. THIS JOB WILL BE CONTINUED.

This message occurs when (1) machine or system failure is detected and (2) the system input unit is a card reader or an end-of-tape was encountered during the job.

MACHINE OR SYSTEM FAILURE HAS OCCURRED. TO RETRY THIS P/A, PRESS START. TO CONTINUE THIS P/A, PRESS START WITH KEY "S" DOWN. TO DELETE THIS P/A, PRESS START WITH KEY "1" DOWN.

Self-explanatory.

NO PROCESSING THIS P/A.

This message occurs when a Processor application consists of a $IBLDR card with the LIBE option, or contains no decks at all.

ON-LINE PRINTER AND PUNCH MAY NOT BE ATTACHED AS SYSOU1 AND SYSPP1. CANNOT PROCEED.

This message occurs if either the system output unit or the system peripheral punch has been assigned to on-line equipment.

ONLY SYSIN1, SYSOU1, SYSPP1 WILL BE REPOSITIONED FOR RETRY.

This message occurs if machine or system failure is detected by some portion of the IBJOB Processor.

PATCH TABLE HAS OVERFLOWED.

This message occurs if more than 50 table words have been generated because of $PATCH cards. Table words are generated as follows: one word is necessary for each $PATCH card and an additional word is necessary for each patch word on the card.

PERMANENT REDUNDANCY OR EOT WHILE READING ALTERNATE UNIT.

Self-explanatory.

PERMANENT REDUNDANCY WHILE READING CONTROL CARDS. THIS P/A CANNOT BE CONTINUED.

Self-explanatory.

PREST CARD CKSUM ERROR, SEQUENCE NUMBER *****.

This message occurs if, while processing a Prest deck, a check-sum error is detected. Processing will continue. Execution is prevented.

PREST CARD FIELD ERROR. SEQUENCE NUMBER *****.

This message occurs if, while processing a Prest deck, an error is detected in the field or string count. Processing continues. Execution is prevented.

PREST CARD SEQ ERROR. SEQUENCE NUMBER *****.

This message occurs if, while processing a Prest deck, an error is detected in the sequence of cards. Processing continues. Execution is prevented.

REDUNDANCY WHILE READING ALTER CARDS. THIS DECK CANNOT BE PROCESSED.

This message occurs when a read redundancy occurs while moving Alter cards to the system utility unit (SYSUT2).

REMAINDER OF JOB DELETED.

This message occurs if the option to delete the remainder of the job is chosen after machine or system failure has occurred.

RETURNING TO IBSYS.

This message occurs when the IBJOB Processor is returning control to the System Monitor.

SCHF OPTION INVALID IF SYSINx IS DISK. PROCEEDING TO NEXT P/A.

This message occurs when the "search option" (SCHFn) is requested on a $IEDIT card and the system unit function requested for the search is assigned to disk.

SYSxxx IN USE. PROCEEDING TO NEXT P/A.

This message occurs when SYSxxx has been requested on either a $IEDIT or $OEDIT card, and it is currently in use due to a previous $OEDIT or $IEDIT card.

SYSxxx NOT ASSIGNED. PROCEEDING TO NEXT P/A.

This message occurs when SYSxxx has been requested on either a $IEDIT or $OEDIT card and no unit is assigned to the function.

THIS DECK CONTROL CARD CANNOT BE PROCESSED. IT MUST APPEAR IN TABLE SSTAB.

This message occurs when a recognized subsystem control card is encountered and the SYSTM routine is not set up properly for the subsystem.

THIS JOB WILL BE CONTINUED.

This message occurs if the option to continue is chosen after machine or system failure has occurred.

UNIT ****** EOT OR EOB EXIT.

This message occurs if the system input/output editor, while trying to write on an input/output unit, receives a signal from IOCS that an end-of-buffer condition exists. If the unit is 1301/2302 Disk Storage, the condition is caused by exceeding the cylinder limits specified for the system function.

UNRECOGNIZED OPTION ON ABOVE CARD.

This message occurs when an unrecognized option is encountered on a $IEDIT or $OEDIT control card. The standard option is used.

The following alphabetic list contains the error messages generated by the FORTRAN IV Compiler and their explanations where necessary. Words in a message that must vary from situation to situation are denoted by "*****". Where asterisks actually appear as a standard part of a message, the condition is specifically noted.

Messages which begin with variable words are alphabetized according to the first word following the variable quantity. Messages which cannot be located alphabetically by the first word should be sought according to the second word of the message.

A DO ENDING IS MISSING OR HAS OCCURRED BEFORE THE DO ITSELF.
Self-explanatory.

A DO ENDING ***** IS MISSING OR IS A NON EXECUTABLE STATEMENT.
Violation of FORTRAN requirements for DO statement. Provide ending for DO; provide an executable statement for end of DO.

A DO ENDING IS A NON EXECUTABLE STATEMENT, OR ANOTHER DO, OR A TRANSFER.
Violation of FORTRAN requirements for DO statement. The DO statement must be terminated correctly.

A DOUBLE PRECISION OR COMPLEX VARIABLE ***** IS BROUGHT INTO COMMON BY AN EQUIVALENCE STATEMENT BUT DOES NOT LIE AN EVEN NUMBER OF LOCATIONS FROM BEGINNING OF THAT BLOCK.
Equivalence statement must be changed so that when the variable is brought in, it is an even number of positions from the head of the block. Each double-precision and complex variable occupies two consecutive positions of core storage.

A FORMAT ***** HAS BEEN ILLEGALLY REFERENCED BY A GO TO OR AN ASSIGN.
FORTRAN does not allow a FORMAT statement to be referenced by a GO TO statement or an ASSIGN statement.

A FORMULA NUMBER IS GREATER THAN 2**15 OR IS NOT AN INTEGER CONSTANT.
A formula number must be an integer constant and must be less than or equal to 2 to the 15th power. (The asterisks here indicate exponentiation).

A FORMULA NUMBER IS NULL.
External Formula Number is incorrect. Processor cannot determine value.

AFTER OR NEAR OPERAND ***** FIND IMPROPER DELIMITER *****.
Self-explanatory.

A GO TO DESTINATION OR AN - ASSIGN - REFERENCE ***** IS MISSING OR IS A NON EXECUTABLE STATEMENT.
Supply label as the object for the GO TO statement or provide label at the correct executable statement.

A JOB BEGINS WITH AN END-OF-FILE OR A NON IBFTC CARD.
Self-explanatory.

ALPHABETIC CHARACTER EXPECTED AFTER A PERIOD. REFERENCE OPERAND *****.
Self-explanatory.

A NAMELIST NAME IS MISSING.
Self-explanatory.

A NAME BEGINS WITH A NUMERIC CHARACTER.
Self-explanatory.

AN ELEMENT IN THE SUBSCRIPT COMBINATION IS MISSING OR IS ZERO FOR ARRAY *****.
Self-explanatory.

AN END CARD IS NOT FOLLOWED BY AN END OF FILE OR A CONTROL CARD - INPUT CARDS WILL BE IGNORED UNTIL NEXT EOF OR $ CARD.
Self-explanatory.

AN ERROR HAS OCCURRED TRYING TO READ IN THE INTERPHASE TABLES.
Internal compiler error. Programmer should attempt rerun. If error persists consult system engineer.

AN ERROR HAS OCCURRED TRYING TO WRITE OUT THE INTERPHASE TABLE.
Internal compiler error. Programmer should attempt rerun. If error persists consult system engineer.

AN ILLEGAL CHARACTER FOLLOWS ROUTINE NAME *****.
Self-explanatory.

AN I/O STATEMENT REFERENCED A MISSING FORMAT OR AN ILLEGAL STATEMENT. THE REFERENCED EFN IS *****.
The External Formula Number must reference a proper statement. The External Formula Number must be assigned to the desired statement.

APPARENT LOGICAL ERROR IN PROCESSING.
General internal error covering a variety of possibilities. Consult system engineer or attempt rerun.

ARGUMENT ***** APPEARS MORE THAN ONCE IN THE SAME SUBROUTINE, FUNCTION, OR ENTRY STATEMENT.
Self-explanatory.

ARGUMENT VARIABLE ***** WAS PREVIOUSLY USED AS AN ENTRY POINT NAME.
Self-explanatory.

ARITHMETIC STATEMENT FUNCTION DEFINITIONS MUST APPEAR BEFORE ANY OTHER EXECUTABLE STATEMENTS.
Place Arithmetic Statement Function definition before any other executable statement.

ARRAYS WITH MORE THAN 3 DIMENSIONS WILL BE ENTERED IN THE DEBUG DICTIONARY AS 1 DIMENSIONAL ARRAYS.
Self-explanatory.

A SLASH IS MISSING AT THE BEGINNING OF THIS NAMELIST STATEMENT.
Self-explanatory.

ASTERISK IS ILLEGAL ARGUMENT IN FUNCTION SUBPROGRAM.
Self-explanatory.

A SUBSCRIPT, A DIMENSION, OR A PARAMETER IS ZERO OR GREATER THAN 32767, OR IS NOT AN INTEGER.
A subscript, a dimension, or a parameter must be an integer greater than zero and less than 32767. FORTRAN requirement.

A VARIABLE HAS TOO MANY ADDENDS.
Self-explanatory.

A VARIABLE NAME IS A NUMERIC.
Self-explanatory.

A ZERO COEFFICIENT IS NOT ALLOWED IN SUBSCRIPT OF ARRAY *****.
If this condition occurs, a 1 is assumed as the coefficient. Execution is permitted.

BUILT-IN FUNCTION NAME ***** APPEARED IN AN EXTERNAL STATEMENT OR IS AN ARGUMENT TO THIS SUBPROGRAM AND WILL BE TREATED AS AN EXTERNAL FUNCTION SUBPROGRAM.
Self-explanatory.

BUILT-IN OR ARITHMETIC STATEMENT FUNCTION NAMES CANNOT BE PASSED AS ARGUMENTS REFERENCE SYMBOL *****.
Self-explanatory.

BUILT-IN OR LIBRARY FUNCTION ***** HAS BEEN INCORRECTLY TYPED. SYSTEM TYPING WILL TAKE PRECEDENCE.
Self-explanatory.

COMMA MISSING AFTER THE INDEX NAME.
Self-explanatory.

COMMA MISSING BEFORE THE VARIABLE NAME *****.
Self-explanatory.

COMMA MISSING BETWEEN EQUIVALENCE GROUPS.
Self-explanatory.

COMMON BLOCK ILLEGALLY EXTENDED BEYOND ORIGIN BY EQUIVALENCE VARIABLE *****.
Self-explanatory.

COMMON BLOCK NAME OR NAMELIST NAME IS MISSING.
Self-explanatory.

COMMON OR EQUIVALENCE STATEMENT SHOULD APPEAR BEFORE THE FIRST DO.
Self-explanatory.

COMPILER EXPECTS A NUMERICAL ADDEND FOLLOWING THE + OR − SIGN IN THE SUBSCRIPT ELEMENT OF THE ARRAY *****.
Self-explanatory.

COMPILER EXPECTS A NUMERICAL FIELD.
Self-explanatory.

COMPILER EXPECTS AN INTEGER VARIABLE NAME AS ONE OF THE SUBSCRIPT ELEMENTS OF THE ARRAY *****.
Self-explanatory.

COMPILER EXPECTS N1, N2, N3.
Programmer must provide 3 branches for Arithmetic IF statement. Consult section on IF statement.

COMPILER EXPECTS SUBROUTINE NAME AFTER THE WORD CALL. NUMERICS OR PUNCTUATION FOUND INSTEAD.
Self-explanatory.

DATA STATEMENT. DATA FOR ARGUMENT VARIABLE ***** DATA NOT COMPILED.
Self-explanatory.

DATA STATEMENT. DATA FOR ASF ARGUMENT ***** DATA NOT COMPILED.
Data for the Arithmetic Statement Function must be provided.

DATA STATEMENT. DATA FOR BLANK-COMMON VARIABLE ***** DATA NOT COMPILED.
Self-explanatory.

DATA STATEMENT. DATA FOR COMMON VARIABLE ***** IN NON-BLOCK DATA PROGRAM. FORTRAN IV LANGUAGE VIOLATION BUT DATA COMPILED.
Self-explanatory.

DATA STATEMENT. DATA FOR ILLEGAL VARIABLE ***** DATA NOT COMPILED.
Self-explanatory.

DATA STATEMENT. DATA FOR NON-COMMON VARIABLE ***** IN BLOCK-DATA PROGRAM. DATA NOT COMPILED.
Self-explanatory.

DATA STATEMENT ***** DOES NOT END WITH /.
Self-explanatory.

DATA STATEMENT FOR GROUP ***** ILLEGAL PUNCTUATION FOLLOWING NAME ***** TREATED AS COMMA.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** EMPTY VARIABLE LIST.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** ERROR IN SUBSCRIPT FOR NAME *****.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** IMPROPER OR MISSING PARAMETER FOR DO ON ***** PARAMETER ASSUMED TO BE 1.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** LITERAL LIST ILLEGAL ALPHABETIC LITERAL *****.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** LITERAL LIST ILLEGAL PERIOD PRECEDING LITERAL.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** LITERAL LIST ILLEGAL PUNCTUATION TAKEN AS COMMA.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** LITERAL LIST ILLEGAL ZERO COUNT FOR - H - FIELD.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** LITERAL LIST INCORRECT LOGICAL CONSTANT TAKEN AS .FALSE.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** LITERAL LIST INCORRECT LOGICAL CONSTANT TAKEN AS .TRUE.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** LITERAL LIST MISSING PERIOD. INSERTED AT END OF LOGICAL CONSTANT.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** LITERAL LIST MISSING PUNCTUATION. COMMA ASSUMED.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** LITERAL LIST REPEAT COEFFICIENT NOT INTEGRAL. TAKEN AS ZERO.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** LITERAL LIST. SUPERFLUOUS PUNCTUATION IGNORED.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** MISSING COMMA ASSUMED FOLLOWING NAME *****.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** MISSING RIGHT PARENTHESIS INSERTED AFTER SPECIFICATION OF DO ON *****.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** SUPERFLUOUS PUNCTUATION IGNORED FOLLOWING NAME *****.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** UNPAIRED LEFT PARENTHESES IGNORED.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** UNPAIRED RIGHT PARENTHESES IGNORED FOLLOWING NAME *****.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** VARIABLE NAME ***** APPEARS ONLY IN DATA STATEMENT.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** VARIABLE NAME ***** IS AN ARGUMENT TO THIS PROGRAM.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** VARIABLE NAME TO LONG TRUNCATED TO *****.
Self-explanatory.

DATA STATEMENT ***** GROUP ***** VARIABLE NAME ***** STARTS WITH NUMERIC CHARACTER AND IS IGNORED.
Self-explanatory.

DATA STATEMENT. IMPLIED-DO NESTING LEVEL EXCEEDS SEVEN. ALL LEVELS ABOVE SEVENTH IGNORED.
Self-explanatory.

DATA STATEMENT. INITIAL OR FINAL DO-INDEX VALUES OUT OF RANGE OF DIMENSIONS FOR VARIABLE *****.
Self-explanatory.

DATA STATEMENT LITERAL LIST LONGER THAN VARIABLE LIST.
Self-explanatory.

DATA STATEMENT. NO SUBSCRIPT CORRESPONDING TO DO INDEX ***** FOR VARIABLE *****.
Self-explanatory.

DATA STATEMENT. NO DO INDEX CORRESPONDING TO SUBSCRIPT ***** FOR VARIABLE *****.
Self-explanatory.

DATA STATEMENT. SHORT-LIST VARIABLE ***** IN DO.
Self-explanatory.

DATA STATEMENT. SUBSCRIPTS OUT OF RANGE OF DIMENSIONS FOR VARIABLE *****.
Self-explanatory.

DATA STATEMENT. TYPE DISCREPANCY. COMPLEX DATA FOR VARIABLE *****.
Self-explanatory.

DATA STATEMENT. TYPE DISCREPANCY DOUBLE PRECISION DATA FOR VARIABLE *****.
Self-explanatory.

DATA STATEMENT. TYPE DISCREPANCY INTEGER DATA FOR VARIABLE *****.
Self-explanatory.

DATA STATEMENT. TYPE DISCREPANCY LOGICAL DATA FOR VARIABLE *****.
Self-explanatory.

DATA STATEMENT. TYPE DISCREPANCY 'REAL' DATA FOR VARIABLE *****.
Self-explanatory.

DATA STATEMENT. UNSUBSCRIPTED VARIABLE ***** IN DO.
Self-explanatory.

DATA STATEMENT VARIABLE LIST LONGER THAN LITERAL LIST.
Self-explanatory.

DATA STATEMENT VARIABLE SUBSCRIPTS OUTSIDE DO FOR VARIABLE *****.
Self-explanatory.

DATA STATEMENT VARIABLE ***** WITH ONLY CONSTANT SUBSCRIPTS IN DO.
Self-explanatory.

DELIMITER DOES NOT FOLLOW SUBSCRIPTED VARIABLE *****.
Self-explanatory.

DIFFERENT RESULTS FOR THE SAME SCAN: LOGICAL ERROR.
Internal compiler error. Programmer should attempt rerun. If error persists consult system engineer.

DOLLAR SIGNS LEGAL ONLY FOR ERROR RETURNS IN SUBROUTINE CALLS.
Self-explanatory.

DO REFERENCE MISSING OR WRONG PUNCTUATION WITHIN A DO.
Self-explanatory.

DO'S INCORRECTLY NESTED, NO FURTHER CHECKING WILL BE DONE FOR THIS ERROR IN SUBSEQUENT DO-NEST.
First DO entered must be last DO satisfied.

DOTAG/MAIN FILES SEQUENCE ERROR.
Internal compiler error. Programmer should attempt rerun. If error persists, consult system engineer.

DOUBLE OPERATOR FOLLOWING SYMBOL ***** ILLEGAL PUNCTUATION *****.
Self-explanatory.

DUPLICATE EFN.
Two statements cannot have the same External Formula Number.

EFN IS ZERO OR GREATER THAN 32,767 OR IS NOT AN INTEGER.
The External Formula Number must be an integer greater than zero or less than 32767.

EFN MISSING.
An External Formula Number must be provided.

EMPTY STATEMENT – PARAMETERS OR ARGUMENT LIST MISSING.
The statement is not recognized in its present form.

ENTRY STATEMENT WITHIN A DO NEST. STATEMENT IGNORED.
Self-explanatory.

EOB OR REFERENCE IOCS MESSAGE FOR DOTAG/ MAIN FILES.
IOCS error return. Internal error can be an end-of-buffer condition or an IOCS error. Consult system engineer or attempt rerun.

EQUIVALENCE GROUPS REQUIRE TWO VARIABLES.
Self-explanatory.

ERROR IN A COMPLEX LITERAL – SCAN WILL RESUME AT THE CHARACTER FOLLOWING THE RIGHT PARENTHESIS, IF ANY, OR WILL STOP IF NO RIGHT PARENTHESIS IS FOUND.
Self-explanatory.

EXPONENT FIELD TOO LONG – ONLY THE FIRST TWO SIGNIFICANT DIGITS HAVE BEEN KEPT.
Self-explanatory.

FOLLOWING SYMBOL ***** SYNTACTICAL USE OF OPERATOR ***** IS INCORRECT.
Self-explanatory.

FOLLOWING SYMBOL ***** THERE IS NON-LOGICAL OPERAND IN A LOGICAL EXPRESSION.
Self-explanatory.

FORMAT STATEMENT ***** CHARACTER ***** ILLEGAL IN CONTEXT.
Self-explanatory.

FORMAT STATEMENT ***** EXTENDS BEYOND RIGHT PARENTHESIS WHICH MATCHES OPENING PARENTHESIS.
Self-explanatory.

FORMAT STATEMENT ***** ILLEGAL CHARACTER *****.
Self-explanatory.

FORMAT STATEMENT ***** NO NUMBER PRECEDING SCALE FACTOR P.
Self-explanatory.

FORMAT STATEMENT ***** NO NUMBER PRECEDING X.
Self-explanatory.

FORMAT STATEMENT ***** NO OPENING PARENTHESIS.
Self-explanatory.

FORMAT STATEMENT ***** NOT PRECEDED BY NUMBER.
Self-explanatory.

FORMAT STATEMENT ***** NUMBER ASSOCIATED WITH ***** EXCEEDS 132.
Self-explanatory.

FORMAT STATEMENT ***** NUMBER FOLLOWING IN D, E OR F FIELD IS GREATER THAN NUMBER PRECEDING.
Self-explanatory.

FORMAT STATEMENT ***** PARENTHESES DO NOT BALANCE.
Self-explanatory.

FORMAT STATEMENT ***** P SCALING FACTOR TOO LARGE.
Self-explanatory.

FORMAT STATEMENT ***** TOO MANY NESTED LEFT PARENTHESES.
Self-explanatory.

FORMAT STATEMENT ***** ZERO - H - SPECIFICATION ILLEGAL. COUNT ASSUMED TO BE ONE.
Self-explanatory.

FUNCTION ***** IS USED IN A SUBROUTINE CONTEXT.
Self-explanatory.

FUNCTION NOT USED IN AN INPUT LIST OR AT THE LEFT OF AN EQUAL SIGN.
Violation of FORTRAN requirements. Function must be used in input list or at the left of an equal sign.

FUNCTION OR SUBROUTINE ***** CALLS ITSELF.
Self-explanatory.

FUNCTION NAME DOES NOT APPEAR ON LEFT OF EQUALS SIGN.
Self-explanatory.

HIGH ORDER POSITION OF VARIABLE ***** IS NOT AN EVEN NUMBER OF WORDS FROM BEGINNING OF COMMON BLOCK ******.
Compiler requirements. Adjust high order position of variable to conform with message.

HOLLERITH LITERALS ARE PERMITTED ONLY AS DIRECT ARGUMENTS IN CALL STATEMENTS.
Self-explanatory.

IBFTC HAS BEEN GIVEN CONTROL WITH A NON-IBFTC CARD.
Message can occur through a machine error or an internal compiler error. Programmer should attempt rerun. If error persists consult system engineer.

ILLEGAL CHARACTER AFTER FORMULA NUMBERS IGNORED.
An external formula number must not be followed by a non-numeric character.

ILLEGAL CHARACTER BEFORE FORMULA NUMBERS TREATED AS A LEFT PARENTHESIS.
Self-explanatory.

ILLEGAL CHARACTER BEFORE VARIABLE NAME.
Self-explanatory.

ILLEGAL CHARACTER IN A NUMBERIC FIELD.
Self-explanatory.

ILLEGAL CHARACTER ***** OCTAL, TREATED AS A BLANK.
Self-explanatory.

ILLEGAL CHARACTER OR PUNCTUATION *****.
Self-explanatory.

ILLEGAL CHARACTER OR PUNCTUATION.
Self-explanatory.

ILLEGAL DO PARAMETER.
Self-explanatory.

ILLEGAL FORMAT REFERENCE
Self-explanatory.

ILLEGAL OR MISSING PUNCTUATION.
Self-explanatory.

ILLEGAL PUNCTUATION DETECTED IN SOME SUBSCRIPT ELEMENT FOR THE ARRAY ***** OR NUMBER OF SUBSCRIPTS DOES NOT AGREE WITH DIMENSIONALITY.
Self-explanatory.

ILLEGAL PUNCTUATION IN THIS STATEMENT.
Self-explanatory.

ILLEGAL TRANSFER FROM OUTER DO TO INNER DO.
Self-explanatory.

ILLEGAL TRANSFER INTO DO NEST FROM OUTSIDE ITS RANGE.
Self-explanatory.

ILLEGAL TRUE CONDITION FOR THIS LOGICAL IF.
Programmer has given an illegal true condition for IF statement. Violation of FORTRAN language requirements.

ILLEGAL USE OF A PERIOD NEAR SYMBOL *****.
Self-explanatory.

ILLEGAL UNIT REFERENCE.
Self-explanatory.

ILLEGAL UNARY OPERATOR MODIFYING LOGICAL VARIABLE OR EXPRESSION.
A logical variable cannot be preceded by + or − sign in a logical expression.

ILLEGAL USE OF EXPONENTIAL OPERATOR.
Self-explanatory.

ILLEGAL USE OF THE DELIMITER ***** EXISTS.
Self-explanatory.

INCOMPLETE STATEMENT – FORMULA NUMBERS MISSING.
Self-explanatory.

INCOMPLETE STATEMENT – VARIABLE MISSING.
Self-explanatory.

INCONSISTENT RECURRENCE IN EQUIVALENCE STATEMENT OF VARIABLE *****.
Self-explanatory.

INCONSISTENT USAGE. ***** CANNOT BE USED AS A SUBROUTINE NAME.
Self-explanatory.

INCONSISTENT USAGE OF ENTRY NAME *****.
Self-explanatory.

INCORRECT EFN IN COLUMNS 1 to 6.
Provide correct External Formula Number.

INCORRECT NUMBER OF ARGUMENTS FOR BUILT-IN FUNCTION *****.
Self-explanatory.

INCORRECT PUNCTUATION PRECEDES OR FOLLOWS NAME *****.
Self-explanatory.

INPUT ERROR–THE FOLLOWING CARD WILL NOT BE PROCESSED.
Check input deck to eliminate faulty cards.

INTEGER GREATER THAN 32767.
Violation of FORTRAN language requirements.

INTEGER GREATER THAN 2**35-1.
Error in fixed-point arithmetic. Violation of FORTRAN language requirements. Asterisks indicate exponentiation.

INVALID ENTRY IN TABLE *****.
Internal compiler error. Programmer should attempt to rerun program. If error persists, consult system engineer.

***** IS A NON LOGICAL OPERAND IN A LOGICAL EXPRESSION.
Self-explanatory.

LEFT PARENTHESIS EXPECTED AFTER IF.
Self-explanatory.

LIBRARY FUNCTION NAME ***** HAS BEEN TYPED AS EXTERNAL AND INCONSISTENTLY WITH SYSTEM TYPING. USER TYPING WILL TAKE PRECEDENCE.
Self-explanatory.

LIBRARY FUNCTION NAME ***** HAS BEEN TYPED INCONSISTENTLY WITH SYSTEM TYPING. SYSTEM TYPING WILL TAKE PRECEDENCE.
Self-explanatory.

MISSING LEFT PARENTHESIS BEFORE FORMULA NUMBERS.
Self-explanatory.

MISSING PARAMETER WITHIN A DO.
Self-explanatory.

MISSING PUNCTUATION OR ILLEGAL USE OF A DELIMITER.
Self-explanatory.

NEAR SYMBOL ***** AN ARGUMENT TO AN ARITHMETIC STATEMENT FUNCTION DEFINITION BEGINS WITH A NUMERIC OR PUNCTUATION CHARACTER.
Self-explanatory.

NO CLOSING RIGHT PARENTHESIS OR ILLEGAL CHARACTER ENDS SUBSCRIPT COMBINATION OF THE ARRAY ***** OR NUMBER OF SUBSCRIPTS DOES NOT AGREE WITH DIMENSIONALITY.
Self-explanatory.

NO FORMULA NUMBER IN THIS GO TO STATEMENT.
Self-explanatory.

NO MORE TABLE SPACE AVAILABLE.
Program cannot be accommodated. Segment programs into subprograms.

NUMBER OF SUBSCRIPTS FOR EQUIVALENCED ARRAY ***** WRONG.
Self-explanatory.

NUMERICAL VARIABLE NAME IN THE STATEMENT.
Self-explanatory.

OCTAL NUMBER MORE THAN FIVE DIGITS.
Self-explanatory.

ODD SEPARATION BETWEEN EQUIVALENCED DOUBLE-PRECISION OR COMPLEX VARIABLES ***** AND *****.
Compiler requirement. Programmer must maintain an even separation.

OPERATOR MISSING BEFORE OR AFTER OPERAND *****.
Self-explanatory.

***** OPTION NOT IN THE DICTIONARY OR WRONG PUNCTUATION.
Self-explanatory.

OPTIONAL ERROR RETURN IS EITHER LARGER THAN 32,767 OR NOT AN INTEGER.
Violation of FORTRAN language. Programmer should correct External Formula Number.

OVERFLOW IN THIS FLOATING POINT NUMBER.
Self-explanatory.

PARENTHESES ARE ILLEGAL.
Self-explanatory.

PARENTHESES DO NOT BALANCE.
Self-explanatory.

PUNCTUATION MISSING OR INCOMPLETE STATEMENT.
Self-explanatory.

REDUNDANT COMMA.
Self-explanatory.

REDUNDANT COMMA IN FORMULA NUMBERS.
Self-explanatory.

REDUNDANT COMMA IN THIS STATEMENT.
Self-explanatory.

REDUNDANT COMMA OR ILLEGAL PUNCTUATION.
Self-explanatory.

REDUNDANT COMMA OUTSIDE THE FORMULA NUMBERS.
Self-explanatory.

REDUNDANT PARENTHESIS IN THIS STATEMENT.
Self-explanatory.

RIGHT AND LEFT PARENTHESES DO NOT BALANCE.
Self-explanatory.

RIGHT PARENTHESIS MISSING AFTER FORMULA NUMBERS OR INCOMPLETE STATEMENT.
Self-explanatory.

RIGHT PARENTHESIS MISSING FOR THE ARRAY *****.
Self-explanatory.

RIGHT PARENTHESES OR COMMA EXPECTED AFTER A DOLLAR SIGN.
Self-explanatory.

***** SHOULD BE FOLLOWED BY LEFT PARENTHESIS OR EQUALS SIGN.
Self-explanatory.

***** SHOULD BE PRECEDED BY COMMA OR LEFT PARENTHESIS OR FOLLOWED BY COMMA OR RIGHT PARENTHESIS.
Self-explanatory.

STATEMENT ILLEGAL IN THIS CONTEXT. STATEMENT IGNORED.
Self-explanatory.

SUBARG TABLE MISSING AT THE BEGINNING OF PHASE B.
Internal compiler error. The subroutine argument table is functioning incorrectly. Programmer should attempt a rerun. If error persists, consult system engineer.

SUBPROGRAM NAME ***** USED PREVIOUSLY AS VARIABLE OR COMMON BLOCK NAME.
Self-explanatory.

SUBROUTINE, FUNCTION, OR ENTRY NAME ***** CANNOT BE SAME AS DECK NAME.
Self-explanatory.

SUBROUTINE NAME ***** USED PREVIOUSLY AS VARIABLE OR COMMON BLOCK NAME.
Self-explanatory.

SUBSCRIPTS ARE NOT PERMITTED IN ARITHMETIC STATEMENT FUNCTION DEFINITIONS.
Self-explanatory.

SUBSCRIPTS NOT PERMITTED IN A NAMELIST FOR THE ARRAY *****.
Self-explanatory.

SYMBOL BEGINNING WITH ***** TRUNCATED TO SIX CHARACTERS.
Self-explanatory.

SYMBOL FOLLOWING ***** UNIDENTIFIABLE.
Self-explanatory.

SYMBOL LEFT OF EQUALS ILLEGAL OR NONEXISTENT.
Self-explanatory.

THE ADDEND ***** IS NOT NUMERICAL.
Self-explanatory.

THE ADJUSTABLE DIMENSION ***** HAS BEEN DIMENSIONED.
Self-explanatory.

THE ADJUSTABLE DIMENSION ***** IS NOT AN ARGUMENT OF THIS PROGRAM.
Calling program must supply value for adjustable dimension.

THE ADJUSTABLE DIMENSION ***** MUST NOT BE DIMENSIONED.
Self-explanatory.

THE ADJUSTABLE DIMENSION ***** IS NOT AN INTEGER VARIABLE.
Self-explanatory.

THE ARRAY ***** HAS ADJUSTABLE DIMENSION(S).
Self-explanatory.

THE ARRAY ***** HAS MORE THAN 7 DIMENSIONS.
Self-explanatory.

THE ARRAY ***** HAS NO DIMENSION.
Self-explanatory.

THE ARRAY ***** HAS NOT BEEN DIMENSIONED.
Self-explanatory.

THE ARRAY ***** IN COMMON HAS AN ADJUSTABLE DIMENSION.
Self-explanatory.

THE ARRAY NAME IN COMMON ***** HAS A VARIABLE DIMENSION *****.
Self-explanatory.

THE ARRAY ***** HAS AN ADJUSTABLE DIMENSION ***** IS NOT AN ARGUMENT OF THIS PROGRAM.
Illegal FORTRAN construction.

THE COMMON BLOCK ***** IS EMPTY.
Area assigned to COMMON has not been utilized.

THE COMMON BLOCK NAME ***** HAS BEEN USED AS A NAMELIST OR ROUTINE NAME.
Self-explanatory.

THE COMPILER EXPECTS A FORMULA NUMBER.
Self-explanatory.

THE COMPILER EXPECTS A NUMERICAL FIELD INSTEAD OF THE VARIABLE *****.
Self-explanatory.

THE COMPILER EXPECTS A VARIABLE NAME AFTER THE WORD – TO –.
Self-explanatory.

THE COMPILER EXPECTS END OF STATEMENT. EXTRANEOUS CHARACTERS IGNORED.
Self-explanatory.

THE COMPILER EXPECTS THE WORD – TO – AFTER THE FORMULA NUMBER.
Self-explanatory.

THE END OF DIMENSION PRODUCT TABLE HAS ILLEGALLY BEEN REACHED. LOGICAL ERROR.
Internal compiler error. Programmer should attempt a rerun. If error persists, consult a system engineer.

THE INDEX NAME ***** MUST BE A VARIABLE.
Self-explanatory.

THE INDEX NAME ***** MUST NOT BE AN ADJUSTABLE DIMENSION.
Self-explanatory.

THE LEFT PARENTHESIS IS ILLEGAL AS A NAMELIST STATEMENT.
Self-explanatory.

THE NAME ***** HAS ALREADY APPEARED IN A DECLARATIVE STATEMENT.
Self-explanatory.

THE NAME ***** HAS ALREADY APPEARED IN AN EXTERNAL STATEMENT.
Self-explanatory.

THE NAME ***** HAS ALREADY BEEN USED AS A VARIABLE IN AN EXECUTABLE OR DATA, OR NAMELIST STATEMENT.
Self-explanatory.

THE NAME ***** HAS BEEN USED AS NAMELIST OR COMMON BLOCK NAME.
Self-explanatory.

THE NAME ***** IS A ROUTINE OR NAMELIST NAME.
Self-explanatory.

THE NAME ***** IS NOT A VARIABLE NAME.
Self-explanatory.

THE NAME ***** IS THE NAME OF THIS SUBROUTINE.
Self-explanatory.

THE NAMELIST NAME ***** HAS ALREADY BEEN USED.
Self-explanatory.

THE NAMELIST ***** HAS NO LIST OF VARIABLES.
Self-explanatory.

THE PRODUCT OF CONSTANT DIMENSIONS IS GREATER THAN 2**15.
Violation of FORTRAN language requirements. Asterisks indicate exponentiation.

THE PRODUCT OF CONSTANT DIMENSIONS IS ZERO.
Violation of FORTRAN requirements.

THE PROGRAM SHOULD END WITH A TRANSFER, A RETURN HAS BEEN GENERATED.
Programmer should provide transfer instruction.

THE STATEMENT WITH EFN ***** CANNOT BE REACHED.
Statement number ***** cannot be reached because of program construction.

THE SUBSCRIPTED VARIABLE ***** IS NOT AN ARRAY NAME.
Self-explanatory.

THE SYMBOL ***** AND ITS SUBSCRIPT OR ARGUMENT LIST SHOULD BE FOLLOWED BY AN EQUALS SIGN.
Self-explanatory.

THE SYMBOL ***** BEGINS WITH A NUMERICAL CHARACTER.
Violation of FORTRAN language requirements.

THE SYMBOL ***** CANNOT PRECEDE ITS USE AS ARITHMETIC STATEMENT FUNCTION NAME IN AN ASF DEFINITION EXCEPT IN A TYPE STATEMENT.
An arithmetic statement function must precede any executable statement. A symbol can precede an Arithmetic statement function definition only in a TYPE statement.

THE SYMBOL ***** HAS PREVIOUSLY APPEARED IN A STATEMENT BEFORE APPEARING IN AN ARGUMENT LIST. INSTRUCTIONS COMPILED MAY BE INCORRECT.
Self-explanatory.

THE SYMBOL ***** WAS USED PREVIOUSLY AS AN ADJUSTABLE DIMENSION FUNCTION, SUBROUTINE, OR NAMELIST NAME.
Self-explanatory.

THE TYPE OF THE NAME ***** IS INCONSISTENT WITH A PREVIOUS DEFINITION.
Self-explanatory.

THE VARIABLE ***** BEGINS WITH A NUMERIC CHARACTER.
Self-explanatory.

THE VARIABLE ***** HAS ALREADY BEEN DIMENSIONED.
Self-explanatory.

THE VARIABLE ***** HAS ALREADY BEEN USED IN COMMON.
Self-explanatory.

THE VARIABLE ***** IS AN ADJUSTABLE DIMENSION.
Self-explanatory.

THE VARIABLE ***** IS NOT AN INTEGER OR IS AN ARRAY.
The Programmer should provide a nonsubscripted integer variable to conform with this message.

THE VARIABLE ***** MUST BE PRECEDED BY A COMMA.
Self-explanatory.

THE VARIABLE NAME ***** HAS APPEARED AS AN ARGUMENT.
Self-explanatory.

THE VARIABLE NAME ***** HAS BEEN USED AS A NAMELIST NAME.
Self-explanatory.

THE VARIABLE NAME ***** HAS BEEN USED AS A ROUTINE NAME.
Self-explanatory.

THE VARIABLE NAME ***** IS AN ADJUSTABLE DIMENSION.
Self-explanatory.

THE VARIABLE NAME ***** IS USED IN A FUNCTION CONTEXT.
Self-explanatory.

THERE IS AN ILLEGAL OR REDUNDANT PUNCTUATION IN THIS STATEMENT.
Self-explanatory.

THERE IS A REDUNDANT PUNCTUATION IN THE ARRAY *****.
Self-explanatory.

THERE IS A SUPERFLUOUS COMMA IN THIS STATEMENT.
Self-explanatory.

THERE IS A TRANSFER TO THE STATEMENT ITSELF.
Self-explanatory.

THIS NAME ***** HAS BEEN USED AS A ROUTINE NAME.
Self-explanatory.

THIS STATEMENT CANNOT BE REACHED.
Programmer is notified of all statements which are isolated through program logic.

THIS STATEMENT IS EMPTY.
Programmer should complete statement. Compiler expects a full statement.

THIS VARIABLE ***** HAS BEEN USED IN AN EXECUTABLE-DATA-NAMELIST STATEMENT OR AS ADJUSTABLE DIMENSION.
Self-explanatory.

TOO MANY CONTINUATION CARDS FOR THIS STATEMENT.
Self-explanatory.

TOO MANY RIGHT PARENTHESES.
Self-explanatory.

TWO IBFTC CARDS FOR THIS JOB.
Self-explanatory.

TYPES COMBINED ILLEGALLY.
Self-explanatory.

TYPES COMBINED ILLEGALLY BY AN EQUAL SIGN.
Self-explanatory.

TYPES COMBINED ILLEGALLY FOR EXPONENTIATION.
Self-explanatory.

UNDERFLOW IN THIS FLOATING POINT NUMBER.
Violation of FORTRAN requirements.

UNDIMENSIONED VARIABLE ***** IS SUBSCRIPTED IN AN EQUIVALENCE STATEMENT. *****.
An undimensioned variable cannot be subscripted in an equivalence statement.

UNEXPECTED END CONDITION.
Internal compiler error. Programmer should attempt rerun. If error persists consult system engineer.

UNEXPECTED END IN TABLE *****.
Internal compiler error. Programmer should attempt rerun. If error persists, consult system engineer.

UNEXPECTED END MARK.
Self-explanatory.

UNEXPECTED END OF CONSTANT TABLE.
Internal compiler error. Programmer should attempt rerun. If error persists, consult system engineer.

UNEXPECTED END OF DO PUSHDOWN LIST TABLE.
Internal compiler error. Programmer should attempt rerun. If error persists, consult system engineer.

UNEXPECTED END OF EFN TABLE.
Internal compiler error. Programmer should attempt rerun. If error persists, consult system engineer.

UNEXPECTED ENTRY IN THE CONVERT ROUTINE.
Internal compiler error. Programmer should attempt rerun. If error persists, consult system engineer.

UNEXPECTED EOF READING DOTAG/MAIN FILES.
Internal compiler error. Programmer should attempt rerun. If error persists, consult system engineer.

UNRECOGNIZABLE STATEMENT IGNORED.
Compiler has not been able to locate statement in its dictionary. The statement is ignored in this case.

VARIABLES ***** AND ***** IN DIFFERENT COMMON BLOCKS ARE ILLEGALLY EQUIVALENCED.
Self-explanatory.

VARIABLES ***** AND ***** IN SAME COMMON BLOCK ARE INCONSISTENTLY EQUIVALENCED.
Self-explanatory.

VARIABLES ***** AND ***** IN SAME COMMON BLOCK CONSISTENTLY BUT REDUNDANTLY EQUIVALENCED.
Self-explanatory.

WRONG DO INDEX.
Self-explanatory.

WRONG PUNCTUATION OR STATEMENT INCORRECTLY WRITTEN.
Self-explanatory.

WRONG PUNCTUATION OR TOO MANY PARAMETERS FOR THIS DO. END OF STATEMENT IGNORED.
Self-explanatory.

ZERO - H - SPECIFICATION ILLEGAL COUNT ASSUMED TO BE ONE.
The numeric value preceding the H-conversion specification must not be zero.

COBOL Compiler Error Messages

The following alphabetic list contains the error messages generated by the COBOL Compiler. Messages involving the linkage-mode language and compile-time debugging language are included in this list, since they are so closely connected to COBOL programming.

In addition to the error message, a card number corresponding to the number of a card in the deck will also be generated. This number does not necessarily mean that the error occurred on that particular card, but simply means that the error occurred in that general area. If the Compiler for some reason cannot determine the number of a card, it lists the number as either 0000 or 9999.

Words in a message that must vary from situation to situation are denoted by "*****." Where asterisks actually appear as a standard part of a message, the condition is specifically noted. In a case where the Compiler inserts variable information at the beginning of a message, the message is listed alphabetically by the next nonvariable word. For this reason, if a COBOL error message cannot be found alphabetically based on the first word, the programmer should use the second, and in exceptional cases, the third word.

All references are to the publication *IBM 7090/7094 IBSYS Operating System: COBOL Language*, Form C28-6391.

NOTE: There is only one possible MAP assembly error message that can result from a source deck error when the source deck is in COBOL. This is: "LOCATION FIELD FORMAT ERROR." If such a source deck error results in any other MAP message, the programmer should consult a system engineer.

A SPACE SHOULD SEPARATE A SUBSCRIPTED NAME FROM THE FOLLOWING LEFT PARENTHESIS. SPACE IS ASSUMED.
Self-explanatory.

'ACCEPT' MAY ONLY BE FOLLOWED BY A DATA-NAME. NOTHING DONE.
See the ACCEPT verb.

ALL CHARACTERS ACCEPTED FOR ***** MUST BE NUMERIC.
See the ACCEPT verb.

ALPHABETIC CLASS SPECIFIED FOR ***** IGNORED SINCE ITEM IS EXTERNAL DECIMAL.
ALPHABETIC CLASS cannot be specified for external decimal (NUMERIC DISPLAY) items.

ALPHABETIC OR ALPHANUMERIC CLASS SPECIFIED FOR ***** IGNORED SINCE ITEM IS INTERNAL DECIMAL.
Internal decimal (NUMERIC COMPUTATIONAL) items cannot be alphabetic or alphanumeric.

ALTER AT ***** DISALLOWED SINCE IT IS NOT SINGLE GO TO SENTENCE.
See the ALTER verb.

ALTER REFERENCE INCORRECT ***** IS NOT A ***** NOTHING DONE.
There has been incorrect use of the ALTER verb. The ALTER statement is ignored.

***** AND ***** HAVE NO CORRESPONDING SUBFIELDS. NO ACTION STATEMENTS GENERATED FOR THIS PAIR.
See the MOVE CORRESPONDING verb.

ARGUMENT NUMBER ***** MAY NOT APPEAR IN A DISPLAY STATEMENT. SPACE ASSUMED INSTEAD.
See the DISPLAY verb.

ARITHMETIC PHRASES IN CONDITIONAL EXPRESSIONS MAY NOT CONSIST OF MORE THAN 500 OPERATORS AND OPERANDS. EXPRESSION DELETED SINCE LIMIT EXCEEDED.
Break expression into smaller parts.

*****, ASSOCIATED WITH OCCURS ... DEPENDING ON ..., IS AN IMPROPER DATA ITEM. CLAUSE IGNORED.
The data-name is required to be a positive integer greater than zero. See OCCURS clause of the data-item description.

*****, ASSOCIATED WITH REDEFINES OR OCCURS ... DEPENDING ON ..., IS AN IMPROPER DATA ITEM. CLAUSE IGNORED.
See the REDEFINES and OCCURS clauses of the data-item description.

ATTEMPTED DIVISION BY ZERO BYPASSED. RESULT TAKEN TO BE ZERO.
Division by zero is mathematically undefined.

BINARY COMPUTATIONAL USAGE OF ***** INCOMPATIBLE WITH BCD RECORDING MODE FOR THIS FILE.
The record must agree with the mode specifications.

BINARY RECORDING MODE SPECIFICATION OF FILE ***** ASSIGNED TO CARD UNIT IS NOT PERMITTED.
Cards coming from the card reader must not be in binary.

BLOCK SIZE (***** COMPUTER WORDS) SPECIFIED FOR FILE ***** IS NOT A MULTIPLE OF RECORD SIZE (***** COMPUTER WORDS). BLOCK SIZE CHANGED TO ***** COMPUTER WORDS.
This message indicates that the block size and the record size are inconsistent. See the BLOCK CONTAINS and RECORD CONTAINS clauses.

BLOCKING OF DISTINCT RECORD TYPES OF DIFFERING SIZES, WITHOUT COUNT CONTROL, IN FILE ***** IS NOT PERMITTED. FILE IS SET UNBLOCKED.
See BLOCK clause of file description entry.

***** CANNOT BE SUBSCRIPTED. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.
See subscripts.

***** CANNOT BE USED AS AN ARGUMENT FOR THE CORRESPONDING OPTION.
See the CORRESPONDING option of the ADD, SUBTRACT, and MOVE verbs.

***** CANNOT HAVE MORE THAN 49 QUALIFIERS. EXTRA ONES DELETED.

Self-explanatory.

CANNOT USE VARIABLE LENGTH ITEMS FOR COMPARISON. NOTHING DONE.

See IF conditional statements in the PROCEDURE DIVISION.

CARD SEQUENCE ERROR IN COLUMNS 1-6. CONDITION IGNORED.

The card sequence has been checked and this error message results. There is no effect on compilation.

CARD UNIT NOT ALLOWED AS SECONDARY UNIT ASSIGNED TO FILE *****. SECONDARY UNIT ASSIGNMENT IGNORED.

See the FILE-CONTROL paragraph in the INPUT-OUTPUT section of the ENVIRONMENT DIVISION.

CAUTION, GROUP ITEM ***** TESTED.

A group item was an operand of an EXAMINE or IF class-test-type statement. This is a warning message.

CAUTION, GROUP LEVEL MOVE FROM ***** TO *****.

See the MOVE CORRESPONDING verb.

CAUTION, MOVE FROM ***** TO ***** CAUSES TRUNCATION.

This message indicates that either the size or number of decimal places of the items did not match. There is a possibility that information will be lost.

CAUTION, MOVE FROM ***** TO ***** CAUSES TRUNCATION EXCEPT IN CASES OF SYNCHRONIZATION.

This message indicates that there might be a loss of significant data.

CHARACTER LOGIC MOVE INVOLVING AN ITEM LONGER THAN 32767 CHARACTERS. NOTHING GENERATED.

This message indicates a compiler limitation has been reached.

CHECKPOINTS DESIGNATED TO BE WRITTEN ON FILE ***** BUT FILE IS NOT LABELED OUTPUT. CHECKPOINTS WILL BE WRITTEN ON STANDARD CHECKPOINT UNIT INSTEAD.

This message indicates that checkpoints cannot be written on an input file or an unlabeled output file.

CLOSE REEL FOR ***** IS ILLEGAL SINCE FILE IS ASSIGNED TO A CARD OR SYSTEM UNIT. REEL OPTION IGNORED.

Self-explanatory.

COBOL COMPILER DOES NOT OBEY THE USE OF XR4, XR5, OR XR6 ON $IBCBC CARD. XR3 IS ASSUMED.

Index register 3 and 7 are the only index register specifications accepted. See the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

COBOL WORD ***** WAS NOT FOUND WHERE REQUIRED IN THIS STATEMENT. STATEMENT DELETED.

This message indicates a language violation. See the rules regarding the use of the particular verb used.

COBOL WORD 'SECTION' MISSING. BEGINNING OF ***** SECTION ASSUMED BY COMPILER.

See "Organization of Source Program" and the ENVIRONMENT DIVISION.

COBOL WORDS 'ASSIGN TO' OMITTED IN SELECT ENTRY *****. ASSUMED UNIT ASSIGNMENTS IS '1 TAPE-UNIT'.

See the FILE-CONTROL paragraph in the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION.

COBOL WORDS 'TO PROCEED TO' NOT FOUND WHERE REQUIRED IN ALTER STATEMENT. STATEMENT DELETED.

See the ALTER verb.

'COLLATE-COMMERCIAL' SHOULD NOT APPEAR IN ENVIRONMENT DIVISION. COLLATING SEQUENCE ASSUMED COMMERCIAL UNLESS 'BINSEQ' APPEARS ON $IBCBC CARD.

This message indicates obsolete wording.

COMMA ILLEGALLY TO RIGHT OF POINT IN PICTURE OF REPORT ITEM *****. +++++ IS ASSUMED PICTURE.

See the PICTURE clause of the data-item description.

COMPILER ***** COUNT CONTROL CONVENTION ***** FILE *****.

See the file description entry in the DATA DIVISION.

COMPILER ***** COUNT CONTROL CONVENTION ***** FILE ***** UNLESS ***** IS ASSIGNED TO A CARD UNIT AT OBJECT-TIME.

See the file description entry in the DATA DIVISION.

COMPILER ALLOWS ONLY 20 CONSECUTIVE IMPLIED BOOLEAN OPERATORS. CONDITIONAL EXPRESSION DELETED SINCE MAXIMUM EXCEEDED.

Break the expression into smaller parts.

COMPILER ASSUMES FILE(S) ASSOCIATED WITH FD ENTRY ***** HAS LABEL RECORD SINCE VALUE OF LABEL GIVEN.

See the LABEL and VALUE clauses of the file description entry.

COMPILER BASE LOCATOR CAPACITY EXCEEDED. TRY SUBDIVIDING INTO SMALLER PROGRAMS FOR SEPARATE COMPILATION WITH COMBINATION AT OBJECT TIME.

This message indicates a compiler limitation has been reached.

COMPILER FORCED TO ASSUME ***** IS A GROUP ITEM DUE TO ERROR IN SUBSEQUENT LEVEL NUMBER.

See level-numbers in the DATA DIVISION.

COMPILER IGNORES ILLEGAL CLAUSES IN DESCRIPTION OF LEVEL 88 CONDITION ***** (ONLY VALUE IS ALLOWED).

See condition-name in the DATA DIVISION.

COMPILER TABLE CAPACITY EXCEEDED. TRY SUBDIVIDING INTO SMALLER PROGRAMS FOR SEPARATE COMPILATION WITH COMBINATION AT OBJECT TIME.

This message indicates that a compiler limitation has been exceeded. Suggested action is included in the message.

COMPILER THWARTED IN SEARCHING DATA STRUCTURE FOR GROUP(S) CONTAINING ARRAY ***** PROBABLY DUE TO TOO MANY SUBSCRIPTS GIVEN. OBJECT PROGRAM USES FIRST ELEMENT OF THE ARRAY.

See subscripts.

CONDITIONAL EXPRESSION TEST CAPACITY EXCEEDED. REWRITE AS TWO OR MORE SEPARATE EXPRESSIONS WITH A MAXIMUM OF 18 OPERATORS. ILLEGAL SENTENCE STRUCTURE. NOTHING DONE.

This message indicates that a compiler limitation has been reached.

CONDITIONAL VARIABLE ***** IMPROPERLY DESCRIBED AS A REPORT, SCIENTIFIC DECIMAL, OR FLOATING POINT ITEM. X IS ASSUMED PICTURE.

See conditional variables.

CONFLICTING 'USE' OPTIONS FOR FILE ***** OVERRIDDEN BY 'USE' STATEMENT(S) FOR ***** FILES.

This message indicates that conflicting USE procedures have occurred. See the USE verb.

CONTINUATION CHARACTER MUST NOT BE USED WITH AN OCCUPIED A MARGIN. CONTINUATION CHARACTER IGNORED.
Continued items must not begin before B margin. See "Reference Format."

CONTROL CARD ENCOUNTERED PRECEDING $CBEND CARD. END OF COBOL TEXT ASSUMED.
It is assumed that the $CBEND card is missing.

COPY OPTION NOT IMPLEMENTED. FD ENTRY IGNORED.
Enter the information in detail.

CORRESPONDING FIELDS OF ***** AND ***** OVERLAP.
See the MOVE CORRESPONDING verb.

CROSS-REFERENCE NAME TOO LONG. FIRST 6 CHARACTERS USED.
A cross-reference name is limited to 6 characters. See the ENTER verb.

DATA DIVISION-HEADER NOT FOLLOWED BY SECTION-NAME. SCAN RESUMED AT NEXT DATA DESCRIPTION ENTRY, SECTION, OR DIVISION.
See the "Organization of Source Program" and DATA DIVISION.

DATA ITEM ***** INVALID AS AN ARGUMENT IN 'EXAMINE' STATEMENT OR CLASS TEST. STATEMENT DELETED.
See the EXAMINE verb.

DATA ITEM ***** IS UNDER INFLUENCE OF INCONSISTENT USAGE AND CLASS CLAUSES. DETERMINING HIERARCHY IS PICTURE, USAGE, CLASS.
See the data-item description in the DATA DIVISION.

DATA ITEM ***** WITH REDEFINES CLAUSE NOT PRECEDED BY AN ELEMENTARY ITEM. REDEFINES IGNORED.
The redefining item must follow the redefined item with no intervening entries. See the REDEFINES clause of the data-item description.

DATA-NAME, NOT ******, EXPECTED AS ARGUMENT IN THIS STATEMENT. STATEMENT DELETED.
See the rules regarding the use of the particular verb referred to in this message.

DATA-NAME ***** REQUIRING CONVERSION, EDITING, OR DEFINITION MAY NOT APPEAR IN AN ACCEPT STATEMENT. NOTHING DONE.
See the ACCEPT verb.

DATA RECORDS CLAUSE COMMITTED IN FD ENTRY *****. CONDITION IGNORED.
See the DATA RECORDS clause of the file description entry.

DECK NAME IN 'CALL' STATEMENT MUST BE ENCLOSED IN QUOTES. STATEMENT DELETED.
This message indicates a language rule violation. See the ENTER verb.

DECK NAME IS MISSING ON $IBCBC CARD. CONDITION IGNORED.
Self-explanatory.

'DECLARATIVES' MUST BE AT BEGINNING OF PROCEDURE DIVISION. STATEMENT DELETED.
This message indicates a language requirement. See "Organization of Source Program."

***** DEFINED IN MORE THAN ONE OUTPUT FILE. INPUT/OUTPUT STATEMENT IGNORED.
The record name used must be unique.

***** DESCRIPTION ENTRY ENCOUNTERED. BEGINNING OF ***** SECTION ASSUMED BY COMPILER.
See "Organization of Source Program" and the DATA DIVISION.

DISCREPANCY BETWEEN LEVELS OF ***** AND THE REDEFINED ITEM. DISCREPANCY IGNORED AT THIS POINT OF ANALYSIS.
The redefining item should match the redefined item. In this case, the level numbers do not match. This is a warning level message. See the REDEFINES clause of the data-item description.

DIVISION NAME SHOULD BE FOLLOWED BY THE WORD DIVISION AND A PERIOD. CONDITION IGNORED.
Self-explanatory.

DIVISIONS MUST BE IN ORDER AND NOT DUPLICATED. COMPILER SKIPS TO NEXT DIVISION.
See "Organization of Source Program."

$ IS A LEGAL CHARACTER ONLY IN THE PICTURE CLAUSE. $ DELETED.
Self-explanatory.

DOUBLE ASTERISKS (INDICATING EXPONENTIATION) SHOULD NOT BE SEPARATED BY SPACE(S). SPACE(S) IGNORED.
See formulas in the COMPUTE statement in the PROCEDURE DIVISION.

DOWNSCALE GENERATED WHICH LOSES ALL SIGNIFICANT DIGITS.
This message indicates that data has been lost due to downscaling.

ELEMENTARY LEVEL CLAUSES IN DESCRIPTION OF GROUP ITEM ***** IGNORED (I.E., VALUE, SIGNED, POINT, SYNCHRONIZED, EDITING, OR PICTURE.)
Certain clauses apply only to elementary items. See the discussion of the particular clause.

END OF COBOL MESSAGES.
Self-explanatory.

ENVIRONMENT ***** NOT FOLLOWED BY ***** SCAN RESUMED AT NEXT PARAGRAPH, SECTION, OR DIVISION.
See "Organization of Source Program" and the ENVIRONMENT DIVISION.

ENVIRONMENT PARAGRAPH-NAME ENCOUNTERED. BEGINNING OF ***** SECTION ASSUMED BY COMPILER.
See the ENVIRONMENT DIVISION.

ERRONEOUS PARENTHESIZATION IGNORED. TRANSLATION CONTINUES ARBITRARILY.
The number of left parentheses should equal the number of right parentheses.

ERROR IN OCCURS . . . DEPENDING ON CLAUSE IN DESCRIPTION OF *****. COMPILER ASSUMES OCCURS EXACTLY 100 TIMES, IGNORING QUANTITY ITEM.
See the OCCURS clause in the data-item description entry.

ERRORS IN OCCURS . . . DEPENDING ON CLAUSE IN DESCRIPTION OF *****. COMPILER IGNORES 'INTEGER −1 TO' SPECIFICATION.
See the OCCURS clause in the COBOL manual.

ERROR MESSAGE NOT YET IN FILE.
This message should never appear. If it does, a systems engineer should be called to initiate an error report.

ERROR NUMBER ** DUMMY ** .
This message precedes a debugging printout.

EXCESSIVE BLOCK SIZE SPECIFIED FOR FILE *****. FILE IS SET UNBLOCKED.
See the BLOCK CONTAINS clause of the file description entry.

EXTRANEOUS 'ELSE' FOUND. IF IN 'AT END' OR 'ON SIZE ERROR' CLAUSE, 'ELSE' TREATED AS A PERIOD, IN OTHER CASES IT IS TREATED AS A COMMA.
This message indicates a language violation. See the rules regarding the particular verb for which ELSE is an option. The statement must be rewritten to eliminate the extraneous ELSE.

FD ENTRY ***** NOT TERMINATED BY A PERIOD. CONDITION IGNORED.
Self-explanatory.

FIGURATIVE CONSTANT OR NON-NUMERIC LITERAL MUST FOLLOW 'ALL'. ALL ZERO ASSUMED IF COBOL WORD NEXT.
See figurative constants.

FILE ***** ASSIGNED TO *****, BUT FILE IS NOT *****. ***** USAGE ASSUMED.
File usage is contradictory to system unit usage.

FILE ***** ASSIGNED TO CARD UNIT HAS RECORD CONTAINING MORE THAN 72 CHARACTERS. MAXIMUM RECORD SIZE PROCESSED IS 72 CHARACTERS.
Card column limitation has been exceeded. There are only 72 columns available. The first 72 characters will be processed.

FILE ***** ASSIGNED TO SYSOU1, BUT RECORDING MODE GIVEN AS BINARY. RECORDING MODE CHANGED TO BCD.
This message indicates a system requirement; SYSOU1 must be BCD.

FILE ***** ASSOCIATED WITH REDUNDANT 'USE' STATEMENT. FIRST ONE RETAINED.
This message indicates that a redundant USE statement has been encountered. See the USE verb.

FILE ***** HAS NO ASSOCIATED FD ENTRY. ARBITRARY SPECIFICATIONS ASSUMED.
See the file description entry.

FILE ***** HAS NO RECORDS. INPUT/OUTPUT STATEMENT IGNORED.
See the DATA RECORDS clause of the file description entry.

FILE ***** IS ASSIGNED TO A CARD OR SYSTEM UNIT. OPTIONS SPECIFIED IN CLOSE STATEMENT ARE IGNORED.
This message indicates that certain system conventions (as indicated) take priority over COBOL options.

FILE ***** IS ASSIGNED TO ***** AND FILE RECORDING MODE IS BINARY. UNIT NOT PERMITTED TO BE CARD AT OBJECT-TIME.
This message indicates that system units SYSIN1, SYSOU1, and SYSPP1, should not be specified as card units at execution time. This is a system restriction.

FILE ***** IS ASSIGNED TO ***** AND MAXIMUM RECORD SIZE EXCEEDS 72 CHARACTERS. UNIT NOT PERMITTED TO BE CARD AT OBJECT-TIME.
This message indicates that system units SYSIN1, SYSOU1, and SYSPP1, should not be specified as card units at execution time. This is a system restriction.

***** FILE ***** IS ASSIGNED TO *****. UNIT NOT PERMITTED TO BE CARD AT OBJECT-TIME.
This message indicates that system units SYSIN1, SYSOU1, and SYSPP1 should not be specified as card units at execution time. This is a system restriction.

***** FILE ***** NOT PERMITTED AS ARGUMENT IN 'USE' OPTION ***** STATEMENT.
See the USE verb.

FILE ***** RETENTION-PERIOD SPECIFICATION IGNORED SINCE ALLOWED ONLY FOR OUTPUT FILE.
See the VALUE clause of the file description entry in the COBOL manual.

FILE ***** SPECIFIED AS ***** INPUT ***** OUTPUT. ***** USAGE ASSUMED.
This message indicates that the rules governing the use of the USE verb have been violated. See the USE verb.

'FILLER' NOT PERMITTED AS FILE-NAME. FD ENTRY IGNORED.
Self-explanatory.

FIRST REPETITION OF SUBSCRIPTING ERROR. SUBSEQUENTLY, MESSAGES REFERRING TO SUBSCRIPTS APPEAR ONLY ONCE CORRESPONDING TO THE FIRST APPEARANCE OF EACH UNIQUE SUBSCRIPT DATA-NAME OR EXPRESSION.
No further messages will be generated for this error.

FLOATING POINT UNDERFLOW CONVERTING LITERAL. ZERO USED.
This message indicates the exponent became less than the minimum exponent which is 2^{-128} (approximately 10^{-38}).

FLOATING POINT OVERFLOW IN CONVERTING LITERAL. MAXIMUM VALUE USED.
This message indicates the exponent became greater than the maximum exponent which is 2^{+127} (approximately 10^{+38}).

***** FORCED TO LEVEL NUMBER OF 01.
Every data organization must have the highest order at the 01 level. See level-numbers in the DATA DIVISION.

***** ***** FROM ***** ***** TO ***** *****. ***** *****.
See the MOVE verb.

'FROM' MAY ONLY BE FOLLOWED BY IBJOB STANDARD MNEMONIC NAME 'SYSIN1'. SYSIN1 ASSUMED.
See the rules for the ACCEPT verb.

GROUP ITEM ***** USED AS A SUBSCRIPT. OBJECT PROGRAM USES SUBSCRIPT VALUE OF 1.
See subscripts.

***** HAS AN ILLEGAL LEVEL NUMBER. ASSUMED LEVEL NUMBER IS 49.
Valid level-numbers are 01 through 49, 77, and 88. If any other number is specified, 49 will be assumed. See level-numbers.

***** HAS AN ILLEGAL PICTURE. ***** IS ASSUMED PICTURE.
See the PICTURE clause of the data-item description.

***** HAS NO FILE DESCRIPTION. INPUT/OUTPUT STATEMENT IGNORED.
See the verb being used. Also check for proper wording of the file description entry and the FILE-CONTROL paragraph.

***** HAS NOT APPEARED IN A SELECT ENTRY IN ENVIRONMENT DIVISION. FD ENTRY IGNORED.
Self-explanatory.

***** HAS NOT BEEN DEFINED IN A SELECT ENTRY AND IS IGNORED.

See the FILE-CONTROL paragraph of the INPUT-OUTPUT SECTION in the ENVIRONMENT DIVISION.

***** HAS VALUE CLAUSE TOGETHER ILLEGALLY WITH OCCURS OR REDEFINES. VALUE ACCEPTED IF OCCURS – APPLIED TO FIRST ELEMENT.

The OCCURS clause or the REDEFINES clause has been used illegally with the VALUE clause. See the three clauses in the data-item description entry.

HYPHENATED FORM OF ***** DESIGNATION PREFERRED.

See the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

I-O-CONTROL OPTIONS FOR ***** IGNORED SINCE ALLOWED ONLY FOR BINARY FILES.

This message reflects a system IOCS requirement.

I-O-CONTROL PARAGRAPH NOT FOLLOWING FILE-CONTROL PARAGRAPH IGNORED.

See "Organization of Source Program" and the ENVIRONMENT DIVISION.

ILLEGAL ***** UNIT ASSIGNED TO FILE *****. *****

See the FILE-CONTROL paragraph in the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION.

ILLEGAL ARGUMENT IN 'ON' STATEMENT. STATEMENT DELETED.

Refer to "Debugging Package" in the COBOL manual.

ILLEGAL ARITHMETIC PHRASE, ENDING WITH AN OPERATOR OTHER THAN RIGHT PARENTHESIS. PHRASE DELETED.

See formulas in the COMPUTE statement of the PROCEDURE DIVISION.

ILLEGAL CHARACTER IN COLUMN 7. SPACE IS ASSUMED.

A hyphen or a blank are the only characters allowed in column 7. See "Reference Format."

ILLEGAL CHARACTER IN LITERAL. CHARACTER IGNORED.

This message indicates one of the basic rules has been violated. See literals.

ILLEGAL CHARACTER ON CARD DELETED.

Self-explanatory.

ILLEGAL CLAUSE(S) DESCRIBING ***** IGNORED. LENGTH 1, VALUE OF R. M. ASSIGNED BY COMPILER. ONLY SYNCHRONIZATION, IF ANY, RETAINED.

Consult the specific format for the record mark (PICTURE J).

ILLEGAL CLAUSE(S) IN DESCRIPTION OF FLOATING POINT ITEM ***** IGNORED.

See the discussion on the specific format for floating-point items in the USAGE clause of the data-item description entry.

ILLEGAL CLAUSE(S) IN DESCRIPTION OF SCIENTIFIC DECIMAL ITEM ***** IGNORED.

See the discussion on the specific format for scientific decimal items in the PICTURE clause of the data-item description entry.

ILLEGAL CONDITIONAL EXPRESSION IN AT END OR ON SIZE ERROR CLAUSE.

This message has been generated because of a COBOL language restriction, but this 7090/7094 compiler accepts the statement. See the description of the AT END clause of the READ verb or of the ON SIZE ERROR clause relating to the ADD, SUBTRACT, MULTIPLY, or DIVIDE verbs.

ILLEGAL CONDITIONAL EXPRESSION IN TEXT. EXPRESSION IGNORED.

See conditional statements in the PROCEDURE DIVISION.

ILLEGAL CONTROL SECTION NAME FOR DEBUG REQUEST. NAME IGNORED.

Refer to "Debugging Package" in the COBOL manual.

ILLEGAL DESIGNATION OF SIGN CONVENTION IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED.

See the report form of the PICTURE clause of the data-item description entry.

ILLEGAL FORM OF VALUE FOR ***** *****. VALUE IGNORED.

Self-explanatory.

ILLEGAL INSERTION POINT SPECIFICATION FOR DEBUG REQUEST. REQUEST WILL NOT BE EXECUTED.

Refer to "Debugging Package" in the COBOL manual.

ILLEGAL MIXTURE OF DIGIT POSITION CHARACTERS (9 Z *) AFTER POINT IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.

See the report form of the PICTURE clause of the data-item description entry.

ILLEGAL MIXTURE OF ORDER OF DIGIT POSITION CHARACTERS IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.

See the report form of the PICTURE clause of the data-item description entry.

ILLEGAL PICTURE OF SCIENTIFIC DECIMAL ITEM *****. +99999999E+99 IS ASSUMED PICTURE.

See the scientific decimal form of the PICTURE clause of the data-item description entry.

ILLEGAL PICTURE (OR NEITHER PICTURE NOR LEGAL SIZE GIVEN) FOR ***** DECIMAL ITEM *****. 999999 IS ASSUMED PICTURE.

The size must be specified by either the SIZE or the PICTURE clause.

ILLEGAL POINT OR SIGNED CLAUSE IN DESCRIPTION OF NON-REPORT DISPLAY ITEM *****.

See the data-item description entry in the DATA DIVISION.

ILLEGAL REDEFINITION IGNORED FOR FILE RECORD (01 LEVEL) NAMED *****.

The REDEFINES clause cannot be used with logical records (01 level) associated with the same file. See the REDEFINES clause of the data-item description entry.

ILLEGAL SENTENCE STRUCTURE. NOTHING DONE.

See "Punctuation."

ILLEGAL SUBSCRIPT STRUCTURE. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.

See subscripts.

ILLEGAL USAGE OR CLASS CLAUSE (OR BLANK WHEN ZERO) IN DESCRIPTION OF ALPHANUMERIC ITEM *****. CLAUSE IGNORED IN FAVOR OF PICTURE.

This warning message indicates a violation of a language rule.

ILLEGAL USAGE OR CLASS CLAUSE(S) IGNORED IN FAVOR OF PICTURE OF REPORT ITEM *****.

The PICTURE clause overrides contradictory USAGE and CLASS clauses.

ILLEGAL USE OF COMMA IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.

Self-explanatory.

ILLEGAL USE OF $ IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.
See replacement characters under the PICTURE clause of the data-item description entry.

ILLEGAL USE OF UNALTERABLE 'GO TO' STATEMENT. 'GO TO' STATEMENT DELETED.
See the GO TO and ALTER verbs.

ILLEGAL USE OF V OR POINT IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.
See the report form of the PICTURE clause of the data-item description entry.

IMPROPER CHARACTER INTERRUPTS STRING OF + OR − OR $ IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.
See the floating +, −, or $ in the report form of the PICTURE clause of the data-item description entry.

IMPROPER LABEL CLAUSE IGNORED. COMPILER ASSUMES LABEL RECORD(S) OMITTED UNLESS VALUE CLAUSE PRESENT.
See the LABEL clause of the file description entry.

IMPROPER RECORDING CLAUSE IGNORED. BCD, HIGH DENSITY ASSUMED.
See the RECORDING MODE clause of the file description entry.

***** IN THE ENVIRONMENT DIVISION MUST NOT BE REPEATED. SCAN RESUMED AT NEXT PARAGRAPH, SECTION, OR DIVISION.
See "Organization of Source Program" and the ENVIRONMENT DIVISION.

INCOMPLETE STATEMENT DELETED
This message indicates invalid sentence structure. Consult the rules regarding the particular verb in the COBOL manual.

INELIGIBLE DATA-NAME ***** IN RECEIVE OR PROVIDE STATEMENT. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A-MARGIN.
See the "ENTER" verb.

INELIGIBLE DATA-NAME CANNOT BE USED AS AN ARGUMENT FOR THE CORRESPONDING OPTION.
See the CORRESPONDING clause under the MOVE verb.

'INPUT' OR 'OUTPUT' MUST FOLLOW VERB IN AN 'OPEN' STATEMENT. STATEMENT DELETED.
The OPEN statement requires that either INPUT or OUTPUT be specified. See the OPEN verb.

INPUT/OUTPUT STATEMENT IGNORED.
See the PROCEDURE DIVISION discussion on the associated verb. Often associated with errors concerning SELECT or FD entries.

INTEGER MUST NOT EXCEED 32767. INTEGER 1 ASSUMED.
This message indicates that core storage capacity has been exceeded.

INTERNAL FILE ERROR
- INCONSISTENT FILE AND SERIAL
- PERMANENT CLOSE
- UNABLE TO STASH
- READ OUT SEQUENCE
- UTILITY READ
- UTILITY EOF

Possible causes:
1. "short" or faulty utility tapes
2. oversize program
3. machine error

INVALID LITERAL USED IN EXAMINE STATEMENT.
Self-explanatory.

***** IS A NAME DEFINITION AND MUST NOT BE QUALIFIED. DEFINITION FORCED.
This message refers to a data-name, condition-name, or procedure-name that was not properly defined; see the statement being used.

***** IS A TYPE OF ELEMENTARY DATA ITEM THAT MAY NOT BE USED AS A SUBSCRIPT. OBJECT PROGRAM USES SUBSCRIPT VALUE OF 1.
See subscripts.

***** IS A TYPE OF ELEMENTARY DATA ITEM THAT MAY NOT BE USED IN 'RETURN'. SATEMENT DELETED.
See the ENTER verb.

***** IS AN OUT-OF-RANGE REFERENCE.
Refer to "Debugging Package" in the COBOL manual.

***** IS AN UNRECOGNIZABLE ITEM ON CARD. COMPILER SKIPS TO NEXT DIVISION.
The remainder of division is skipped. Check for spelling error.

***** IS GREATER THAN *****. FIRST VALUE USED IN DETERMINING MAXIMUM ***** SIZE.
See "FD."

***** IS IMPROPERLY QUALIFIED. DEFINITION FORCED.
See subscripts.

***** IS IMPROPERLY QUALIFIED. NAME IS NOT UNIQUE. DEFINITION FORCED.
This message refers to a data-name, condition-name, or procedure-name that was not properly defined; see qualification of names.

***** IS NOT *****. INPUT/OUTPUT STATEMENT IGNORED.
The READ verb requires a filename; the WRITE verb requires a record name. See the rules regarding the particular verb.

***** IS NOT A FILE NAME. FD ENTRY IGNORED.
There is no legitimate SELECT entry in FILE-CONTROL paragraph. See the ENVIRONMENT DIVISION and the FILE SECTION of the DATA DIVISION.

***** IS NOT A FILE-NAME. I-O-CONTROL CLAUSE IGNORED.
A spelling error may have occurred in the file-name and no match can be found. The I-O-CONTROL paragraph in the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION is ignored.

***** IS NOT A LEGAL CONDITION-NAME. REMAINDER OF SWITCH-NAME ENTRY IGNORED.
See the SPECIAL-NAMES paragraph in the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

***** IS NOT A LEGAL FILE-NAME. SELECT ENTRY IGNORED.
A language rule has been violated. See the FILE-CONTROL paragraph in the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION.

***** IS NOT A LEGAL MNEMONIC-NAME. REMAINDER OF SWITCH-NAME ENTRY IGNORED.
See the SPECIAL-NAMES paragraph in the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

***** IS NOT A LITERAL. CLAUSE IGNORED.
See the VALUE clause of the data-item description entry.

***** IS NOT A NUMERIC LITERAL AND IS IGNORED.
See the VALUE clause of the file description entry.

***** IS NOT A PROCEDURE NAME. TRANSFER BYPASSING THIS STATEMENT INSERTED.
Incorrect reference. See the structure of the PROCEDURE DIVISION.

***** IS NOT DEFINED. DEFINITION FORCED UNLESS A QUALIFIER.
This message refers to a data-name, condition-name, or procedure-name that was not properly defined; see qualification.

***** IS STRUCTURALLY INCORRECT AT THIS POINT. I-O-CONTROL CLAUSE IGNORED.
A section head has been omitted. See "Organization of Source Program" and the ENVIRONMENT DIVISION.

***** IS STRUCTURALLY INCORRECT AT THIS POINT. REMAINDER OF SWITCH-NAME ENTRY IGNORED.
See the SPECIAL-NAMES paragraph in the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

***** IS STRUCTURALLY INCORRECT AT THIS POINT. SCAN RESUMED AT BEGINNING OF NEXT RERUN CLAUSE, PERIOD, OR PROPER INFORMATION IN THE A MARGIN.
See I-O-CONTROL paragraph of the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION.

***** IS STRUCTURALLY INCORRECT AT THIS POINT. SCAN RESUMED AT BEGINNING OF NEXT SELECT ENTRY, PERIOD, OR PROPER INFORMATION IN THE A MARGIN.
See the FILE-CONTROL paragraph in the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION.

***** IS STRUCTURALLY INCORRECT AT THIS POINT. SCAN RESUMED AT BEGINNING OF NEXT SWITCH-NAME ENTRY, PERIOD, OR PROPER INFORMATION IN THE A MARGIN.
See "Organization of the Source Program" and the ENVIRONMENT DIVISION.

***** IS STRUCTURALLY INCORRECT AT THIS POINT. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.
This message indicates that a language rule has been violated. See the PROCEDURE DIVISION information for the particular verb used.

***** IS UNIDENTIFIABLE. CLAUSE IGNORED.
See the DATA DIVISION.

***** IS UNIDENTIFIABLE. REMAINDER OF CLAUSE IGNORED.
See the FILE SECTION of the DATA DIVISION. This wording may also concern the PROCEDURE DIVISION; see message below.

***** IS UNIDENTIFIABLE. REMAINDER OF CLAUSE IGNORED.
Check spelling. Also study the rules of the particular verb used in this clause. This wording may also concern the FILE SECTION of the DATA DIVISION; see message above.

***** IS UNRECOGNIZABLE IN PROBABLE MULTIPLE REEL OPTION. MULTIPLE REELS ASSUMED.
This message indicates a probable spelling error.

***** IS UNRECOGNIZABLE. SCAN RESUMED AT NEXT DATA DESCRIPTION ENTRY, SECTION, OR DIVISION.
See the DATA DIVISION.

***** IS UNRECOGNIZABLE. SCAN RESUMED AT NEXT PARAGRAPH, SECTION, OR DIVISION.
See the PROCEDURE DIVISION.

ITEM ***** HAS NO SPECIFIED LENGTH. CONDITION IGNORED.
Length of a data-item must be specified. See the PICTURE and SIZE clauses of the data-item description entry.

JUSTIFIED CLAUSE IN DESCRIPTION OF ***** IGNORED. THIS FEATURE NOT IMPLEMENTED.
The JUSTIFIED clause is not a feature of 7090/7094 COBOL.

LABEL CLAUSE OMITTED IN FD ENTRY *****. COMPILER ASSUMES LABEL RECORD(S) OMITTED.
See the LABEL clause of the file description entry.

LENGTH OF NON-NUMERIC LITERAL EXCEEDS LENGTH SPECIFIED BY SIZE OR PICTURE CLAUSE FOR *****. LOW ORDER TRUNCATION DONE.
The alphanumeric constant contained within the VALUE clause should not be greater in size than the item. When it is greater, the low order portion is truncated.

LENGTH OF ***** NOT BOTH CONSTANT AND LESS THAN 73 CHARACTERS. NOTHING DONE.
See the ACCEPT verb.

LENGTH (***** CHARACTERS) OF REDEFINING DATA FIELD HEADED BY DATA-NAME ***** IS GREATER THAN LENGTH (***** CHARACTERS) OF DATA FIELD BEING REDEFINED. DANGEROUS CONDITION IGNORED.
See the REDEFINES clause of the data-item description entry.

LEVEL FD SHOULD APPEAR IN THE A MARGIN. A MARGIN ASSUMED.
This message indicates a violation of a language rule. See "Reference Format."

LEVEL OF ***** CONFLICTS WITH THE PRECEDING LEVEL NUMBER CONDITION IGNORED.
See level-numbers in the data-item description entry.

LEVEL 77 ITEM ***** MAY NOT HAVE OCCURS.
See independent items (level number 77) in the WORKING-STORAGE and CONSTANT SECTION.

LEVEL 77 ITEM ***** APPEARS IN FILE SECTION. INVALID DATA ORGANIZATION RESULTS.
Level 77 items are allowed only in the WORKING-STORAGE and CONSTANT SECTION.

LEVEL 88 CONDITION ***** LACKS MANDATORY VALUE CLAUSE.
Condition-names (level-number 88) must have a VALUE clause. See condition-names.

LEVEL 88 CONDITION ***** APPEARS ILLEGALLY IN CONSTANT SECTION.
Condition-names (level-number 88) have no meaning in the CONSTANT SECTION. See condition-names.

LEVEL 88 CONDITION ***** NOT PRECEDED BY VALID ELEMENTARY ITEM.
See condition-names.

LIMIT (15 BITS) OF SIZE FIELD IN DICTIONARY NECESSITATES TREATING OPERATION LENGTH OF ***** AS MODULO 32768.
This message indicates a compiler limitation has been reached.

LITERAL FOLLOWING ALL IS LIMITED TO ONE CHARACTER. THE FIRST LITERAL CHARACTER IS USED.
See figurative constants.

***** LITERAL IS TOO LONG. FIRST ***** CHARACTERS WILL BE USED.

See the VALUE clause of the file description entry.

MACHINE OR COMPILER ERROR. COMPILATION IS INCOMPLETE.

(1) This message may indicate a machine error. Notify a customer engineer. (2) This message may indicate a compiler error. Consult a system engineer concerning the APAR procedure.

MAXIMUM NUMBER ***** OF DIFFERENT NAMES IN A SOURCE PROGRAM EXCEEDED. COMPILATION TERMINATED.

This message indicates that a compiler limitation has been exceeded. Rework program into smaller individual programs.

MAXIMUM RECORD SIZE (***** COMPUTER WORDS) EXCEEDS SPECIFIED BLOCK SIZE (***** COMPUTER WORDS) OF FILE *****. FILE IS SET UNBLOCKED.

This message indicates that the blocksize specified was not large enough. See the BLOCK CONTAINS clause of the file description entry.

MAXIMUM RECORD SIZE (***** COMPUTER WORDS) SPECIFIED IN FD ENTRY ASSOCIATED WITH FILE ***** IS NOT EQUAL TO SIZE OF MAXIMUM RECORD (***** COMPUTER WORDS). FD RECORD CLAUSE IGNORED.

This message indicates that there is an inconsistency between the record size specified in the file description entry and the record size given in the descriptions of the record. See RECORD CONTAINS and SIZE clauses in the FILE SECTION.

.....MESSAGE CAPACITY EXCEEDED.

This message indicates a compiler limitation. No more error messages can be generated. At least one further error has not been recorded.

MISUSE OF PERIOD, SIGN, OR E IN LITERAL. ILLEGAL CHARACTER(S) IGNORED.

This message indicates one of the mentioned rules has been violated. See literals in the COBOL manual.

MIXED CONTIGUOUS INSERTION-CHARACTERS IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.

See insertion characters under the PICTURE clause of the data-item description entry.

MNEMONIC ***** NOT UNIQUE, CONDITION IGNORED.

See the SPECIAL-NAMES paragraph in the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

MOVE FROM A FIGURATIVE CONSTANT TO A VARIABLE LENGTH GROUP ITEM NOT ALLOWED.

See the MOVE verb.

MOVE FROM A FIGURATIVE CONSTANT TO AN ITEM LONGER THAN 32767 CHARACTERS NOT ALLOWED.

This message indicates a compiler limitation has been reached. Suggest dividing data-item into smaller parts.

MULTIPLE ***** CLAUSES IN ***** DATA DESCRIPTION. FIRST ONE RETAINED.

This message indicates an extraneous clause has appeared. Only the first clause is retained.

MULTIPLE ***** CLAUSES IN FD ENTRY *****. FIRST ONE RETAINED.

See the file description entry.

MULTIPLE CONTIGUOUS INSERTION-CHARACTERS IN PICTURE OF REPORT ITEM ***** CHANGED TO A SINGLE CHARACTER.

This message indicates a compiler convention. See the report form of the PICTURE clause in the data-item description entry.

MULTIPLE REEL OPTION FOR FILE ***** OMITTED WHERE REQUIRED BUT IS ASSUMED.

MULTIPLE REEL must be specified if a file is on two or more reels of magnetic tape. See the FILE-CONTROL paragraph of the INPUT-OUTPUT SECTION in the ENVIRONMENT DIVISION.

NEITHER PICTURE NOR SIZE CLAUSE GIVEN FOR NON-REPORT DISPLAY ITEM *****. X IS ASSUMED PICTURE.

Self-explanatory.

NEITHER PICTURE NOR SIZE CLAUSE GIVEN FOR REPORT ITEM *****. ******* IS ASSUMED PICTURE.

See the data-item description entry.

NESTED REDEFINES ILLEGAL. REDEFINES CLAUSE IGNORED FOR *****.

Redefining is not allowed at a subordinate level to another REDEFINES. No more than one REDEFINES in one organization is permitted.

NO DIGIT POSITIONS IN PICTURE OF REPORT ITEM *****. ++++++ IS ASSUMED PICTURE.

See the report form of the PICTURE clause in the data-item description entry.

NO ERRORS WERE DETECTED BY THE COMPILER

Self-explanatory.

NO PARAGRAPH NAME FOUND PRECEDING 'EXIT' STATEMENT. CONDITION IGNORED.

EXIT must always be a one-word paragraph. See the discussion of the EXIT verb.

NO RECORD DESCRIPTION ENTRIES FOLLOW FD ENTRY *****.

See "Organization of Source Program" and the DATA DIVISION.

NON-ALPHABETIC LITERAL GIVEN FOR ALPHABETIC ITEM *****. CONDITION IGNORED.

The value given for an alphabetic item may not contain non-alphabetic characters.

NON-NUMERIC LITERAL CONTINUATION MUST BEGIN WITH A QUOTE, QUOTE ASSUMED PRECEDING FIRST NON-SPACE CHARACTER.

See "Reference Format."

NON-NUMERIC LITERAL LONGER THAN 120 CHARACTERS OR NAME LONGER THAN 30 CHARACTERS TRUNCATED.

This message indicates a language restriction. See literals.

NON-NUMERIC LITERAL VALUE OF NUMERIC ITEM ***** IGNORED.

Numeric items may have only numeric values. See the VALUE clause in the DATA DIVISION.

***** NOT A LABEL-DATA-NAME. REMAINDER OF VALUE CLAUSE IGNORED.

See the VALUE clause of the file description entry.

NOT IN 'DECLARATIVES' MODE. STATEMENT DELETED.

The USE verb may be used only in the declarative section. See the USE verb.

NOTE...FILE-NAME CHANGED FOR INTERNAL PURPOSES TO *****.

Internal limitations make change of file name mandatory. No information is lost.

NUMBER OF DIGITS IN FIELD OF LITERAL EXCEEDS LIMIT OF 18. 18 DIGITS USED.

A numeric literal may not contain more than 18 characters. See the rules for literals.

NUMBER OF FILES NAMED IN FILE-CONTROL EXCEEDS MAXIMUM OF 63. COMPILATION TERMINATED.

This message indicates that a compiler limitation has been reached. This is a D-level message causing compilation to stop. A dump will also result. It is suggested that the job be broken into smaller jobs.

NUMBER OF OCCURRENCES OF ***** IS ILLEGAL. COMPILER ASSUMES OCCURS EXACTLY 100 TIMES.

See the OCCURS clause of the data-item description entry.

NUMBER OF OCCURRENCES OF ITEM ***** DEPENDS ON A FOLLOWING ITEM IN THE SAME DATA GROUP. 'DEPENDING ON' CLAUSE IGNORED.

The description of the item that determines the number of repetitions cannot follow the item with the OCCURS DEPENDING ON clause. See the OCCURS clause of the data-item description entry.

OBJECT-COMPUTER DESIGNATION OVERRIDDEN BY ***** OPTION ON $IBCBC CARD.

This message indicates that the number of index registers specified on the $IBCBC card does not agree with the number of index registers designated by the object-computer.

OCCURS CLAUSE IGNORED FOR CONDITIONAL VARIABLE *****.

See conditional variables.

OCCURS CLAUSE NOT PERMITTED FOR HIGHEST LEVEL DATA ITEM *****.

This message indicates a language limitation. An OCCURS clause may not be used at 01 level.

***** OF FILE ***** ASSIGNED TO ***** NOT PERMITTED.

This message indicates a system function error. Blocking is not permitted on system units.

***** OF GROUP ITEM ***** IGNORED DUE TO CONFLICT WITH A HIGHER ORDER GROUP.

Self-explanatory.

ONLY FILE-NAMES MAY BE USED AS ARGUMENTS IN 'OPEN' OR 'CLOSE' STATEMENTS. STATEMENT DELETED.

See OPEN and CLOSE statements in the PROCEDURE DIVISION of the COBOL manual.

OPERAND TABLE OVERFLOW TRANSLATING EXPRESSION. STATEMENT DELETED.

The statement is too large. Rearrange the statement in smaller parts.

OPERATION IGNORED BECAUSE ***** HAS IMPROPER DATA FORMAT.

See the rules regarding the particular verb.

OPERATION IGNORED BECAUSE ILLEGAL USE OF FIGURATIVE CONSTANT.

See figurative constants.

***** ORDER TRUNCATION OCCURS IN GENERATION OF INITIAL VALUE FOR *****.

See VALUE clause of the data-item description entry.

***** PARAGRAPH APPEARS ILLEGALLY IN ***** SECTION. SCAN RESUMED AT NEXT PARAGRAPH, SECTION, OR DIVISION.

See "Organization of Source Program" and ENVIRONMENT DIVISION.

PERFORM STATEMENT STRUCTURALLY INCORRECT. STATEMENT DELETED.

See the PERFORM verb.

PERMANENT READ ERROR DURING PROCEDURE INSTRUCTION GENERATION PHASE. COMPILATION IS SUSPECT.

This message indicates a bad utility tape. Do not trust compilation. Suggest re-compilation.

PICTURE OF ALPHANUMERIC ITEM ***** CONTAINS A MIXTURE OF A'S AND 9'S – TREATED AS ALL X'S.

This message indicates a language rule violation.

PICTURE OF REPORT ITEM ***** HAS ILLEGAL USE OF SCALING CHARACTER P. +++++++ IS ASSUMED PICTURE.

See the PICTURE clause of the data-item description entry.

PICTURE OF REPORT ITEM ***** HAS SCALING CHARACTER P EMBEDDED ILLEGALLY BETWEEN NUMERIC CHARACTER POSITIONS. +++++++ IS ASSUMED PICTURE.

See the PICTURE clause of the data-item description entry.

PICTURE OF REPORT ITEM ***** IS ILLEGAL. +++++++ IS ASSUMED PICTURE.

See the report form of the PICTURE clause in the data-item description entry.

PREVIOUS DATA DESCRIPTION NOT TERMINATED BY A PERIOD. PERIOD ASSUMED AND PROCESSING OF ***** BEGUN.

Self-explanatory.

PRIMARY AND SECONDARY UNITS ASSIGNED TO FILE ***** CONFLICT. SECONDARY UNIT ASSIGNMENT IGNORED.

Self-explanatory.

PROCEDURE STATEMENT TO ***** FILE ***** NOT GIVEN.

Every file named in a SELECT statement in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION must be opened and closed in the PROCEDURE DIVISION. See the verbs OPEN and CLOSE.

QUANTITY ITEM ***** SHOULD NOT BE USED WITH 'REPLACING' OPTION IN 'EXAMINE' STATEMENT. CONDITION IGNORED.

See the EXAMINE verb.

RECORD ***** APPEARS IN RECORD DESCRIPTION ENTRY BUT WAS NOT LISTED IN THE FD DATA RECORD(S) CLAUSE. CONDITION IGNORED AND RECORD DESCRIPTION RETAINED.

See the DATA RECORDS clause of the file description entry.

RECORD HEADED BY DATA ITEM ***** EXCEEDS 32767 WORDS. LENGTH MODULO 32768 USED.

This message indicates a compiler limitation has been reached.

RECORD ***** LISTED IN THE FD DATA RECORD(S) CLAUSE DOES NOT APPEAR IN A RECORD DESCRIPTION ENTRY. CONDITION IGNORED.

See the DATA RECORDS clause of the file description entry.

REDEFINES DESCRIBING ***** NOT FOLLOWED BY A PREVIOUSLY DEFINED DATA-NAME. CLAUSE IGNORED.

See the REDEFINES clause of the data-item description entry.

REDUNDANCY ON SYSTEM INPUT UNIT. CONDITION IGNORED.

One card image may have been lost. Suggest recompilation.

REDUNDANT FD ENTRY ***** IGNORED.

Self-explanatory.

REDUNDANT FD ENTRY ***** IGNORED. ONLY ONE FD ENTRY MAY DESCRIBE A SET OF RENAMED SELECT ENTRIES.

See the description of RENAMING clause of the SELECT entry.

REDUNDANT I-O-CONTROL SPECIFICATION.

SEQUENCE-CHECK, CHECK-SUM, or RERUN has been specified more than one time for a given file in the I-O-CONTROL, paragraph of the ENVIRONMENT DIVISION. For example, the following two cards in the same program will cause check-sum to be specified for FILE-NAME-1 twice:

APPLY CHECK-SUM ON FILE-NAME-1.
APPLY CHECK-SUM ON ALL FILES.

The card whose number is given with the message contains the SELECT entry for the file involved.

REDUNDANT SELECT ENTRY ***** IGNORED.

A file-name has been selected more than once in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION.

REDUNDANT SWITCH-NAME ENTRY FOR KEY ***** IGNORED.

See the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

REDUNDANT 'USE' STATEMENT. Statement Deleted

This message indicates that a redundant USE statement has been encountered. See the USE verb.

'RENAMING' MAY ONLY BE FOLLOWED BY A FILE-NAME, REMAINDER OF SELECT ENTRY IGNORED. ASSUMED UNIT ASSIGNMENT IS '1 TAPE-UNIT'.

If the RENAMING option is used in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION, it must be followed by a file-name.

SECTION-HEADER ***** SECTION NOT FOLLOWED BY ***** DESCRIPTION ENTRY. SCAN RESUMED AT NEXT ***** DESCRIPTION ENTRY, SECTION, OR DIVISION.

See "Organization of Source Program."

SECTIONS IN THE DATA DIVISION MUST ***** SCAN RESUMED AT NEXT SECTION, OR DIVISION.

See "Organization of Source Program."

SENTENCE LENGTH EXCEEDS COMPILER CAPACITY. SUGGEST SUBDIVIDING SENTENCE INTO SMALLER COMPONENTS.

Self-explanatory.

SEQUENCE-CHECK MUST BE SPECIFIED WHEN CHECK-SUM IS SPECIFIED, SEQUENCE-CHECK ASSUMED.

This message reflects a system (IOCS) requirement.

729-MODEL NO. ***** ASSIGNED TO FILE *****, NOT ACCEPTABLE TO LOADER, CHANGED BY COMPILER TO *****.

This message indicates a system restriction. Certain 729 Magnetic Tape Units cannot be used.

***** SHOULD BE FOLLOWED BY A SPACE. SPACE IS ASSUMED.

See "Punctuation."

***** SHOULD NOT BE FOLLOWED BY A SPACE. CONDITION IGNORED.

See "Punctuation."

***** SHOULD NOT BE IN THE A MARGIN. B MARGIN ASSUMED.

See "Reference Format."

SIZE OF ***** GIVEN AS 0. COMPILER ASSUMES SIZE IS 6.

Zero is an illegal size.

SIZE, POINT, SIGNED, OR EDITING CLAUSES IGNORED IN FAVOR OF PICTURE IN *****.

This message indicates that some information has been duplicated. In this case, the PICTURE clause information is contrary to similar information specified in another clause. The PICTURE is correct.

SOURCE-COMPUTER IS IMPROPERLY SPECIFIED. IBM-7090 ASSUMED.

The hyphen must be specified in "IBM-7090" in the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

***** SPECIFICATION ***** REMAINDER OF VALUE CLAUSE IGNORED.

See the VALUE clause of the file description entry.

***** SPECIFICATION IN ***** CLAUSE NOT AN UNSIGNED INTEGER. CLAUSE IGNORED.

See the DATA DIVISION.

***** SPECIFICATION IN ***** CLAUSE NOT AN UNSIGNED INTEGER. REMAINDER OF CLAUSE RETAINED.

See the DATA DIVISION.

STATEMENT CONTAINS TOO FEW RIGHT PARENTHESES. COMPENSATING PARENTHESES ADDED AT END OF STATEMENT.

The number of right parentheses must equal the number of left parentheses.

STATEMENT CONTAINS TOO MANY RIGHT PARENTHESES. EXTRA PARENTHESES IGNORED.

The number of right parentheses must equal the number of left parentheses.

STATEMENT REQUIRES A DATA-NAME, LITERAL, OR FIGURATIVE CONSTANT, NOT *****, AS AN ARGUMENT, STATEMENT DELETED.

See the rules regarding the use of the particular verb referred to in this message.

STATEMENT REQUIRES A PROCEDURE NAME, NOT *****, AS AN ARGUMENT. STATEMENT DELETED.

See the rules regarding the use of the particular verb referred to in this message.

SUBORGANIZATION OF ITEM ***** WITH OCCURS CLAUSE CONTAINS A VALUE CLAUSE. VALUE GIVEN TO FIRST ELEMENT ONLY.

A VALUE clause cannot appear in a description subordinate to one containing an OCCURS clause. See the VALUE and OCCURS clauses of the data-item description entry.

SUBSCRIPT COUNT EXCEEDS 3. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.

Only three levels of subscripting are allowed. See "Subscripts."

SUBSCRIPT INTEGER MUST NOT EXCEED 32767. INTEGER 1 ASSUMED.

This message indicates that a compiler limitation has been exceeded.

SUBSCRIPT MISSING AFTER LEFT PARENTHESIS. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.

Information has been omitted.

SYNCHRONIZED ITEM ***** HAS REDEFINES CLAUSE. STORAGE ASSIGNMENT MIGHT NOT BEGIN WITH THE FIRST CHARACTER POSITION OF THE REDEFINED AREA. CONDITION IGNORED.

This message indicates the characteristics of the REDEFINES clause take priority.

SYSIDR ACCOUNTING ROUTINE ERROR.

Contact your systems engineer whenever this message occurs.

T2 WORD – ********** L(TSX) – ***** L(SKEL) – *****

Consult your systems engineer concerning APAR procedure whenever this message occurs. It is always accompanied by the MACHINE OR COMPILER ERROR message.

TERMINATION OF LITERAL FORCED AT END OF CARD.

This message indicates that either a quote mark or a continuation mark has been omitted. See literals.

TOO FEW OPERANDS IN ADD STATEMENT. STATEMENT DELETED.

A minimum of two operands is allowed in an ADD statement.

TOO FEW SUBSCRIPTS GIVEN FOR ***** COMPILER ASSUMES MISSING LEFTMOST SUBSCRIPTS TO BE 1.

See subscripting.

TRANSFER BYPASSED BECAUSE ***** IS NOT A STATEMENT OR SECTION NAME.

Self-explanatory.

21 PERMANENT READ REDUNDANCIES ON SYSTEM INPUT UNIT. COMPILATION TERMINATED.

This message indicates 21 card read errors. Compilation is terminated and a dump is taken.

UNIDENTIFIABLE WORD ***** IN DATA DESCRIPTION. WORD OR CLAUSE IGNORED.

This message indicates a program error. See the DATA DIVISION.

'UPON' MAY ONLY BE FOLLOWED BY IBJOB STANDARD MNEMONIC-NAME 'SYSOU1'. SYSOU1 ASSUMED.

SYSOU1 is the only unit permitted for DISPLAY statements.

UPSCALE MAY CAUSE HIGH ORDER TRUNCATION FOR STORE INTO *****.

High-order truncation may occur because of decimal point alignment in a receiving area that is too small. An error will probably result from this operation.

'USE' NOT PRECEDED BY SECTION-NAME.

See the rules regarding the USE verb in the declarative section of the PROCEDURE DIVISION.

***** USED AS A SUBSCRIPT HAS AN ALPHANUMERIC PICTURE, BUT VALUES MUST BE RESTRICTED TO INTEGERS.

The data-item referred to contains non-numeric characters, possibly blanks. This message is a warning only, but the results obtained during execution are likely to be wrong. See "Subscripts."

***** USED AS A SUBSCRIPT HAS AN ALPHABETIC PICTURE. INVALID SUBSCRIPT. OBJECT PROGRAM USES SUBSCRIPT VALUE OF 1.

See "Subscripts."

***** USED AS SUBSCRIPT IS A SIGNED EXTERNAL DECIMAL ITEM. NEGATIVE VALUES CAUSE ERRORS.

Subscripts must have positive integer values. See "Subscripts."

***** USED AS A SUBSCRIPT IS IN BCD. OBJECT-TIME CONVERSION AND/OR UNPACKING IS REQUIRED FOR SUBSCRIPTS NOT COMPUTATIONAL, SYNCHRONIZED RIGHT.

In most cases computational, synchronized right items are more efficient as subscripts.

***** USED AS A SUBSCRIPT. IS INVALID DUE TO NON-ZERO SCALING. OBJECT PROGRAM USES SUBSCRIPT VALUE OF 1.

Subscripts must be positive integers. See "Subscripts."

***** USED AS A SUBSCRIPT. IS NOT SYNCHRONIZED RIGHT. OBJECT-TIME UNPACKING IS REQUIRED.

In most cases synchronized right items are more efficient as subscripts.

*****, USED TO CONTROL A GO TO, HAS ILLEGAL FORMAT. GO TO STATEMENT IGNORED.

See the GO TO DEPENDING ON statement in the PROCEDURE DIVISION.

*****, USED TO CONTROL A GO TO, IS NOT AN INTEGER. INTEGER PART USED.

See the GO TO DEPENDING ON statement in the PROCEDURE DIVISION.

'USING' MUST BE FOLLOWED BY THE NAME OF A FIXED-LOCATION DATA-ITEM. STATEMENT DELETED.

See the description of the CALL form of the ENTER verb.

VALUE CLAUSE OF ITEM ***** IGNORED SINCE IT IS EITHER REDEFINED, PRECEDED BY A VARIABLE LENGTH ITEM, OR IS IN THE FILE SECTION.

This message indicates that certain clauses are contradictory. See the VALUE clause of data-item description.

VALUE CLAUSE OMITTED IN FD ENTRY *****. LEGAL BUT UNUSUAL.

See the VALUE clause of the file description entry.

WARNING. LISTING OF THIS NAME HAS BEEN TERMINATED. PROGRAM IS NOT AFFECTED.

The REF has been used. Because the number of items cross-referenced exceeds compiler limitations, the REF list may be incomplete.

***** WITH REDEFINES CLAUSE NOT IMMEDIATELY PRECEDED BY THE REDEFINED AREA. REDEFINES IGNORED.

The redefining item must follow the redefined item with no intervening entries. See the REDEFINES clause of the data-item description entry.

'WITHOUT' MUST BE FOLLOWED BY COBOL WORDS 'COUNT CONTROL' IN RECORDING CLAUSE. NOTHING DONE.

See the RECORDING MODE clause of the file description entry.

WORDING 'EVERY BEGINNING OF REEL' PREFERRED IN RERUN CLAUSE AND IS ASSUMED.

This message indicates obsolete wording. See the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION.

NOTE: If a compilation is terminated and a dump taken with no error message being printed, a systems engineer should be notified.

If the Assembler encounters a card in error, it will be flagged with a number. At the end of the assembly listing, the number is printed with an error message explaining the error, and a severity indication is given.

The severity indication is a numerical code:

LEVEL	MEANING
1	Trivial error. Does not affect assembly or execution.
2	Definite error. Assembly supplies probable interpretation. Execution is not permitted.
3	Serious error. Assembly supplies guess interpretation. Execution is not permitted.
4	Unrecoverable error. No interpretation attempted. Assembly continues. Execution is not permitted.
5	Useless to continue assembly. Return to Processor Monitor after printing of all errors detected.

In a normal assembly, messages are printed just after the assembly listing. All messages for a given card are printed together, and the card groups are printed in ascending sequence. Correlation with the listing is accomplished by printing the card number assigned by the assembly, in the left margin of the listing, for each card that requires a message. Messages printed when the NOLIST option is in effect will have no listing for visual correlation. However, since the card numbers are assigned sequentially for every card processed (including duplicate sequences and macro-generated cards) correlation can be made, though perhaps with difficulty.

The following alphabetical list contains the messages generated by the Assembler. The symbol "******" indicates a location in the error message where the Compiler inserts a variable word. Where a variable word is the first in the message, the message is listed alphabetically on the next nonvariable word.

NOTE: One of the Assembler error messages, "LOCATION FIELD FORMAT ERROR" has a special application when the original source deck is coded in the COBOL language. If any other MAP message occurs in connection with a COBOL source deck error, the programmer should call a systems engineer.

7094 INSTRUCTION ONLY.
Level 1. Assembled as -NOP-.

ADDRESS NOT ALLOWED.
Level 1. Zero used for address.

ADDRESS NOT EXPECTED.
Level 1. Address used.

ADDRESS REQUIRED.
Level 1. Zero used for address.

BAD ADJECTIVE CODE USED FOR -OPD- OR -OPVFD-.
Level 4.

BIT COUNT TOO LARGE FOR -VFD- OR -OPVFD- SUBFIELD.
Level 4. 864 is the maximum allowed.

BOOLEAN CONSTANT TRUNCATED TO SIX DIGITS.
Level 2.

BOOLEAN FIELD MUST BE CONSTANT.
Level 2.

-CALL- OPERATION WITH BLANK VARIABLE FIELD IGNORED.
Level 3.

CONTROL DICTIONARY ENTRY FOR ****** ELIMINATED. IMPROPER REFERENCE.
Level 4. References to symbols which are absolute or which are defined by EQUs, BOOLs, and SETs are invalid.

CONTROL DICTIONARY NOT CORRECTLY PROCESSED. ASSEMBLER OF MACHINE ERROR.
Level 4. Bad text encountered.

COUNT FIELD ON -BCI- CARD CANNOT EXCEED SIX DIGITS.
Level 4. A count of 1 is used.

COUNT FIELD ON -BCI- CARD IMPROPERLY TERMINATED. COUNT OF ONE USED.
Level 4.

COUNT FIELD ON -VFD- CANNOT EXCEED SIX DIGITS.
Level 4. First six digits used.

DEBUGGING FORMAT ERROR.
Level 1. Analysis of variable field is completed.

DEBUGGING TABLE OVERFLOW DEBUGGING IGNORED.
Level 1. Table length is approximately 6,200 locations.

DECIMAL CONSTANT TOO LARGE.
Level 2. Truncated to 15 bits.

DECIMAL POINT CAN OCCUR ONLY IN PRINCIPAL PART OF LITERAL OR CONSTANT.
Level 2.

DECREMENT MUST BE CONSTANT AND CANNOT EXCEED ****** BITS.
Level 2.

DECREMENT NOT ALLOWED.
Level 1. Zero used for decrement.

DECREMENT NOT EXPECTED.
Level 1. Low order 6 bits used.

DECREMENT REQUIRED.
Level 1. Zero used for decrement.

DUBIOUS USE OF * ASSUMED TO BE LOCATION COUNTER.
Level 4.

DUBIOUS USE OF ****** ON -FILE- CARD.
Level 1. Indicates 'HYPER' option missing. The name shown where ****** appears in the message will be processed.

-DUP- CARD PUSH DOWN TABLE OVERFLOW.
Level 5. Limit of nested DUP's is 32.

-END- CARD SHOULD FOLLOW.
Level 4. An -END- card is simulated.

END OF FILE WHILE PROCESSING -DUP- CARD.
Level 4. An -END- card is simulated.

END OF FILE WHILE PROCESSING SOURCE INPUT.
Level 4. An -END- card is simulated.

-END- SHOULD NOT OCCUR IN MACRO DEFINITION.
Level 5.

-END- SHOULD NOT OCCUR IN RANGE OF A -DUP-.
Level 4. An END card is processed.

-ETC- CARD SHOULD FOLLOW.
Levels 1 and 2.

EXTERNAL LABEL FOR FILE TOO LONG. FIRST 18 CHARACTERS USED.
Level 3.

EXTERNAL NAME ****** SUPPLIED FOR THIS CDICT ENTRY.
Level 1.

-FILE- CARD MUST HAVE SYMBOL IN LOCATION FIELD.
Level 4. Card ignored.

FLOATING POINT OVERFLOW IN CONVERTING LITERAL OR CONSTANT.
Level 2. Value set to highest possible (all binary 1's).

FLOATING POINT UNDERFLOW IN CONVERTING LITERAL OR CONSTANT.
Level 2. Value set to zero.

FORMAT ERROR FOLLOWING = SIGN ON -FILE- CARD.
Level 3.

FORMAT ERROR IN FIRST SUBFIELD.
Levels 2 and 3. Card ignored.

I, D OR E OCCURRED MORE THAN ONCE FOR -SAVE- OPN.
Level 1. Redundant field ignored.

ILLEGAL BCD CHARACTER TREATED AS IF BLANK.
Level 4.

ILLEGAL COUNT FIELD ON -BCI- CARD.
Level 4. Count of 1 used.

ILLEGAL INTERNAL CONDITION. ASSEMBLER OR MACHINE ERROR.
Level 5.

ILLEGAL OPERAND IN THE VARIABLE FIELD.
Level 4.

ILLEGAL QUALIFICATION USAGE ON THIS CARD.
Levels 1 and 3.

ILLEGAL SCAN CONDITION. ASSEMBLER OR MACHINE ERROR.
Level 5.

ILLEGAL SUBFIELD IGNORED FOR -SAVE- OPN.
Level 3.

ILLEGAL TERMINATOR FOR DECIMAL CONSTANT.
Level 4.

ILLEGAL USE OF -ETC- CARD.
Level 1. Card ignored.

ILLEGAL VARIABLE FIELD CONDITION AT START OF -ETC- CARD.
Level 4. Ignore this and remaining -ETC- cards.

IMPROPER PUNCTUATION FOLLOWING HOLLERITH LITERAL.
Level 4. Punctuation ignored.

INCORRECT DUPLICATE SEQUENCE.
Level 5. The range of a -DUP- that occurs within the range of another -DUP- must be fixed before the outer -DUP- is encountered. See *IBM 7090/7094 IBSYS Operating System: Macro Assembly Program (MAP) Language,* Form C28-6392, for further details.

INCORRECT FORM OF VARIABLE FIELD ON -CONTRL- OR -FILE- CARD.
Level 3. Card ignored.

INCORRECT FORM OF VARIABLE FIELD ON -ORGCRS- CARD.
Level 1. Card treated as if variable field were blank.

INDEX SPECIFIED MORE THAN ONCE FOR -SAVE- OPN.
Level 1. Redundancies ignored.

INDIRECT ADDRESS NOT ALLOWED ON THIS INSTRUCTION.
Level 1.

INTERNAL DICTIONARY FULL. ASSEMBLE PROGRAM IN SMALLER PARTS.
Level 5.

INTERNAL DICTIONARY OVERFLOW WHILE PROCESSING CONTROL DICTIONARY.
Level 4. Assemble program in smaller parts.

INTERNAL TEXT SYNCHRONIZATION FAILURE. ASSEMBLER OR MACHINE ERROR.
Level 4.

INVALID INDEX SPECIFIED FOR -SAVE- OPN.
Level 1. Guess used for index value.

-IRP- IGNORED. VARIABLE FIELD ERROR.
Level 1.

LOCATION FIELD FORMAT ERROR.
Illegal characters ignored. If this message refers to the initial SAVE card of an assembly following a COBOL compilation, however, a numeric deck name has been used on the $IBCBC card for the deck. If there is no reference to the deck name in the program, this error should not be harmful.

LOCATION FIELD REQUIRED ON THIS CARD.
Levels 2 and 4. Card ignored.

MACRO CALL HAS TOO MANY PARAMETERS.
Level 1. Excess parameters ignored.

MACRO DEFINITION CANNOT HAVE MORE THAN 63 PARAMETERS.
Level 4. Excess parameters ignored.

MACRO DEFINITION CARD WITHOUT NAME IGNORED.
Level 1.

MACRO DEFINITION NESTING ERROR.
Level 1. Missing -ENDM- cards simulated.

MACRO DEFINITION TERMINATED BY -ENDM- CARD WITH BLANK VARIABLE FIELD.
Level 1. Macro definition terminated.

MACRO DEFINITION TERMINATED BY -ENDM- CARD WITH WRONG NAME.
Level 1. Macro definition terminated.

MACRO GENERATION SYNCHRONIZATION FAILURE. ASSEMBLER OR MACHINE ERROR.
Level 4. Expansion of current macro terminated.

MACRO PARAMETER EXPANSION TABLE OVERFLOW.
Level 5. Results from substitution being too long and/or too many parameters in macro call.

MACRO PARAMETER PUSH DOWN TABLE OVERFLOW.
Level 5. Results from substitution being too long and/or too many parameters in macro call.

MACRO SKELETON TABLE OVERFLOW. NO MORE DEFINITIONS ACCEPTED.
Level 4. Too many macro definitions. Try PURGE.

MISUSE OF E OR B IN LITERAL OR CONSTANT.
Level 2. Invalid characters ignored.

MISUSE OF PARENTHESES IN MACRO DEFINITION.
Level 3.

MISUSE OF PARENTHESES ON -CALL- OPERATION.
Level 3.

MIXED BOOLEAN EXPRESSION. LEFT BOOL USED.
Level 1.

MORE THAN ONE -BEGIN- FOR THIS LOCATION COUNTER. ALL BUT FIRST IGNORED.
Level 4.

MORE THAN ONE DECIMAL POINT FOR LITERAL OR CONSTANT. ALL BUT FIRST IGNORED.
Level 2.

MORE THAN ONE SIGN FOR EXPONENTIAL OR BINARY-PLACE PART OF LITERAL OR CONSTANT.
Level 2.

MORE THAN ONE SIGN FOR PRINCIPAL PART OF LITERAL OR CONSTANT.
Level 2.

****** IS AN IMPROPERLY QUALIFIED NAME.
Level 1. Frequently caused by multiple definitions of names.

****** IS AN IMPROPER QUALIFIER.
Level 1.

****** IS AN UNDEFINED SYMBOL.
Level 1.

NAME SUPPLIED FOR -SAVE- OPN.
Level 1.

NAME TABLE FULL. ASSEMBLE PROGRAM IN SMALLER PARTS.
Level 5. Approximately 3,500 programmer symbols may be used.

NON-NUMERIC CHARACTER IN 'ID' FIELD OF -CALL-.
Level 1. 'ID' field ignored and line number used.

NO PREVIOUS LOCATION COUNTER BLANK COUNTER USED.
Level 2.

NUMERIC FIELD CONTAINS NON-NUMERIC CHARACTER. FIELD IGNORED.
Level 1.

OCTAL CONSTANT CANNOT EXCEED 12 DIGITS.
Level 2. Field truncated to 12 digits.

OCTAL CONSTANT CONTAINS NON-OCTAL CHARACTER.
Level 2.

ONE OR MORE QUALIFICATION SECTIONS NOT CLOSED.
Level 4.

ONLY FIRST -LDIR- EFFECTIVE.
Level 1.

ONLY FIRST -LORG- EFFECTIVE.
Level 1.

OPERATION CODE NOT IN DICTIONARY.
Level 1. Card ignored.

OPERATION CODE NOT IN DICTIONARY. PURGED.
Level 1. Card ignored.

OPERATION CODE REDEFINED.
Level 1. New definition used.

OPERATION FIELD FORMAT ERROR.
Level 1.

PRINCIPAL, EXPONENTIAL OR BINARY-PLACE PART OF LITERAL OR CONSTANT TOO LONG. FIELD TRUNCATED TO MAXIMUM DIGIT SIZE.
Level 2.

PERMANENT READ ERROR ON SECOND PASS TEXT INPUT.
Level 4.

PRINCIPAL PSEUDO-OPERATION CANNOT BE DEFINED.
Levels 2 and 4. Usually due to circular definition.

PSEUDO-OPERATION DICTIONARY FULL. ASSEMBLE PROGRAM IN SMALLER PARTS.
Level 5.

QUALIFICATION ILLEGAL ON THIS CARD.
Level 4. Card ignored.

QUALIFICATION SECTION NESTING CAPACITY EXCEEDED.
Level 4.

QUALIFICATION SECTION NESTING ERROR.
Level 4.

QUALIFICATION SECTION TERMINATED BY -ENDQ- CARD WITH BLANK VARIABLE FIELD.
Level 1. Innermost qualification eliminated.

QUALIFICATION SECTION TERMINATED BY -ENDQ- WITH WRONG NAME.
Level 4.

SERIALIZATION GROUP ON -LBL- CARD TOO LARGE. FIRST EIGHT CHARACTERS USED.
Level 1.

SIGNIFICANT DIGITS LOST IN SHIFTING FIXED POINT LITERAL OR CONSTANT.
Level 2.

SUBFIELD IGNORED BECAUSE OF FORMAT ERROR.
Level 2. Caused by two left parentheses with no intervening right.

SUBFIELD ****** FORMAT ERROR ON -LABEL- CARD.
Level 2.

S-VALUE INDETERMINATE. ASSEMBLER OR MACHINE ERROR.
Level 4.

SYMBOL TOO LONG. FIRST SIX CHARACTERS USED.
Levels 1 and 4.

TAG NOT ALLOWED.
Level 1. Zero used for tag.

TAG NOT EXPECTED.
Level 1. Tag used.

TAG REQUIRED.
Level 1. Zero used for tag.

THIS CARD NOT EFFECTIVE IN AN ABSMOD ASSEMBLY.
Level 1.

THIS CARD NOT EFFECTIVE IN A RELMOD ASSEMBLY.
Level 1.

THIS CARD NOT EFFECTIVE UNLESS WITHIN A MACRO DEFINITION.
Level 1.

THIS INSTRUCTION CANNOT HAVE AN ASSOCIATED SYMBOL.
Level 1. Symbol ignored.

TOO MANY SUBFIELDS ON -DUP- CARD. FIRST TWO USED.
Level 3.

TOO MANY SUBFIELDS ON THIS CARD.
Levels 1 and 2. Excess subfields ignored.

TOTAL BIT COUNT FOR -OPVFD- MUST NOT EXCEED 36.
Level 3. Card ignored.

VARIABLE FIELD FORMAT ERROR FOR -IFF- OR -IFT- CARD.
Level 4. Card ignored.

VARIABLE FIELD TOO COMPLEX. SIMPLIFY AND REASSEMBLE.
Level 3.

VARIABLE FIELD TOO LONG.
Level 4. Variable field truncated to 127 operands.

VIRTUAL CANNOT APPEAR IN -END- OR -TCD- CARD.
Level 2. Control section value used.

VIRTUAL CANNOT APPEAR IN TAG.
Level 1. Zero used for tag.

ZERO SUPPLIED FOR REQUIRED OPERAND.
Level 1.

Following is an alphabetic list of Load-Time Debugging Processor error messages. The symbol "******" in a message indicates the location where the processor will insert variable information.

*** BCI STRING NOT TERMINATED BY '. TERMINATED BY END OF CARD.
Self-explanatory.

*** CAUTION. THIS * REDEF CARD DEFINES SYMBOLS FOR REQUESTS AFTER THIS CARD AND NOT BEFORE.
Self-explanatory.

CK2 READ ERROR. DUMP TRANSLATION STOPPED.
This is a message from the postprocessor routines and it indicates that the routines cannot process further dump information.

*** COLUMNS 1-5 SHOULD BE BLANK ON CONTINUATION CARDS (IGNORED).
Self-explanatory.

*** CONDITIONAL NOT FOLLOWED BY UNCONDITIONAL STATEMENT DELETED.
This message indicates that an IF or an ON statement is not followed by an unconditional action, such as SET or DUMP.

*** *DEND CARD SIMULATED. ***
Self-explanatory.

*** DUPLICATE LIST NUMBERS.
Self-explanatory.

*** DUPLICATE STATEMENT NUMBERS.
Self-explanatory.

END OF DUMP RECORD NOT ENCOUNTERED. DUMP INFORMATION MAY BE INCOMPLETE.
This is a message from the postprocessor routines and indicates an irregular job termination.

*** ERROR APPROXIMATELY AT V.
The following output appears after this message:

```
V
**************************
```

where the V is an arrow pointing to the approximate part of the source statement, represented by the line of X's, in which an error occurred.

*** *ETC CARD SHOULD HAVE COLUMNS 1-5 BLANK (IGNORED).
Self-explanatory.

*** EXECUTION TERMINATED DUE TO NESTED DEBUG REQUESTS.
Either a debugging request CALL has been made to a subroutine which in turn executes another debugging request, or a trap has occurred during the execution of a debugging request and the routine to which control is thereby passed executes another debugging request. Pairs of requests in such a relationship are called "nested." The above message is connected with a message occurring later in the output, "*** NESTED DEBUG REQUESTS."

*** EXTRA RIGHT PARENTHESIS IN SET STATEMENT. IGNORED.
Self-explanatory.

*** GO TO NOT FOLLOWED BY STATEMENT NUMBER. DELETED.
Self-explanatory.

*** IBDBL ENCOUNTERED $-CARD.
Self-explanatory.

*** IBDBL TERMINATED BY UNEXPECTED EOF.
Self-explanatory.

*** ILLEGAL CALL ARGUMENT. STATEMENT DELETED.
Self-explanatory.

*** ILLEGAL CARD. SKIPPING TO NEXT LEGAL *-CARD.
Self-explanatory.

*** ILLEGAL INSERTION POINT. DELETED.
Self-explanatory.

*** ILLEGAL LIST ELEMENT. ELEMENT DELETED.
Self-explanatory.

*** ILLEGAL MODE IN AREA DUMP SPECIFICATION. OCTAL ASSUMED.
Self-explanatory.

*** ILLEGAL NAME FIELD. DEFINITION DELETED.
Self-explanatory.

*** ILLEGAL ON STATEMENT. STATEMENT DELETED.
Self-explanatory.

*** ILLEGAL OPTION ON $IBDBL CARD. OPTION IGNORED.
Self-explanatory.

*** ILLEGAL *REDEF FIELD. DEFINITION DELETED.
Self-explanatory.

*** ILLEGAL STATEMENT. STATEMENT DELETED.
Self-explanatory.

*** INTEGER TOO BIG FOR STATEMENT NUMBER. DELETED.
Self-explanatory.

INVALID INFORMATION ENCOUNTERED ON SYSCK2. DUMP TRANSLATION STOPPED.
Self-explanatory.

*** LEFT HAND SIDE OF SET IS ILLEGAL. DELETED.
Self-explanatory.

*** LEFT PARENTHESIS IS UNMATCHED.
Self-explanatory.

*** LEFT PARENTHESIS MISSING IN CALL STATEMENT. IGNORED.
Self-explanatory.

*** LEFT PARENTHESIS MISSING IN IF STATEMENT. IGNORED.
Self-explanatory.

LINE MAX EXCEEDED.
This is a message from the postprocessor routines. This message indicates that no more information will be dumped because the programmer-specified LINE MAX has been exceeded.

*** LIST NUMBER - - - - IS UNDEFINED.
Self-explanatory.

*** LIST STATEMENT HAS NO NUMBER.
Self-explanatory.

*** LIST STATEMENT MAY NOT REFER TO ANOTHER LIST. REFERENCE DELETED.
Self-explanatory.

*** NESTED DEBUG REQUESTS.
FIRST REQUEST AT *****, REL LOC *****, ABS LOC *****, IN DECK *****
SECOND REQUEST AT *****, REL LOC *****, ABS LOC *****, IN DECK *****
The variable information after the word "AT" in each half of the message refers to the symbolic location of the request point in the format symbol + k. This message is connected with a message that appears earlier in the debugging output, "*** EXECUTION TERMINATED DUE TO NESTED DEBUG REQUESTS."

*** RIGHT HAND SIDE OF SET IS ILLEGAL. DELETED.
Self-explanatory.

*** RIGHT PARENTHESIS MISSING IN CALL STATEMENT. IGNORED.
Self-explanatory.

*** RIGHT PARENTHESIS MISSING IN IF STATEMENT. IGNORED.
Self-explanatory.

*** STATEMENT NUMBER ****** IS UNDEFINED.
Self-explanatory.

*** SYMBOL HAS MORE THAN 6 CHARACTERS. TRUNCATED TO ******.
Self-explanatory.

*** SYSCK2 IS NOT ATTACHED. JOB WILL RUN WITHOUT DEBUGGING.
Self-explanatory.

*** TEXT BAD. DEBUG INSERTIONS DELETED.
All debug statements following the previous *DEBUG card are erroneous; therefore, this *DEBUG card and all debug statements following it are deleted.

TRAP MAX REACHED.
This is a message from the interpreter routines. This message indicates that no further debugging will take place.

The following alphabetical list contains the messages generated by the Loader. A row of asterisks appears wherever the Loader inserts a word of variable content. When a message begins with variable information, the message is listed alphabetically by the next nonvariable word. Thus, if a message cannot be found in the list alphabetically by the first word, the programmer should look for the message listed alphabetically by the second word.

A SUBROUTINE TO BE INSERTED HAS THE SAME NAME AS AN EXISTING SUBROUTINE WHICH HAS NOT BEEN DELETED.
Self-explanatory.

ABS.PROG.LD.FAILS.NO.EXEC.
A permanent redundancy occurred in the loading of final text overflow tape.

CALL TO OBJECT PROGRAM UNDEFINED.
The absolute location of the first instruction in the object program could not be identified.

$***** CARD ENCOUNTERED, RETURNING TO MONITOR FOR PROCESSING OF THIS CARD.
Either a $JOB, $IBSYS, $EXECUTE, or $STOP card was encountered.

CONTROL DICTIONARY CONTAINS UNDEFINED VIRTUAL.
This message is printed only if LOGIC is specified and the control dictionary contains an undefined virtual.

CONTROL SECTION '******' IS AN UNDEFINED ENTRY POINT.
A section specified on the $ENTRY card does not exist.

CONTROL SECTION '******' REQUIRED BY SUBROUTINE '******' IS VIRTUAL IN THE SUBROUTINE LIBRARY.
Self-explanatory.

CONTROL SECTION '******' SPECIFIED ON A $USE CARD WAS DELETED, TEXT ERRORS MAY OCCUR.
A section specified on a $USE card was nested within a section deleted by equality reduction.

CYLINDER COUNT SPECIFIED FOR FILE '**********' EXCEEDS DISK LIMITS. THE MAXIMUM WILL BE USED.
A maximum of 250 cylinders is permitted on a $FILE card for a disk file.

CYLINDER COUNT SPECIFIED FOR FILE '**********' EXCEEDS DRUM LIMITS. THE MAXIMUM WILL BE USED.
A maximum of 10 cylinders is permitted on a $FILE card for a drum file.

DECK '******' DOES NOT EXIST IN SOURCE INPUT. $OMIT ENTRY IS IGNORED.
A $OMIT specification is in error.

DECK '******' DOES NOT EXIST IN SOURCE INPUT, SECTION RENAMED IS IGNORED.
A $NAME specification is in error.

DECK '******' DOES NOT EXIST IN SOURCE INPUT. $USE ENTRY IS IGNORED.
A $USE specification is in error.

DECK FORMAT ERROR – PROCESSING DECK '******' A CARD SEQUENCE BREAK IN ****** SEQUENCE NUMBER ******.
The input deck contains an error in sequence numbers.

DECK FORMAT ERROR – PROCESSING DECK '******' A CARD SEQUENCE BREAK IN ****** SEQUENCE NUMBER ****** THE LAST CORRECT CARD IS ****** SEQUENCE NUMBER ******.
Self-explanatory.

DECK FORMAT ERROR – PROCESSING DECK '******' ****** BINARY CARD IS NOT IN PROPER PLACE. CANNOT FOLLOW ****** CARD.
A binary card improperly follows BCD card.

DECK FORMAT ERROR – PROCESSING DECK '******' ****** BINARY CARD IS NOT IN PROPER PLACE. THE LAST CORRECT CARD IS ****** SEQUENCE NUMBER ******.
A binary card improperly follows a binary card.

DECK FORMAT ERROR – PROCESSING DECK '******' CARD IS NOT IN PROPER PLACE. THIS CARD IGNORED CANNOT FOLLOW ****** CARD.
A BCD card improperly follows a BCD card.

DECK FORMAT ERROR – PROCESSING DECK '******' CARD IS NOT IN PROPER PLACE. THE LAST CORRECT CARD IS *****- SEQUENCE NUMBER******.
A BCD card improperly follows a binary card.

DECK FORMAT ERROR – PROCESSING DECK '******' CHECKSUM ERROR – DOES NOT COMPARE IN BITS **********.
Self-explanatory.

DECK FORMAT ERROR – PROCESSING DECK '******' CHECKSUM ERROR – DOES NOT COMPARE IN BITS ********** THE LAST CORRECT CARD IS *****- SEQUENCE NUMBER ******.
Self-explanatory.

DECK FORMAT ERROR – PROCESSING DECK '******' ********** IS AN ILLEGAL 9L FORMAT.
Absolute decks and Prest decks cannot be processed by the Loader.

DECK FORMAT ERROR – PROCESSING DECK '******' ********** IS AN ILLEGAL 9L FORMAT. THE LAST CORRECT CARD IS ****** SEQUENCE NUMBER ******.
Absolute decks and Prest decks cannot be processed by the Loader.

DECK FORMAT ERROR – PROCESSING DECK '******' TEXT ENCOUNTERED IN READING CONTROL INFORMATION.
Subroutine library format error-text should not appear in control information file. Probable machine error during library edit.

DECK ****** IS ASSEMBLED FOR IBM 7094 AND CANNOT BE RUN ON IBM 7090.
Self-explanatory.

DECK NAME APPEARING ON THE ABOVE CARD DOES NOT AGREE WITH NAME FROM $IBLDR CARD.
All BCD cards ($IBLDR, $FDICT, $TEXT, $CDICT, $DDICT, and $DKEND) must contain the same deck name in columns 8-13.

DECKNAME CONTAINS ILLEGAL CHARACTER OR BLANK. A DECKNAME OF ALL BLANKS WILL BE USED.
The deck name on $IBLDR card is missing or in error. A parenthesis, slash, quotation mark, equal sign, or embedded blanks are not permitted.

DECK NAME '******' ON $GROUP CARD IS IGNORED.
Self-explanatory.

DECK NAME '******' ON $POOL CARD IS IGNORED.
Self-explanatory.

DECK NAME ON $TEXT OR $DKEND CARD UNRECOGNIZABLE.
No $IBLDR card appeared with the above name.

DISREGARD MOUNTING INSTRUCTIONS.
This message is printed on-line when execution is suppressed after file list has been printed on-line.

END OF FILE IN READING INPUT (GO) TAPE.
End of file on input tape instead of or immediately following the $IBJOB card.

END OF FILE NOT PERMITTED AT THIS POINT.
Input to the Loader is incomplete because of a probable setup error. An end of file may not be embedded within a deck, or before the first $IBLDR card or $INCLUDE card within a link.

END OF TAPE CONDITION OCCURRED IN WRITING 'SUBROUTINE LIBRARY'.
Retry subroutine edit.

ENTRY POINT SPECIFIED IS NOT IN MAIN LINK.
An initial transfer to other than the main link is not permitted.

ERROR IN COMPLEX OPERATOR AT REL LOC xxxxx, TEXT FOLLOWING MAY BE IN ERROR. (DECK 'DECKNM').
Invalid binary text, probable machine error.

ERROR IN COMPLEX RESULT STORAGE REF AT REL LOC xxxxx, TEXT FOLLOWING MAY BE IN ERROR. (DECK 'DECKNM').
Invalid binary text, probable machine error.

ERROR IN FILE NAME ENCOUNTERED ON A $LABEL CARD. THE CARD WILL BE IGNORED.
Self-explanatory.

ERROR IN VARIABLE FIELD OF ABOVE CARD. THE FIELD IS IGNORED.
Self-explanatory.

$ETC CARD NOT FOLLOWING $FILE CARD WHEN REQUIRED. ERRORS MAY OCCUR.
Self-explanatory.

$FILE CARD ACTIVITY SPECIFIED EXCEEDS 99. THIS MAXIMUM WILL BE USED.
Self-explanatory.

FILE **********– 729 UNITS ONLY MAY BE USED WHEN ALTIO OPTION IS SPECIFIED.
Self-explanatory.

FILE '**********' BASE OF 'BLOCK SIZE A MULTIPLE OF' IS INCONSISTENTLY SPECIFIED.
File dictionaries from different decks do not agree on the blocksize definition.

$FILE CARD BLOCK SIZE SPECIFIED EXCEEDS 9999. THIS MAXIMUM WILL BE USED.
Self-explanatory.

FILE ********** CHANNEL IS ILLEGITIMATE.
The channel specified cannot be used.

FILE CHECK-FILE '**********' DEVIATION FROM BASE OF 'BLOCK SIZE A MULTIPLE OF'.
The file dictionary blocksize does not agree with the $FILE card for the same file.

FILE CHECK-FILE '**********' DEVIATION FROM 'EXACT BLOCK SIZE.'
The file dictionary blocksize does not agree with the $FILE card for the same file.

FILE CHECK-FILE '**********' DEVIATION FROM FILE TYPE.
The file type (input, output, checkpoint) in the file dictionary does not agree with the $FILE card for the same file.

FILE CHECK-FILE '**********' DEVIATION FROM 'MINIMUM BLOCK SIZE'.
The file dictionary blocksize does not agree with the $FILE card for the same file.

FILE CHECK-FILE '**********' DEVIATION FROM MIXED MODE.
The file dictionary does not agree with the $FILE card for the same file.

FILE CHECK-FILE '**********' DEVIATION FROM MODE.
The file dictionary does not agree with the $FILE card for the same file.

FILE CHECK-FILE '**********' DEVIATION FROM UNIT ASSIGNMENT (CARD UNIT NOT ALLOWED).
Self-explanatory.

FILE CHECK-FILE '**********' DEVIATION FROM UNIT ASSIGNMENT (TAPE UNIT NOT ALLOWED).
Self-explanatory.

FILE '**********' DISK MODULE REQUESTED IS NOT AVAILABLE.
Self-explanatory.

FILE '**********' DRUM MODULE REQUESTED IS NOT AVAILABLE.
Self-explanatory.

FILE '**********' 'EXACT BLOCKSIZE' IS INCONSISTENTLY SPECIFIED.
File dictionaries from different decks do not agree on blocksize definition.

FILE '**********' ILLEGAL SECONDARY UNIT (REQUEST IS IGNORED).
The secondary unit may not be card equipment, disk, or internal file. The secondary unit must be compatible with primary unit: '*' may not be used to specify secondary unit when primary unit is an intersystem reserve unit.

FILE '**********' ILLEGAL SYSUNI CODE.
A machine or system error.

FILE '**********' INTERSYSTEM INPUT FILE HAS NOT BEEN RESERVED.
Self-explanatory.

FILE '**********' I/O UNIT TYPE REQUIREMENT IS INCONSISTENTLY SPECIFIED.
File dictionaries from different decks do not agree on an input/output unit type.

FILE '**********' MODE OR FILE I/O TYPE IS INCONSISTENTLY SPECIFIED.
File dictionaries from different decks do not agree.

FILE '**********' NO UNIT IS ASSIGNED TO ****** (REQUEST IS IGNORED).
Self-explanatory.

FILE '**********' PRINTER ILLEGAL AS AN INPUT
Self-explanatory.

FILE '**********' PROCESSING ERROR.
Machine or system error during unit assignment.

FILE '**********' PUNCH ILLEGAL AS AN INPUT.
Self-explanatory.

FILE '**********' READER ILLEGAL AS AN OUTPUT.
Self-explanatory.

FILE RENAME FOR FILE '**********' IS IGNORED. DECK '******' DOES NOT EXIST.
Self-explanatory.

FILE RENAME FOR FILE ********** IS IGNORED. FILE DOES NOT EXIST IN ANY FILE DICTIONARIES.
Self-explanatory.

FILE RENAME FOR FILE ********** IS IGNORED. FILE DOES NOT EXIST IN DECK '******'.
Self-explanatory.

FILE '**********' RESERVE UNIT NAME IS ILLEGAL.
Intersystem reserve channels are designated by letters J through Q. Unit numbers range from 0 through 9.

FILE '**********' SPECIFIED AS NOPOOL IS REFERENCED BY A $POOL OR $GROUP CARD. NOPOOL IS IGNORED.
Self-explanatory.

FILE '**********' SPECIFIED ON $GROUP CARD DOES NOT EXIST.
Self-explanatory.

FILE '**********' SPECIFIED ON $LABEL CARD DOES NOT EXIST.
Self-explanatory.

FILE '**********' SPECIFIED ON $POOL CARD DOES NOT EXIST.
Self-explanatory.

FILE '**********' UNIT ****** ILLEGAL AS AN INPUT.
Self-explanatory.

FILE '**********' UNIT ****** ILLEGAL AS AN OUTPUT.
Self-explanatory.

FILE '**********' UNIT ****** ILLEGAL FOR BCD MODE USE (STANDARD OPTION IS ASSUMED).
File mode is binary.

FILE '**********' UNIT ****** ILLEGAL FOR BINARY MODE USE (STANDARD OPTION IS ASSUMED).
File mode is BCD.

FILE '**********' UNIT ****** IS AN ILLEGAL SECONDARY UNIT (REQUEST IS IGNORED), SECOND UNIT ASSIGNED SAME AS FIRST.
The secondary unit may not be card equipment, disk, or internal file. The secondary unit must be compatible with primary unit. '*' may not be used to specify secondary unit when primary unit is an intersystem reserve unit.

FILE '**********' UNIT ****** NOT ALLOWED FOR LABELLED FILE USE.
Self-explanatory.

FILE '**********' UNIT REQUESTED IS NOT AVAILABLE.
Self-explanatory.

FILE '**********' UNIT2 CHANNEL IS ILLEGITIMATE (REQUEST IS IGNORED).
The channel specified cannot be used.

FILE '**********' UNIT2 REQUESTED IS NOT AVAILABLE (REQUEST IS IGNORED).
Self-explanatory.

FIRST CARD READ FROM 'INPUT'/'GO TAPE' IS NOT A $IBJOB CARD.
Probable machine error.

FORMAT ERROR ENCOUNTERED ON A $LABEL CARD. THE CARD WILL BE IGNORED.
Self-explanatory.

FORMAT ERROR ENCOUNTERED ON A $SIZE CARD. THE CARD WILL BE IGNORED.
Self-explanatory.

FORMAT ERROR FOR FIELD '******' OF $NAME CARD. THE REMAINDER OF THIS CARD AND ASSOCIATED $ETC CARDS WHICH FOLLOW WILL BE IGNORED.
Self-explanatory.

FORMAT ERROR FOR FIELD '******' OF $OMIT CARD. THE REMAINDER OF THIS CARD AND ASSOCIATED $ETC CARDS WHICH FOLLOW WILL BE IGNORED.
Self-explanatory.

FORMAT ERROR FOR FIELD '******' OF $USE CARD. THE REMAINDER OF THIS CARD AND ASSOCIATED $ETC CARDS WHICH FOLLOW WILL BE IGNORED.
Self-explanatory.

$GROUP CARD BUFFER COUNT SPECIFIED EXCEEDS 999. THE FIELD WILL BE OMITTED.
Self-explanatory.

$GROUP CARD OPEN FILE COUNT SPECIFIED EXCEEDS 99. THE FIELD WILL BE OMITTED.
Self-explanatory.

$IBLDR CARD ENCOUNTERED WHICH SPECIFIES 'LIBE' DURING PROCESSING OF 'NOLIBE' OPTION CARDS ONLY. THE CARD WILL BE IGNORED.
Mixture of 'LIBE' and 'NOLIBE' decks is not permitted.

$IBLDR CARD ENCOUNTERED WHICH SPECIFIES 'NOLIBE' DURING PROCESSING OF 'LIBE' OPTION CARDS ONLY. THE CARD WILL BE IGNORED.
Mixture of 'LIBE' and 'NOLIBE' decks is not permitted.

$IBLDR CARD ENCOUNTERED WHILE PROCESSING SUBROUTINE WHICH HAS THE SAME NAME AS $IBLDR CARD FROM SOURCE INPUT WHERE 'LIBE' OPTION WAS NOT SPECIFIED.
Duplicate deck names are not permitted.

$IBLDR CARD WITH DUPLICATE NAME ENCOUNTERED WHILE PROCESSING SOURCE INPUT.
Duplicate deck names are not permitted.

ILLEGAL BCD VALUE ENCOUNTERED ON A $ETC CARD FOLLOWING A $FILE CARD. THE $FILE CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL BCD VALUE ENCOUNTERED ON A $ETC CARD FOLLOWING A $GROUP CARD. THE $GROUP CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL BCD VALUE ENCOUNTERED ON A $ETC CARD FOLLOWING A $POOL CARD. THE $POOL CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL BCD VALUE ENCOUNTERED ON A $FILE CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL BCD VALUE ENCOUNTERED ON A $GROUP CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL BCD VALUE ENCOUNTERED ON A $POOL CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL CHARACTER ENCOUNTERED ON A $ETC CARD FOLLOWING A $FILE CARD. THE $FILE CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL CHARACTER ENCOUNTERED ON A $ETC CARD FOLLOWING A $GROUP CARD. THE $GROUP CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL CHARACTER ENCOUNTERED ON A $ETC CARD FOLLOWING A $POOL CARD. THE $POOL CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL CHARACTER ENCOUNTERED ON A $FILE CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED
Self-explanatory.

ILLEGAL CHARACTER ENCOUNTERED ON A $GROUP CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL CHARACTERS ENCOUNTERED ON A $POOL CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL CHARACTER. REMAINDER OF CARD IGNORED.
$ORIGIN or $INCLUDE card contains an invalid character or a blank in column 16.

ILLEGAL FILE NAME ENCOUNTERED ON A $ETC CARD FOLLOWING A $FILE CARD. THE $FILE CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL FILE NAME ENCOUNTERED ON A $ETC CARD FOLLOWING A $GROUP CARD. THE $GROUP CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL FILE NAME ENCOUNTERED ON A $ETC CARD FOLLOWING A $POOL CARD. THE $POOL CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL FILE NAME ENCOUNTERED ON A $FILE CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL FILE NAME ENCOUNTERED ON A $GROUP CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL FILE NAME ENCOUNTERED ON A $POOL CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.
Self-explanatory.

ILLEGAL SECTION NAME ENCOUNTERED ON A $ENTRY CARD. THE CARD WILL BE IGNORED.
The section name specified on the $ENTRY card may not contain a parenthesis, equal sign, quotation mark, comma, or slash.

IMPROPER FORMAT.
Leading, trailing, or multiple field separators occur in the variable field of the $ORIGIN or $INCLUDE card.

IMPROPER SYMBOLIC ORIGIN.
The symbolic origin on a $ORIGIN card is all numeric or greater than six characters.

INCOMPLETE DDICT ENTRY IN DECK '******'.
More debugging dictionary binary cards are expected before $DKEND card.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE AND PERMANENT REDUNDANCY OCCURRED IN READING GENERATED TIF.
Probable machine error.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE AND PERMANENT REDUNDANCY OCCURRED IN READING INTERMEDIATE TEXT.
Probable machine error.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE AND PERMANENT REDUNDANCY OCCURRED IN READING LIBRARY CTRL FILE.
Probable machine error.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE AND PERMANENT REDUNDANCY OCCURRED IN READING LIBRARY SRNT/SRDT.
Probable machine error.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE AND PERMANENT REDUNDANCY OCCURRED IN READING LIBRARY TEXT FILE.
Probable machine error.

INPUT/OUTPUT ERROR – END OF BUFFERS CONDITION OCCURRED IN READING GENERATED CIF.
Probable machine error.

INPUT/OUTPUT ERROR – END OF BUFFERS CONDITION OCCURRED IN READING GENERATED TIF.
Probable machine error.

INPUT/OUTPUT ERROR – PERMANENT REDUNDANCY OCCURRED IN READING GENERATED CIF.
Probable machine error.

INPUT/OUTPUT ERROR – PERMANENT REDUNDANCY OCCURRED IN READING GENERATED TIF.
Probable machine error.

INPUT/OUTPUT ERROR – PERMANENT REDUNDANCY OCCURRED IN READING INTERMEDIATE TEXT.
Probable machine error.

INPUT/OUTPUT ERROR – PERMANENT REDUNDANCY OCCURRED IN READING LIBRARY CTRL FILE.
Probable machine error.

INPUT/OUTPUT ERROR – PERMANENT REDUNDANCY OCCURRED IN READING LIBRARY SRNT/SRDT.
Probable machine error.

INPUT/OUTPUT ERROR – PERMANENT REDUNDANCY OCCURRED IN READING LIBRARY TEXT FILE.
Probable machine error.

INPUT/OUTPUT – END OF FILE IN READING INTERMEDIATE TEXT.
Probable machine error.

INPUT/OUTPUT ERROR – UNEXPECTED END OF FILE IN READING LIBRARY CTRL FILE.
Nonexistent subroutine or a routine which had been passed was specified on a librarian control card.

INPUT/OUTPUT ERROR – UNEXPECTED END OF FILE IN READING LIBRARY TEXT FILE.
Probable machine error.

INSUFFICIENT STORAGE FOR CONTROL DICTIONARIES AND CONTROL INFORMATION DETECTED BY ******.
Program is too large for the Loader to handle. Decks must be restructured to reduce cross referencing.

INSUFFICIENT STORAGE TO GENERATE SUBROUTINE SECTION NAME TABLE AND SUBROUTINE DEPENDENCE TABLE.
The library contains too many control sections. Some control sections must be deleted.

INSUFFICIENT WORKING STORAGE.
Decks must be restructured to reduce cross referencing.

I/O ERROR – EOB IN WRITING FINAL TEXT.
Probable machine error.

I/O ERROR – TEXT – EOB.
Probable machine error.

I/O ERROR – TEXT PERM. REDUNDANCY.
Probable machine error.

LIBRARIAN CONTROL CARD WITH BLANK VARIABLE FIELD.
Only $INSERT card may have blank variable field.

LOADING TERMINATED DUE TO HASH TABLE OVERFLOW. THERE IS AN EXCESSIVE NUMBER OF UNIQUE CONTROL SECTION NAMES.
Only 1,000 unique control section names are allowed in loading. (2,000 unique control section names are permitted by the librarian.) The Loader must be reassembled.

LOADING TERMINATED DUE TO IMPROPERLY DEFINED OVERLAY STRUCTURE.
Overlay structure could not be defined because of invalid symbolic origins on one or more links.

LOADING TERMINATED DUE TO TOO MANY VIRTUAL SECTIONS.
If not a system or machine error, IBLDR must be reassembled. Current limit is 350 virtual sections.

NO DECKNAME IN SPECIFICATION ON $USE CARD. '******' ENTRY IS IGNORED.
Self-explanatory.

NON STANDARD LABEL ROUTINE FOR FILE '**********' WAS DELETED BY LOAD CONTROL CARDS.
Self-explanatory.

NO SYMBOLIC ORIGIN SPECIFIED.
A symbolic origin is not first in the variable field of a $ORIGIN card.

NOT ENOUGH UNITS AVAILABLE.
This general error message is generated if there were not enough units to complete total unit assignment requests.

'NOTEST' IGNORED BECAUSE 'LIBE' OPTION IS SPECIFIED.
Self-explanatory.

NOT IB LOADER CONTROL CARD. CARD IGNORED.
Self-explanatory.

OBJECT PROGRAM EXCEEDS AVAILABLE STORAGE.
Self-explanatory.

OPTIONS OTHER THAN ABSOLUTE ORIGIN IGNORED ON MAIN LINK $ORIGIN CARD.
Self-explanatory.

ORIGIN IS INCORRECTLY SPECIFIED. ORIGIN IS IGNORED.
Librarian control card ($INSERT, $REPLACE, $ASSIGN), used to assign an absolute origin to a Library routine, is incorrect.

ORIGIN MUST BE SPECIFIED ON $ASSIGN CARD.
Librarian control card error.

ORIGIN SPECIFIED FOR LINK *** IS TOO LOW. LOWEST ALLOWABLE FOR THIS SYSTEM CONFIGURATION IS ***** (OCTAL) AND HAS BEEN ASSIGNED.
An absolute origin specified on a $ORIGIN card is too low.

OVERLAY SUBROUTINE .LOVRY NOT DEFINED.
LOVRY is required to perform overlay link loading.

PARAMETER '******' NOT RECOGNIZED. IGNORED.
Unrecognizable parameter found on $ORIGIN card.

PERM REDUN IN READING INPUT (GO) TAPE.
Probable machine error.

$POOL CARD BLOCK SIZE SPECIFIED EXCEEDS 9999. THE FIELD WILL BE OMITTED.
Self-explanatory.

$POOL CARD BUFFER COUNT SPECIFIED EXCEEDS 999. THE FIELD WILL BE OMITTED.
Self-explanatory.

POOLING ERROR GROUPING FILE '**********'.
Files of one group are not in the same buffer pool.

'******' PREVIOUSLY SPECIFIED, IGNORED.
Section or deck appears more than once on $INCLUDE cards for the same link.

PROGRAM EXCEEDS ABSOLUTE LOCATION ******.
Program may use but not load above specified value.

PROGRAM REQUIRES IOCS, IOCS MAY NOT BE LOADED WHEN THE ALTIO OPTION HAS BEEN SPECIFIED.
Self-explanatory.

RELATIVE LOCATION OF TEXT CONTAINING ILLEGAL CONTROL GROUP. (DECK '******').
A list of relative locations follows this message.

RELATIVE LOCATION OF TEXT CONTAINING ILLEGAL LOCATION COUNTER CONTROL. (DECK '******').
Self-explanatory.

RELATIVE LOCATION OF TEXT CONTAINING UNDEFINABLE FIELD. (DECK '******').
A list of relative locations follows this message.

******, REQUIRED AS A LOAD-TIME DEPENDENCY BY ROUTINE ***** IS VIRTUAL IN IBLIB. LIBRARIAN PROCESSING CONTINUES WITH THIS DEPENDENCY IGNORED.
The input/output routine required from analysis of a $LABEL or $FILE card does not exist in the Library.

SECONDARY $ENTRY CARD ENCOUNTERED AND IGNORED.
Self-explanatory.

SECTION ****** IS AN UNDEFINED SYSTEM SYMBOL.
Faulty system subroutines or machine error.

SECTION '******' DOES NOT EXIST IN DECK '******'. $OMIT ENTRY IS IGNORED.
A $OMIT specification is in error.

SECTION '******' DOES NOT EXIST IN DECK '******'. SECTION RENAME IS IGNORED.
A $NAME specification is in error.

SECTION '******' DOES NOT EXIST IN DECK '******'. $USE ENTRY IS IGNORED.
A $USE specification is in error.

SECTION '******' DOES NOT EXIST IN SOURCE INPUT. $OMIT ENTRY IS IGNORED.
A $OMIT specification is in error.

SECTION '******' DOES NOT EXIST IN SOURCE INPUT. SECTION RENAME IS IGNORED.
A $NAME specification is in error.

SECTION '******' IN DECK '******' HAS BEEN MARKED FOR DELETION AND CANNOT BE SPECIFIED ON $USE CARD.
A $USE specification is in error.

SECTION – 2 I/O ERROR EOB IN SRDICT.
Probable machine error.

SECTION – 2 I/O ERROR EOF IN SRDICT.
Probable machine error.

SECTION NAME OF '000000' OR '//' CANNOT BE SPECIFIED. $NAME ENTRY IS IGNORED.
Self-explanatory.

SECTION NAME OF '000000' OR '//' CANNOT BE SPECIFIED. $USE ENTRY IS IGNORED.
Self-explanatory.

SECTION NAME OF '000000' OR '//' CANNOT BE SPECIFIED. $OMIT ENTRY IS IGNORED.
Self-explanatory.

SECTION OR DECK '******' HAS BEEN SPECIFIED TO BE ASSIGNED TO MORE THAN ONE LINK.
Section or deck appears on more than one $INCLUDE card in different links.

SEQUENCE FAILURE IN ORDERING OF SR LIBRARY.
Probable machine error.

STARTING CYLINDER SPECIFIED FOR FILE '**********' EXCEEDS DISK LIMITS. ZERO WILL BE USED.
Starting cylinder specified on a $FILE card for a disk file cannot exceed 249.

STARTING CYLINDER SPECIFIED FOR FILE '**********' EXCEEDS DRUM LIMITS. ZERO WILL BE USED.
Starting cylinder specified on a $FILE cards for a drum file cannot exceed 9.

STORAGE ALLOCATION ERROR – BUFFER COUNT SPECIFIED ON A POOL CARD IS INSUFFICIENT.
Number of buffers must be larger to handle the required number of open files.

STORAGE ALLOCATION ERROR–INSUFFICIENT INPUT/OUTPUT BUFFER STORAGE.
Files must be pooled.

SUBROUTINE DICTIONARY FORMAT ERROR.
Machine or system error.
A). EOF in middle of subroutine name table
B). No subroutine name table
C). Subroutine dependence table not complete
D). Subroutine dependence table has invalid format
1) Invalid operation
2) Comma was encountered when bracket count not greater than zero
3) Too many right brackets

SUBROUTINE NAME IS INCORRECTLY SPECIFIED.
Invalid format of deck name on librarian control card.

SYMBOL '******' NOT DEFINED IN DECK '******'.
Debugging symbol was not found in debugging dictionary for this deck. Debugging activity needing this symbol will be ineffective.

SYMBOL '******' NOT DEFINED IN DECK '******' DECK NOT ENCOUNTERED.
A deck for which a debugging dictionary was required was not in this program load. Debugging activity needing this symbol will be ineffective.

SYSTEM ERROR OR CPU MAINFRAME FAILURE. ENTRY IN SUBROUTINE SECTION NAME TABLE DOES NOT APPEAR AS REAL RETAINED SECTION IN ANY CONTROL DICTIONARY.
Self-explanatory.

THE ABOVE CARD IS NOT A LIBARIAN CONTROL CARD.
Self-explanatory.

THE ABOVE CARD IS NOT PERMITTED IN THE LIBRARY. IT IS IGNORED.
Subroutines may not contain $USE, $OMIT, or $NAME cards.

TOO MANY LEVELS SUBROUTINE DEPENDENCE.
More than 25 dependent subroutines detected due to calling some subroutine.

TOO MANY REQUIRED SUBROUTINES.
The program uses so many subroutines from the Subroutine Library that the storage allotted for the list of subroutine names is exhausted. This list, called the required subroutine package number list, is described in "Section 2" under "Loader Information" and in "Subroutine Library Information."

UNDEFINED CONTROL DICTIONARY ENTRIES REFERENCED.
A list of control section names always follows this message.

UNDEFINED FILE '**********'. $FILE card is missing.
Self-explanatory.

UNDEFINED SECTION OR DECK NAME '******'.
A section or deck name on a $INCLUDE card is undefined.

UNDEFINED VIRTUAL CONTROL SECTION '******'.
This message is not printed if LOGIC is requested.

UNEXPECTED END OF BUFFERS CONDITION ENCOUNTERED IN WRITING OF INTERMEDIATE TEXT FILE.
Probable machine error.

UNIT SYSxxx NOT ATTACHED AND READY.
Unit specified for overlay links has not been attached to a physical drive.

UNRECOGNIZABLE PARAMETER ENOUNTERED ON A $FILE CARD. '******' WILL BE IGNORED.
Self-explanatory.

UNRECOGNIZABLE PARAMETER ENCOUNTERED ON A $GROUP CARD. '******' WILL BE IGNORED.
Self-explanatory.

UNRECOGNIZABLE PARAMETER ENCOUNTERED ON A $IBLDR CARD. '******' WILL BE IGNORED.
Self-explanatory.

UNRECOGNIZABLE PARAMETER ENCOUNTERED ON A $LABEL CARD. THE FIELD IS STORED AS ZERO.
Self-explanatory.

UNRECOGNIZABLE PARAMETER ENCOUNTERED ON A $POOL CARD. '******' WILL BE IGNORED.
Self-explanatory.

UNRECOGNIZABLE PARAMETER ON $IBJOB CARD. ****** IS IGNORED.
Self-explanatory.

VALUE SPECIFIED ON $SIZE CARD EXCEEDS FIELD SIZE. THE CARD WILL BE IGNORED.
Self-explanatory.

VIRTUAL SECTION NAME LIST IS FULL.
If not system or machine error, the Loader must be reassembled. Limit is 350.

Subroutine Library Error Messages

The following list includes system subroutine, FORTRAN IV subroutine, and COBOL subroutine error messages. Following each message is an explanation or a suggested action. The symbol "******" indicates the location in a message where the Subroutine Library inserts variable information.

System Subroutine Messages

The subroutine in which the error was encountered precedes each message.

.LXCON

****** LINES OUTPUT
Self-explanatory.

STR AT ***** XR1 = ***** XR2 = ***** XR4 = *****
This message occurs only on the IBM 7090 Operating System and is followed by a dump.

SYSTEM STOP XR1 = ***** XR2 = ***** XR4 = *****
Self-explanatory. This message occurs only on the IBM 7090 Operating System.

STR AT ***** XR1 = ***** XR2 = ***** XR4 = *****
XR3 = ***** XR5 = ***** XR6 = ***** XR7 = *****
This message occurs only on the IBM 7094 Operating System and is followed by a dump.

SYSTEM STOP XR1 = ***** XR2 = ***** XR4 = *****
XR3 = ***** XR5 = ***** XR6 = ***** XR7 = *****
Self-explanatory.

.LOVRY

UNABLE TO INTERPRET OVERLAY COMMUNICATION REGION WHILE LOADING LINK. CANNOT PROCEED.
This message is generated when the overlay tables, .LDT, .LVEC, or .LRECT, are destroyed, or when a tape read error occurred while overlay tables were being destroyed.

.LXSL

.LXSEL FOR UNIT REQUESTED IS NOT IN LIBRARY.
This message indicates that a reassembly is required for the equipment requested. .LXSL is a modular assembly based on parameters for disk/drum and Hypertape.

SYSOU, SYSIN, OR SYSPP CANNOT BE ASSIGNED TO DISK/DRUM.
Since the operating system does not support peripheral functions on disk or drum, .LXSL does not support them either. Reassignment of the function is required.

FORTRAN IV Subroutine Messages

The following list gives the subroutine in which the error was encountered, the error code, the error message, and the optional exit message. Execution is terminated after each error message is written unless the optional exit is used. The optional exit message is written below the error message to indicate the action taken before execution is resumed. (See the discussion of optional exits under "FORTRAN Utility Library.") An explanation follows each error message and each optional exit message.

FXP1 1
EXPONENTIATION ERROR 0**0.
For I^J where I=0, J=0.

SET RESULT =0.
Set I^J=0.

FXP1 2
EXPONENTIATION ERROR 0**(-J).
For I^J where I=0, J<0.

SET RESULT =0.
Set I^J=0.

FXP2 3
EXPONENTIATION ERROR 0**0.
For B^J where B=0, J=0.

SET RESULT =0.
Set B^J=0.

FXP2 4
EXPONENTIATION ERROR 0**(-J).
For B^J where B=0, J<0.

SET RESULT =0
Set B^J=0.

FXP3 5
EXPONENTIATION ERROR (-B)**C.
For B^C where B<0, C≠0.

EVALUATE FOR +B.
Evaluate for | B |.

FXP3 6
EXPONENTIATION ERROR 0**0.
For B^C where B=0, C=0.

SET RESULT =0.
Set B^C=0.

FXP3 7
EXPONENTIATION ERROR 0**(-C).
For B^C where B=0, C<0.

SET RESULT =0.
Set B^C=0.

FXPF 8
EXP(X), X GRT THAN 88.029692 NOT ALLOWED.
For e^x where X>88.029692.

SET RESULT = +OMEGA.
SET e^x = +Ω.

FLOG 9
ALOG(0) OR ALOG10(0) NOT ALLOWED.
For $\log_e X$ or $\log_{10} X$ where X=0.

SET RESULT = −OMEGA.
Set $\log_e X$ or $\log_{10} X$ = −Ω.

FLOG 10
ALOG (−X) OR ALOG10 (−X) NOT ALLOWED.
For $\log_e X$ or $\log_{10} X$ where X < 0.

EVALUATE FOR +X
Evaluate for | X |.

FATN 11
ATAN2 (0,0) NOT ALLOWED.
For arctan (Y,X) where Y = 0, X = 0.

SET RESULT = 0.
Set angle = 0.

FSCN 12
SIN(X) OR COS(X), | X | GRT THAN OR EQ TO 2**25 NOT ALLOWED.
For sin(x) or cos(x) where $|X| \geq 2^{25}$.

SET RESULT = 0.
Set sin(X) = 0 or cos(X) = 0.

FSQR 13
SQRT (−X) NOT ALLOWED.
For $X^{1/2}$ where X <0.

EVALUATE FOR +X.
Evaluate for | X |.

FDX1 14
EXPONENTIATION ERROR 0**0.
For D^J where D=0, J=0.
Also, for Z^J where Z = 0 + 0i, J = 0.

SET RESULT =0.
Set $D^J = 0$ or set $Z^J = 0 + 0i$.

FDX1 15
EXPONENTIATION ERROR 0**(-J).
For D^J where D=0, J<0.
Also, for Z^J where Z = 0 + 0i, J < 0.

SET RESULT =0.
Set $D^J = 0$ or set $Z^J = 0 + 0i$.

FDX2 16
EXPONENTIATION ERROR (-B)**C.
For $D_1^{D_2}$ where $D_1<0$, $D_2 \neq 0$.

EVALUATE FOR +B.
Evaluate for $|D_1|$.

FDX2 17
EXPONENTIATION ERROR 0**0.
For $D_1^{D_2}$ where $D_1=0$, $D_2=0$.

SET RESULT =0.
Set $D_1^{D_2}=0$.

FDX2 18
EXPONENTIATION ERROR 0**(−C).
For $D_1^{D_2}$ where $D_1=0$, $D_2<0$.

SET RESULT =0.
Set $D_1^{D_2}=0$.

FDXP 19
DEXP(X), X GRT THAN 88.029692 NOT ALLOWED.
For e^x where X>88.029692.

SET RESULT = +OMEGA.
Set $e^x = +\Omega$.

FDLG 20
DLOG(0) OR DLOG10(0) NOT ALLOWED.
For $\log_e X$ or $\log_{10} X$ where X=0.

SET RESULT = −OMEGA.
Set $\log_e X$ or $\log_{10} X = -\Omega$.

FDLG 21
DLOG (−X) OR DLOG10 (−X) NOT ALLOWED.
For $\log_e X$ or $\log_{10} X$ where X < 0.

EVALUATE FOR +X.
Evaluate for | X |.

FDSQ 22
DSQRT(−X) NOT ALLOWED.
For $X^{\frac{1}{2}}$ where X<0.

EVALUATE FOR +X.
Evaluate for | X |.

FDSC 23
DSIN(X) OR DCOS(X), | X | GRT THAN OR EQ TO PI*2**50 NOT ALLOWED.
For sin(X) or cos(X) where $|X| \geq 2^{50}\pi$.

SET RESULT = 0.
Set sin(X) = 0 or cos(X) = 0.

FDAT 24
DATAN2(0,0) NOT ALLOWED.
For arctan (Y,X) where Y=0, X=0.

SET RESULT =0.
Set angle = 0.

FCXP 26
CEXP(X+IY), X GRT THAN 88.029692 NOT ALLOWED.
For e^{X+iY} where X>88.029692.

SET RESULT = OMEGA*(COSY+ISINY)
Set $e^{X+iY} = \Omega[\cos(Y) + i\sin(Y)]$.

FCXP 27
CEXP(X+IY), | Y | GRT THAN OR EQ TO 2**25 NOT ALLOWED.
For e^{x+iY} where $|Y| \geq 2^{25}$.

SET RESULT = 0 + 0I.
Set $e^{x+iY} = 0 + 0i$.

FCLG 28
CLOG(0+0I) NOT ALLOWED.
For $\log_e(X+iY)$ where X=0, Y=0.

SET RESULT = −OMEGA + 0I.
Set $\log_e(X+iY) = -\Omega+0i$.

FCSC 29
CSIN(X+IY) OR CCOS(X+IY), | X | GRT THAN OR EQ TO 2**25 NOT ALLOWED.
For sin(X + iY) or cos(X + iY) where $|X| \geq 2^{25}$.

SET RESULT = 0 + 0I.
Set sin(X + iY) or cos(X + iY) = 0 + 0i.

FCSC 30
CSIN (X+IY) OR CCOS (X+IY), | Y | GRT THAN 88.029692 NOT ALLOWED.
For sin(X + iY) or cos(X + iY) where | Y | > 88.029692.

REF IBLIB ERR MSG LIST FOR EVALUATION METHOD.

For Y > 88.029692, $\sin(X + iY) = \frac{\Omega}{2}(\sin X + i\cos X)$ and $\cos(X + iY) = \frac{\Omega}{2}(\cos X - i\sin X)$.

For Y < −88.029692, $\sin(X + iY) = \frac{\Omega}{2}(\sin X - i\cos X)$ and $\cos(X + iY) = \frac{\Omega}{2}(\cos X + i\sin X)$.

FIOH 31
FORMAT AT xxxxxx, FIRST WORD xxxxxx, HAS ILLEGAL CONTROL CHARACTER OR SPECIFIED TOO LONG A LINE.
Either invalid control character in FORMAT statement or too long a line is specified.

TREAT AS END OF FORMAT.

FCNV 32
ILLEGAL CHAR IN DATA BELOW OR BAD FORMAT.
Record containing invalid character written on-line following message.

TREAT ILLEGAL CHARACTER AS ZERO.

FCNV 33
ILLEGAL CHAR IN DATA BELOW OR BAD FORMAT.
Record containing invalid character written on-line following message.

TREAT ILLEGAL CHARACTER AS ZERO.

FIOS. 35
PERMANENT READ REDUNDANCY UNITxxx.
or
RECORD USED AS READ 100TH TIME.

FIOB 38
PHYSICAL RECORD SIZE EXCEEDS BUFFER SIZE.
This message pertains to binary input data.
PROCESS PORTION OF RECORD IN BUFFER.

FIOB 39
INTERNAL LABEL WORD COUNT DOES NOT MATCH IOCS WORD COUNT.
or
PROCESS RECORD READ.

FIOB 40
LIST EXCEEDS LOGICAL RECORD LENGTH.
or
STORE ZEROS IN REMAINING LIST ITEMS.

FIOS }
FIOS. } 41
END OF FILE READING UNITxx.
or
READ NEXT FILE.

FIOS }
FIOS. } 42
PERMANENT READ REDUNDANCY UNITxx.
or
RECORD USED AS READ THE 100th TIME.

FIOS 43
END-OF-BUFFER EXIT READING UNITxx.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FIOS 44
END-OF-BUFFER EXIT WRITING UNITxx.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FRCD 46
END-OF-FILE CARD READER.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FVIO }
FVIO. } 47
LOGICAL UNIT NOT DEFINED FOR VALUE xx.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FBST 48
PERMANENT READ REDUNDANCY UNITxx.
or
RECORD USED AS READ THE 100th TIME.

FBST 49
END-OF-BUFFER EXIT READING UNITxx.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FDMP 50
TAPE REDUNDANCY ON SYSUT4 ATTEMPTING TO WRITE MEMORY SAVE.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FSLITE 51
REFERENCE TO NONEXISTENT SENSE LIGHT.
(a) For I larger than 4 when *setting* the sense light.
(b) For I equal to 0 or larger than 4 when *testing* the sense light.
DECLARED 'OFF' IF TESTING. IGNORED IF SETTING.
(a) No action is taken.
(b) Set J equal to 2 (OFF).

FIOU 52
PUNCTUATION ERROR OR ZERO SUBSCRIPT.
NO OPTIONAL EXIT–EXECUTION TERMINATED.
Results from error conditions during read of BCD data referencing NAMELIST due to punctuation error in variable name, subscript, or literal, or to absence of a literal.

FSSWTH 53
NONEXISTENT SENSE SWITCH TESTED.
For I equal to 0 or larger than 6.
SWITCH DECLARED 'UP'.
Set J equal to 2 (OFF).

FIOS }
FIOS. } 54
WRITE REQUEST ON UNIT DEFINED AS SYSIN1 ILLEGAL.
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FXEM 55
ILLEGAL VALUE FOR COMPUTED GO TO AT IFN xxxxx. EXECUTION TERMINATED.

FIOS }
FIOS. } 56
READ REQUEST ON UNIT DEFINED AS SYSOU1 ILLEGAL.
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FCNV 57
ILLEGAL CHAR FOR L CONVERSION IN DATA BELOW.
Record containing invalid character written on-line following message.
Invalid character in logical input data.
TREAT ILLEGAL CHAR AS BLANK.

FIOU 58
NO DOUBLE PRECISION COMPLEX ALLOWED.
For (C_1, C_2) where C_1 and/or C_2 can be single-precision real numbers only.
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FIOU 59
NAMELIST NAME NOT FOUND OR PUNCTUATION ERROR.
For input variable name which does not match a NAMELIST name, possibly due to blank invalidly placed in variable name or misspelling.
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FIOU 60
EMBEDDED BLANKS
For input literals containing blanks.
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FIOU 61
NUMERICAL FIELD MISSING OR PUNCTUATION ERROR.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FIOU 62
UNEXPECTED END OF NAMELIST OR END OF RECORD.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FIOU 63
NAMELIST NAME NOT FOLLOWED BY MATCHING VARIABLE NAME.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FIOU 64
SUBSCRIPTS TOO LARGE OR TOO MANY SUBSCRIPTS OR INCONSISTENT DIMENSIONING.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FIOU 65
THE SIZE OF AN ARRAY HAS BEEN EXCEEDED.
or
READING OF ARRAY VALUES CONTINUES.
Literals are accepted and stored successively in locations following array block.

FIOU 66
LITERAL NOT PRECEDED BY VARIABLE NAME.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FIOU 67
PUNCTUATION MISSING FOR COMPLEX LITERAL.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FIOU 68
ILLEGAL CHARACTER IN LOGICAL INPUT DATA.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FIOU 69
DATA TYPE DOES NOT MATCH VARIABLE NAME.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FIOU 70
ILLEGAL CHARACTER OR ALL BLANKS IN LITERAL OR VARIABLE NAME BEGINS WITH NUMERICAL CHARACTER.
or
NO OPTIONAL EXIT–EXECUTION TERMINATED.

FCNV 71
INPUT DATA NOT WITHIN PERMISSIBLE RANGE OF FLOATING POINT NUMBERS.
or
READING OF INPUT DATA CONTINUES.

FASC 72
ARSIN(X) OR ARCOS (X), | X | GRT THAN 1 NOT ALLOWED.
For arcsin(X) or arccos(X), where | X | > 1.
SET RESULT = 0
Set arcsin(X) or arccos(X) = 0.

FTNC 73
TAN(X) OR COTAN(X), | X | GRT THAN OR EQ TO 2**20 NOT ALLOWED.
For tan(X) or cot(X) where $|X| \geq 2^{20}$.
SET RESULT = 0
Set tan(X) or cot(X) = 0.

FTNC 74
TAN(X) OR COTAN(X), X TOO CLOSE TO SINGULARITY, NOT ALLOWED.
For tan(X) where X is near an odd multiple of $\frac{\pi}{2}$ or cot(X) where X is near a multiple of π.
SET RESULT = +OMEGA
Set tan(X) or cot(X) = + Ω.

FSCH 75
SINH(X) OR COSH(X), | X | GRT THAN 88.029692 NOT ALLOWED
For sinh(X) = ½ ($e^x - e^{-x}$) or cosh(X) = ½ ($e^x + e^{-x}$) where $|X| > 88.029692$.
SET RESULT = +OMEGA
Set sinh(X) or cosh(X) = + Ω.

FGAM 76
GAMMA(X), X LESS THAN OR EQ TO 2**−127 OR GRT THAN OR EQ TO 34.843 NOT ALLOWED.
For $\Gamma(X) = \int_0^\infty u^{x-1} e^{-u}\, du$ where
$X \leq 2^{-127}$ or $X \geq 34.843$.
SET RESULT = +OMEGA
Set Γ(X) = + Ω.

FGAM 77
ALGAMA(X), X NON POSITIVE OR GRT THAN OR EQ TO 2.0593*10**36 NOT ALLOWED.
For $\log_e \Gamma(X) = \int_0^\infty u^{x-1} e^{-u}\, du$ where
$X \leq 0$ or $X \geq 2.0593(10^{36})$.
SET RESULT = +OMEGA
Set $\log_e \Gamma(X)$ = + Ω.

Added to the preceding messages, which result from the recognition of an FXEM error condition, the following FPTRP and FXEM messages may also be written:

FPTRP
UNDRFLOW AT ***** IN AC
or
UNDRFLOW AT ***** IN AC AND MQ
or
UNDRFLOW AT ***** IN MQ
or
OVERFLOW AT ***** IN AC
or
OVERFLOW AT ***** IN AC AND MQ
or
OVERFLOW AT ***** IN MQ
or
ADDRESS AT ***** ODD

FXEM
(NOTE: The following table is always contained in the FXEM message.)
ERROR TRACE CALLS IN REVERSE ORDER

CALLING ROUTINE	IFN OR LINE NO.	ABSOLUTE LOCATION
*****	******	******
*****	******	******
.	.	.
.	.	.
.	.	.
*****	******	******

(After the above table, one of the following three items is printed.)
ERROR CODE ***** NOT A STANDARD CODE.
or
A line of data containing an invalid character, if pertinent to the error.
or
A message from the routine that called the FXEM routine, if pertinent to the error.
(The following item may also appear.)
EXECUTION TERMINATED.

FOUT

END-OF-BUFFER EXIT WRITING SYSOU1. EXECUTION TERMINATED.

FBST

BACKSPACE REQUEST IGNORED ON UNIT**.

FEFT }
FEFT.}

REQUEST TO WRITE EOF ON UNIT ASSIGNED AS SYSIN1, SYSOU1, OR SYSPP1 HAS BEEN IGNORED.

FRWT }
FRWT.}

REQUEST TO REWIND UNIT ASSIGNED AS SYSIN1, SYSOU1, SPSPP1 HAS BEEN IGNORED.

FDMP

EXECUTION TERMINATED BY DUMP–DISK ERROR.

or

EXECUTION TERMINATED BY DUMP–UNUSUAL END SYSUT4.

or

EXECUTION TERMINATED BY DUMP–EWA FLAG ON–SYSUT4.

or

EXECUTION TERMINATED BY DUMP–LESS THAN 12 TRACKS ATTACHED TO SYSUT4.

DMPR

PLEASE SUPPLY CORRECT CALLING SEQUENCE FOR DUMP.

or

EXECUTION TERMINATED BY DUMP–SYSUT4 REDUNDANCY.

or

EXECUTION TERMINATED BY DUMP–UNUSUAL END–SYSUT4.

or

EXECUTION TERMINATED BY DUMP–DISK ERROR.

EXECUTION TERMINATED BY DUMP–EWA FLAG ON–SYSUT4.

or

EXECUTION TERMINATED BY DUMP–SYSOU1 REDUNDANCY.

or

EXECUTION TERMINATED BY DUMP–UNUSUAL END–SYSOU1.

or

SYSOU1 IS NOW TAPE ON XHK/S

COBOL Subroutine Messages

CBLER

PROCESSING TERMINATED–DATA ITEM REFERENCED BEFORE ITEM IS LOCATED.

An attempt has been made in the program to refer to (1) a data-item in a file that is not in OPEN status or (2) a data-item that follows a group defined by an OCCURS . . . DEPENDING ON data-name clause where data-name does not contain a non-zero value at the time of reference.

CEOBP

PROCESSING TERMINATED DUE TO TAPE CHECK SUM AND REDUNDANCY ERRORS.

or

PROCESSING TERMINATED DUE TO TAPE SEQUENCE AND REDUNDANCY ERRORS.

or

PROCESSING TERMINATED DUE TO UNRECOVERABLE TAPE REDUNDANCY ERRORS.

or

PROCESSING TERMINATED DUE TO TAPE CHECK SUM ERROR.

or

PROCESSING TERMINATED DUE TO TAPE SEQUENCE ERROR.

or

PROCESSING TERMINATED DUE TO TAPE RECORD LENGTH ERROR.

Each of the above error messages is followed by several lines of output of the following form:

THIS ERROR IS ASSOCIATED WITH AN I/O VERB AT CARD NUMBER ******. THE FOLLOWING INFORMATION IS ASSOCIATED WITH THE FILE IN ERROR.....
FILE NAME **********.
REEL SEQUENCE NUMBER....******.
FILE SERIAL NUMBER.....******.
FILE BLOCK COUNT.....******.

After each message is written, the job is terminated and a full core dump is taken.

CEXPR

EXPONENTIAL OVERFLOW AT CARD NUMBER ******.

Results from accumulator oveflow caused by floating-point number which exceeds maximum value allowed. The largest possible floating-point number is assumed as the result of the operation. Should an underflow occur, no message is written but the result is set to zero.

ERROR IN EXPONENTIAL AT CARD NUMBER ******.

In evaluating A**B, the above message is written when the exponent is not valid. In particular, a noninteger exponent may not be used with a negative base.

CBDCV

JOB TERMINATED–ENCOUNTERED INPUT RECORD LENGTH NOT A MULTIPLE OF SIX.
JOB TERMINATED–COUNT CONTROL CONTAINS A NON-NUMERIC BCD CHARACTER.

These messages are issued by CBDCV which processes the count control word from variable length records. When the message is written, the count control word is faulty in the manner indicated.

ACEPT

ACCEPT FROM ****** ENCOUNTERED END OF FILE. ZERO VALUE PROVIDED.

An EOF indication in the input during execution of an ACCEPT statement causes this message to be written. In the case of an ACCEPT from the card reader, the message results when card reader is empty. The area into which the data is to be accepted is loaded with zeros.

APPENDIXES

Appendix A: Control Card Format Index

Refer to the given page reference for a description of each card.

CARD FORMAT			PAGE REFERENCE
1 3			
$* any text			10
1		16	
$AFTER		srname	123
1	8	16	
m	*ALTER	n1	15
1	8	16	
m	*ALTER	n1,n2	15
1		16	
$ASSIGN		srname, ORG = nnnnn	123
1			
$CBEND			20
1			
$DATA			11
1		16	
$DELETE		srname	123
1			
*DEND			26
1 6	8	16	
$DUMP n	cxxxxx	loc1/loc2, loc3/loc4....	59
1		16	
$EDIT		[LOGIC]	122

CARD FORMAT		PAGE REFERENCE
1 8		
*ENDAL		15
1		
$ENDREEL		11
1 16		
$ENTRY	[{exname / deckname}]	10
1 16		
$ETC	variable field information	37
1 16		
$EXECUTE	subsystem name	8
1 16		
$FILE	'filename' [,unit1, unit2] [,{LIST / NOLIST}]	31
	[,{MOUNT / DEFER / READY / or / MOUNTi / DEFERi / READYi}] [,{INPUT / OUTPUT / INOUT / CHECKPOINT / or / CKPT}]	32
	[,{BLOCK / BLK}=xxxx]	
	[,ACT=xx] [,{ONEREEL / MULTIREEL / or / REELS}] [,{NOSEARCH / SEARCH}]	32
	[,{IIIGII / LOW / 200 / 556 / 800}] [,{BCD / BIN / MXBCD / MXBIN}] [,{SLABEL / HILABEL / LOLABEL / FLABEL}]	33

CARD FORMAT			PAGE REFERENCE
		[,{NOSEQ / SEQ / or / SEQUENCE}] [,{NOCKSUM / CKSUM}] [,{NOCKPTS / CKPTS}]	33
		[,AFTERLABEL] [,{SCRATCH / PRINT / PUNCH / HOLD}]	33
		[,{CYLINDER / or / CYL} = {x / xxx}]	
		[{CYLCOUNT / CYLCT} = {xx / xxx}] [,WRITECK] [,{HNRNFP / HRFP / HRNFP / HNRFP}]	34
1		16	
$GROUP		'filename$_1$', . . . 'filename$_n$' [,OPNCT=xx] [,BUFCT=xxx]	35
1	8	16	
$IBCBC	deckname	[,{NOLIST / LIST / FULIST}] [,{NOREF / REF}] [,{DECK / NODECK}]	19
		[,{M90 / M94 / M94/2}] [,{XR3 / XR7}] [,{INLINE / TIGHT}] [,{IOEND / READON}]	
		[,{COMSEQ / BINSEQ}] [,{NODD / DD}]	
1		16	
$IBDBL		[,TRAP MAX=n_1] [,LINE MAX=n_2] [,NOMES]	25
1	8	16	
$IBDBC	[name]	location [,FATAL]	25

CARD FORMAT			PAGE REFERENCE
1	8	16	
$IBFTC	deckname	[,{NOLIST / LIST / FULIST}] [,{DECK / NODECK}] [,{M90 / M94 / M94/2}] [,{XR3 / XRn}] [,{NODD / DD / SDD}]	17
1		16	
$IBJOB		[,{GO / NOGO}] [,{NOLOGIC / LOGIC / DLOGIC}] [,{NOMAP / MAP}] [,{NOFILES / FILES}] [,{SOURCE / NOSOURCE}] [,{IOEX / MINIMUM / BASIC / LABELS / FIOCS / ALTIO}] [,{FLOW / NOFLOW}]	8
1	8	16	
$IBLDR	deckname	[,{NOLIBE / LIBE}] [,{TEXT / NOTEXT}]	31
1	8	16	
$IBMAP	deckname	[, count] [,{LIST / NOLIST}] [,{REF / NOREF}] [,{DECK / NODECK}] [,{NOSYM / MONSYM / JOBSYM}] [,{M90 / M94 / M94/2}] [,{RELMOD / SYSMOD / ABSMOD}] [,{NO ( ) / ( ) OK}] [,{NOMFTC / MFTC}] [,{NODD / DD / SDD}]	22
1			
$IBREL			11
1			
$IBSYS			10
1	7-72		
$ID	any text		10

CARD FORMAT			PAGE REFERENCE
1		16	
$IEDIT		[,{SYSIN1 / SYSxxx}] [,{NOSRCH / SRCHn / SCHFn}] [,{NOALTER / ALTER}]	12
1		16	
$INCLUDE		{exname / deckname},...	42
1		16	
$INSERT		[srname] [, ORG=nnnnn]	123
1		16	
$JOB		any text	8
1		16	
$LABEL		'file name', [{serial / or / home address}], [reel], [{date / or / days}], [name]	34
1		16	
$NAME		[{deckname (exname) = exname,... / exname = exname / 'deckname (filename)' = 'filename',.. / 'filename' = 'filename',...}]	36
1		16	
$OEDIT		[,{SYSOU1 / SYSxxx}] [,{NOPREST / PREST}] [,{NOCPR / CPREST}]	13
1		16	
$OMIT		{exname / deckname (exname),...}	36
1		16	
$ORIGIN		logical origin [absolute / 'origin] [{SYSUT2 / SYSxxx}] [{NOREW / REW}]	41
1	8	16	
$PATCH	cxxxxx	instr. 1, instr. 2,...	60

CARD FORMAT			PAGE REFERENCE
1		16	
$PAUSE		instructions to operator	10
1		16	
$POOL		'filename$_1$', . . . 'filename$_n$' $\left[,\begin{Bmatrix}\text{BLOCK}\\\text{BLK}\end{Bmatrix}=\text{xxxx}\right]$ [, BUFCT=xxx]	35
1			
$POST			11
1		16	
$REPLACE		srname [, ORG=nnnnn]	122
1		16	
$SIZE		/ /=n	36
1			
$STOP			10
1	8	16	
$TITLE	[NODAT]	any text	11
1		16	
$USE		deckname (exname), . . .	36

Appendix B: Control Card Check List

	SOURCE LANGUAGE PROGRAMS			RELOCATABLE BINARY	
	COBOL	FORTRAN	MAP	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 routine.
$EXECUTE	x	x	x	x	Causes the loading of the Processor Monitor.
$*	o	o	o	o	Comments card.
$PAUSE	o	o	o	o	Permits operator action.
$STOP	o	o	o	o	Transfers control to the System Monitor for processing.
$IBSYS	o	o	o	o	Next job segment will not be processed by the IBJOB Processor; control is transferred to the System Monitor.
$IBJOB	x	x	x	x	Initiates an IBJOB Processor application; one required for each Processor application.
$ENDREEL	o	o	o	o	Causes a reel switch involving SYSIN1 and SYSIN2.
$DATA	o	o	o	o	Indicates the beginning of a data file.
$IBREL	o	o	o	o	Indicates that the decks that follow do not need compiling or assembling.
$TITLE	o	o	o	o	Causes the information in columns 16-72 to be printed on the next Processor or Assembler listing output.
$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 program to be loaded.
$ENTRY	o	o	o	o	Specifies the location to which the initial transfer to the object program will be made.
$IEDIT	o	o	o	o	Sets input specifications other than standard.
$OEDIT	o	o	o	o	Sets output specifications other than standard.
$FILE	o	o	o	o	Provides file specification; supersedes some assembled specifications.
$LABEL	o	o	o	o	Provides label information for files.
$POOL	o	o	o	o	Designates files to share common buffer areas.
$GROUP	o	o	o	o	Designates how buffers are to be shared by a group of files.
$NAME	o	o	o	o	Used to change control section names of file names.
$USE	o	o	o	o	Specifies that a particular control section is to be used.
$OMIT	o	o	o	o	Specifies that a particular control section is to be deleted.
$SIZE	o	o	o	o	Specifies the size of Blank COMMON.
$ETC	o	o	o	o	Extends the variable field of a $FILE, $POOL, $GROUP, $USE, $OMIT, $NAME, or $ETC card.
$ORIGIN	o	o	o	o	Used to define the structure of an overlay deck.
$INCLUDE	o	o	o	o	Specifies the decks or control sections to be included in a link.
$AFTER	o	o	o	o	Causes the Librarian to copy the Library from its current position through the named subroutine.
$ASSIGN	o	o	o	o	Causes the Librarian to copy the current Library up to, but not including, the named subroutine.
$DELETE	o	o	o	o	Causes the Librarian to copy the Library from its current position up to, but not including, the named subroutine.
$EDIT	o	o	o	o	Causes the Librarian to be called for an editing run.
$INSERT	o	o	o	o	Causes the Librarian to place the subroutine deck that follows the card into the Library at the current position of the Library file.
$REPLACE	o	o	o	o	Permits a subroutine in the Subroutine Library to be replaced.
$IBDBC	x				Precedes each compile-time debugging request and defines the point where the request is to be executed.
$IBDBL	o	o	o	o	Precedes each load-time debugging request packet.
*DEND	o	o	o	o	Terminates the load-time debugging package.
$POST		o	o	o	Causes the load-time debugging postprocessor routines to be called.
*ALTER	o	o	o	o	Used to alter a source, symbolic, or Prest deck.
*ENDAL	o	o	o	o	Required to end an alter deck.
$DUMP					Causes portions of system records to be dumped.
$PATCH					Used to insert temporary patches in system records.

Notation: x—necessary; o—optional; blank—does not apply.

Appendix C: IBJOB Communication Region

The components of the IBJOB Processor System transfer control and information to each other and to the IBSYS system through specified words in the IBJOB communication region. These words are:

SYMBOL	LOCATION RELATIVE TO COMMUNICATION REGION BASE	OCTAL LOCATION
SYSLOC	IBJCOM	21234
SYSFAZ	IBJCOM+1	21235
IBJCOR	IBJCOM+2	21236
IBJDAT	IBJCOM+3	21237
.JLDAT	IBJCOM+4	21240
.JTYPE	IBJCOM+6	21242
.JLIN	IBJCOM+7	21243
.JVER	IBJCOM+8	21244
.JKAPU	IBJCOM+9	21245
SYSDSB	IBJCOM+10	21246
.FDPOS	IBJCOM+12	21250
SSTRA	IBJCOM+15	21253
ACTION	IBJCOM+16	21254
JOBIN	IBJCOM+17	21255
JOBOU	IBJCOM+18	21256
JOBPP	IBJCOM+19	21257
IOEDIT	IBJCOM+20	21260
JREEL	IBJCOM+21	21261
SUBSP	IBJCOM+22	21262
PUNCH	IBJCOM+23	21263
SYSSHD	IBJCOM+24	21264
LILDMP	IBJCOM+25	21265
IBSLB	IBJCOM+26	21266
PRSW	IBJCOM+27	21267
JLNSIZ	IBJCOM+28	21270
INPOP	IBJCOM+29	21271
SUBSYS	SUBSYS	21412
DEFINE	IOCS	21347
JOIN	IOCS+2	21351
ATTACH	IOCS+4	21353
CLOSE	IOCS+6	21355
OPEN	IOCS+8	21357
READ	IOCS+10	21361
WRITE	IOCS+12	21363
STASH	IOCS+14	21365

The contents of some of the more important IBJOB communication region locations are described in the following two lists. The first list contains locations available only up until execution of object programs. The octal address of the location is given in column 1. Column 2 gives the symbolic name of the location. The third column provides information regarding the use of the location.

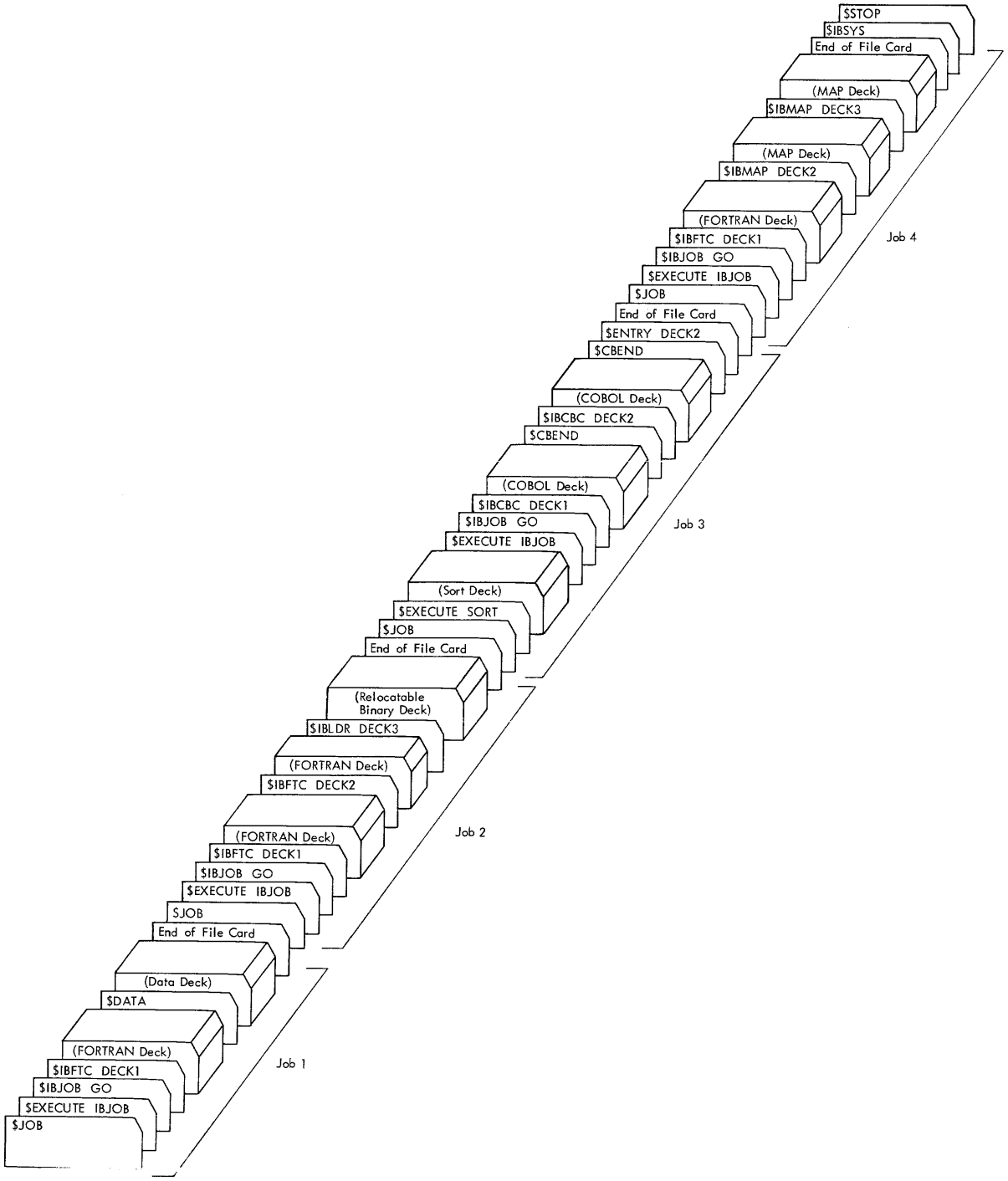
OCTAL LOCATION	SYMBOLIC NAME	REMARKS
21263	PUNCH	Set to nonzero by IBMAP if NODECK option is requested by a compilation or an assembly. This location is tested by the JOBPP punching routine in the IBJOB Monitor. It is set to zero if a punched deck is wanted.
21264	SYSSHD (System subhead)	Set by a subsystem under the IBJOB Monitor. Contains the location and length of a subheading to be used in the listing.
21267	PRSW (Print switch)	Set to nonzero on the distributed IBJOB system tape. It may be changed at an installation. If the location has a value of zero, most IBJOB Monitor messages will occur on-line as well as off-line. If the location has a nonzero value, on-line printing of IBJOB messages is minimized.
21272	COMCEL (Communication word)	Set and used by the IBJOB Monitor. Contains flag bits relating to control card options, input/output editor uses, loading, execution, and other functional operations.
21273	EOFPP (End of file on peripheral punch)	Set to nonzero on the distributed IBJOB system tape. It may be changed at an installation. If the location has nonzero value, end-of-file mark is put on the peripheral punch tape at the end of each Processor application (that is, the job is contained between a $IBJOB card and the next end of file, including the DATA file if any). If the value is zero, no end-of-file mark is put on the peripheral punch tape.
21301	DECK	Contains the current deck name from columns 8-13 of the last deck control card (for example, $IBLDR, or $IBFTC) encountered by IBJOB.
21315	JBNAM (Job name)	Contains job name from columns 8-13 of last $IBJOB control card.
21323	TYPOU (Type of output)	Determines type of output to be generated. When the location has a value of zero, output is in BCD mode, with blocking of up to five lines per block. When the location has a nonzero value, output is in binary mode, with blocking of up to five lines per block. This output can be printed off-line.

The second list of IBJOB communication words contains locations available both before and after object program loading. In this table, column 1 contains the octal address of the location only up until execution.

Column 2 gives the symbolic name of the location. In two cases the symbolic name is different after loading. These changes are given in parentheses.

OCTAL LOCATION	SYMBOLIC NAME	REMARKS
21234	SYSLOC	Contains the complemented location of last MAP language CALL pseudo-operation that has been executed.
21235	SYSFAZ	Contains record name of last system read. At execution time, this location contains .OBJPR.
21236	IBJCOR (.JOR)	Contains upper and lower limits of core storage currently available to IBJOB.
21237	IBJDAT (.JDATE)	Contains the date appearing in the IBSYS date location SYSDAT. Format is YY DDD, where YY is year and DDD is day of year.
21240 21241	.JLDAT	Contains same date as IBJDAT but in form MM/DD/YY, where MM is month and DD is day of month.
21242	.JTYPE	Contains zero if system is on 7090 and nonzero if it is on 7094.
21243	.JLIN	Contains count of lines output on SYSOU during current Processor application.
21244	.JVER	Contains version number of IBJOB system in the form: BCI 1,VERxxx
21246	SYSDSB	Contains an enable instruction from a location containing zero and is used by a variation of the SAVE pseudo-operation to disable traps.
21250	.FDPOS	Contains location of FORTRAN dump record. Prefix is device type (PZE for 729, MZE for 1301, or PTW for 7340). Tag is SYSUNI index for the unit. If record is on tape decrement and address are file and record positions. If record is disk, address is track address and there is no decrement.

Appendix D: Sample Control Card Deck



Appendix E: Procedure for Selecting the 7094 Optional Conversion Routine

A new release procedure provided with Version 5 of the Subroutine Library permits the 7094 customer to select the optional conversion routine (FCNV).

Both the standard conversion routine and the optional conversion routine are on the released symbolic tape. The standard conversion routine is automatically provided when creating a system tape, unless otherwise specified by the customer.

In order to specify the optional conversion routine, the following three cards must be included in the special deck for 7094 IBLIB assembly used to create the new 7094 system tape. The format of these cards is:

1	8	16	72
M9094	SET	94	3F300015
M9094	SET	94	3F4C0015
M9094	SET	94	3F4M0015

The inclusion of these three cards causes the optional conversion routine FCNV, as well as modified versions of the routines FIOH and FWRO, to be selected from the symbolic tape.

NOTE: The 7090/7094 user will receive another deck (called the 7090 Asterisk Deck) which will generate asterisks if an insufficient field width is specified in a program using the standard 7090 conversion routines. Output through the standard 7090/7094 conversion routines will thus be made consistent with output from the 7094 optional conversion routine. Appendix H contains instructions on how to use the 7090 Asterisk Deck.

Appendix F: Core Storage Load Map

```
                    * MEMORY MAP *

SYSTEM                                  00000 THRU  02717
FILE BLOCK ORIGIN                       02720
FILES       1.     INFILE
            2.     OUFILE
FILE LIST ORIGIN                        02750
PRE-EXECUTION INITIALIZATION            02754
CALL ON OBJECT PROGRAM                  02777
OBJECT PROGRAM                          03004 THRU  12573

LINK  DECK  ORIGIN      CONTROL SECTIONS  (/NAME/=NON 0 LENGTH, (LOC)=DELETED, *=NOT REFERENCED)

  0  DECKA   03004       H       03051     CONSTA  03140     CONSTB  03141     CONSTC  03142     CONSTD  03143
                         G       03154
     .LINK   03173      /.LDT  / 03173    /.LRECT/ 03177    /.LVEC / 03205
     .LXCON  03217       .LXSTR  03217      .LXSTP  03223      .LXOUT  03271 *    .LXRTN  03303      IBEXIT  03303 *
                         .LXCAL  03306 *    .LXERR  03306      .DBCLS  03501 *    .LXARG  03650 *    .LO     03673 *
                         .CLSE   03701      .LFBL   03702 *    .LUNB   03703      .DFOUT  03704
     .IODEF  03710       .DEFIN  03710      .ATTAC  03714 *    .CLOSE  03716      .OPEN   03720      .READ   03722
                         .WRITE  03724      .BSR    03734 *    .READR  03744      .RELES  03746 *    .LAREA  03757
                         .LFBLK  03775      .LTSX   04000 *    .AREA1  04012      .LUNBL  04020      .ENTRY  04024
                         .GOA    04055      .GO     04061      .DERR   04075      .NOPXI  04076      .COMXI  04100
                         .EX34   04122
     .LOVRY  04127       .LOVRY (04127)     .LDT   (03173)     .LRECT (03177)     .LVEC  (03205)
     .LXSL   04506       .LXSEL  04506      .LXTST  04521 *    .LXOVL  04561 *    .LXRCT  04567 *    .LXIND  04636
                         .LXDIS  04641      .LXFLG  04642      .LTCH   04643
     .FPTRP  04650       .FFPT.  04650 *    .FPOUT  04777 *    .FPARG  05005 *   /.COUNT/ 05007 *    OVFLOW  05053 *
     XIT     05060       EXIT    05060      .EXIT.  05060 *
     FXEM    05061       .FXEM.  05061      .FXOUT  05411 *    .FXARG  05417 *   /.OPTW./ 05473 *
     FDMP    05504       DUMP    05504      PDUMP   05506 *
     .IOCS   06742       .L(O)   06742      .MONSW  06762      .TEOR   07031      .DEFI.  07111      .JOINX  07155 *
                         .CLOS.  07174      .ATTC.  07207      .SH1    07421 *    .SH9    07463 *    .OPEN.  07504
                         .OP4    07532 *    .OP7    07563 *    .OP9.2  07577 *    .RLSE.  07643      .RER2.  07643
                         .READ.  07644      .RER1.  07667      .WRIT.  07671      .MNTIA  10057 *    .EOFBX  10140 *
                         .FEEIT  10210      .GTIOX  10231      .RW7    10347 *    .RE7    10766 *    .ENDTR  11427
                         .SEL59  11431 *    .BSR.   12042      .EOTOF  12165      .ETOF3  12173 *    .SWITC  12222
                         .TCHEX  12523      .BASIO  12526 *
     .IOCSM  12527

  1  DECK1   12527      /RTNEA / 12527

  2  DECK2   12527      /RTNEB / 12527    /RTNEC / 12547
     DECK3   12555      /RTNED / 12555

  3  DECK4   12565      /RTNEE / 12565

I/O BUFFERS                             12574 THRU  77765

UNUSED CORE                             77766 THRU  77777
```

A load map of core storage prior to object program execution is generated by the Loader in response to the MAP option punched on a $IBJOB card. In the "System" section of the load map are contained those parts of the IBSYS Monitor, IOCS, and IOEX needed for job-to-job operation. "File Block Origin" refers to 12-word working file blocks, one block for each file named in the program. The "File List" contains two-word entries, one entry for each file. The contents of the "Pre-Execution Initialization" section of the load map are described in the "Loader Information" section of this manual under "Control of Program Execution." The CALL on the object program is also described there. The object program itself consists of the instructions as generated by the Assembler or the FORTRAN IV Compiler supplemented by subroutines loaded from the Subroutine Library. In the load map shown, the subroutine names begin with .LINK. The column entitled "LINK" refers to overlay links.

The control sections in each deck are listed in rows of five. The names of the control sections with lengths greater than zero are bounded by slashes. The names of control sections that are not referred to during execution are followed by asterisks. A deleted control section would appear as follows:

```
CONSTA    (02763)
```

An expanded EVEN control section would appear as follows:

```
EVEN    02763      CONSTA    02764
```

Appendix G: Machine Configuration Required for IBJOB Processor Operation

The following machine configuration is required for the operation of the IBJOB Processor:

An IBM 7090 or 7094 Data Processing System

An IBM 716 Printer

An IBM 711 Card Reader

Required for Installations Using Only IBM 729 (II, IV, V, or VI) Magnetic Tape Units and/or IBM 7340 Hypertape Drives: Eight units are required ordinarily. If an IBM 1401 with its attached IBM 1402 Card Read Punch and IBM 1403 Printer is available for processing system output, and a single tape unit is assigned by the System Monitor to both SYSOU1 and SYSPP1 (list and punch functions), only seven units are required. When load-time debugging is used, an additional unit is necessary, attached as SYSCK2.

Required for Installations using IBM 1301/2302 Disk Storage or IBM 7320 Drum Storage: Four tape units are required ordinarily. IBM 729 Magnetic Tape Units or IBM 7340 Hypertape Drives can be assigned in any combination. If an IBM 1401 with its attached card read punch and printer is available for processing system output and a single tape unit is assigned by the System Monitor to both SYSOU1 and SYSPP1 (list and punch functions), only three units are required.

Five other units are required ordinarily. IBM 729 Magnetic Tape Units, IBM 7340 Hypertape Drives or selected cylinders of IBM 1301/2302 Disk Storage or of IBM 7320 Drum Storage can be assigned to the installation in any combination. When load-time debugging is used, an additional unit is necessary, attached as SYSCK2.

Appendix H: FORTRAN IV Mathematics Subroutines—Algorithms, Accuracy, and Speeds

This appendix presents the algorithms, statistics on accuracy, and average speed for most of the FORTRAN IV mathematics subroutines.

Algorithms

Some of the formulas are widely known; those that are not widely known are derived from more commonly known formulas. The steps leading from the common formulas have been detailed so that derivation can be reconstructed by anyone who has a basic understanding of mathematics and who has access to the common texts on numerical analysis. Background information for algorithms involving continued fractions may be found in the publication entitled *Analytic Theory of Continued Fractions*, written by H. S. Wall and published in 1948 by the D. Van Nostrand Co., Inc., of Princeton, N. J.

Accuracy

Because the size of a machine word is limited, small errors may be generated by mathematical subroutines. In an elaborate computation, slight inaccuracies can accumulate to become larger errors. Thus, in interpreting final results, the user should take into account any errors introduced during the various intermediate stages.

The accuracy of an answer by a subroutine is influenced by two factors: (1) the accuracy of the argument and (2) the performance of the subroutine.

Accuracy of the Argument

Most arguments contain errors. An error in a given argument may have accumulated over several steps prior to the use of the subroutine. Even data fresh from input conversion contain slight errors since decimal data cannot usually be exactly converted into the binary form required by the processing unit; the conversion process is usually only approximate. Argument errors always influence the accuracy of answers. The effect of an argument error on the accuracy of an answer depends solely on the nature of the mathematical function involved and not on the particular coding by which that function is computed within a subroutine. In order to assist users in assessing the accumulation of errors, a guide on the propagational effect of argument errors is provided for each function. Wherever possible, this is expressed as a simple formula.

Performance of the Subroutine

The performance statistics supplied in this appendix are based upon the assumption that arguments are perfect (i.e., without errors, and therefore have no argument error propagation effect upon answers). Thus, the only errors in answers are those introduced by the subroutines themselves.

For each subroutine, accuracy figures are given for one or more segments throughout the valid argument range(s). The particular statistics given are those most meaningful to the function and range under consideration. For example, the maximum relative error and standard deviation of the relative error of a set of answers are generally useful and revealing statistics, but useless for the range of a function where its value becomes 0, since the slightest error of the argument value can cause an unbounded fluctuation on the relative magnitude of the answer. Such

is the case with $\sin(x)$ for x near π, and in this range it is more appropriate to discuss absolute errors.

Symbols Used in Describing Accuracy

In the presentation of error statistics, the following symbols are employed:

$g(x)$ = the answer given by the subroutine for the mathematical function under discussion

$f(x)$ = the correct extra-precision answer for the mathematical function under discussion

$\epsilon = \left| \dfrac{f(x) - g(x)}{f(x)} \right|$, the relative error of the answer

δ = the relative error of the argument

$E = \left| f(x) - g(x) \right|$, the absolute error of the answer

Δ = the absolute error of the argument

$M(E) = \text{Max} \left| f(x) - g(x) \right|$, the maximum absolute error produced during testing

$M(\epsilon) = \text{Max} \left| \dfrac{f(x) - g(x)}{f(x)} \right|$, the maximum relative error produced during testing

$\sigma(E) = \sqrt{\dfrac{1}{N} \sum_i \left| f(x_i) - g(x_i) \right|^2}$, the root-mean-square (standard deviation) absolute error

$\sigma(\epsilon) = \sqrt{\dfrac{1}{N} \sum_i \left| \dfrac{f(x_i) - g(x_i)}{f(x_i)} \right|^2}$, the root-mean-square (standard deviation) relative error

When applied to complex numbers, the absolute value signs in the above formulas should be regarded as denoting complex absolute value. Thus, the above formula for E represents the vector error when applied to a complex function.

Algorithms, Accuracy, and Speeds

The algorithms and performance statistics for most of the FORTRAN IV mathematics subroutines are shown in the following section. The subroutines appear in the same order as in Figures 25A, 25B, 25C, and 26. The subroutines not described are FXP1, FXP2, FXP3, FDX1, FDX2, and FDMD.

Single-Precision Subroutines

The following information describes single-precision subroutines listed in Figure 25A.

Square Root — FSQR

Algorithm

1. Write

   $x = (2^{2p-q})m$, where p is an integer, $q = 0$ or 1, and $\frac{1}{2} \leq m < 1$.

   Then

   $\sqrt{x} = 2^p \sqrt{(2^{-q})m}$, where $\frac{1}{4} \leq (2^{-q})m < 1$.

2. Take the first approximation y_0 to be

$y_0 = 2^p \left(\frac{m}{2} + \frac{1}{2}\right)$, if $q = 0$, and

$y_0 = 2^p \left(\frac{m}{2} + \frac{1}{4}\right)$, if $q = 1$.

The relative error of this approximation is less than 2^{-4}.

3. Apply the Newton-Raphson iteration 3 times to y_0 as follows:

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{x}{y_n}\right).$$

Using the rule $\epsilon_{n+1} \sim \frac{1}{2}\epsilon_n^2$, the relative error of y_3 is down to 2^{-39}.

Effect of Argument Error

$\epsilon \sim \frac{1}{2}\delta$.

Performance Statistics

Performance statistics for the single-precision square root subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error σ (E)	Maximum Relative Error M (E)	Average Speed in Microseconds	
			7090	7094
$X > 10^{-80}$	3.09×10^{-9}	7.25×10^{-9}	238	149
Accuracy is the same for arguments less than 10^{-80}, but speed is slower because of floating-point underflow.				

The sample arguments upon which these performance statistics are based were exponentially distributed over the specified range.

Exponential — FXPF

Algorithm:

1. Write

$x\,[\log_2 (e)] = n + r$, where n is the integer part and r is the fraction part.

Then

$e^x = (2^n)\,(2^r)$, where $-1 < r < 1$.

2. Compute 2^r by the means of a rational approximation formula where $-1 < r < 1$. This formula was derived in the following way. Take the Gaussian continued fraction

$$e^z = \frac{1}{1} - \frac{z}{1} + \frac{z}{2} - \frac{z}{3} + \frac{z}{2} - \frac{z}{5} + \frac{z}{2} - \frac{z}{7} + \frac{z}{2} - \ldots,$$

truncate at the ninth term and rewrite to obtain

$$e^z \cong \frac{1680 + 840z + 180z^2 + 20z^3 + z^4}{1680 - 840z + 180z^2 - 20z^3 + z^4}$$

Substituting $r\,[\log_e (2)]$ for z and rewriting the above, we get

$$2^r \cong 1 + \frac{2r}{cr^2 - r + D - \frac{B}{r^2 + A}}$$, where A, B, C, and D are constants.

The maximum relative error of this formula is 1.6×10^{-9}.

3. If $x < -89.415987$, 0 is given as the answer.
4. The computation is carried out in fixed-point to minimize truncation errors.

Effect of Argument Error

$\epsilon \sim \Delta$. Since $\Delta = \delta \cdot x$, for the larger value of x, even the round-off error of the argument causes a substantial relative error in the answer.

Performance Statistics

Performance statistics for the single-precision exponential subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$0<X<1$	3.36×10^{-9}	7.32×10^{-9}	288	188
$-88.028<X<88.028$	4.80×10^{-9}	1.50×10^{-8}	288	188

The sample arguments upon which the above statistics are based were uniformly distributed over the specified range.

Logarithm — FLOG

Algorithm

1. If $|1 - x| < 2^{-7}$, use the polynomial approximation

   $\log(1 + z) \cong z - [2^{-1} + 3(2^{-18})]z^2 + \frac{1}{3}z^3$, where $z = x - 1$.

The maximum relative error of this formula for $|z| < 2^{-7}$ is 3×10^{-8}.

2. If $|1 - x| \geq 2^{-7}$, reduce the case as follows:

   Write

   $x = (2^p)\,m$, where $\frac{1}{2} \leq m < 1$ and $z = \left(m - \frac{1}{\sqrt{2}}\right) \Big/ \left(m + \frac{1}{\sqrt{2}}\right)$.

   Then

   $|z| < 0.1716$.

   Also

   $$\frac{1+z}{1-z} = \left(\sqrt{2}\right) m.$$

   And

   $$\log_e x = \left[p - \tfrac{1}{2} + \log_2\left(\frac{1+z}{1-z}\right)\right] \log_e 2.$$

3. By transforming the Taylor series into a continued fraction (see Wall's *Analytic Theory of Continued Fractions*, page 196), we obtain

$$\log_e\left(\frac{1+z}{1-z}\right) = 2z\left[1 + \tfrac{1}{3}z^2 + \frac{\frac{z^2}{5}}{-\frac{5}{7} + z^{-2} + w(z)}\right].$$

Replacing the remainder term $w(z)$ with its approximate value of

$-\frac{11}{(7)(9)(140)}$ for

$z = \sqrt{\frac{(7)(11)}{2843}}$, we obtain the approximation

$$\log_e\left(\frac{1+z}{1-z}\right) \cong \frac{(3^3)(4)(5)(7^2) - (3)(3371)(z^2) - (1019)(z^4)}{(3^3)(4)(5)(7^2) - (3)(6311)(z^2)}(2z). \quad (*)$$

This reduces to the form

$$\log_2\left(\frac{1+z}{1-z}\right) \cong z\left[c_1 + c_2 z^2 + \frac{c_3}{z^2 + c_4}\right].$$

The maximum relative error of the formula (*) is 0.62×10^{-9} for $|z| < 0.1716$. However, since the procedure in item 2 above may involve cancellation of significant digits, this relative accuracy cannot be maintained for the final result if the argument is near 1.

Effect of Argument Error

$E \sim \delta$. In particular, if δ is the round-off error of the argument, say $\delta \sim 7 \cdot 10^{-9}$, then $E \sim 7 \cdot 10^{-9}$. This means that if the argument is close to 1, the relative error can be very large, since the function value is very small.

Performance Statistics

Performance statistics for the natural logarithm function of the single-precision logarithm subroutine are as follows:

Argument Range	Root-Mean-Square Absolute Error $\sigma(E)$	Maximum Absolute Error $M(E)$	Average Speed in Microseconds	
			7090	7094
$15/16 < x < 17/16$	1.18×2^{-33}	1.20×2^{-31}	361	208

The sample arguments upon which the above statistics are based were uniformly distributed over the specified range.

Argument Range	Root-Mean-Square Relative Error $\sigma(E)$	Maximum Relative Error $M(E)$	Average Speed in Microseconds	
			7090	7094
All positive numbers outside $(15/16, 17/16)$	3.21×10^{-9}	7.24×10^{-9}	390	226

The sample arguments on which the above statistics are based were exponentially distributed over the specified range.

Performance statistics for the common logarithm function of the single-precision logarithm subroutine are as follows:

Argument Range	Root-Mean-Square Absolute Error $\sigma(E)$	Maximum Absolute Error $M(E)$	Average Speed in Microseconds	
			7090	7094
$15/16 < x < 17/16$	1.95×2^{-34}	1.03×2^{-31}	405	234

The sample arguments on which the above statistics are based were uniformly distributed over the specified range.

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
All positive numbers outside $15/16, 17/16$	4.70×10^{-9}	1.57×10^{-8}	434	252

The sample arguments upon which the above statistics are based were exponentially distributed over the specified range.

Sine/Cosine — FSCN

Algorithm

1. $\sin(-x) = -\sin(x), \cos(-x) = \cos(x)$. Assume $x \geq 0$.
2. Write

   $x = \frac{\pi}{4}(q) + r$, where q is an integer and $0 \leq r < \frac{\pi}{4}$. Let $q_0 \equiv q \pmod 8$.

3. If $\cos(x)$ is desired, raise q_0 by 2, reduce it modulo 8 and compute sine. If $\sin(x)$ is desired and if $x < 2^{13}$, give x as the answer.
4. Now the case is reduced to the computation of $\sin\left(\frac{\pi}{4} q_0 + r\right)$, where $0 \leq q_0 \leq 7$.

Using the formulas

$$\sin\left[\frac{\pi}{4}(4+q_0)+r\right] = -\sin\left[\frac{\pi}{4}q+r\right], \text{ where } 0 \leq q_0 \leq 3$$

$$\sin\left[\frac{\pi}{4}+r\right] = \cos\left[\frac{\pi}{4}-r\right]$$

$$\sin\left[\frac{\pi}{2}+r\right] = \cos(r)$$

$$\sin\left[\frac{3\pi}{4}+r\right] = \sin\left[\frac{\pi}{4}-r\right],$$

the case is reduced to the computation of $\sin(r)$ or $\cos(r)$ for $0 \leq r \leq \frac{\pi}{4}$.

5. The coefficients of approximation

$$\sin(r) \cong r(s_0 + s_1r^2 + s_2r^4 + s_3r^6)$$

were obtained by the Chebyshev interpolation over the range $0 \leq r \leq \frac{\pi}{4}$.

The coefficients of approximation

$$\cos(r) \cong 1 + c_1r^2 + c_2r^4 + c_3r^6 + c_4r^8$$

were obtained by the Chebyshev interpolation over the range $-0.01 \leq r \leq \frac{\pi}{4}$.

The relative error of the sine formula is less than 0.34×10^{-8}. The relative error of the cosine formula is less than 0.73×10^{-10}.

6. The computations are carried out in fixed-point to minimize truncation errors.

Effect of Argument Error

$E \sim \Delta$. As the argument gets larger, Δ grows, and since the function value is periodically diminishing, no consistent relative error control can be maintained outside the principal range $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$. This holds true for the cosine functions as well.

Performance Statistics

Performance statistics for the sine function of the single-precision sine/cosine subroutine are as follows:

Argument Range	Root-Mean-Square Absolute Error $\sigma(E)$	Maximum Absolute Error $M(E)$	Average Speed in Microseconds	
			7090	7094
$\lvert x\rvert \leq \frac{\pi}{2}$	1.12×2^{-29}	1.00×2^{-27}	381	240
$\frac{\pi}{2} < \lvert x\rvert \leq 10$	1.51×2^{-29}	1.65×2^{-27}	418	260
$10 < \lvert x\rvert \leq 100$	1.50×2^{-29}	1.69×2^{-27}	415	258

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$\lvert x\rvert \leq \frac{\pi}{2}$	3.65×10^{-9}	1.34×10^{-8}	381	240

The sample arguments on which the above statistics are based were uniformly distributed over the specified range.

The performance statistics for the cosine function of the sine/cosine subroutine are as follows:

Argument Range	Root-Mean-Square Absolute Error σ (E)	Maximum Absolute Error M (E)	Average Speed in Microseconds	
			7090	7094
$0 \leqslant x \leqslant \pi$	1.56×2^{-29}	1.72×2^{-27}	429	267
$-20 \leqslant x \leqslant 0$, $\pi \leqslant x \leqslant 20$	1.53×2^{-29}	1.60×2^{-27}	430	266

The sample arguments on which the above statistics are based were uniformly distributed over the specified range.

Tangent/Cotangent — FTNC

Algorithm

1. If $x < 0$, use

$\tan(x) = -\tan(x)$,
$\cot(-x) = -\cot(x)$.

Assume $x \geq 0$ now.

2. Write

$x = \frac{\pi}{4} q + r$, where q is an integer and $0 \leq r < \frac{\pi}{4}$.

Let $q_0 \equiv q \pmod 4$.

3. If $q_0 = 0$ or 2 (i.e., octant 1 and 3), define $r_0 = r$.

If $q_0 = 1$ or 3 (i.e., octant 2 and 4), define $r_0 = \frac{\pi}{4} - r$.

4. Define the case number s as follows:

If $\tan(x)$ is desired,

$s = q_0$.

If $\cot(x)$ is desired,

$s = 1$, if $q_0 = 0$,
$s = 0$, if $q_0 = 1$,
$s = 3$, if $q_0 = 2$,
$s = 2$, if $q_0 = 3$.

5. Compute the factor F as follows:

$$F = 1 + \frac{13.946\, r_0^2 - 313.11}{r_0^2 - 104.46 + \frac{939.33}{r_0^2}}, \text{ if } r_0 > 2^{-14}.$$

$F = 1$, if $r_0 \leq 2^{-14}$.

This approximation can be obtained by rewriting the continued fraction

$$\frac{\tan(r_0)}{r_0} \cong \frac{1}{1 - \frac{r_0^2}{3 - \frac{r_0^2}{5 - \frac{r_0^2}{7 - \frac{r_0^2}{8.946}}}}} .$$

The maximum relative error of this formula is 10^{-9}.

6. Now the answer is $\frac{r_0}{F}$ for $s = 0$, $\frac{F}{r_0}$ for $s = 1$, $-\frac{F}{r_0}$ for $s = 2$, and $-\frac{r_0}{F}$ for $s = 3$.

Relative Error Control

Let $x = (2^n)m$. If the case number s above is 1 or 2 and if the reduced argument r_0 is less than 2^{-26+n} (with the exception of cotangent entry with small arguments), an execution error is signaled. In such a case, when the argument is so close to a singularity that the minimal indeterminary of the argument (due to prerounding) can cause a relative error of up to ⅓. No screening is given for arguments near a zero of the function.

Control can be strengthened or eliminated by the use of the subroutine FMTN.

If $|x| \geqq 2^{20}$ or if cotangent is asked with $|x| < 2^{-126}$, an execution error is also signaled.

Effect of Argument Error

$E \sim \Delta/\cos^2 x$, $\epsilon \sim 2\Delta/\sin 2x$ for tan x. Thus, near the singularities $x = (k + \frac{1}{2})\pi$, where k is an integer, neither absolute error control nor relative error control can be maintained. This is also true for cotan x, where $x \cong k\pi$ and k is an integer.

Performance Statistics

Performance statistics for the tangent function of the single-precision tangent/cotangent subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$\lvert x\rvert \leqslant \frac{\pi}{4}$	4.02×10^{-9}	1.23×10^{-8}	392	252
$\frac{\pi}{4} < \lvert x\rvert \leqslant \frac{\pi}{2}$	5.48×10^{-9}	(6.45×10^{-8})*	470	304
$\frac{\pi}{2} < \lvert x\rvert \leqslant 10$	6.40×10^{-9}	(8.53×10^{-8})*	457	295
$10 < \lvert x\rvert \leqslant 100$	6.85×10^{-8}	(9.97×10^{-8})*	458	296

*Note: The figures cited as the maximum relative errors are those encountered among 2500 random samples in the respective ranges. For all the perfect arguments in the full range (legitimate under the standard error control), the maximum relative error is estimated to be 2×10^{-4}.

Performance statistics for the cotangent function of the tangent/cotangent subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$\lvert x\rvert \leqslant \frac{\pi}{4}$	4.75×10^{-9}	1.43×10^{-8}	408	264
$\frac{\pi}{4} < \lvert x\rvert \leqslant \frac{3\pi}{4}$	1.23×10^{-8}	(5.39×10^{-7})*	459	298
$\frac{3\pi}{4} < \lvert x\rvert \leqslant 10$	6.37×10^{-9}	(2.17×10^{-6})*	470	306

*Note: The figures cited as the maximum relative errors are those encountered among 2500 random samples in the respective ranges. For all the perfect arguments in the full range (legitimate under the standard error control), the maximum relative error is estimated to be 2×10^{-4}.

The sample arguments upon which the above statistics are based were distributed uniformly over the specified range.

Arctangent — FATN

Algorithm

1. Use

$$\arctan(-x) = -\arctan(x),$$

$$\arctan\left(\frac{1}{|x|}\right) = \frac{\pi}{2} - \arctan(|x|)$$

Assume $0 \leq x \leq 1$.

2. If $[\tan(15°)] \leq x \leq 1$, reduce further to the range $|x| \leq [\tan(15°)]$ by using $\arctan(x) = 30° + \arctan(x)$, where $x = \sqrt{3} - \dfrac{4}{x + \sqrt{3}}$.

3. By transforming the Taylor series into a continued fraction, we obtain

$$\arctan(x) = x \left[1 - \frac{1}{3}x^2 + \frac{\frac{1}{5}x^2}{\frac{5}{7} + x^{-2} - \dfrac{\frac{(4)(5)}{(7)(7)(9)}}{\frac{43}{7.11} + x^{-2} + w}} \right],$$

where w abbreviates further terms.

Dropping w and rewriting the formula, we get

$$\arctan(x) = x \left[-\frac{64}{(3)(5^2)(7^2)}x^2 + \frac{(41)(64)}{(5^3)(7^2)} + \frac{\frac{(3^5)(13)(79)}{(5^4)(7^3)}}{x^2 + \frac{34063}{(3)(5)(13)(79)} - \dfrac{\frac{(14641)(1100)}{(3^2)(7)(13^2)(79^2)}}{x^2 + \frac{(11)(389)}{(3)(13)(79)}}} \right].$$

For $|x| \leq [\tan(15°)]$, the maximum relative error of this approximation is 6×10^{-11}.

4. Fixed-point computation is used to minimize truncation errors.

5. ATAN2 provides the extended answer range $-\pi < y \leq \pi$, depending on the combination of signs of the two arguments.

Effect of Argument Error

$E \sim \Delta/(1 + x^2)$. For small x, $\epsilon \sim \delta$; and as x becomes large, the effect of δ on ϵ diminishes.

Performance Statistics

Performance statistics for the single-precision arctangent subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
The entire range	2.93×10^{-9}	1.39×10^{-8}	419	269

The sample arguments upon which the above statistics are based were tangents of uniformly distributed numbers between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ (i.e., the argument sample was such that function values were uniformly distributed over the answer range).

Arcsine/Arccosine — FASC

Algorithm

1. If $0 \leq x \leq \frac{1}{2}$, compute $\arcsin(x)$ by use of the Chebyshev interpolation polynomial of degree 5 over this range.

2. If $\frac{1}{2} < x \leq 1$,

$$\arcsin(x) = \frac{\pi}{2} - 2\left[\arcsin\left(\sqrt{\frac{1-x}{2}}\right)\right].$$

Since in this range we have $0 \leq \sqrt{\frac{1-x}{2}} < \frac{1}{2}$, this case is reduced to that of item 1 above.

3. If $0 \leq x \leq 1$,

$$\arccos(x) = \frac{\pi}{2} - \arcsin(x).$$

This reduces the case of arccosine to that of arcsine.

4. If $-1 \leq x < 0$, use $\arcsin(x) = -\arcsin(-x)$ and $\arccos(x) = \pi - \arccos(-x)$ to reduce to the earlier cases.

5. The routine FSQR is used in item 2 above.

Effect of Argument Error.

$E \sim \Delta/\sqrt{1-x^2}$. Thus, for small x, $E \sim \Delta$. For ARSIN with small x, $\epsilon \sim \delta$. Toward the limit of the range, a small argument error causes a substantial error in the answer.

Performance Statistics

Performance statistics for the arcsine function of the single-precision arcsine/arccosine subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$\lvert x\rvert \leq 1$	5.47×10^{-9}	3.15×10^{-8}	533	286

Performance statistics for the arccosine function of the single-precision arcsine/arccosine subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$\lvert x\rvert \leq 1$	5.54×10^{-9}	1.98×10^{-8}	545	291

The sample arguments upon which the above statistics are based were uniformly distributed over the specified range.

Hyperbolic Sine/Cosine — FSCH

Algorithm

1. $\cosh(x) = \frac{e^x + e^{-x}}{2}$.

2. If $|x| > 0.3465736$, use

$$\sinh(x) = \frac{e^x - e^{-x}}{2}.$$

3. If $|x| \leq 0.3465736$, use

$$\sinh(x) \cong x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!}.$$

The maximum relative error of this approximation is 6×10^{-10}.

4. This routine uses the subroutine FXPF.

Effect of Argument Error

For the hyperbolic sine function,

$$E \sim \Delta \cdot \cosh x + \frac{\Delta^2}{2} \sinh x, \epsilon \sim \Delta \cdot \coth x + \frac{2}{\Delta^2}.$$

For the hyperbolic cosine function,

$$E \sim \Delta \cdot \sinh x + \frac{\Delta^2}{2} \cosh x, \epsilon \sim \Delta \cdot \tanh x + \frac{\Delta^2}{2}.$$

In particular, for the hyperbolic cosine function, $\epsilon \sim \Delta$ over the entire range. On the other hand, for the hyperbolic sine function with small value of x, $\epsilon \sim \delta$.

Performance Statistics

Performance statistics for the hyperbolic sine function of the single-precision hyperbolic sine/cosine subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$\lvert x\rvert \leqslant 0.3466$	5.36×10^{-9}	1.42×10^{-8}	262	153
$0.3466 < \lvert x\rvert \leqslant 10$	4.79×10^{-9}	2.61×10^{-8}	446	299

Performance statistics for the hyperbolic cosine function of the single-precision hyperbolic sine/cosine subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$\lvert x\rvert \leqslant 10$	4.41×10^{-9}	1.35×10^{-8}	426	285

The sample arguments upon which the above statistics are based were uniformly distributed over the specified range.

Hyperbolic Tangent — FTNH

Algorithm

1. For $|x| < 0.5493$, use the modified continued fraction

$$\tanh(x) = \frac{x}{1} + \frac{x^2}{3} + \frac{x^2}{5} + \frac{x^2}{7} + \frac{x^2}{9.02743}.$$

The maximum relative error of this approximation is 4×10^{-9}.

2. For $0.5493 \leq x < 10.4$, use

$$\tanh(x) = 1 - \frac{2}{e^{2x} + 1}.$$

The routine FXPF is used in this step.

3. For $10.4 \leq x$, give
   $\tanh(x) = 1.$
4. For $x \leq -0.5493$, use
   $\tanh(-x) = -\tanh(x).$

Effect of an Argument Error

$E \sim (1 - \tanh^2 x)\,\Delta$, $\epsilon \sim 2\,\Delta/\sinh 2x$. Thus, for small x, $\epsilon \sim \delta$, and as x gets larger, the effect of δ on ϵ diminishes.

Performance Statistics

Performance statistics for the single-precision hyperbolic tangent subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$\lvert x\rvert \leqslant 0.5493$	5.70×10^{-9}	1.43×10^{-8}	349	228
$0.5493 < \lvert x\rvert \leqslant 10.4$	2.63×10^{-9}	1.45×10^{-8}	438	292

The sample arguments upon which the above statistics are based were uniformly distributed over the specified range.

Error Function — FERF

Algorithm

1. $\mathrm{Erf}(-x) = -\mathrm{Erf}(x)$.

   Assume $x \geq 0$ now.

2. If $x > 4.17$,

   $\mathrm{Erf}(x) \cong 1$.

3. If $4.17 \geq x > 1.51$, use the following Gaussian type continued fraction:

$$1 - \mathrm{Erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-u^2}\,du$$

$$\int_x^\infty e^{-u^2}\,du = e^{-x^2}\left[\frac{0.5\,x}{x^2 + 0.5 -}\ \frac{0.5}{x^2 + 2.5 -}\ \frac{3}{x^2 + 4.5 -}\ \cdots\right.$$

$$\left.\frac{(n - \tfrac{1}{2})n}{-x^2 + 2n + \tfrac{1}{2} -}\ \cdots\right]$$

$$\cong e^{-x^2}\left[\frac{0.5\,x}{x^2 + 0.5 -}\ \frac{0.5}{x^2 + 2.5 -}\ \frac{3}{x^2 + 4.5 -}\ \frac{7.5}{x^2 + 6.5 -}\ \frac{10.803}{x^2 + 4.269}\right]. \quad (*)$$

The two constants in the last term are obtained by requiring that the formula give the exact values at $x = \sqrt{2.3125}$ and $x = \sqrt{2.75}$.
The maximum absolute error of this approximation (*) is 1.1×10^{-9}.

4. If $1.51 \geq x \geq 0$, use the continued fraction obtained in the following way: Transform the Taylor expansion of Erf into a continued fraction (Wall, page 196) as follows:

$$\frac{\sqrt{\pi}}{2x}\left[\mathrm{Erf}(x)\right] = 1 - \frac{x^2}{3} + \frac{x^4}{2!5} - \frac{x^6}{3!7} + \frac{x^8}{4!9} - \cdots$$

$$= 1 - \frac{1.0281x^2}{x^2 + 10.216} + \frac{-167.17}{x^2 + 9.8103} + \frac{201.39}{x^2 + 11.570}$$

$$+ \frac{31.228}{x^2 - 1.7730} + \frac{64.244}{x^2 + 5.578 + w(x)}.$$

The constants in the last formula are approximate.
Finally, drop $w(x)$ and instead modify the two constants of the last term so that the formula is exact at $x = 1$ and $x = \sqrt{2.}$ The maximum relative error of the formula obtained this way is 2×10^{-9}.

5. Fixed-point computation is employed to minimize truncation errors. This routine uses the exponential subroutine FXPF.

Effect of Argument Error

$E \sim \Delta \cdot e^{-x^2}$. As the magnitude of the argument increases away from 1, the effect of an argument error on the final accuracy diminishes rapidly. For small x, $\epsilon \sim \delta$.

Performance Statistics

Performance statistics for the single-precision error function subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$\|x\| \leq 4$	3.20×10^{-9}	1.27×10^{-8}	681	438

The sample arguments upon which the above statistics are based were uniformly distributed over the specified range.

Gamma/Loggamma — FGAM

Algorithm

1. If $0 < x \leq 2^{-127}$, for ALGAMA use
   $\log[\Gamma(x)] \cong -\log(x)$.
2. If $2^{-127} < x < 4$, reduce the case to $1 \leq x \leq 2$, using
   $x[\Gamma(x)] = \Gamma(x+1)$.

Compute $\Gamma(x)$ for $1 \leq x \leq 2$, using a continued fraction obtained in the following manner:

For $1 \leq x \leq 2$,

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt$$

$$= \sum_{k=0}^{\infty} \frac{2^{k+1} a k}{k!} (x-1.5)^k \quad (*)$$

where: $a_k = \int_0^\infty [\log(t)]^k (e^{-t^2}) t^2 dt.$

$a_0 = \frac{1}{4}\sqrt{\pi}$,

$a_1 = 0.80845993 \times 10^{-2}$,

$a_{21} = -0.1628 \times 10^{10}$.

By transforming the formula $(*)$ into a continued fraction, we obtain

$$\Gamma(z+1.5) = \frac{\alpha_1}{z+\beta_1} + \frac{\alpha_2}{z+\beta_2} + \frac{\alpha_3}{z+\beta_3} + \frac{\alpha_4}{z+\beta_4} + \frac{\alpha_5}{z+\beta_5+w(z)} \quad (**)$$

where:

$\alpha_1 =$	-1.2581927	$\beta_1 =$	-11.746649
$\alpha_2 =$	33.814358	$\beta_2 =$	-4.8326355
$\alpha_3 =$	59.285853	$\beta_3 =$	8.1171874
$\alpha_4 =$	-3.6512651	$\beta_4 =$	0.20317850
$\alpha_5 =$	6.0985782	$\beta_5 =$	1.4063097

Finally, drop $w(z)$ in formula $(**)$ and compensate for it by modifying the constants β_4, α_5, β_5 to obtain an approximation formula accurate to within absolute error of 2.3×10^{-9}.

3. If $2^{-127} < x < 4$ for ALGAMA, compute $\log[\Gamma(x)]$ by first computing $\Gamma(x)$ as in item 2 and then taking the logarithm of the result.
4. If $4 \leq x < 1.54926 \times 2^{120}$, compute $\log[\Gamma(x)]$ as follows:
   $\log[\Gamma(x)] \cong x[\log(x) - 1] - \frac{1}{2}\log(x) + \frac{1}{2}\log(2\pi) + F(x)$

where:

$F(x) \cong 0$, if $x \geq 2^{12}$.

$$F(x) \cong \frac{1}{12x} - \frac{1}{360x^3} + \frac{1}{1260x^5} - \frac{1}{1760x^7}, \text{ if } x < 2^{12}.$$

This formula is the result of economizing Stirling's asymptotic series. For the range considered, its absolute error is less than 2.1×10^{-9}.

5. If $4 \leq x < 34.843$, compute $\Gamma(x)$ by first computing $\log[\Gamma(x)]$ as in item 4 and then taking its exponential base e.

6. The routines FLOG and FXPF are used by this subroutine.

Effect of Argument Error

For $\Gamma(x)$, $\epsilon \sim \Psi(x) \cdot \Delta$, and for $\log \Gamma(x)$, $E \sim \Psi(x) \cdot \Delta$, where Ψ is the digamma function.

For $\frac{1}{2} < x < 3$, $-2 < \Psi(x) < 1$, and $E \sim \Delta$ for $\log \Gamma(x)$.

However, since $x = 1$ and $x = 2$ are zeros of $\log \Gamma(x)$, even a small δ can cause a substantial ϵ in this interval.

For large values of x, $\Psi(x) \sim \log x$. Hence, for $\Gamma(x)$, $\epsilon \sim \delta \cdot x \cdot \log x$. This shows that even the round-off error of the argument contributes greatly to the relative error of the answer. On the other hand, for $\log \Gamma(x)$ with large value of x, $\epsilon \sim \delta$.

Performance Statistics

Performance statistics for the gamma function of the single-precision gamma/loggamma subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$2^{-127} < x < 1$	4.55×10^{-9}	1.22×10^{-8}	474	316
$1 \leq x < 2$	2.51×10^{-9}	6.16×10^{-9}	435	289
$2 \leq x < 4$	4.76×10^{-9}	1.59×10^{-8}	519	335
$4 \leq x < 10$	6.03×10^{-8}	2.26×10^{-7}	1110	676
$10 \leq x < 34$	3.18×10^{-7}	1.27×10^{-6}	1110	678

Performance statistics for the loggamma function of the single-precision gamma/loggamma subroutine are as follows:

Argument Range	Root-Mean-Square Absolute Error $\sigma(E)$	Maximum Absolute Error $M(E)$	Average Speed in Microseconds	
			7090	7094
$\frac{1}{2} < x < 3$	1.02×2^{-28}	1.72×2^{-27}	858	532

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$0 < x \leq \frac{1}{2}$	5.09×10^{-9}	1.72×10^{-8}	871	544
$3 \leq x < 4$	5.87×10^{-9}	2.43×10^{-8}	945	579
$4 \leq x < 10$	7.81×10^{-9}	2.67×10^{-8}	808	480
$10 \leq x < 100$	6.53×10^{-9}	2.09×10^{-8}	811	485

The sample arguments upon which the above statistics are based were uniformly distributed over the specified range.

Double-Precision Subroutines

The following information describes double-precision subroutines listed in Figure 25B.

Square Root — FDSQ

Algorithm

1. Write

$x = (2^{2p-q})m$, where p is an integer, $q = 0$ or 1, and $\frac{1}{2} \leq m < 1$.

Then

$$\sqrt{x} = 2^p\left[\sqrt{(2^{-q})m}\right].$$

2. Take the first approximation y_0 to be

$$y_0 = 2^p\left(\frac{m}{2} + \frac{1}{2}\right), \text{ if } q = 0.$$

$$y_0 = 2^p\left(\frac{m}{2} + \frac{1}{4}\right), \text{ if } q = 1.$$

The relative error of y_0 is less than 2^{-4}.

3. Apply the Newton-Raphson iteration

$y_{n+1} = \frac{1}{2}\left(y_n + \frac{x}{y_n}\right)$ 4 times to y_0, 3 times in single precision and the last time in double precision:

The maximum relative error of y_3 is 2^{-53}.

Effect of Argument Error

$\epsilon \sim \frac{1}{2}\,\delta$.

Performance Statistics

Performance statistics for the double-precision square root subroutine are as follows:

Argument Range		Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
				7090	7094
$10^{-30} \leq x < 10^{37}$	7090	4.26×10^{-17}	1.59×10^{-16}	448	
	7094	4.03×10^{-17}	1.56×10^{-16}		220
*For arguments that are less than 10^{-30}, speed is substantially slower because of floating-point underflow. For accuracy in this range, see the section "Alternate 7094 Floating-Point Trap Subroutine."					

The sample arguments on which the above statistics are based were exponentially distributed over the specified range.

Exponential — FDXP

Algorithm

1. $e^x = 2^y$, where $y = x(\log_2 e)$.

Write

$y = y_1 + y_2$, where y_1 is the integer part and y_2 is the fraction part.

Define

$z_1 = y_1, \quad z_2 = y_2, \quad \text{if } y \geq 0.$

$z_1 = y_1 - 1, \quad z_2 = y_2 + 1, \quad \text{if } y < 0.$

Then

$2^y = 2^{z_1}(2^{z_2})$, where z_1 is an integer and $0 \leq z^2 \leq 1$.

2. 2^{z_2} for $0 \leq z_2 \leq 1$ is computed by the use of the Chebyshev interpolation polynomial of degree 11 for the interval. The maximum relative error of this polynomial is 2^{-57}.

3. If $x \leq -89.415987$, 0 is given as the answer.

Effect of Argument Error

$\epsilon \sim \Delta$. Since $\Delta = \delta \cdot x$, for the larger value of x, even the round-off error of the argument causes a substantial relative error in the answer.

Performance Statistics

Performance statistics for the double-precision exponential subroutine are as follows:

Argument Range		Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
				7090	7094
$0 \leqslant x \leqslant 1$	7090	4.24×10^{-17}	1.68×10^{-16}	3016	
	7094	3.74×10^{-17}	1.07×10^{-16}		559
$-70.0 < x < 88.028$	7090	4.82×10^{-17}	1.96×10^{-16}	3021	
	7094	4.38×10^{-17}	1.39×10^{-16}		564

The sample arguments on which the above statistics are based were uniformly distributed over the specified range.

Logarithm — FDLG

Algorithm

1. Write

$x = (2^n)m$, where n is the exponent and $\frac{1}{2} \leq m < 1$.

Define the base value F as

$F = \frac{1}{2}$, where $\frac{1}{2} \leq m < \frac{\sqrt{2}}{2}$.

$F = 1$, where $\frac{\sqrt{2}}{2} \leq m < 1$.

Let $z = \frac{m - F}{m + F}$. Then

$m = F\left(\frac{1 + z}{1 - z}\right)$ and $|z| \leq 0.1716$.

$\log(x) = n[\log(2)] + \log(F) + \log\left(\frac{1 + z}{1 - z}\right)$.

$\log(F) = -\log(2)$, where $\frac{1}{2} \leq m < \frac{\sqrt{2}}{2}$.

$\log(F) = 0$, where $\frac{\sqrt{2}}{2} \leq m < 1$.

2. $\log\left(\frac{1 + z}{1 - z}\right) = 2z\left(1 + \frac{z^2}{3} + \frac{z^4}{5} + \ldots\right) \cong z(c_0 + c_1 z^2 + \ldots + c_7 z^{14})$,

where coefficients $c_0, c_1, \ldots, c_7$ are obtained by Chebyshev interpolation.

The maximum relative error of this approximation is $2^{-59.5}$.

Effect of Argument Error

$E \sim \delta$. In particular, if δ is the round-off error of the argument, say $\delta \sim 5.6 \times 10^{-17}$, then $E \sim 5.6 \times 10^{-17}$. This means that if the argument is close to 1, the relative error can be very large, since the function value is very small at that point.

Performance Statistics

Performance statistics for the natural logarithm function of the double-precision logarithm subroutine are as follows:

Argument Range		Root-Mean-Square Absolute Error $\sigma(E)$	Maximum Absolute Error $M(E)$	Average Speed in Microseconds	
				7090	7094
$\frac{1}{2} \leqslant x \leqslant 2$	7090	1.30×2^{-55}	1.64×2^{-53}	2723	
	7094	1.76×2^{-56}	1.89×2^{-54}		463

The sample arguments on which the above statistics are based were uniformly distributed over the specified range.

Argument Range		Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
				7090	7094
Full range excluding the interval $(\frac{1}{2}, 2)$	7090	9.65×10^{-17}	2.95×10^{-16}	2736	
	7094	6.09×10^{-17}	1.80×10^{-16}		474

The sample arguments on which the above statistics are based were exponentially distributed over the specified range.

Performance statistics for the common logarithm function of the double-precision logarithm subroutine are as follows:

Argument Range		Root-Mean-Square Absolute Error $\sigma(E)$	Maximum Absolute Error $M(E)$	Average Speed in Microseconds	
				7090	7094
$\frac{1}{2} < x < 2$	7090	1.84×2^{-56}	1.12×2^{-53}	2886	
	7094	1.66×2^{-56}	1.64×2^{-54}		489

The sample arguments upon which the above statistics are based were uniformly distributed over the specified range.

Argument Range		Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
				7090	7094
All positive numbers outside $(\frac{1}{2}, 2)$	7090	1.18×10^{-16}	4.46×10^{-16}	2899	
	7094	9.12×10^{-17}	3.31×10^{-16}		500

The sample arguments upon which the above statistics are based were exponentially distributed over the specified range.

Sine/Cosine — FDSC

Algorithm

1. If $\cos(x)$ is desired, reduce the case to a sine function by
   $\cos(x) = \sin\left(\frac{\pi}{2} - x\right)$.
2. If $x < 0$, use
   $\sin(-x) = -\sin(x)$.

Assume $x \geq 0$. If $|x| < 2^{-26}$, give x as the answer.

3. Divide x by $\frac{\pi}{4}$ and separate the integer part q_1 and the fraction part q_2 of the quotient.

Let $q_0 = q_1$ [modulo 8]. Then

$$\sin(x) = \sin\left(\frac{\pi}{4} q_0 + \frac{\pi}{4} q_2\right).$$

4. Further reduce the case to the computation of either $\sin\left(\frac{\pi}{4}r\right)$ or $\cos\left(\frac{\pi}{4}r\right)$ with $0 \leq r \leq 1$ by the formulas

$$\sin\left[\frac{\pi}{4}(4+q_0)+\frac{\pi}{4}q_2\right] = -\sin\left(\frac{\pi}{4}q_0+\frac{\pi}{4}q_2\right), \text{ where } 0 \leq q_0 < 3.$$

$$\sin\left(\frac{\pi}{4}+\frac{\pi}{4}q_2\right) = \cos\left[\frac{\pi}{4}(1-q_2)\right].$$

$$\sin\left(\frac{\pi}{2}+\frac{\pi}{4}q_2\right) = \cos\left(\frac{\pi}{4}q_2\right).$$

$$\sin\left(\frac{3\pi}{4}+\frac{\pi}{4}q_2\right) = \sin\left[\frac{\pi}{4}(1-q_2)\right].$$

5. For $0 \leq r \leq 1$,

$$\sin\left(\frac{\pi}{4}r\right) = S_0r + S_1r^3 + S_2r^5 + \ldots. + S_6r^{13}.$$

$$\cos\left(\frac{\pi}{4}r\right) = 1 + C_1r^2 + C_2r^4 + \ldots. + C_6r^{12}.$$

Coefficients $S_0, \ldots, S_6$ are obtained by Chebyshev interpolation over the range $0 \leq r \leq 1$, and the coefficients $C_0, \ldots, C_6$ are obtained by Chebyshev interpolation over the range $-0.01 \leq r \leq 1$.
The maximum relative error of the sine approximation is 2^{-58}. The maximum relative error of the cosine approximation is 2^{-54}.

Effect of Argument Error
$E \sim \Delta$. As the argument gets larger, Δ grows, and since the function value is periodically diminishing, no consistent relative error control can be maintained outside the principal range $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$. This observation holds true for cosine as well.

Performance Statistics
Performance statistics for the sine function of the double-precision sine/cosine subroutine are as follows:

Argument Range		Root-Mean-Square Absolute Error $\sigma(E)$	Maximum Absolute Error $M(E)$	Average Speed in Microseconds	
				7090	7094
$\lvert x\rvert \leq \frac{\pi}{2}$	7090	1.85×2^{-55}	1.75×2^{-53}	2141	
	7094	1.12×2^{-54}	2^{-52}		414
$\frac{\pi}{2} < \lvert x\rvert \leq 10$	7090	1.63×2^{-52}	1.52×2^{-49}	2184	
	7094	1.16×2^{-52}	1.32×2^{-50}		441
$10 < \lvert x\rvert \leq 100$	7090	1.50×2^{-49}	1.42×2^{-46}	2180	
	7094	1.08×2^{-49}	1.21×2^{-47}		441

Argument Range		Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
				7090	7094
$\lvert x\rvert \leq \frac{\pi}{2}$	7090	9.14×10^{-17}	4.72×10^{-16}	2141	
	7094	9.08×10^{-17}	3.81×10^{-16}		414

The sample arguments on which the above statistics are based were uniformly distributed over the specified range.

Performance statistics for the cosine function of the double-precision sine/cosine subroutine are as follows:

Argument Range		Root-Mean-Square Absolute Error $\sigma(E)$	Maximum Absolute Error $M(E)$	Average Speed in Microseconds	
				7090	7094
$0 \leqslant x \leqslant \pi$	7090	1.26×2^{-54}	1.05×2^{-52}	2241	
	7094	1.52×2^{-54}	1.14×2^{-52}		427
$-20 \leqslant x < 0$ and $\pi < x \leqslant 20$	7090	1.21×2^{-50}	1.57×2^{-48}	2280	
	7094	1.63×2^{-52}	1.33×2^{-49}		453

The sample arguments on which the above statistics are based were uniformly distributed over the specified range.

Arctangent — FDAT

Algorithm

1. Reduce the general case to $0 \leq x \leq 1$ by using the following formulas:

$$\arctan(-x) = -\arctan(x).$$

$$\arctan\left(\frac{1}{|x|}\right) = \frac{\pi}{2} - \arctan(|x|).$$

2. Then reduce the case further to $|x| \leq \tan(15°)$ by using

$$\arctan(x) = 30° + \arctan\left(\frac{(\sqrt{3})x - 1}{x + \sqrt{3}}\right), \text{ if } \tan(15°) < x < \tan(45°).$$

3. For the basic range $|x| \leq \tan(15°)$, a continued fraction approximation of the following form is used:

$$\frac{\arctan(x)}{x} = 1 + \frac{\alpha_1 x^2}{\beta_1 + x^2} + \frac{\alpha_2}{\beta_2 + x^2} + \frac{\alpha_3}{\beta_3 + x^2} + \frac{\alpha_4}{\beta_4 + x^2}. \quad (*)$$

This formula can be derived by transforming the Taylor series into the following fraction:

$$\frac{\arctan(x)}{x} = 1 - \frac{\frac{1}{3}x^2}{\frac{3}{5} + x^{-2}} - \frac{\frac{(3)(4)}{(5^2)(7)}}{\frac{23}{(5)(9)} + x^{-2}} - \frac{\frac{(5^2)(2^4)}{(7)(9^2)(11)}}{\frac{59}{(9)(13)} + x^{-2}}$$
$$- \frac{\frac{(4)(7^2)(3)}{(5)(11)(13^2)}}{\frac{(3)(37)}{(13)(17)} + x^{-2} + w}.$$

As the approximation of the value $w = w(x)$, pick -0.00398. When we rewrite the above fraction with this value of w, we obtain the formula $(*)$.
The maximum relative error of the formula $(*)$ is less than 2^{-55}.

4. DATAN2 provides the extended answer range $-\pi < y \leq \pi$, depending on the combination of signs of the two arguments.

Effect of Argument Error:

$E \sim \Delta/(1 + x^2)$. For small x, $\epsilon \sim \delta$; and as x becomes large, the effect of δ on ϵ diminishes.

Performance Statistics

Performance statistics for the double-precision arctangent subroutine are as follows:

Argument Range		Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error M (E)	Average Speed in Microseconds	
				7090	7094
Full range	7090	6.47×10^{-17}	2.21×10^{-16}	2739	
	7094	6.61×10^{-17}	3.92×10^{-16}		548

The sample arguments on which the above statistics are based were tangents of random numbers uniformly distributed over $-\frac{\pi}{2}$, $\frac{\pi}{2}$ (i.e., the argument sample was such that the function values were uniformly distributed over the answer range).

Complex Subroutines

The following information describes complex subroutines listed in Figure 25C.

Square Root — FCSQ

Algorithm

1. Write

$$\sqrt{x+iy} = \xi + i\,\eta.$$

2. If $x \geq +0$, use

$$\xi = \sqrt{\frac{|x| + |x+iy|}{2}}.$$

$$\eta = \frac{y}{2\xi}.$$

3. If $x \leq -0$, use

$$\xi = \frac{y}{2\eta}.$$

$$\eta = \operatorname{sgn}(y)\sqrt{\frac{|x| + |x+iy|}{2}}.$$

Thus, if $x < 0$, the case $y = -0$ is differentiated from the case $y = +0$. That is,

$$\sqrt{x-0i} = \lim_{\epsilon \to +0} \sqrt{x-\epsilon i}.$$

$$\sqrt{x+0i} = \lim_{\epsilon \to +0} \sqrt{x+\epsilon i}.$$

4. If in the foregoing computation $\frac{|x| + |x+iy|}{2} < 2^{-129}$, then $0 + 0i$ is given as the answer.

5. The routines FSQR and FCAB are used by this subroutine.

Limitation: If $|x| + |x+iy| \geq 2^{127}$, floating-point overflow is caused.

Effect of Argument Error

If $x + iy = r \cdot e^{ih}$ and $\sqrt{x} + iy = R \cdot e^{iH}$, then $\epsilon(R) \sim \frac{1}{2}\,\delta(r)$ and $\epsilon(H) \sim \delta(h)$.

Performance Statistics

Performance statistics for the complex square root subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error M (ϵ)	Average Speed in Microseconds	
			7090	7094
$10^{-30} \leq \lvert x_1 + ix_2 \rvert \leq 10^{37}$	4.31×10^{-9}	1.44×10^{-8}	821	503

The distribution of sample arguments upon which the above statistics are based is radially exponential and is uniform around the origin.

Exponential — FCXP

Algorithm

1. $e^{x+iy} = e^x \cdot \cos(y) + ie^x \cdot \sin(y)$.
2. The routines FXPF and FSCN are used by the subroutine.

Effect of Argument Error

If $e^{x+iy} = \mathrm{R}(e^{iH})$, then $H = y$ and $\epsilon(R) \sim \Delta(x)$.

Performance Statistics

Performance statistics for the complex exponential subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $\mathrm{M}(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$\lvert x_1\rvert \leqslant 85, \lvert x_2\rvert \leqslant 10$	6.64×10^{-9}	1.97×10^{-8}	1345	835

The sample arguments on which the above statistics are based were uniformly distributed over the specified range.

Natural Logarithm — FCLG

Algorithm

1. Write

$$\log(x + iy) = \xi + i\eta.$$

Then in the sense of ATAN2,

$$\xi = \log(|x + iy|) \text{ and}$$

$$\eta = \arctan \frac{y}{x}.$$

2. This routine differentiates the argument $x - 0i$ from the argument $x + 0i$, That is,

$$\log(x - 0i) = \lim_{\epsilon \to +0} \log(x - \epsilon i).$$

$$\log(x + 0i) = \lim_{\epsilon \to +0} \log(x + \epsilon i).$$

3. The routines FCAB, FLOG, and FATN are used by this subroutine.

Limitation: If $| x + iy | \geq 2^{127}$, floating-point overflow is caused.

Effect of Argument Error

If $x + iy = r \cdot e^{ih}$ and $\log(x + iy) = \xi + i\eta$, then $h = \eta$ and $E(\xi) \sim \delta(r)$. When the argument is near $1 + 0i$, the answer is almost 0, and therefore a small δ can cause a large ξ.

Performance Statistics

Performance statistics for the complex logarithm subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $\mathrm{M}(\epsilon)$	Average Speed in Microseconds	
			7090	7094
Full range, except the immediate neighborhood of $1+0i$.	3.45×10^{-9}	3.03×10^{-8}	1469	906

The distribution of sample arguments on which the above statistics are based is radially exponential and is uniform around the origin.

Sine/Cosine —FCSC

Algorithm:

1. $\sin(x + iy) = [\sin(x)]\,[\cosh(y)] + [i]\,[\cos(x)]\,[\sinh(y)]$
   $\cos(x + iy) = [\cos(x)]\,[\cosh(y)] - [i]\,[\sin(x)]\,[\sinh(y)]$
2. The routines FSCN and FSCH are used by the subroutine.

Effect of Argument Error

Combine the effects of the sine and cosine functions of the single-precision sine/cosine subroutine with the hyperbolic sine and cosine functions of the single-precision hyperbolic sine/cosine subroutine according to step 1, in the algorithm above.

Performance Statistics

Performance statistics for the sine function of the complex sine/cosine subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error M (ϵ)	Average Speed in Microseconds	
			7090	7094
$\lvert x_1\rvert \leqslant 10, \lvert x_2\rvert \leqslant 1$	7.53×10^{-9}	(2.81×10^{-8})	1868	1169

Note: The maximum relative error cited here is based upon a set of 2500 random samples in the range. In the immediate neighborhood of the points $n\pi + 0i$, $n = \pm 1, \pm 2, \ldots$, the relative error can be quite high, although the absolute errors are small there. For the same value of x_2, the relative error increases with $\lvert x_1\rvert$. For the same value of x_1, the relative error stays substantially constant as $\lvert x_2\rvert$ increases.

Performance Statistics

Performance statistics for the cosine function of the complex sine/cosine subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error M (ϵ)	Average Speed in Microseconds	
			7090	7094
$\lvert x_1\rvert \leqslant 10, \lvert x_2\rvert \leqslant 1$	7.44×10^{-9}	(2.72×10^{-8})	1871	1169

Note: The maximum relative error cited here is based upon a set of 2500 random samples in the range. In the immediate neighborhood of the points $(n + 1/2)\pi + 0i$, $n = 0, \pm 1, \pm 2 \ldots$, the relative error can be quite high, although the absolute errors are small there. For the same value of x_2, the relative error increases with $\lvert x_1\rvert$. For the same value of x_1, the relative error stays substantially constant as $\lvert x_2\rvert$ increases.

The sample arguments on which the above statistics are based were uniformly distributed over the specified range.

Absolute Value — FCAB

Algorithm

1. If $\lvert x\rvert \geq \lvert y\rvert$,

$$\lvert x + yi\rvert = \lvert x\rvert \sqrt{1 + \left(\frac{y}{x}\right)^2} + 0i.$$

2. If $\lvert y\rvert > \lvert x\rvert$,

$$\lvert x + yi\rvert = \lvert y\rvert \sqrt{1 + \left(\frac{x}{y}\right)^2} + 0i.$$

3. The routine FSQR is used by this subroutine.

NOTE: If the result is greater than Ω, the floating-point overflow is caused.

Performance Statistics

Performance statistics for the complex absolute value subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\sigma(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
$10^{-30} \leqslant \lvert x_1 + ix_2 \rvert \leqslant 10^{37}$	5.80×10^{-9}	2.44×10^{-8}	421	250

The distribution of sample arguments on which the above statistics are based is radially exponential and is uniform around the origin.

Arithmetic — FCAS

Algorithm

1. $(a + bi) \pm (c + di) = (a \pm c) + (b \pm d)i$
2. $(a + bi) \times (c + di) = (ac - bd) + (ad + bc)i$
3. If $\lvert c \rvert \geq \lvert d \rvert$,

$$(a + bi) \div (c + di) = \frac{a/c + bd/c^2}{1 + (d/c)^2} + \frac{-ad/c^2 + b/c}{1 + (d/c)^2} xi.$$

If $\lvert d \rvert > \lvert c \rvert$, then reduce to the case above by transforming $(a + bi) \div (c + di) = (b - ai) \div (d - ci)$.

NOTE: When the magnitude of the result is close to Ω, a floating-point overflow is possible. When the magnitude of the result is close to the underflow threshold, the accuracy of the answer diminishes.

Performance Statistics

Performance statistics for the multiply function of the complex arithmetic subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\delta(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
For pairs of operands which are away from the overflow or underflow thresholds	1.09×10^{-8}	2.62×10^{-8}	324	174

Performance statistics for the divide function of the complex arithmetic subroutine are as follows:

Argument Range	Root-Mean-Square Relative Error $\delta(\epsilon)$	Maximum Relative Error $M(\epsilon)$	Average Speed in Microseconds	
			7090	7094
For pairs of operands which are away from the overflow or underflow thresholds	1.3×10^{-8}	3.23×10^{-8}	513	299

The distribution of sample operands upon which the above statistics are based is radially exponential and is uniform around the origin.

Miscellaneous Subroutines

The following information describes miscellaneous subroutines listed in Figure 26.

Error Control Modification for the FTNC Subroutine — FMTN

The FMTN subroutine modifies error control of the single precision tangent/cotangent subroutine (FTNC). Its calling sequence is:

```
CALL MTAN(k)
```

When the $k = 0, 1, 2, 3, 4$, or 5, the FTNC subroutine is modified to give a minimum relative accuracy guarantee of $1/(2^{k+2}-1)$ for correctly rounded arguments near the singularities. When k is greater than 5, the FTNC subroutine is modified to suspend the accuracy guarantee feature completely; any argument less than 2^{20} in magnitude (and greater than 2^{-126} for the cotangent function) will be accepted.

The MAP programmer can accomplish the same effect as the FMTN subroutine by coding the following instructions:

1. When the accuracy guarantee is to be modified:

```
CLA     =1
ALS     k
STO     CRIT
```

Here $k = 0, 1, 2, 3, 4$, or 5. Values higher than 5 should not be used.

2. Where the guarantee is to be eliminated:

```
STZ     CRIT
```

7090 Double-Precision Arithmetic Simulator — FDAS

The FDAS subroutine was designed to perform double-precision addition, multiplication, and division for the 7090 double-precision mathematical functions. Since it saves space as a closed subroutine in comparison with macro-expansions of double-precision instructions, it is also useful to the MAP programmer. The FDAS subroutine cannot be called by FORTRAN IV or COBOL programs.

The calling sequence for the FDAS subroutine is:

```
TSX     entry point,4
PZE     AD1,T1
PZE     AD2,T2
```

where the entry points are DFAD for addition, DFMP for multiplication, and DFDP for division. The first operand must be contained in the AC-MQ. AD1 modified by the tag T1 is the core storage address of the high-order portion of the second operand; AD2 modified by the tag T2 is the address of the low-order portion. The FDAS subroutine leaves the result of the computation in the AC-MQ.

NOTE: The FDAS algorithms are similar to the macro-expansions of double-precision instructions. The subroutine always acts as an extension of the calling program, i.e., it does not update system locations. If a floating-point trap occurs during its operation, the name printed in an error message is that of the calling program. FDAS uses only index register 4 and contains no entry point for subtraction.

Appendix I: Storage Requirements for FORTRAN IV Mathematics Library Subroutines

The following chart shows the octal and decimal storage requirements for the FORTRAN IV mathematics subroutines. An asterisk beside the name of a subroutine indicates that, in the 7090 Subroutine Library, double-precision operations for that subroutine will be performed by the FDAS subroutine.

In each pair of figures, the figure to the left of the slash represents the number of storage words used in a 7090. The figure to the right of the slash applies to a 7094.

	STORAGE REQUIREMENTS	
SUBROUTINE NAME	OCTAL	DECIMAL
FASC	131/131	89/89
FATN	232/232	154/154
FCAB	44/40	36/32
FCAS	115/102	77/66
FCLG	61/60	49/48
FCSQ	55/54	45/44
FCSC	163/157	115/111
FCXP	130/125	88/85
FDAS (7090 library only)	46/—	38/—
FDAT*	314/231	204/153
FDLG*	231/170	153/120
FDSQ	100/67	64/55
FDSC*	250/211	168/137
FDXP*	205/170	133/120
FERF	141/141	97/97
FGAM	302/302	194/194
FLOG	177/177	127/127
FMTN	13/13	11/11
FSCH	121/121	81/81
FSCN	173/173	123/123
FSQR	53/53	43/43
FTNC	222/222	146/146
FTNH	72/72	58/58
FXPF	121/121	81/81

Appendix J: Procedure for Using the 7090 Asterisk Deck

The 7090 Asterisk Deck generates asterisks if an insufficient field width is specified in a program using the standard 7090 conversion routines. Output through the standard 7090/7094 conversion routines is thus made consistent with output from the 7094 optional conversion routine.

The deck is used as follows:

1. Mount the 7090/7094 IBSYS Operating System system tape as SYSLB1 on A1.

2. Mount a tape on A3 containing the Subroutine Library as the first file. This tape is made by copying the Subroutine Library from Symbolic Tape 1, where it is the fifth file.

3. Prepare a tape from the 7090 Asterisk Deck and mount the tape as SYSIN1 on A2. Press LOAD TAPE.

4. The following message should occur:

   $* MOUNT SCRATCH TAPE ON A3 WHEN IT UNLOADS.

Remove the tape containing the Subroutine Library from A3 and mount a scratch tape. The new 7090/7094 IBSYS Operating System system tape will be produced on A3 when the job is completed.

5. The following message should now occur:

   $* MOUNT SCRATCH TAPE ON B3 WHEN IT UNLOADS.

Remove B3. This is the new Subroutine Library symbolic tape.

6. Mount a scratch tape on B3 and continue. At the end of job the tape on A3 will contain the new 7090/7094 IBSYS Operating System tape.

Glossary

The definitions in this glossary apply to these terms as they are used in this manual. The reader may also refer to the IBM Reference Manual, *Glossary for Information Processing*, Form C20-8089.

absolute
: A term referring to specific core storage addresses. *Absolute text* and *absolute program* refer to machine language instructions that can be loaded directly into specific areas of core storage for execution.

AC
: See "register."

accumulator
: See "register."

AC-MQ
: See "register."

alternate unit
: An input/output unit that is substituted for another such unit as the result of programmer direction.

application
: Any single use of a subsystem or user's program.

argument
: An independent variable.

assembly
: The process of translating into machine language a program coded in a symbolic language that parallels machine language coding.

Assembler
: See "Macro Assembly Program."

bit
: A binary digit. In 7090/7094 core storage a bit is represented by a magnetic core. If the computer circuits have made the core positive, the bit is a 1; if the core is negative, the bit is 0. A bit can be represented on external storage such as tape.

blank COMMON
: A storage area in upper core to provide temporary storage for data that will be used by several program segments. The blank COMMON area is lost after the job is completed.

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.

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 within core storage itself or to or from an input/output device.

calling sequence
: The instructions and data words that establish the linkage to and from a subroutine.

CALL transfer vector
: The data words in a calling sequence that contain the information used to identify the position of routines whose location is out of the regular sequence of instructions.

chaining
: A technique for associating two or more table entries with each other. A word in each chained entry contains the address of the next entry in the chain.

checksum
: A summation of digits or bits used primarily for checking purposes and summed according to an arbitrary set of rules.

closed subroutine
: A subroutine that is a separate routine. 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 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 component of the IBJOB Processor that translates a COBOL program into MAP language as input to the Macro Assembly Program.

communication word
A permanent storage location used to transfer information from one set of coding to another.

compile
To transform a problem-oriented program into a symbolic language that parallels machine language, or into machine language itself.

component
A part of the IBJOB Processor that is called to perform a specific task. For example, the COBOL Compiler is called to translate a COBOL deck into the MAP language.

constant addend
That part of a subscript which is a constant and is connected to a variable using an additive operation.

control dictionary
The portion of a relocatable binary 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 whose name is entered in the control dictionary for the program deck that contains the control section. It can be referred to from outside this program deck and can be deleted or replaced with a control section from another program deck.

core storage
The form of high-speed main storage using magnetic cores that is found in a 7090/7094 Data Processing System.

debug
To detect, locate, and remove the mistakes from a program.

dictionary
A table of entries that define symbolic names.

double-precision
A term used to describe computations in which the arguments are numbers contained in two adjacent machine words for greater accuracy.

embedded blank
A blank between two characters.

end-of-file
An end-of-file refers to a file mark (tape mark) which signals the end of a file of information on a tape unit.

entry point
A location in a program deck to which control can be transferred from another program deck. Also used to describe the location of the first instruction to be executed in a program.

error-flow trace
A routine used to determine the logical order of program execution that occurred before an error. See the "FORTRAN Utility Library" under "Subroutine Library Information."

even location
A core storage location the address of which is an even number. Core storage locations are numbered 0 through 32,767.

EVEN storage feature
The provision for assigning an even location to instructions, double-precision floating-point operands, or other data to satisfy machine requirements or to increase efficiency of operation.

execution
The computer operation performed in response to program instructions.

external storage for text
Temporary storage areas on a peripheral unit used to contain program instructions which cannot be contained in main storage during loading.

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
Twelve words in storage containing the control information and characteristics of a file to be processed by IOCS.

file dictionary
A table of entries that define files.

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, and 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.

floating-point
A form of notation wherein numbers are represented by a number multiplied by a power of ten. For example, 99.1 would be represented as 9.91×10^1. The portion of the number to the left of the multiplication sign is the mantissa. The portion to the right is the exponent shown as a power of 10. In a 7090/7094 36-bit machine word the exponent of a floating-point

binary number is expressed in the upper eight bits as a power of two and the mantissa is contained in the remaining portion.

FORTRAN
A programming language that closely resembles the ordinary language of mathematics and is designed primarily for scientific and technical applications.

FORTRAN Compiler
A component of the IBJOB Processor that translates FORTRAN IV programs into relocatable binary input to the Loader.

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.

initialize
To set an instruction, counter, switch, or address to a specific starting condition in preparation for operation.

input editor
A part of the input/output editor under the IBJOB Processor which regulates input to the Processor.

intersystem unit
An input/output unit that is to be reserved so that information may be passed between jobs.

job
All card images from one $JOB card up to but not including the next $JOB card. Within a job, one or more applications are executed as a logical unit.

job control
A component of the IBJOB Monitor that supervises the overall operations of the Processor and communicates with the IBSYS System Monitor.

library
An organized collection of operating programs, subroutines, and data.

links, overlay
Segments into which a program can be divided when it exceeds core storage capacity. The main link is resident in core storage while a job is being executed. Dependent links are stored on external storage units and can be brought into core storage when needed. See "Overlay Feature of the Loader."

load
To take information from auxiliary or external storage and place it into core storage. Loading under the IBJOB Processor consists of combining separately assembled program decks and required subroutines, establishing an input/output mechanism for the program, and placing the program in the necessary core storage sequence for execution.

load map
A listing by the Loader of core storage allocation just prior to execution of a program.

Loader
The component under the IBJOB Processor that loads programs.

location counter
A counter that is incremented by one for each word the assembly program generates in the object program.

machine language
The binary data that can be executed or used directly by the processing unit of the 7090/7094.

Macro Assembly Program
The component of the IBJOB Processor that translates MAP language source programs into either relocatable binary machine language as input to the Loader, or absolute binary machine language. Often called the Assembler.

MAP
A symbolic language that closely parallels machine coding on a 7090/7094 machine.

monitor
A program or routine to control operation of several other programs or routines.

MQ
See "register."

nucleus (IBSYS)
The portion of the System Monitor that remains in core storage at all times during use of the operating systems to provide common data areas, pointers, tables, and routines.

object program
The output from an assembler or compiler, usually in machine language.

off-line
A term pertaining to operation of input/output devices or auxiliary equipment not under direct control of the central processing unit.

on-line
A term pertaining to operation of input/output devices under direct control of a program being executed in the central processing unit.

open subroutine
A subroutine that is 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.

operating system
A collection of monitors, subsystems, data control programs, and user's programs that permit unin-

terrupted computer operation during the processing of a variety of jobs.

origin
The address of the beginning of a program section.

overflow
1. During loading: an operation that transfers to an external storage unit the reloctable binary text that exceeds its allocated storage area.
2. During computation: a condition in which a computed result is too large for the register(s) used to contain it.

overlay
The technique of using the same block of core storage for two or more program's segments, called links, that are executed at different times. These links are on external units and are called into core storage by the program when needed.

parameter
A quantity to which arbitrary values may be assigned.

patch
An instruction or group of instructions inserted in a program to correct or change coding temporarily. See "$PATCH card."

peripheral
A term pertaining to operation independent from the main computer.

Polish notation
A notation used in reducing arithmetic expressions into a form suitable for translation by a processor.

prepositioning feature
A routine provided by the Processor Monitor that causes the next needed component to be positioned on the system tape for immediate access.

process control
A component of the Processor Monitor that supervises the assembling and compiling of a source program.

processor
A program that compiles, assembles, and loads a program and usually supervises its execution.

program
1. A plan for the solution of a problem, including data-gathering, processing, and reporting.
2. A group of related routines that solve a given problem.

program deck
A section of coding headed by a component control card and terminated by an "end" card appropriate to the language in which the deck is coded.

program set
A set of contiguous programs in the same job.

pseudo-operation
Any operation available in the MAP language that is not an actual machine operation, special instruction, IOCS operation, prefix code, or macro-operation. It directs the Macro Assembly Program in the process of assembly, rather than the computer in the process of program execution.

qualification
The process of uniquely identifying the symbols defined in a given section of a program by appending another symbol.

relocatable
A term referring to core storage addresses that must be incremented to absolute addresses before use in program execution. A *relocatable binary deck* is a deck coded in machine language that is produced by the FORTRAN IV Compiler or Assembler as Loader input. The deck may consist of up to four sections: control dictionary, file dictionary, text of program instructions, and a dictionary of debugging symbols. The location of each instruction in the text section is numbered relative to the first instruction in the deck. Each reference to an address is also relative. Each relative address is incremented to an absolute address during the loading process.

register
A device used to store data while it is being either processed or transferred from one location to another. The *accumulator* (also called the AC) contains the results of single-precision addition and subtraction. The *multiplier-quotient* (MQ) contains the results of single-precision multiplication and division. When used together the combined registers (AC-MQ) contain the results of double-precision and complex computations. *Index registers* are used to contain quantities for address modification and for purposes such as the determination of loop exits.

routine
A sequence of machine instructions that carry out a specific function.

scatter-loading
A technique for loading sections of a record into different core storage areas. Each record section is headed by a separate input/output command.

single-precision
A term used to describe computations in which each argument is contained in one word.

source program
A program written in a problem-oriented or symbolic language, and which is not in a form that can be directly executed by a computer.

spill
An operation during loading that transfers to an external storage unit all the text of a relocatable binary program. This transfer occurs when the core storage area for text is in danger of being overlaid by storage areas for other data.

statement scanner
A compiler routine which accepts as input a statement written in symbolic language and classifies and performs preliminary processing on the statement.

storage map
A listing by the FORTRAN IV Compiler or Assembler of the relative contents of core storage.

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.

subsystem
A major component of the IBSYS Operating System, such as the IBJOB Processor or the Generalized Sorting Program.

symbol
A character or combination of characters used to represent the address of a storage location, an input/output device, or any other program parameter. An *immediate symbol* is assigned a specific value during the first pass of the assembly program. A *real symbol* is a symbol defined in the deck in which it appears. A *virtual symbol* is a symbol that is not defined in the deck in which it appears.

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.

trailer label
A record with identification and control data related to previous records of a labeled file. It occurs at the end of the file, or at the end of each reel of a multireel labeled file.

trap
An interruption of program execution in response to a specific condition, such as underflow or overflow. A trap is usually followed by a transfer to a routine to investigate and take action on the condition causing the trap.

underflow
A condition in which a computed result is too small for the register(s) used to contain it.

unit control block
A nine-word area of core storage describing an input/output device connected to the computer.

utility unit
A unit that is available for use by the system or by the programmer for any purpose.

word
In 7090/7094 core storage a set of 36 bits that are taken by the computer to express data in binary form. A word can be represented on external storage such as tape.

Index

CUT ALONG LINE

COMMENT SHEET

IBM 7090/7094 IBSYS OPERATING SYSTEM, VERSION 13
IBJOB PROCESSOR

FORM C28-6389-1

FROM

NAME _____

OFFICE NO. _____

FOLD

CHECK ONE OF THE COMMENTS AND EXPLAIN IN THE SPACE PROVIDED

FOLD

☐ SUGGESTED ADDITION (PAGE)

☐ SUGGESTED DELETION (PAGE)

☐ ERROR (PAGE)

EXPLANATION

FOLD

FOLD

NO POSTAGE NECESSARY IF MAILED IN U.S.A.

FOLD ON TWO LINES, STAPLE, AND MAIL

C28-6389-1

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 33504

NEW YORK, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

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

ATTN: PUBLICATIONS, DEPARTMENT D39

CUT ALONG LINE

FOLD

FOLD

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601

STAPLE

Printed in U.S.A. C28-6389-1